2025년도 기본사업 세부(협약)과제 최종보고서

보안등급
일반[√], 보안[ ]

부처명	과학기술정보통신부	사업명	한국전자통신연구원 연구운영비지원 (기본사업)
전문기관명	한국전자통신연구원		

과제유형		■ 기초 □ 응용 □ 개발 □ 기타		
대과제	과제명	성능한계를 극복하는 초성능 컴퓨팅 실현 (Realization of Future Computing to overcome Performance Limitations)		
	과제책임자	조일연	소속 및 부서	인공지능컴퓨팅연구소
			전화번호	[redacted]
세부(협약)과제	과제명 국문	인공지능 처리성능 한계를 극복하는 고성능 컴퓨팅 기술 연구		
	과제명 영문	Research on High-Performance Computing to overcome Limitations of AI		
	과제책임자	김강호	소속 및 부서	초성능컴퓨팅연구본부
			전화번호	[redacted]
과제수행기간	총 과제수행 기간	2020. 01. 01 - 2025. 12. 31 (6년 0개월)		
	단계 1단계	2020. 01. 01 - 2021. 12. 31 (2년 0개월)		
	단계 2단계	2022. 01. 01 - 2024. 12. 31 (3년 0개월)		
	단계 3단계	2025. 01. 01 - 2025. 12. 31 (1년 0개월)		

연구개발비 (단위:천원)		정부출연금	민감부담금		합계		
			현금	현물	현금	현물	계
총계		14,516,000		19,200	14,516,000	19,200	14,535,200
1단계	1년차	1,315,000	-	19,200	1,315,000	19,200	1,334,200
	2년차	2,921,000	-	-	2,921,000	-	2,921,000
2단계	3년차	2,874,000	-	-	2,874,000	-	2,874,000
	4년차	2,953,000	-	-	2,953,000	-	2,953,000
	5년차	2,236,000	-	-	2,236,000	-	2,236,000
3단계	6년차	2,217,000	-	-	2,217,000	-	2,217,000

참여인력(M/Y)	총 참여인력	135명(69M/Y)			
			1단계 ('20~'21)	1년차	11명(6.6M/Y)
				2년차	18명(11.6M/Y)
			2단계 ('22~'24)	3년차	24명(13.2M/Y)
				4년차	24명(13.2M/Y)
				5년차	33명(12.9M/Y)
			3단계('25)	6년차	26명(11.5M/Y)

공동연구개발기관 등 (해당 시 작성)	기관명	책임자	직위	휴대전화	전자우편	비고 역할	비고 기관유형
공동연구개발기관	아주대학교	[redacted]	조교수	[redacted]		공동	대학
	성균관대학교		부교수			공동	대학
	UNIST		부교수			공동	정부출연연
	웁살라대학교		조교수			공동	해외대학
	UT달라스		조교수			공동	해외대학
	한국항공대학교		부교수			공동	대학
위탁연구개발기관	대구대학교		부교수			위탁	대학
	경북대학교		조교수			위탁	대학
	인하대학교		교수			위탁	대학
	경상대학교		부교수			위탁	대학
	경희대학교		부교수			위탁	대학
	한국항공대학교		부교수			위탁	대학
	충북대학교		부교수			위탁	대학
	고려대학교		부교수			위탁	대학
	포항공과대학교		부교수			위탁	대학

ETRI 실무담당자	성명	안신영	소속 및 부서	슈퍼컴퓨팅시스템연구실
			전화번호	[redacted]

이 보고서에 기재된 내용이 사실임을 확인하며, 만약 사실이 아닌 경우 관련 법령 및 규정에 따라 과제 중단, 협약 해약, 제재처분 등의 불이익도 감수하겠습니다.

2026 년 2 월 27 일

세부과제책임자 : 김 강 호 (인)

대과제책임자 : 조 일 연 (인)

한국전자통신연구원장 귀하

세부(협약)과제 최종보고서 요약문

대과제명	성능 한계를 극복하는 초성능 컴퓨팅 실현						
세부(협약)과제명	인공지능 처리성능 한계를 극복하는 고성능 컴퓨팅 기술 연구	과제 유형	기초[√] 응용[ ] 개발[ ] 기타[ ]	TRL	시작	2
					종료	6
1 세세부과제명	패브릭 메모리 컴퓨팅 핵심 기술 연구 (2020.01.01.~2023.12.31)	과제 유형	기초[√] 응용[ ] 개발[ ] 기타[ ]	TRL	시작	2
					종료	6
2 세세부과제명	Massive Things의 초근접 서비스를 위한 지능형 EdgeCPS 플랫폼 개발 (2021.01.01.~2023.12.31)	과제 유형	기초[√ 응용[ ] 개발[ ] 기타[ ]	TRL	시작	2
					종료	6
3 세세부과제명	이종 가속기 기반 대규모 AI 병렬컴퓨팅 기술 개발 (2024.01.01.~2025.12.31)	과제 유형	기초[√] 응용[ ] 개발[ ] 기타[ ]	TRL	시작	2
					종료	6

국가과학기술 표준분류	EE0203	50 %	EE0108	30%	EE0110	20%

총 과제수행 기간	2020. 01. 01 - 2025. 12. 31 (6년 0개월)

총연구개발비

총 14,535,200천원
* 정부출연금 : 14,516,000 천원
* 민간부담금 : 19,200 천원

연구개발비 (단위:천원)		정부출연금	민감부담금	합계
총계		14,516,000	19,200	14,535,200
1단계	1년차	1,315,000	19,200	1,334,200
	2년차	2,921,000	0	2,921,000
2단계	1년차	2,874,000	0	2,874,000
	2년차	2,953,000	0	2,953,000
	3년차	2,236,000	0	2,236,000
3단계	1년차	2,217,000	0	2,217,000

과제수행 목표 및 내용		
최종 목표		ㅇ Trillion급 대규모 생성형 인공지능 모델의 연산 한계를 극복하는 이종 가속기 기반 대규모 AI 병렬컴퓨팅 기술 개발
1단계	목표	ㅇ 인공지능 서비스와 데이터 처리를 가속하는 메모리 중심 컴퓨팅 기술 개발 및 저지연·실시간 인프라 구현을 위한 지능형 EdgeCPS 플랫폼 구조 설계
	내용	ㅇ 고성능·고효율 메모리 중심 컴퓨팅 핵심 기술 연구 ㅇ 메모리 중심 컴퓨팅 기반 초고속 데이터 분석 기술 연구 ㅇ 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 구조 설계 ㅇ 엣지 사이버물리시스템을 위한 대규모 디바이스 가상화 기술 설계
2단계	목표	ㅇ 계산 성능한계를 극복하기 위한 AI 병렬컴퓨팅 구조 연구
	내용	ㅇ Peta-Scale 메모리 중심 컴퓨팅 핵심 기술 연구 ㅇ 메모리 중심 컴퓨팅 기반 스케일러블 인메모리 저장소 연구 ㅇ 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 핵심 알고리즘 개발 ㅇ 엣지 사이버물리시스템을 위한 대규모 디바이스 가상화 플랫폼 핵심 알고리즘 개발 ㅇ AI 병렬컴퓨팅 구조 연구
3단계	목표	ㅇ 이종 인공지능 프로세서 가속기 기반 Trillion급 초성능 AI 컴퓨팅 시스템 연구
	내용	ㅇ 클러스터 레벨 대규모 생성형 AI 고성능 분산/병렬 컴퓨팅 기술 연구 ㅇ T급 생성형 모델의 이종 가속기 통합 AI 병렬 컴퓨팅 기술 연구 ㅇ 이종 가속기간 고성능 집합통신 기술 연구

과제 수행과정 및 내용

ㅇ(1세세부과제) 패브릭 메모리 컴퓨팅 핵심 기술 연구 (1~4차년)

ㅇ 핵심 요소 기술의 개념

- 메모리 중심 컴퓨팅 구조: 원격 메모리 및 CXL 메모리 등을 활용하여 대용량·공유 메모리 풀을 통해 다수의 연산장치들이 연결된 새로운 컴퓨팅 구조 연구
- 메모리 중심 컴퓨팅 기반 패브릭 메모리 관리 프레임워크: 패브릭 메모리 기반 응용 메모리 및 IO 메모리 활용 프레임워크 연구
- 메모리 중심 컴퓨팅 기반 데이터 처리 가속: 메모리 중심 컴퓨팅을 기반으로 응용을 재설계하여 성능 혁신

ㅇ 과제 수행과정

- ETRI에서 데이터 중심 컴퓨팅을 위한 새로운 컴퓨팅 구조와 시스템SW 원천기술 연구
- 대학은 메모리 중심 컴퓨팅 기반 초고속 데이터 분석 기술 연구
- 기업은 메모리 중심 컴퓨팅 시스템SW 사업화 관련 기술 연구 및 실증 진행

ㅇ 과제 수행내용

- 고성능·고효율 메모리 중심 컴퓨팅 핵심 기술 연구

- (ETRI) 384TB 규모 패브릭 메모리 시스템 구축
- (ETRI) 응용/컨테이너 기반 실행환경 지원 및 CXL장치 등 차세대 메모리 지원
- (아주대) 성능 오차 1% 이내의 차세대 디바이스 에뮬레이터 기술 연구

	- 패브릭 메모리 관리 프레임워크 기술 연구 • (ETRI) 가상머신 메모리 확장 성능 29% 향상 • (ETRI) 패브릭 메모리 보안 기술 개발로 분산 자원 활용으로 인한 보안성 문제 극복 • (ETRI) 가상머신 실시간 이주 최적화로 이주시 데이터 전송향 2배 이상 절약 • (성균관대학교) 패브릭 메모리 기반 응용 메모리 및 파일시스템 캐시 연구로 DB성능 1.8배 향상 **- 메모리 중심 컴퓨팅 기반 초고속 데이터 분석 기술 연구** • (ETRI) 1T 엣지 그래프 분석 559s • (UNIST) AI 지속학습 에너지 효율 46% 향상 및 데이터 증강 최적화로 학습 성능 2.8배 향상 • (웁살라대학) 다계층 메모리 기반 그래프 분석 성능 1.65배 향상 **(2세세부과제) Massive Things의 초근접 서비스를 위한 지능형 EdgeCPS 플랫폼 개발 (2~4차년)** **ㅇ 핵심 요소 기술의 개념** - 다양한 성능의 디바이스가 대규모로 활용되는 환경에서 저자원/중간사양/고사양 디바이스에 가상화된 증강자원을 부여하여 디바이스 성능한계를 극복하는 AI서비스 제공이 가능한 EdgeCPS 플랫폼 개발 - Deviceless Computing : 디바이스 운영/관리 및 용적 계획 결정이 개발자나 운영자로부터 완전히 숨겨져 있는 시스템. 플랫폼에서 모든 번거로운 일을 처리하기 때문에 서비스가 실행되는 디바이스의 하드웨어나 운영체제(OS)에 대해서 알지 못해도 됨 **ㅇ 과제 수행과정** - 주관기관인 ETRI를 중심으로 産學硏 협업형 연구개발 체계를 구축하고, 기수행 연구 및 기보유 기술에 기반하여 EdgeCPS를 위한 새로운 아키텍쳐 연구 및 플랫폼 개발 - 세계 기술을 선도하고 있는 대학(UT 달라스, 미국)과 지속적인 협력을 통해 선행연구 진행 - 연구 개발 단계별로 産學硏의 역할과 연구 범위를 명확히 정의하여 과제 종료 시점에 상용화 수준의 EdgeCPS플랫폼을 개발하고 사업화 유망 도메인에 기술 적용 추진 **ㅇ 과제 수행내용** - 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 프로토타입 개발 • (ETRI) 저자원 디바이스의 자원 증강 프레임워크 개발 • (ETRI) FPGA 가속기 자원 프로비저닝 기술 연구 • (ETRI) 이종 디바이스를 위한 가상 자원 동적 생성 기술 연구 • (대구대) 저자원 디바이스 자원 증강 가상화 인프라 구조 연구 • (경북대) EdgeCPS 기반의 자원 제약 디바이스의 분할 실행 기술 연구 • (인하대) EdgeCPS디바이스의 AI 서비스를 위한 AI 전용 지식 공유 모델 연구 - 엣지사이버물리 시스템을 위한 대규모 디바이스 제어 플랫폼 프로토타입 개발 • (ETRI) 실-가상 디바이스간 컴퓨팅 자원/서비스 연계 기술 연구 • (ETRI) Inter-EdgeCPS의 디바이스 증강을 위한 데이터 전처리 기술 연구 • (ETRI) 실디바이스의 원격 활용을 위한 디바이스 미러링 기술 연구 • (경희대) 로컬 모델 기반의 디바이스 데이터 증강 기술 연구 • (경상대) EdgeCPS 디바이스의 자율성 증강 기술 연구 • (UTD) EdgeCPS를 활용한 가상공간 사용자들간 멀티모달 상호작용 기술 연구 **(3세세부과제) 이종 가속기 기반 대규모 AI 병렬컴퓨팅 기술 개발 (5~6차년)** **ㅇ 핵심 요소 기술의 개념** - 인공지능 처리성능 한계를 극복하기 위해 이종의 연산 자원을 통합 운용하는 패브릭 컴퓨팅 구조 분석 - 이종가속기(GPU, 이종 NPU)를 통합하여 대규모(생성형) AI 모델의 고성능 병렬컴퓨팅을 제공하기 위한 SW 기반 구조 연구 - 동종 가속기(NPU)간 그리고 이종가속기(GPU, NPU)간 병렬컴퓨팅을 가능하게 하는 고성능 집합통신 기술 연구 **ㅇ 과제 수행과정** - ETRI에서 대규모 인공지능 모델의 학습/추론을 지원하는 이종 가속기 기반 AI 컴퓨팅 시스템 구조 및 SW 원천 기술 연구 - 국내 대학과의 협력을 통해 요소기술의 선행 연구 진행 - 기관(기업) 협력을 통해 AI 병렬 컴퓨팅 핵심 기술의 활용 및 검증이 가능하도록 요구사항을 수집하는 등의 협력 체계 구축 **ㅇ 과제 수행내용** - 계산 성능한계를 극복하기 위한 AI 병렬 컴퓨팅 구조 기술 연구 • (ETRI) 이종 가속기 기반 AI 분산/병렬 컴퓨팅 구조 연구 • (ETRI) PCIE 가속 장치간 고성능 집합통신 구조 연구 • (ETRI) 분리 자원 노드 추상화 구조 기술 연구 • (항공대) 데이터 처리 장치(DPU) 중심의 AI 분산 컴퓨팅 SW 구조 연구

	• (ETRI) 패브릭 컴퓨팅 아키텍처 연구 : 오픈소스 GiantVM기반 가능성 분석 • (충북대) CXL 기반 계층 메모리 플랫폼에서 커널 데이터 배치 연구 : 패브릭OS의 커널데이터 관리 연구 • (ETRI) 이종 성능 GPU 활용, NPU 활용, GPU-NPU 통합 활용 환경 구축 - 이종 가속기 기반 AI 병렬 학습 기술 연구 • (ETRI) 데이터 중심 AI 분산 학습 SW 구조 연구 • (ETRI) 이종 성능 GPU 기반 고확장/고성능 가중치 업데이트 알고리즘 개발 • (ETRI) 이종 성능 GPU 통합 데이터 병렬 분산 학습 SW(EDDIS) 기능 확장 개발 • (ETRI) EDDIS를 위한 K8S기반 DLOps 개발 • (ETRI) 이종 성능 GPU 환경(A100x4 + H100x4 환경)에서 기존 FSDP 대비 6.22배 빠른 학습 속도 달성 : EDDIS FSYNC 모드, Bert 모델 기준 - 이종 가속기 기반 LLM 병렬 추론 기술 연구 • (ETRI) 이종 성능 GPU 활용 비균등 분할 텐서 병렬처리기반 추론 PoC • (ETRI) 이종 가속기(GPU/NPU) 통합 활용 분리형 프리필 추론 기술 개발(vLLM v0.10.0 기반) • (ETRI) 이종 가속기(GPU/NPU)간 RDMA기반 통신 기술 개발 • (항공대) NPU간 집합통신 기반 모델 자동병렬화 연구 • (고려대) GPU/NPU 통합 생성형 모델 병렬 추론 연구 • (포항공대) 긴 문맥 추론을 위한 KV 캐시 관리 기술 연구 • (ETRI) 이종 가속기 환경(GPU 4개 장착된 노드 + NPU 4개 장착 노드)에서 Llama 3.1 70B에 대한 추론 서빙 확인

과제 수행결과 및 목표달성도

ㅇ 과제 수행결과

1단계

정량

논문		특허				기술이전	
SCI(건)	비SCI(건)	해외(건)		국내(건)		건수	금액(백만원)
		출원	등록	출원	등록		
4	14	5	5	5	4	3	190

•(사업화) 기술이전
- ㈜시스기어(1억원): 2-계층 메모리 시스템 기술 v1
- ㈜시스기어(0.7억원): 2-계층 메모리 시스템 기술 v2
- ㈜테라텍(0.2억원): PCIe 기반 확장 메모리 기술
•(기타) 국제MoU체결: 미국, TidalScale (‘20.02)
•(기타) 실증사업: 에너지기술연구원, 한컴위드, ㈜녹십자지놈

정성

•2021년 국가연구개발사업 우수성과 100선 선정

2단계

정량

논문		특허				기술이전	
SCI(건)	비SCI(건)	해외(건)		국내(건)		건수	금액(백만원)
		출원	등록	출원	등록		
10	44	9	4	22	7	2	60

•(사업화) 기술이전
- ㈜크래프트X(3천만원) : 컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술
- ㈜알티스트(3천만원) : 컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술
•(표준화) ITU-T SG16 Q5 (SG16-C243R1 : Reference model of AI knowledge sharing for AI computing platform)
•(기타) IPR확보를 위한 프로그램 등록 7건

정성

•패브릭 메모리 컴퓨팅 핵심 기술 사업화를 위한 애로기술지원사업 진행 (1건)
•5G 연계형 디바이스의 혁신적 서비스 개발 지원 협력 (경기도경제과학진흥원)
•다양한 디바이스가 설치되는 대규모 생산환경에 적용 가능성 검토 협력 (RIST, POSCO)
•GPU 가속 기술을 이용한 대규모 생산 설비 데이터 분석 기술 협력 (한국생산기술연구원)
•레거시 및 신규 제작하는 자동화 설비의 지능 제어 기술 협력 (비엠아이)
•본부내 디바이스시뮬레이션연구실 및 ㈜뉴블라: 장치간 집합통신 기술 관련 연구 협력 추진
•텐스토렌트와 연구협력 추진(NPU서버 구축 및 Metalium 오픈소스 기반 CCL 연구)
•NPU IP 업체인 Semidynamics (RISC-V Core 및 Tensor 엔진 IP 기술 보유 기업)와 LoI 체결
•㈜딥인스펙션: EDDIS 분산 학습 플랫폼 관련 기술 지원

3단계

정량

논문		특허				기술이전	
SCI(건)	비SCI(건)	해외(건)		국내(건)		건수	금액(백만원)
		출원	등록	출원	등록		
2	9	2	0	3	0	0	0

•(기타) IPR확보를 위한 프로그램 등록 8건

정성

•㈜하이퍼엑셀과 연구협력 신규 추진(HyperAccel LPU 기반 CCL 및 추론기술 연구)
•텐스토렌트와 연구협력 지속 추진(NPU 추가 구축 및 추론 기술 개발)
•㈜딥인스펙션에 EDDIS 분산 학습 플랫폼 관련 기술 지원 및 기술이전 추진중

○ **과제 수행 목표달성도**

가. 공통지표

구분	기본지표							
	지표명	총사업 연도	1단계		2단계			3단계
			′20년도	′21년도	′22년도	′23년도	′24년도	′25년도
과학적 성과	SCI(E) 논문	16	2	2	4	2	4	2
기술적 성과	국내특허(출원)	30	2	3	12	7	3	3
	국내특허(등록)	11	0	0	0	2	1	8
	국제특허(출원)	16	2	3	4	3	2	2
	국제특허(등록)	9	0	0	1	2	4	2
경제적 성과	기술이전(건)	5	1	2	2	0	0	0
	기술료(억원)	2.5	1.0	0.9	0.6	0	0	0

구분	심화지표							
	지표명	총사업 연도	1단계		2단계			3단계
			′20년도	′21년도	′22년도	′23년도	′24년도	′25년도
과학적 성과	표준화된 IF 상위 20% SCI 논문(건)	2	0	1	0	0	1	0
기술적 성과	특허활용률 (기술이전건수/ 특허등록보유건수)	25%	-	-	200%	0	0	0
	국제표준승인표준 기고서(건)	0	0	0	0	0	0	0
	3극 특허(건)	0	0	0	0	0	0	0
경제적 성과	연구비 대비 기술료 수입(%)	1.72	7.5	3.08	2.09	0	0	0

나. 자율지표

구분	자율지표			
	지표명	연간 평균 목표	1단계	
			′20년도	′21년도
과학적 성과	국제 (최)우수학술대회 논문 발표(건)	평균 2.2건	7건	2건

구분	자율지표				
	지표명	연간 평균 목표	2단계		
			′22년도	′23년도	′24년도
과학적 성과	국제 (최)우수학술대회 논문 발표(건)	평균 1.66건	10건	16건	3건

구분	자율지표		
	지표명	연간 평균 목표	3단계
			′25년도
과학적 성과	국제 (최)우수학술대회 논문 발표(건)	평균 2건	6건

관련 분야에 대한 기여	○ 관련 분야 과학적·기술적·경제적·사회적 기여 [메모리 중심 컴퓨팅 분야]
과학적	• 메모리 중심 컴퓨팅 기반 SW 연구 선도: **상위20%SCI 1편**, SCIE 3편, **(최)우수학회 9편** • 유전체 분석 속도 3.34배 향상: **최우수학회 1편,** ㈜녹십자지놈 PoC(기술성/사업성 최고 평가) • 미국 실리콘밸리 TidalScale **국제 MoU 체결** (′20.02), 비대면 연구회의 26회 • 대규모 양자 컴퓨팅 시뮬레이션 **용량 효율성 64배 향상**: ETRI 양자컴퓨팅 연구실 PoC • 대용량 메모리 컴퓨팅의 효율적 수행환경 제공: SCIE 1편, **(최)우수국제학회 9편** • 대규모 메모리 기반 그래프 분석 가속 연구: ETRI연구사업계획서 정량 목표 달성 - 1 trillion edge 그래프 분석 소요 시간: 559초
기술적	• ETRI IP-R&D 프로그램 기반 메모리 중심 컴퓨팅 및 유전체 분석 핵심 특허 확보 - 국제 특허 (미국) 9건 등록 / 14건 출원, 국내 특허 8건 등록 / 13건 출원 • **2021년 국가 연구개발사업 우수성과 100선 선정** (정보·전자 분야)

	• CXL 디바이스 기반 메모리 확장 기술 개발: REDHAT 연구 협력
경제적	• **기술이전: 3건 (1.9억원)** - ㈜시스기어: 2건 . "탄력적 블록 관리 기반 2-계층 메모리 시스템 기술" **(1.0억원)** . "단력적 블록 관리 기반 2-계층 메모리 시스템 기술 V2" **(0.7억원)** · **사업 부문 Valuation 240억원** 평가 ('21.11) / **코오롱베니트** PoC ('21.11) - ㈜테라텍: "PCIe 기반 확장 메모리 기술" **(0.2억원)** • **KIER, 한컴위드 PoC**: 대규모 과학계산 및 대용량 디지털포렌식 비용 절감 확인 • **생명공학연구원 PoC**: 국가 바이오 빅데이터 구축 사업에 유전체 분석 가속 기술 적용 논의 • **수자원공사 메모리 관리 기술 사업화**: 그린IT를 위한 On-Premise 시스템 구축 진행 • 기술이전 업체의 제품 개발 및 사업화 지원, 추가 기술이전을 위한 시장 요구사항 분석 - ㈜시스기어의 시스얼라인 애로기술지원사업 진행, TTA 인증 지원 - 클라우드 컴퓨팅 풀 서비스 모델 지원 플랫폼 제공 • 대용량 메모리 기반 응용 가속기술을 바탕으로 기술사용 비용절감 및 빠른 의사결정도구로 활용

[EdgeCPS 분야]

과학적	• 저자원 디바이스 성능 한계 극복 위한 디바이스리스 컴퓨팅 기술 및 대규모 디바이스 가상화 연구 선도: **최우수학회 8편** • 컴퓨팅 컨티뉴엄의 새로운 패러다임 선도: AI 반도체, 엣지 서버, 클라우드가 혼재된 환경에서 적용 분야별 다양한 성능 요구사항을 만족시키는 하이퍼오토메이션 제공 • EdgeCPS기술 연구 홍보 및 확산: "EdgeCPS 워크샵" 2회 개최" ('21.11, '23.11)
기술적	• **클라우드/엣지 서비스 커버리지 극대화**로 부가가치 창출 - 기존 클라우드 기술로는 수행하지 못하는 실시간 학습 및 추론 분야 가능 • 디바이스리스 컴퓨팅을 위한 서버 자원 증강지원 기술 원천 기술 및 특허 확보 - 연구 초기 단계에서부터 선도적인 IPR확보 전략 수행: 특허 국제 특허 3건 출원, 국내 특허 14건 출원, 소프트웨어 등록(4건) - 국제 표준 (ITU-T SG16) 1건 제안 • 디바이스**/엣지 연계형 서비스를 통해 사용자 QoE 극대화**로 부가가치 창출 - 기존 디바이스로는 수행하지 못하는 실시간 학습 및 추론 분야 가능
경제적	• 기술이전 : 2건(0.6억원) - ㈜크래프트X : "컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술"(0.3억원) - ㈜알티스트 : "컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술"(0.3억원) • 디바이스리스 컴퓨팅을 통해 데이터를 수집하는 엔드포인트에서 직접 학습 및 추론이 가능해지면서 자율주행자동차, 스마트팩토리 등의 산업 활성화 • 다양한 디바이스에서 발생하는 대규모 AI데이터를 CPU/GPU 자원을 활용하여 고속처리하는 기술 사업화 추진 (KISTI, CraftX, RTST, 한국플랫폼서비스기술 등) • 5G/6G 등 차세대 이동통신과 저지연 엣지 서비스를 통해 제조업, ICT, 헬스 케어, 에너지 등 4차 산업혁명과 관련된 광범위한 분야에서 다양한 가치 창출 가능

[이종가속기AI컴퓨팅 분야]

과학적	• 이종 성능 GPU 기반 분산 학습 기술 및 데이터 이동 최적화 연구 선도 - SCIE 논문 2편 게재, (최)우수국제학술대회 3편 발표 • LLM 학습과 추론 과정에서 발생하는 메모리 병목을 완화하여 자원 활용 효율과 메모리 접근 지연 단축에 기여 - SCIE 논문 2편 게재, (최)우수국제학술대회 3편 발표 • 지속 학습(Continual Learning)의 프롬프트 선택 과정과 업데이트 과정에서 요구되는 자원과 정확도 간의 trade-off를 분석하여 자원 효율성을 극대화함으로써, 제한된 자원 환경에서의 지속 학습 수행에 기여 - (최)우수국제학술대회 1편 발표 • 성능이 다른 가속기(GPU)가 장착된 다수의 노드를 통합 활용하여 분산 병렬 처리함으로써 저비용·고효율 AI 모델의 학습/추론에 기여 - (최)우수국제학술대회 2편 발표
기술적	• 다수의 물리 노드들의 가속기 자원을 통합하여 하나의 생성형 모델을 분산처리 할 수 있는 SW 기반 기술 확보 • PCI 기반 가속기의 집합통신 구조연구를 통한 기반 기술 확보 : HW PoC • 이종성능 GPU기반 분산학습 기술 보완 개발 및 E-패밀리기업을 통한 수자원공사 실증 - 이종 OS(Ubuntu, Rocky) 지원 및 FP16/BF16학습 지원 기능 - 국제 특허 1건 / 국내 특허 1건 출원 • 서로 다른 노드에 장착된 가속기간 데이터를 CPU 개입없이 저지연·저비용으로 전달하는 방법에 기반한 AI 모델 학습/추론 가속 기술의 경쟁력 확보 가능 - 국내 특허 1건 출원 완료, 국제 특허 1건 연구원 내부 출원 절차 진행 중 • 메모리 병목을 완화하여 자원 활용 효율을 높이는 기술 확보 - 국내 특허 1건 출원, 국제 특허 1건 연구원 내부 출원 절차 진행 중 • LLM 추론 서빙의 이종 가속기 통합 활용에 관한 기술 선점 - 국내 특허 2건, 국제 특허 2건 연구원 내부 출원 절차 진행 중

경제적	• 다수의 이종 성능 GPU 기반 분산학습 기술 개발로 기투자 가속기들을 통합한 대규모 AI 학습이 가능하게 되어 신규 GPU 투자비용 감소 • 이종 가속기(GPU, NPU) 통합 컴퓨팅이 가능하게 되면 이미 GPU를 가지고 있는 국내 기업들도 가속기의 이종성 때문에 망설이던 국산 NPU 구매가 가능해져 특정 글로벌 대기업으로의 락인 효과 감소 • 이종 가속기 AI 병렬컴퓨팅 기술을 확보하여 대규모 컴퓨팅 자원이 필요한 서비스를 손쉽게 개발할 수 있는 SW 기술을 확보함으로써 국가 경쟁력 향상 • 국내외적으로 NPU 자원의 투자에 더 적극적으로 바뀌어 국내 NPU 기업들의 성장 견인 가능 • AI 분산 학습 관련 기술에 대한 수요기업(E-패밀리기업)의 협력 요청 대응을 통한 성장 유도

○ 후속 과제에 도움을 줄 수 있는 연구 결과

	연구 결과	적용 가능 후속 과제
메모리 중심 컴퓨팅	메모리 중심 시스템SW	저탄소 데이터센터 자원 관리, 분산 인공지능 기반 메타버스 인프라
	유전체 분석 가속	유전자 분석 기반 실시간 감염병 추적, 저비용 유전체 분석
	대용량 양자 시뮬레이션	양자 암호화 알고리즘 연구, 양자 암호화 기반 통신 기술 연구
Edge CPS	디바이스리스 컴퓨팅	도심 물류 제어, 자율형 디바이스 고장 감지, 컴퓨팅 컨티뉴엄 플랫폼
	대규모 디바이스 가상화	메타버스/아크버스 플랫폼 연구, 디바이스 연계형 군집 제어 기술
	저지연 멀티모달 데이터 처리	택타일 인터넷 중계서버, 네트워크 적응형 멀티모달 데이터 처리
이종가속기AI 컴퓨팅	고성능 AI 분산/병렬 컴퓨팅	이종 성능 GPU 기반 데이터 병렬 분산 학습/추론
	이종 가속기 통합 AI 병렬 컴퓨팅	이종가속기기반 LLM 학습/추론, 데이터 처리 장치 기반의 AI 분산 컴퓨팅
	이종 가속기간 고성능 집합통신	PCIe기반 가속기 PoC 기반의 집합통신, 이종가속기기반 LLM 학습/추론 연구

성과관리 및 활용계획

○ 성과관리 현황

- (데이터 생산 및 관리) 소스코드 개발, 실험 평가를 위한 벤치마크 응용 그룹 선정, 원격 노드의 메모리나 로컬노드의 고성능SSD를 활용하여 분리 메모리를 구성, 벤치마크 응용의 수행을 통해 메모리 구성에 따른 성능 결과 저장, 성능 평가를 위한 공유 모델 구조를 선정하고 정적/동적 메모리 할당 정책 및 HW 특성에 따른 자원 공유 정책에 따른 시스템을 구성하여 컨테이너 형태의 응용을 수행하여 발생하는 성능 결과를 저장, 이종성능GPU기반 분산 학습시 발생하는 학습 모델 데이터는 평가후 삭제하고, 학습중 성능 결과 및 평가 결과를 저장
- (연구데이터 저장 및 보존) 소스코드는 기관 소스코드 리포지터리에 수시 저장, 연구 수행 중 자체 연구 개발 관리 서버에 연 2회 실험 결과 데이터 저장, 기관 내부 연구 데이터 리포지터리에 과제 종료 후 소스코드 및 데이터 저장하며 데이터 훼손을 대비한 제어 저장 관리 시스템에 저장함
- (데이터 공동활용) 소스코드, 수치, 텍스트 자료에 대해 원내 범위에서 당해 연구종료 후 엠바고 지나서 데이터를 기관 절차에 따라 데이터 개방 및 공유함, 데이터는 인적사항이 포함된 개인정보와 지적재산권 및 잠재적인 상업적 가치 보호에 대한 공개 및 공유의 제한이 있음

○ 성과 관리 및 활용 계획

- (기술적) 기존 시스템의 한계를 극복하는 대용량 컴퓨팅의 효율적 수행 환경 제공으로 국내 클라우드 서비스 산업과 대규모 과학 계산, 금융 분석 등의 산업의 글로벌 경쟁력 증대. 다양한 가속 HW 자원들의 공유 기술 확보로 최대한 많은 수의 저성능 단말이 인공지능 기술을 처리 할 수 있도록 지원하여 초대규모 디바이스들 연계를 통해 온 디바이스 컴퓨팅과 서버 중심 컴퓨팅 기술의 한계를 넘는 군집/복합 인공지능 서비스를 실현하는데 기여, 다양한 이종성능 가속기를 보유한 다양한 기업, 연구소 등을 위한 고효율 분산 학습/추론 기술 제공을 통해 비용 효율적인 인공지능 컴퓨팅 실현에 기여
- (사회문제해결) 대규모 데이터에 대한 효율적인 분석 시스템을 제공함으로써 유전자 분석, 사회 데이터 분석 등의 비용이 낮추어 사회문제 해결에 기여할 수 있으며, 해외 클라우드 서비스 의존도를 낮춤으로써 데이터 안보에도 이바지. 고성능 엣지 시스템의 가속 HW 자원 공유로 국가적으로 전력 및 CO2 절감 효과에 기여
- (확보된 기술의 사업화 전략) 대규모 메모리 확장 시스템에 대해서 기술이전 업체와의 협력을 지속하여 사업화를 추진하고, 학회 등을 통한 관련 기술 홍보를 통해 사업화의 추진력을 더함. 이종가속기 기반 학습 핵심 요소기술의 개발 및 프로토타입 개발을 추진하고, 지속적인 고도화를 통해 사업 종료시 사업화를 위한 최적화 기술을 확보함

국문핵심어 (5개 이내)	메모리중심컴퓨팅	엣지 사이버물리시스템	디바이스리스컴퓨팅	AI병렬컴퓨팅	집합통신
영문핵심어 (5개 이내)	Memory-centric Computing	EdgeCPS	Deviceless Computing	AI Parallel Computing	Collective Communication

목 차

1. 과제 개요 ····· **10**
1-1. 과제 수행계획 ····· 10
1-2. 현황 및 접근방법 ····· 19

2. 과제목표 및 수행과정 ····· **50**
2-1. 과제목표 ····· 50
2-2. 과제 연차별 수행과정 및 내용 ····· 51
2-3. 과제 수행기간 추진체계 및 방법 ····· 453

3. 과제 수행결과 및 목표달성도 ····· **458**
3-1. 과제 수행결과 ····· 458
3-2. 목표달성도 ····· 474
3-3. 목표 미달 시 원인분석 ····· 475

4. 관련 분야에 대한 기여 ····· **476**
4-1. 과학적·기술적·경제적·사회적 파급효과 ····· 476
4-2. 후속 과제에 도움을 줄 수 있는 연구 결과 ····· 480

5. 성과관리 및 활용계획 ····· **481**
5-1. 성과관리 현황 ····· 481
5-2. 성과관리 및 활용 계획 ····· 483

6. 연구개발비 사용실적 ····· **484**

7. 중요 연구변경 사항 ····· **485**

1 과제 개요

1. 과제 수행계획

가. 과제 수행의 필요성

○ (정부 추진정책) 정부는 4차 산업혁명이 촉발하는 산업·경제, 사회·제도, 과학·기술 전 분야의 변화에 맞춰 각 분야가 긴밀히 연계된 종합 정책을 추진하는 4차 산업혁명위원회를 출범(2017.10.)

- 4차 산업혁명 기술기반 강화를 위해 데이터를 쉽게 찾고 거래하여 가치를 창출하도록 AI 학습 형태로 공공데이터 개방, 산업별 빅데이터 육성 등 '데이터 활용 강화' 정책을 수립

※ 전 산업구조가 데이터에 대한 자가 학습을 통해 지속적으로 AI 알고리즘 성능을 강화하므로 데이터가 산업의 새로운 경쟁 원천으로 부각

○ (정부 추진정책) 정부는 경제 패러다임 전환 추진과정에서 코로나19 사태로 인한 극심한 경기침체 극복 및 구조적 대전환 대응이라는 이중과제를 해결하기 위해 한국판 뉴딜 정책을 추진(2020.7.)

- D·N·A(Data·Network·AI)기반을 바탕으로 혁신과 역동성이 확산되는 디지털 중심지로써, 글로벌 메가트랜드를 주도하는 "똑똑한 나라"로 변신을 추구
- 디지털 新제품·서비스 창출 및 우리 경제의 생산성 제고를 위해 全 산업의 데이터·5G·AI 활용 및 융합을 가속화하는 정책을 추진

※ 전 산업구조가 데이터에 대한 자가 학습을 통해 지속적으로 AI 알고리즘 성능을 강화하므로 데이터가 산업의 새로운 경쟁 원천으로 부각

○ (정부 추진정책) 정부는 '2030년 과학기술 5대 강국 도약'을 위해 '제1차 국가연구개발 중장기 투자전략(2023~2027)'을 발표('23.03.07.), 4대 전략과 23개 과제 추진을 결정

- 국가 혁신역량 강화 전략의 일환으로 '디지털 혁신'을 지정하고 AI를 이의 핵심 기술로 정의함에 따라, 더 적은 에너지로 더 빠르게, 더 효율적으로 AI를 수행하게 하는 'AI 병렬 컴퓨팅' 기술 확보는 국가 혁신역량 강화에 필수 핵심 요소로 부각됨

○ (정부 정책 연계) 제2차 국가초고성능컴퓨팅 육성 기본계획(미래 초고성능 컴퓨팅 패러다임 변화대응) 등 연계 추진

- 제2차 국가초고성능컴퓨팅 육성 기본계획(미래 초고성능 컴퓨팅 패러다임 변화대응): 인간의 뇌와 동등한 사고능력을 구현하는 지능형 초고속 컴퓨팅 역량 강화에 기여(과기정통부, 2018.02)
- 정부에서 국정과제 34번 미래형 신산업 발굴 및 육성을 추진하고 있는 고부가가치 창출 미래형 신산업 발굴・육성과 연계하여 활용이 가능한 핵심 기술

※ 국정과제 34번 內 첨단기술산업 : 지능형 로봇, 3D프린팅, AR・VR, IoT가전, 스마트선박, 나노・바이오, 항공・우주 등 첨단기술 산업 육성을 위해 R&D 및 실증・인프라 구축 지원

○ (정부 정책 연계) 국가과학기술자문회의 국가전략기술 특위는 미래혁신을 이끌 초격차 기술확보 전략으로 '인공지능·첨단바이오 전략로드맵'을 심의·의결('23.10.31.)

- 인공지능의 국가임무로 '학습데이터 및 자원 소모량을 50% 이상 경량화'가 선정되어, 절감하는 것을 핵심 목표로 임무중심 전략 로드맵에서 AI를 '필수기반 기술'로 지정하고 이의 주요 임무에 '효율적 학습 및 인프라', 'AI 산업활용' 등을 추진 목표로 선정함에 따라 AI 병렬 컴퓨팅 핵심 기술 및 데이터 중심 컴퓨팅 핵심기술 개발의 중요성 증대
- 또한 AI반도체는 12대 국가전략기술 중 글로벌 기술패권 경쟁 핵심 기술로 선정되어, AI 반도체에 필수적으로 요구되는 데이터 이동 최적화 컴퓨팅 기술의 중요성은 더욱 커지고 있음

○ (정부 정책 연계) 제3차 국가초고성능컴퓨팅 육성 기본계획(엑사스케일 시대 대비 초고성능컴퓨팅 발전전략) 등 연계 추진

- '초고성능컴퓨팅 기술 자립화로 컴퓨팅 기술강국 도약 추진'을 통한 기술자립화를 위해, 자체기술 적용 시스템 개발을 추진
- 패브릭 컴퓨팅 기술은 초고성능컴퓨팅 기술 자립화에 필요한 핵심 IPR로서, AI 적용 확대를 위한 자체 초고성능컴퓨팅 기술의 활용을 촉진할 것임

○ (ETRI R&R) 2019년 수립된 과학기술 출연연 R&R 계획에서 한국전자통신연구원(ETRI)는 6대 중점추진분야를 설정하였고, 초지능 서비스 확대에 따른 정보처리 능력의 한계극복을 위해서 초성능 분야를 설정하였음

- 고성능 및 변혁적 컴퓨팅 시스템구현을 통해 인류 난제와 국가 사회 문제 해결에 기여하는 컴퓨팅 인프라 제공을 목표로 함

  ※ 세부역할 2-1-1 : 지능정보 데이터 고속처리를 위한 메모리 중심 컴퓨팅 기술 연구
- ETRI R&R 상위역할2(초성능 컴퓨팅 실현) 內 세부역할 2-1-2(멀티 클라우드 컴퓨팅) 및 2-1-3(임베디드 지능화 컴퓨팅)을 연계하여 Massive Thing의 지능화 서비스 인프라 구축이 가능

  ※ 세부역할 2-1-2 : 지능정보 서비스의 성능 최적화를 위한 멀티 클라우드 기술 연구

  ※ 세부역할 2-1-3 : 자원제약 환경에서 높은 복잡도를 가진 인공지능 연산의 성능 문제를 해결하는 임베디드 지능화 컴퓨팅 기술 연구
- '이종가속기 기반 AI 컴퓨팅' 기술 연구는 주요역할 2-1 인공지능 처리성능 한계를 극복하는 고성능컴퓨팅기술 연구에 해당

○ (ETRI연구사업계획서) 연구사업계획서에는 R&R 상위역할2에 해당하는 전략목표로 "성능 한계를 극복하는 초성능 컴퓨팅 실현"을 설정하고, 구체적인 성과목표 2-1에서 "인공지능 성능 혁신을 위해 폰노이만 병목을 해소하는 데이터 중심 컴퓨팅 원천기술"을 제시하며 본연구는 성과목표 2-1 달성을 위한 핵심 기술임

나. 과제 수행의 중요성

○ (플랫폼을 통한 AI 산업 생태계 선순환) AI 응용들과 사용자가 증가하면 플랫폼에 축적되는 데이터가 증가하게 되고, 다양한 데이터 축적이 이루어질수록 인공지능 플랫폼의 경쟁력이 강화되어 경쟁에서 선점 효과가 큰 기술 분야

- 기존 딥러닝과 같은 기계 학습 방법에 기반을 둔 AI 분석 플랫폼은 방대한 데이터 학습을 통해 AI 알고리즘 성능이 발전
- 구글, 아마존, 페이스북 등은 기술적 준비를 마치고 디지털 지능 생태계 확장에 나서고 있으나, 우리는 AI 기술 자체뿐만 아니라 AI 학습을 위한 데이터 확보 측면에서도 열세
- 구글/애플이 OS를 통해 주도한 모바일 플랫폼 경쟁에 밀려 주도권을 놓쳤던 과오가 반복되지 않도록 AI 분석 플랫폼에 대한 기술 역량을 자체적으로 개발하는 선제적 대응이 중요한 시기

○ (인공지능 플랫폼 원천기술 확보) 방대한 데이터 수집과 컴퓨팅 자원 기반 데이터 분석과 학습에 기반을 둔 인공지능 플랫폼 기술을 혁신하는 원천 기술 IPR 확보

- 사람과 환경 피드백 데이터를 실시간으로 적시에 분석하여 지능화하기 위해서는 메모리 처리 성능 수준의 초저지연·분산/병렬 컴퓨팅 기술은 반드시 확보하여야 할 핵심 기술

○ (6G 시대 신산업 육성 가능) 2028년 6G 세계 최초 상용화 목표 달성에 맞춰 대규모 무선 디바이스의 종단간 초저지연 서비스 제공을 위한 신개념 서비스 제공 플랫폼(Device to the Edge) 기술 개발이 시급히 필요

- 2022년까지 70%의 IoT 단말이 셀룰러 기술을 사용할 것으로 예상되는 등 5G/6G 이동통신 망을 통한 디바이스의 초저지연 서비스 핵심 기술 필요

  ※ 출처: Ericsson, Internet of Things Forecast

- 대규모 디바이스에서 발생하는 데이터를 저지연·고속 처리하며 5G/6G 망 연동과 AI 기술을 통한 자율권을 가진 엣지 역할 수행이 가능한 초성능 컴퓨팅 플랫폼 필요
- 센서, IoT 기기 등 초연결(최대 100만개/km^2) 환경에서 발생하는 대용량 트래픽 처리를 위해 초고속·대용량 연산처리 능력 필요

○ (신산업 경쟁력 제고 가능) 무수히 많은 사물로 구성된 스마트 공간(공장, 쉽야드, 도심 등)에 대한 자율권을 가진 EdgeCPS 구현을 통해 다양한 신산업 융합서비스 경쟁력 제고 가능

- 2025년까지 75조 개 이상의 IoT 단말이 동작하는 환경에서 양적으로 많은 단말의 지능 정보처리를 위한 EdgeCPS 기술 필수

  ※ 출처: Statista, Internet of Things – number of connected devices worldwide 2015-2025

- 2019년 10대 주요 기술로 엣지 컴퓨팅을 선정한 가트너는 2020년에는 스마트 공간에 대한 자율권을 가진 엣지(Empowered Edge) 역할의 중요성 강조

  ※ 출처: Gartner Top 10 Strategic Technology Trends for 2020

○ (원천기술 확보) 단순한 통계적/확률적 시뮬레이션을 이용하는 초기 단계의 디지털트윈을 넘어서 방대한 실세계의 데이터를 활용하여 인공지능 기반의 실시간 정밀 제어가 가능한 엣지

사이버물리시스템 기반 디바이스리스 컴퓨팅 원천기술 IPR 확보

※ 디바이스리스 컴퓨팅: 서버리스 컴퓨팅과 대비되는 개념으로 가상화된 공간에서 자원 공유의 탄력성을 제공하여 저성능 단말이 가속기와 같은 고가의 고성능 디바이스 없이도(Deviceless) 학습·추론 등 AI 연산을 가능하게 하는 컴퓨팅 방식

- 실세계의 환경, 사람 및 다양한 디바이스를 실시간으로 분석하여 적절한 데이터를 수집하고 가상화된 디바이스트윈을 통해 인공지능 기반의 실세계를 정밀 제어하는 기술은 반드시 확보해야할 핵심 기술임
- 수집된 데이터에 기반하여 디바이스를 제어하는 초보적인 CPS를 넘어서 모든 세상이 디지털화되는 미래세계에서 필수적으로 필요한 기술임

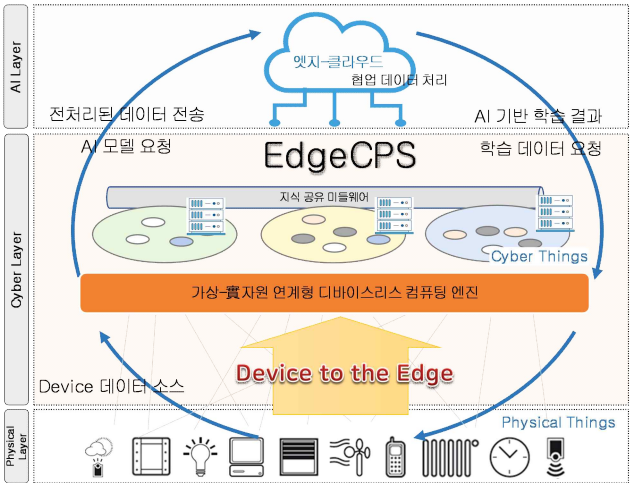
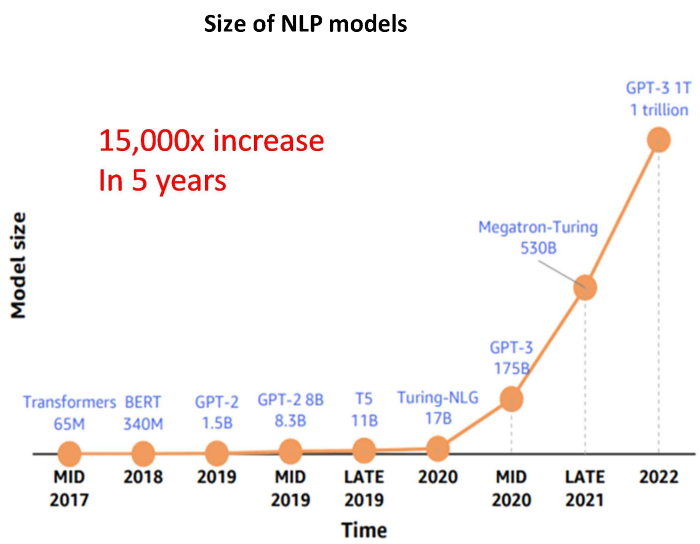


그림 2. Massive Things의 초근접 서비스를 위한 지능형 EdgeCPS 플랫폼 기술 개념

○ (초고성능 컴퓨팅 자원 제공 필요성) 초거대화되는 LLM의 학습/추론에 요구되는 연산량이 기하급수적으로 증가함에 따라 이를 안정적이고 효율적으로 지원할 수 있는 초고성능 컴퓨팅 시스템의 확보 필요

- 자연어 처리 모델(NLP)들의 규모는 지난 5년간 15,000배 증가하였으며, 대표적인 대규모 언어 모델(Large Language Model, 이하 LLM)인 Transformer 계열 모델의 학습에 필요한 컴퓨팅 연산량은 불과 2년만에 275배 증가하였음
- LLM 기반 서비스가 활성화됨에 따라 향후 추론용 컴퓨팅이 학습용 컴퓨팅 요구를 추월할 것으로 예상되며, 이에 따라 지속적인 대규모 연산 자원 제공이 요구됨
- 또한 초거대 기업을 중심으로 한 LLM 개발 및 서비스의 과점 구조가 심화될 것으로 예상되는 바, 과점 구조 극복을 위해 거대 AI 모델을 비용 효율적으로 학습/추론할 수 있는 컴퓨팅 기술의 확보가 필요함



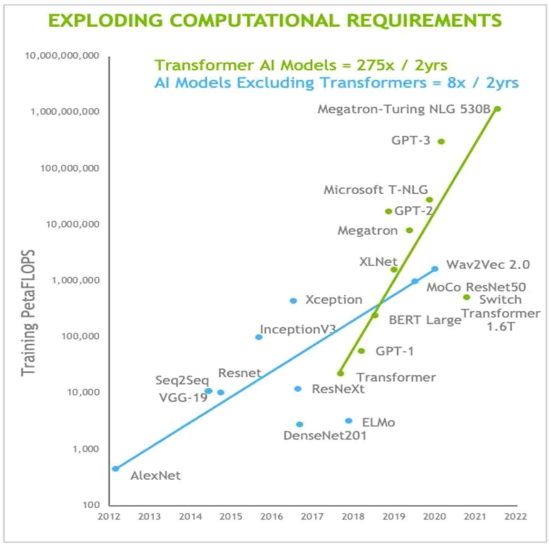


그림 3. AI 모델 규모 증가에 따른 컴퓨팅 요구 증가

○ (AI 병렬컴퓨팅 시스템 기술 개발의 필요성) AI 모델의 대형화·복잡화로 인해 단일 가속기 기반 컴퓨팅의 한계가 명확해지고 있으며, 다중 노드 환경에서 다양한 이종 가속기를 통합하여 활용할 수 있는 AI 병렬컴퓨팅 시스템 기술의 개발이 필수적임

- 지난 수십년간 고성능 컴퓨팅 시스템은 고성능을 낮은 비용으로 구축하기 위해 스케일-아웃 구조를 채택하여 왔으나, GPU를 포함한 다양한 가속기들의 등장으로 연산 자원이 노드 및 장치 단위로 분절되는 파편화가 심화됨
- 최근 고속 인터커넥트(e.g. 800GbE), CXL(v3.0표준화), DPU 등의 기술 성숙으로, 기존 물리 노드간 자원의 공유가 제한적이었던 구조에서 벗어나 다중 노드간 이종 가속기를 통합할 수 있는 기술적 토대가 마련되고 있음
- 이러한 환경변화 속에서 기존 프로그램의 수정없이 더 큰 문제를 더 빠르게 해결하기 위해서는 다중 노드에 분산된 이종 자원(CPU, GPU, NPU 등)들을 하나의 고성능·고효율 AI 병렬 시스템으로 통합하는 기술 개발이 요구됨

다. 과제 수행의 제약요인

○ 수행과정에서 예측되는 장애요소

- 엣지 디바이스, 엣지 서버, 클라우드, MEC 등 이종 플랫폼 연계를 위한 기반 기술 확보 필요
  - 본 연구개발과제는 기존의 연구개발과제들이 목표로 한 엣지 디바이스, 엣지 서버, 클라우드, MEC 등 이종의 컴퓨팅 플랫폼과의 연계를 바탕으로 한 사이버물리시스템으로서의 실행이 필수적
  - 이종 플랫폼 간의 연계를 위한 기반 기술 확보가 선행되지 않으면 이종 기기 통합과 상호운용 이슈가 과제 기간 중 지속적으로 발생할 가능성이 있음
- 엣지 사이버물리시스템을 위한 초고속 이동통신 인프라 연계의 어려움
  - 본 연구개발과제를 통해 추진하고자 하는 엣지 사이버물리시스템의 시나리오들은 고도의 안정성과 대역폭 및 저지연성을 지원하는 통신 인프라를 가정하며, 이는 현재 통신 기술 현황상 5G 및 향후 6G를 가정해야 함

- 초고속 이동통신 인프라를 기반으로 한 디바이스리스 컴퓨팅 기술의 개념 검증을 위해서는 일부가 이동통신 인프라에 내재화되어야 함을 의미하는데, 이는 이동통신사의 적극적 협조가 필요한 제약이 있음

- 이종가속기(GPU, NPU) 통합 기술 개발시 국내 NPU업체들의 연구 협력을 전제로 함
  - 국내 NPU 업체들이 저수준 API를 공개해줘야 고성능 통합 연구가 가능하나, 비공개시 최상위 레벨에서의 통합으로 고성능 통합이 어려울 수 있음

○ 제약요인을 해결 또는 최소화하기 위한 방안

- 관련 연구를 수행한 대학 및 기업의 참여를 적극 유도하여 이종 플랫폼 연계를 위한 기반 기술 확보
  - 본 연구개발과제를 수행하기 위해 엣지 디바이스, 엣지 서버, 클라우드, MEC 과제를 수행하였거나 수행중인 연구원의 참여를 유도하고, 과제 초기단계부터 현재 기술 파악과 통합, 그리고 향후 원천성 확보 가능성을 적극 검토 예정
  - 산업도메인별 데이터의 종류 및 통신 프로토콜이 상이함에 따라 도메인 전문가 그룹 및 관련 연구 사업과 협업을 추진
- 기확보한 5G 연계 인프라 및 인적 네트워크를 활용하여 초고속 이동통신 인프라 연계 추진
  - 본 연구개발과제 수행 부서의 이전 과제를 통해 확보한 5G 연계 인프라 및 인적 네트워크를 활용해 기간 내 개념 검증이 이루어질 수 있도록 협력 관계를 유지
  - 수 년내 상용화가 예상되는 사설 5G 기술과 연계할 수 있는 기자재를 확보하고, 이를 위한 연구 계획과 협력 관계를 지속 확보
- 오픈소스 API를 공개한 NPU를 이용 먼저 기술을 개발하고 국내 NPU들의 연구협력을 강화
  - 텐스토런트 NPU와같이 관련 소프트웨어 스택을 모두 오픈소스로 제공하는 NPU를 기반으로 GPU+NPU 통합 연구를 수행하고, 추후 이를 토대로 국내 NPU를 단계적으로 통합하는 전략 추진

라. 과제 수행결과 기대효과

○ 기술적 기대효과

- [메모리 중심] 프로세서 종속형 폰-노이만 컴퓨팅 구조인 현재의 고성능 컴퓨팅 패러다임을 메모리 중심의 새로운 컴퓨팅 아키텍처로 전환하여 지능정보 데이터의 실시간 고속 처리를 가능하게 하는 컴퓨팅 원천 기술을 확보
  - 메모리 중심 컴퓨팅 기술은 수십년 이어진 CPU 중심의 컴퓨팅 구조가 갖는 CPU-D램-스토리지로 고착화된 컴퓨팅 구조의 비효율성을 제거하여 성능 병목을 근본적으로 해소하는 원천 기술

    ※ 메모리 중심 컴퓨팅: 대용량·공유 메모리 풀이 중심이 되고, 고속 인터커넥트를 통해 메모리와 다수의 연산장치(CPU, GPU 등)가 연결되는 컴퓨팅 구조
  - 데이터 중심 아키텍처를 통해 데이터를 여러 프로세싱 유닛이 메인 메모리 수준에서 공유함으로써 대용량 데이터의 이동 시간을 제거하여 데이터 분석 및 인공지능 처리 성능을 획기적으로 개선
- [EdgeCPS] 자원 및 연산 능력 제한으로 인해 고성능 인공지능 기술 지원이 불가능한

디바이스들의 성능 한계를 극복하는 EdgeCPS 플랫폼 원천 기술확보

- 초대규모 디바이스들 연계를 통해 온 디바이스 컴퓨팅과 서버 중심 컴퓨팅 기술의 한계를 넘는 군집/복합 인공지능 서비스를 실현

- [EdgeCPS] 적용 분야별 다양한 성능 요구사항 만족을 위해 AI 반도체, 엣지 서버, 클라우드가 혼재된 환경에서 하이퍼오토메이션을 제공하는 새로운 컴퓨팅 패러다임 선도
  - 다양한 플랫폼을 통해 저성능 단말에 초저지연성, 고대역폭, 이종 가속기 지원 등을 안정적으로 지원하는 오케스트레이션 기술 확보
- [이종가속기AI컴퓨팅] 초거대 생성형 인공지능 모델의 병렬 추론을 위해 다수의 이종 가속기간 데이터 이동을 최적화하고, 이종 실행환경을 통합하는 AI 병렬 컴퓨팅 원천 기술 확보
  - 다수의 물리 노드들의 가속기 자원을 통합하여 하나의 생성형 모델을 분산 처리할 수 있는 SW 기술 개발
  - 사용되는 가속기 종류가 변경되더라도 인공지능 모델 및 실행 환경은 변경하지 않는 기술 확보

○ 산업적 기대효과

- [메모리 중심] 프로세서 종속형 폰-노이만 컴퓨팅 구조인 현재의 고성능 컴퓨팅 패러다임을 메모리 중심의 새로운 컴퓨팅 구조로 전환하여 지능정보 사회에서 요구되는 초성능 데이터 처리를 가능하게 하는 컴퓨팅 원천 기술을 확보
- [메모리 중심] 대용량·비휘발성 메모리로 인해 대용량 데이터 처리의 비용경쟁력 강화로 인공지능 관련 스타트업 및 중견기업이 보다 손쉽게 데이터를 가공함으로써 가치 창출 및 성장 기회의 향상

※ 비휘발성 메모리(Non-Volatile Memory, NVM): MRAM, PRAM 등의 소자를 기반으로 데이터 영속성을 보장하면서도 DRAM에 비해 높은 집적도를 제공하는 메모리

※ 세계 62개국 855개의 인공지능 스타트업 중 인공지능 응용 기술이 201개 차지

- [EdgeCPS] 엣지 사이버물리시스템 기반 서비스는 타 산업과의 연계성이 매우 높은 기술로 5G/6G등을 활용할 국방, 로봇, 도시, 자동차 등 다양한 분야에 활용 가능
  - 산업 분야 별로 요구되는 다양한 종류 AI 디바이스들의 특성 활용을 지원함으로써 산업 분야별 지능화 및 지식의 융합을 제공하는 기회 제공
- [EdgeCPS] 기존의 클라우드 산업에 적용할 수 없었던 실시간 학습 및 추론 분야에서 중요 매개체 역할을 함으로써 클라우드/엣지의 서비스 커버리지를 극대화해 부가가치를 창출
  - 데이터를 수집하는 엔드포인트 영역에서 컴퓨팅을 자체적으로 수행하는 엣지 컴퓨팅을 통해 자율주행자동차 산업, 스마트팩토리 산업 등의 발전에 기여
  - 저성능 단말을 통한 고성능 엣지 시스템을 사용하고, 시스템 자원 가상화를 통한 자원 공유로 국가적으로 전력 및 CO2 절감 효과
- [이종가속기AI컴퓨팅] 각 산업 도메인 전문가의 IT 기술 장벽을 제거하여 기업의 혁신적 아이디어를 낮은 비용으로 쉽게 실현할 수 있는 기술 기반 마련 가능
- [이종가속기AI컴퓨팅] 이종 가속기를 이용한 고효율 컴퓨팅을 통해 대용량 인공지능 데이터 처리의 비용경쟁력 강화로 인공지능 관련 스타트업 및 중견기업이 보다 손쉽게 데이터를 가공함으로써 가치 창출 및 성장 기회의 향상

○ 경제적 기대효과

- [메모리 중심] 대용량·비휘발성 메모리를 이용한 대용량 메모리 기반 응용 가속 기술은 은행/금융/보험, 판매, 헬스케어/의료, 미디어, 교통/물류 등 산업뿐만 아니라 정부까지를 포함하는 매우 광범위한 시장에서 장시간 운영되어야 했던 대용량 금융 시뮬레이션의 성능을 극대화함으로써 경제적 문제에 대해서 빠른 의사 결정 도구로 활용
- [EdgeCPS] IoT/5G 시장의 성장 방향에 부합하는 서비스 플랫폼 기술 개발을 통해 관련 매출 및 세계 시장 점유 효과
  - 2013년 IoT 분야의 디바이스 관련 매출이 97.1%로 대부분을 차지했으나, 2022년에는 서비스 관련 매출의 비중이 52.6%를 차지하면서 성장을 주도할 것으로 예상
  - 국내 이동통신 관련 기업들이 5G/6G 관련 제품을 생산하고 고신뢰/저지연 모바일 데이터 서비스를 제공함에 따라, '17～'21년간 총 20조 원의 엣지 장비 시장과 총 10조 원의 국내 엣지 서비스 시장 창출 전망
- [EdgeCPS] 신규 망 투자비(CAPEX)와 기존 망 운영비 절감 효과와 산업 전반의 새로운 비즈니스 모델 창출 가능
  - 저성능 단말을 통한 고성능 엣지 시스템을 사용하고, 시스템 자원 가상화를 통한 자원 공유로 국가적으로 전력 및 CO2 절감 효과가 있으며, 통신사 입장에서는 관리비용 절감 및 용량의 변화를 능동적으로 대처해가는 것이 가능
  - 데이터를 수집하는 엔드포인트 영역에서의 컴퓨팅 성능을 개선하는 디바이스리스 컴퓨팅을 통해 자율주행자동차 산업, 스마트팩토리 산업 등의 발전에 기여
- [이종가속기AI컴퓨팅] 이종 가속기 AI 병렬컴퓨팅 기술에 기반하여 대규모 컴퓨팅 자원이 필요한 서비스를 손쉽게 개발할 수 있는 SW 기술을 확보함으로써 국가 경쟁력 향상 및 경제적 파급 효과 기대
- [이종가속기AI컴퓨팅] 이종 가속기 통합 인공지능 응용 가속 기술은 고효율 컴퓨팅을 통해 메타버스, 자율주행/교통, 의료, 미디어, 물류 등 산업뿐만 아니라 매우 광범위한 시장에서 장시간 운영되는 인공지능 서비스들의 경제성을 개선 가능

○ 사회적 기대효과

- [메모리 중심] 대용량 메모리 기반 유사 이미지 검색 알고리즘은 대용량 참조 데이터를 활용하여 기존 유사 이미지 검색 방법에 비해 100배 이상의 성능을 제공하므로, 이에 기반한 전국적으로 설치되어 있는 CCTV 등과 같은 IoT 장치로부터의 정밀하고 고속의 유사 이미지 검색을 구현할 수 있음으로 범죄 예방 등 사회 안정화에 기여
- [EdgeCPS] 사용자 체감 품질을 극대화할 수 있는 EdgeCPS기반의 초저지연 서비스를 구축함으로써 기존 산업의 활성화뿐만 아니라 창의적인 아이디어를 가진 국민들의 새로운 서비스 창출・창업을 통한 신산업 육성 및 신규 고용 확대가 가능
  - 초고속 차세대 이동통신과 저지연 엣지 서비스를 통해 제조업, ICT, 헬스 케어, 에너지 등 4차 산업혁명과 관련된 광범위한 분야에서 다양한 가치 창출 가능
- [EdgeCPS] 데이터 이용에 관한 규제 혁신과 개인정보 보호 협치를 목적으로하는 데이터 3법과 같이 안전한 데이터 이용을 위한 사회적 규범 정립이 진행되는 기조에 이바지
  - 엣지 사이버물리시스템이라는 분야별/지역적 최적화를 통해 민감한 정보는 내부에

서만 처리하는 지역성 서비스가 가능하고, 보안이 필요한 사용자에게 안전한 통신 서비스 제공

- 광범위한 디바이스 및 데이터센터에 프로세싱, 스토리지, 응용들을 분산시켜 단일 장애로 인한 네트워크 다운 방지나 안정적 서비스를 통해 정보 보호 강화

- [이종가속기AI컴퓨팅] 이종 가속기 통합 AI 병렬 컴퓨팅 기반 빅데이터 및 인공지능 기술은 국가적 복지 및 삶의 질 향상과 밀접히 관련되어 더욱 강력한 공공성이 요구되며 파급효과가 매우 큼
  - 정책적 지원을 통한 이종 가속기 통합 AI 병렬 컴퓨팅 기반 초대규모 데이터 분석용 범용 플랫폼을 구축하여 다양한 분야의 영세한 중소기업 활용성 제고
  - 의료, 보건 분야의 데이터 분석 및 인공지능 기반 기술의 공공성 확보를 통한 국민 건강, 복지 향상 기대

2. 현황 및 접근방법

가. 국내.외 현황

1) 국내.외 기술동향 및 수준

○ 데이터 처리 인공지능 컴퓨팅 기술

[국내・외 기술 동향]

- 대용량의 데이터 처리 및 분석에 최적화된 메모리 중심 컴퓨팅을 위한 시스템 인터커넥션 기술, 비휘발성 메모리 기술 융합, 이를 운영하는 SW 기술에 대한 기초 연구가 진행
- (시스템 인터커넥션 기술) 프로세서 중심 컴퓨팅에서 데이터 중심 컴퓨팅으로 변화하기 위해서 CPU, 가속기(GPGPU, FPGA, ASIC 등), 메모리(DRAM, SCM, NVM 등)를 연결하는 메모리 버스/인터커넥트 기술인 CCIX, OpenCAPI, Gen-Z, CXL 규격이 개발 중

※ Gen-Z는 직접 연결, 스위칭 또는 패브릭 토폴로지를 통해 메모리 또는 장치에 메모리 시맨틱 접근을 제공하는 개방형 시스템 인터커넥트로서 랙스케일 구조에 적합

※ CCIX는 프로세서, 가속기 그리고 메모리를 연결하는 버스 인터페이스

※ OpenCAPI는 프로세서가 캐쉬 일관성을 유지하면서 가속기 또는 입출력 장치에 부착할 수 있도록 하는 공개 인터페이스 아키텍처

※ CXL은 Intel에서 제공하는 캐쉬 일관성을 유지하며 프로세서, 메모리 및 가속기 연결하는 인터페이스

- (차세대 비휘발성 메모리) 비휘발성 메모리는 기존 DRAM에 비해 집적도가 높아 대용량화가 용이하고 데이터 보존을 위한 Refresh가 불필요하여 전력소모 면에서 유리하므로 차세대 메모리 기술로 주목

※ NVM은 연결 지점에 따라 메모리 버스 혹은 PCIe 버스에 연결되며 메모리 버스에 연결되는 DIMM Form Factor는 낮은 지연시간과 높은 대역폭을 제공, Load/Store 명령어를 통해 접근을 장점으로 하나, 각각의 NVDIMM이 메모리 슬롯을 차지, 상대적으로 작은 용량을 제공하는 단점을 가짐

※ PCIe 버스에 연결되는 NVM은 적절한 대역폭과 지연시간을 제공하고 프로그래밍 인터페이스가 잘 알려져있으나, CPU에 비해 낮은 성능을 제공하고 Multi-Path를 지원하지 않음

- (분리형 자원에 기반한 차세대 컴퓨팅 구조로의 변화) 데이터센터의 스케일은 크게 증가하고 있으나 서버의 물리적 경계를 넘은 자원 활용의 제약으로 인해 서버 내 활용되지 못한 채 남겨지는 유휴 자원으로 인한 비효율 문제가 대두되고 있음. 이를 극복하기 위해 컴퓨팅 자원을 종류 별로 모아 구성, 추상화하고, 이를 동적으로 할당받아 사용하는 분리형 컴퓨팅 구조(disaggregated computing)가 LegoOS(OSDI18) 연구를 통해 각광을 받고 있으며 관련 연구가 활발히 수행되고 있음
- (분리형 메모리 시스템) 대규모의 메모리를 장착한 메모리 서버와 이를 RDMA등을 통해 원격에서 접근하는 분리형 메모리 시스템을 통해 단일 기기에 설치된 메모리 자원의 양적인 한계를 극복하려는 연구가 활발히 수행되어 왔음. 최근에는 메모리 서버의 장애에 대응하고 저렴한 비휘발성 메모리나 스토리지를 활용하려는 연구가 수행되고 있음
- (인공지능 파이프라인 가속화 기술) 데이터 증강 기법이 복잡해짐에 따라 인공지능 데이터 로딩 파이프라인을 가속화하기 위한 연구가 주목받고 있음. 기존 CPU 자원 이외에 여러 GPU 자원을 활용하는 연구가 활발히 진행

○ 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 기술

[국내 기술 동향]

- 디바이스리스 컴퓨팅 기술은 관련연구가 전무하며 현재는 5G를 활용한 엣지 컴퓨팅, MEC에 대한 기술 개발이 활발히 진행 중
- 삼성, SK하이닉스 등 반도체 업종의 제품 불량 검출에 활용되는 이미지 검사 장비를 시작으로 엣지 컴퓨팅 기술 도입 중
- 서버 시스템, 통신 장비, 소프트웨어 솔루션 등을 공급하는 중소/중견 기업들은 자사의 솔루션을 엣지 컴퓨팅 개념 및 시나리오에 맞춰 고도화 진행 중
- 제조 기업은 시스템 통합(SI) 업체를 통해 산업용 사물 인터넷(IIoT)을 위한 엣지 컴퓨팅을 도입 준비 중이나, 제조 설비·공정의 글로벌 업체 종속성으로 인해 엣지 컴퓨팅 기술도 종속화 되고 있음
- 제조 및 생산 공정 분야의 일부 기업에서 산업용 엣지 컴퓨팅 도입을 준비 중이며, MEC의 경우 이동 통신사를 중심으로 향후 5G 서비스를 위해 기술 개념에 대한 실제 적용 준비를 진행 중
- 다양한 Edge 디바이스에 대한 Profiling 과 Modeling을 위한 지능형 Agent 기술은 사용자 및 시스템 요구사항에 따라 Profiling과 Modeling 기술을 자신들만의 목적에 맞게 기술하여 성능 측정, 성능 예측, 시스템 이상 징후 탐지 등에 활용 하고 있음
- 한국전자통신연구원에서는 2017년부터 수행한 “저지연 융합서비스를 위한 모바일 에지 컴퓨팅 플랫폼 기술 개발”을 통해 ETSI MEC 표준 기반의 MEC 플랫폼과 SDK 및 기술 검증을 위한 스마트 팩토리 PoC를 개발하였음

그림 4. ETRI-TTA MEC 5G 서비스 테스트베드 (판교 HPC 이노베이션 허브)

[국외 기술 동향]

- 인텔, 페이스북, 화웨이 등 글로벌 ICT 기업들은 차세대 인공지능 반도체 개발을 위한 기술 개발 및 투자 확대를 통해 독자적인 AI 생태계를 구축하고 있으며, 유럽에서는 유

럽 클라우드 파트너십과 ETSI를 통해 포그 및 엣지 컴퓨팅 기술을 연구 개발하고 있음

표 1. 유럽 지역 엣지 컴퓨팅 연구 동향

구분	주요 내용
유럽 클라우드 파트너십/ETSI	Network Edge Europe와 Cloud for Europe 등 유럽 클라우드 파트너십과 유럽전기통신표준협회(ETSI)를 통해 포그 및 엣지 컴퓨팅 지원 계획
유럽 위원회 (5G-PPP)	멀티 테넌시와 엣지 서비스를 위한 스몰 셀 협력(SESAME, Small-cell coordination for Multi-tenancy and Edge Services) 프로젝트에 자금 지원 중
	해당 프로젝트는 네트워크 가능 가상화(NFV)와 모바일 엣지 컴퓨팅(MEC), 자동 구성(SONs)으로 네트워크 정보 주위를 혁신하는 것을 목표로 함

※ 출처: KEIT(2017)자료를 기반으로 재정리

- 5G에 대한 기술 도입이 활성화 됨에따라 아시아 지역에서도 엣지 컴퓨팅에 대한 인식이 높아지면서 한국과 중국, 일본을 중심으로 빠르게 응용되고 있음

표 2. 아시아 지역 엣지 컴퓨팅 연구 동향

구분	주요 내용
중국	베이징에서 화웨이, 중국 과학원 심양 연구소, 중국정보통신 기술원(CAICT), Intle, ATM, ISoftStone이 공동으로 엣지 컴퓨팅 컨소시엄(ECC) 설립
	ECC는 제13차 5개년 계획의 일환인 디지털과 산업의 통합을 목표로 함
	중국은 엣지 컴퓨팅이 분산된 산업 자기 동체 아키텍처를 권장하고, 운영기술과 정보기술 간의 융합을 지원할 수 있다고 확신
일본	일본의 오픈 포그 컨소시엄은 미래 인터넷 인프라 구축을 위한 새로운 비즈니스 모델 개발과 표준화 수립에 기여할 수 있도록 미국전기전자학회(IEEE)와 제휴 체결
	해당 제휴는 Sakura Internet, Fujitsu, Toshiba가 참여했으며, 또한 오픈 테스트베드를 통한 기술 선진화와 개념화를 위해 IoT Acceleration Consortium(ITAC)과도 협력 중

※ 출처: 엣지컴퓨팅(Edge Computing)산업동향과 기술개발 및 R&D 동향, 2018

- 시스코(Cisco), 마이크로소프트(MS), 인텔(Intel), 델(Dell), IBM, HP 등 최고의 혁신 기술 서비스를 제공하는 기업들이 엣지 컴퓨팅 시장을 장악하고 있으며, 오픈 포그 컨소시엄(Open Fog Consortium)은 포그 또는 엣지 컴퓨팅 응용을 가속화 하는데 주력하고 있음
- 국립 대만 과학기술대학에서는 Edge 디바이스의 초기 모델 훈련을 위한 지도학습의 비용문제를 해결하기 위해 엘리트 기반 매칭 및 지식 재이용성을 접목한 Edge-to-Edge 협업 Transfer Learning을 제안하였음
- 광저우 대학교에서는 데이터 분류의 일반적인 AI 애플리케이션에서 Edge 디바이스의

학습 모델 품질과 연산 기능의 다양성을 높이기 위해 클라우드의 연합 학습 정확성과 효율성을 높이는 2차원 계약 선택적 모델 통합 접근법을 제안함

- 구글에서는 클라우드와 Edge의 장점을 결합하기 위해 다중 Edge 서버가 부분 모델 집계를 수행할 수 있는 HierFAVG 기반 클라이언트-에지 클라우드 계층적 연합 학습 시스템을 제안하고, 사용자가 직접 사용하는 스마트폰에서 데이터를 처리하고 모델을 강화하고 이 모델을 한 곳에 모아 더 정교한 모델을 만들어 다시 배포하는 연합학습 시스템을 개발중임
- 미국의 Visva에서는 지식 네트워크 구축, 고품질 추천, 새로운 유형의 검색, 사회적 기능을 지원하기 위한 Agent-Device 간의 쌍방향 알고리즘인 Evolutional Collective Intelligence에 의해 구성된 새로운 유형의 지식 그래프인 knowledge Network System을 제안함
- 토론토 대학교에서는 자율성 기반 자기최적화 공장 제조 과정을 위해 지능형 인지, 시뮬레이션, 이해, 예측, 최적화 및 제어 전략에 의한 자율적 제조를 지원할 수 있는 인텔리전트 디바이스트윈 제조 셀의 일반적인 프레임워크를 개발함

○ 엣지 사이버물리시스템을 위한 서버자원융합형 대규모 디바이스 가상화 기술 연구

[국내 기술 동향]

- 국내 기업의 디지털트윈은 걸음마 단계로 대기업, 통신사 또는 모델링 시뮬레이션 전문기업을 중심으로 성장하고 있으며, 수요기업에서는 구축 설비와의 호환과 신속한 보급이 가능한 외산 플랫폼을 통한 디지털트윈 구축을 선호하고 있음
- 제조 기업을 중심으로 생산공정의 디지털化에 대한 기술 개발이 진행되고 있으나 본격적인 디지털 트윈에 대한 기술개발은 진행되지 않음
- 한국전자통신연구원에서는 CPS(Cyber-Physical Systems) 기술을 바탕으로 개발된 모델링 시뮬레이션 기술을 디지털 트윈에 적용을 시도하고 있으나 초기 단계임
- 포스코는 다쏘시스템의 3D익스피리언스 플랫폼을 활용하여 ‘가상 시운전 시스템 및 가상 훈련 시스템’을 구축하여 제조공정 표준화와 시설 성능 개선에 활용하고 있으나 단순한 3D그래픽적인 요소를 적용한 수준에 그치고 있음
- CJ는 물류 풀밀먼트 서비스 분야에 디지털 트윈을 기반으로 한 가상 ‘쌍둥이 창고’를 구축하여 시뮬레이션을 통한 물류 과정의 변화를 진행할 계획임
- 현대자동차는 유니티(Unity)를 파트너사로 하여 제작 공장에 대한 공간, 기기, 센서들을 디지털 트윈 상에 설계하여 자체적인 제조 혁신을 추구할 예정임

[국외 기술 동향]

- 국외 시장에서는 글로벌 기업들의 내재화된 기술 개발 역량과 자사 및 타사의 클라우드 내에서 충분한 데이터로 디지털트윈을 구축을 바탕으로 디지털트윈 구축하고 세계 시장을 선점하고 있음
- 소프트웨어의 경우, 3D CAD/CAE 전문기업들이 3D 모델링 기능에 디지털 정보관리 및 시뮬레이션 기능을 확장해 PLM 기반의 디지털트윈 소프트웨어 시장을 선점

※ 가트너에 따르면 디지털트윈 활용 기업은 크게 5개로 분류(분석 및 데이터, 애플리케이션, 비즈니스 프로세스 관리, IoT 플랫폼, 서비스 제공자)

- 미국의 국가과학기술위원회(National Science and Technology Council)는 첨단 제조업 리더십 확보 전략을 통해 빅데이터 분석, 첨단 센서 및 제어기술, 모델링 및 시뮬레이션 등의 디지털트윈 관련 스마트 제조를 포함한 5대 전략 목표를 추진 하고 있음
- 미국의 Open Mobility Foundation은 디지털트윈을 이용하여 실시간 관리 및 안전하고 효율적인 도시 교통 시스템 구축 및 시뮬레이션 개발을 진행중임

※ Open Mobility Foundation: 미국 도시 교통망 관리를 위해 디지털트윈을 활용하고자 출범한 민간·지자체 협력 재단

- 미국에서 추진중인 Smart America는 도시의 도로 교통 시스템을 디지털트윈화하여 여정 시간· 안전 개선, 도로 혼잡 감소, 도로 시스템의 성능 개선을 통한 경제적 이익 변환을 위해 노력
- 독일에서는 디지털전략2025(digital strategy 2025)을 통해 CPS 기반 스마트팩토리 구축으로 개인 맞춤형 제품 기반의 비즈니스 모델 혁신을 통하여 새로운 시장 창출 및 매출 증대와 스마트팩토리 상품화를 통해 세계 제조장비 시장 주도권 확대 추진이며, 연구기관인 프라운호퍼에서는 다수의 독일 기업(지멘스, 보쉬 등)과 협업을 통해 제조에 알맞은 디지털트윈 구현 및 AI 데이터 인프라 플랫폼 구축을 진행 중임
- 2017년 영국 국가 인프라위원회(British National Infrastructure Commission)는 데이터 공유의 중요성과 디지털트윈의 필요성을 강조하며 '국가 디지털트윈(National Digital Twin)' 권고하고 런던, 맨체스터 등 영국의 도시들을 3D 모델로 구축, 교통, 날씨, 환경 정보 등 실시간 데이터를 연동하여 도시 운영 효율화를 진행 중임
- 일본에서는 IoT 기반 스마트팩토리를 위해 인프라 구축부터 데이터 확보와 디지털화를 위한 전문가 양성 등을 지원하는 Society 5.0 추진을 통해 데이터 구동형 사회를 전면에 내세움으로써 그간 디지털 기술 혁신에 따라 산업 현장에서 수집된 방대한 데이터를 활용한 경제사회의 발전 도모 의지를 표명
- 싱가포르에서는 14년 말부터 '18년까지 약 7300만 달러 투입하여 Virtual Singapore의 명칭으로 도시의 모든 구조물과 맵핑된 디지털트윈을 구현하여 도시계획·교통·환경 등 다양한 분야의 가상 플랫폼으로 활용하고 있음
- 두바이에서는 제브랄리 터미널을 대상으로 하는 버츄얼터미널은 항만 장비/차량의 상태 정보 등을 GPS 기반 3차원 지도 상에 가시화 하는 항만관제 모니터링 시스템 개발을 진행하고 제브랄리 터미널 3에 시범 적용하여 항만 생산성이 65% 이상 향상됨을 보였음
- 인도의 안드라프라데시 주정부는 65억 달러르 투입하여 주도인 아마라바티 개발을 위해 디지털트윈으로 구현, 기존 농작물을 재배하던 도시에서 친환경적인 최첨단 도시로 변화를 시도하고 있음
- 중국에서는 디지털트윈을 사이버-물리세계의 데이터 융합을 위한 방법으로 소개하고 전통산업과 첨단사업에 디지털 기술을 도입하여 중국 기업의 경쟁력 제고에 주력
- 지멘스 그룹은 아마존 웹 서비스(AWS)와의 협업을 통해 클라우드-엣지-디바이스를 아우르는 대규모 계층적 산업용 디지털 트윈 솔루션 개발을 진행할 예정임
- 롤스로이스사는 디바이스 트윈 기술을 활용하여 실기기에서 직접적으로 얻어지는 연소, 공기, 소재 상태와 함께 가상화된 센서에서 얻어지는 정보를 융합하여 엔진 제작의 오류 검출 테스트 과정에 활용하고 있음
- Amazon Web Service는 Amazon Elsatic Container Service(ECS)와 Amazon Elastic

Kubernetes Service (EKS)를 통해 클라우드 및 온프레미스 환경에서의 컨테이너 오케스트레이션 서비스를 제공하고, 컨테이너 애플리케이션의 개발 및 관리, 배포 서비스를 제공하며, AWS App2Containter 서비스를 통해 기존 애플리케이션을 컨테이너화를 지원함

- Google Cloud Platform은 Google에서 개발한 Kubernetes를 통해 클라우드 및 온프레미스 환경에서 컨테이너 오케스트레이션 서비스를 제공하고, Google Kubernetes Engine (GKE)을 이용해 자동확장, 출시 채널, 멀티 클러스터를 지원하는 완전 관리형 서비스를 제공함으로써 사전 빌드된 Kubernetes 애플리케이션 및 템플릿을 제공하여 생산성을 증대시키고, Migrate for Authos and GKE를 통해 가상 머신에서 기본 컨테이너로 기존 애플리케이션을 컨테이너화 기능을 지원함
- 미국의 ANSYS는 단일 워크플로 안에서 시뮬레이션 기반의 디지털 트윈을 구축해 검증, 배치할 수 있는 '앤시스 트윈 빌더(ANSYS Twin Builder)'를 탑재해 엔지니어가 물리적 제품의 디지털 트윈을 빠르게 구축, 검증하고 배치할 수 있도록 지원함
- BMW 사에서는 디지털 트윈을 이용한 시뮬레이션 기술을 이용하여 인력, 로봇, 공정 등 공장의 요소에 관한 계획 시간 단축과 유연성, 정밀도를 개선하여 전체 효율을 30% 증가시키는 계획을 수행 중임

○ 인공지능 연산 오프로딩 기술

[국내 기술 동향]

- SAPEON은 2020년에 AI 추론을 가속화하는 X220 NPU를 발표함. X220은 INT4, INT8, INT16 정수형 데이터 타입 연산을 수행하며, INT8 타입에서 최대 213TOPS 성능을 보임. X220의 전력 소모는 최대 135W 임. 2023년에는 추가로 부동소수점 데이터 타입 연산과 AI 학습을 가속화하는 X330을 발표함. X330은 INT8, FP8, FP16 데이터 타입 연산을 수행하며, FP8 타입에서 최대 734TFLOPS, FP16 타입에서 최대 368 TFLOPS 성능을 보임. X330의 열 설계 전력인 TDP(Thermal Design Power)는 최대 250W임. 2025년에는 HBM3를 탑재한 X430을 발표할 예정임.
- FuriosaAI는 2022년에 이미지 및 비디오 분석, 지능형 교통 관리, 초고해상도, 광학 문자 인식 및 자율 주행을 포함한 컴퓨터 비전 영역의 AI 추론을 가속화하는 Warboy를 발표함. INT8 데이터 타입 연산을 수행하며, INT8 타입에서 64TOPS 성능을 보임. Warboy의 TDP는 최대 60W임. 2024년에는 HBM3를 탑재하여 GPT 수준의 초거대 인공신경망 모델에 대한 학습을 목표로 발표할 예정임.
- Rebellions은 2021년에 엣지 컴퓨팅 시스템 환경에서 파이낸스 트레이딩, 컴퓨터 비전, 언어 처리 등을 위한 ION을 출시함. ION은 INT4, INT8, FP16 데이터 타입 연산을 수행하며, INT4 타입에서 32TOPS, INT8 타입에서 16TOPS, FP16 타입에서 4TFLOPS 성능을 보임. ION의 TDP는 최대 6W임. 2023년에는 엣지 컴퓨팅 시스템과 클라우드 컴퓨팅 시스템 환경에서 CNN(Convolutional Neural Network), LSTM(Long Short-Term Memory), BERT(Bidirectional Encoder Representations from Transformers) 및 최신 트랜스포머 네트워크 (T5, GPT 등) 등 다양한 컴퓨터 비전, 언어 처리를 가속화하는 ATOM을 출시함. ATOM은 INT8, FP16 데이터 타입 연산을 수행하며, INT8 타입에서 128TOPS, FP16 타입에서 32TFLOPS 성능을 보임. ATOM의 TDP는 최대 150W임.

- 삼성전자와 네이버는 2023년에 AI 반도체 칩 솔루션 시험 평가를 완료했다고 발표함. 양사가 개발한 AI 반도체는 네이버 하이퍼클로바X 등 AI 모델 구동에 활용될 것으로 예상됨. AI 경량화 기술 중 파인 그레인드 프루닝 분야에서 세계 최고 수준 성능을 확보하며, 저전력으로 구현하여 에너지 효율도 개선한 것으로 보임.
- 삼성전자는 세계 최초로 2021년에 개최한 Hot Chips 33에서 메모리 반도체와 인공지능 프로세서를 결합한 HBM-PIM(Processing-in-Memory)를 발표함. AI 시스템에 HBM-PIM을 설치할 경우, 기존 HBM2를 이용한 시스템 대비 성능은 2배 이상 증가, 에너지 소모는 70% 이상 감소함. 2023년 개최한 Hot Chips 35에서는 AMD의 GPU인 MI100 가속기 카드에 HBM-PIM 메모리를 탑재한 내용을 발표함. HBM-PIM 클러스터를 구현하여 대규모 AI 및 HPC 애플리케이션에 적용한 결과, 기존 GPU 가속기 대비 성능은 2배 증가, 에너지 소모는 50% 감소함. 또한, HBM 뿐만 아니라 LPDDR 및 GDDR 메모리에도 PIM을 적용할 수 있음을 보임.
- SK하이닉스는 2022년에 개최한 ISSCC에서 GDDR6-AiM(Accelerator in Memory) 샘플을 발표함. GDDR6-AiM은 머신러닝, 고성능 컴퓨팅, 빅 데이터의 연산과 저장 등에 활용될 전망임. GDDR6-AiM을 CPU 및 GPU와 함께 탑재하면 일반 DRAM 대비 특정 연산의 성능은 최대 16배 증가함. GDDR6의 기존 동작 전압인 1.35V보다 낮은 1.25V에서 동작하며, 자체 연산을 하는 AiM이 CPU 및 GPU로의 데이터 이동을 줄여 기존 제품 대비 에너지 소모는 80% 감소함. 2023년에 개최한 AI Hardware & Edge AI Summit에서는 GDDR6-AiM 기반의 생성형 가속기 카드인 AiMX 시제품을 공개함. Meta의 생성형 AI인 OPT(Open Pre-trained Transformer) 13B 모델을 AiMX 시제품을 탑재한 서버 시스템에서 시연한 결과, 기존 GPU를 탑재한 시스템 대비 성능은 10배 증가, 전력 소모는 1/5 수준을 보임

[국외 기술 동향]

- NVIDIA는 2021년에 데이터센터에서 AI 학습 및 추론, 데이터 분석, HPC 등을 수행하는 A100 텐서 코어 GPU를 발표함. A100은 NVIDIA 암페어 아키텍처, 최대 7개의 GPU 인스턴스로 파티셔닝, 초당 2TB/s 이상의 메모리 대역폭을 지원하고, 데이터 타입 연산은 INT8, FP16, BF16, TF32, FP32 FP64를 수행함. INT8 타입에서 최대 1,248TOPS, FP16 및 BF16 타입에서 최대 624TFLOPS, TF32 타입에서 최대 312TFLOPS, FP32 타입에서 19.5TFLOPS, FP64 타입에서 최대 19.5TFLOPS 성능을 보임. A100의 TDP는 최대 400W임. 2023년에는 A100보다 대규모 언어 모델에서 최대 30배 더 높은 성능을 제공하는 H100 텐서 코어 GPU를 출시함. H100은 NVIDIA NVLink 스위치 시스템을 사용하여 최대 256개의 H100 연결을 지원하고, 데이터 타입 연산은 INT8, FP8, FP16, BF16, TF32, FP32, FP64를 수행함. INT8 타입에서 최대 7,916TOPS, FP8 타입에서 최대 7,916TFLOPS, FP16 및 BF16 타입에서 최대 3,958TFLOPS, TF32 타입에서 최대 1,979TFLOPS, FP32 타입에서 최대 134TFLOPS, FP64 타입에서 최대 134TFLOPS 성능을 보임. H100의 TDP는 최대 700W임. 2024년에는 H200, 2025년에는 X100을 출시할 예정임
- AMD는 2021년에 HPC 및 AI 데이터센터에서 사용되는 MI250 가속기를 출시함. MI250은 AMD CDNA2 아키텍처, AMD Infinity 아키텍처, AMD ROCm 기술을 지원하고, 데이터 타입 연산은 INT4, INT8, FP16, BF16, FP32, FP64를 수행함. INT4 및

INT8 타입에서 최대 383TOPS, FP16 및 BF16 타입에서 최대 383TFLOPS, FP32 및 FP64에서 최대 95.7TFLOPS 성능을 보임. MI250의 TDP는 최대 560W 임. 2023년에는 생성형 AI 워크로드 및 HPC 어플리케이션에서 성능을 향상시킨 MI300 가속기를 출시함. MI300은 AMD CDNA3, AMD Infinity 아키텍처, AMD ROCm 기술을 지원하고, 데이터 타입 연산은 INT8, FP8, FP16, BF16, TF32, FP32, FP64를 수행함. INT8 타입에서 최대 5.22POPS, FP8 타입에서 최대 5.22PFLOPS, FP16 및 BF16 타입에서 최대 2.61PFLOPS, TF32 타입에서 최대 1.3PFLOPS, FP32 및 FP64에서 최대 163.4TFLOPS 성능을 보임. MI300의 TBP(Typical Board Power)는 최대 750W 임

- Intel은 2022년에 데이터센터 전용 GPU인 Flex Series를 출시함. Flex Series는 INT8, FP32 데이터 타입 연산을 수행하며, INT8 타입에서 최대 250TOPS, FP32 타입에서 16TFLOPS 성능을 보임. Flex Series의 TDP는 최대 150W임. 2023년에는 Max Series를 출시함. Max Series는 INT8, BF16, FP32, FP64 데이터 타입 연산을 수행하며, INT8 타입에서 최대 1,664TFLOPS, BF16 타입에서 최대 832TFLOPS, FP32 및 FP64 타입에서 최대 52TFLOPS 성능을 보임. Max Series의 TDP는 최대 600W 임
- Google의 TPU(Tensor Processing Unit)는 2016년 TPU v1부터 2023년 TPUv5e까지 공개함. TPU는 ASIC(Application Specific Integrated Circuits)으로 설계되었으며, 기계학습 워크로드를 가속화하는데 사용됨. 2021년에 발표한 TPU v4는 2개의 텐서 코어로 구성되며, 각 텐서 코어는 4개의 MXU (Matrix Multiplication Unit), 1개의 벡터 유닛, 1개의 스칼라 유닛으로 구성됨. TPU v4는 INT8, BF16 데이터 타입 연산을 수행하며, INT8 및 BF16 타입에서 최대 275TFLOPS 성능을 보임. TPU v4의 전력 소모는 최대 192W 임. 2023년에 발표한 TPU v5e는 1개의 텐서 코어로 구성되며, 각 텐서 코어는 4개의 MXU, 1개의 벡터 유닛, 1개의 스칼라 유닛으로 구성됨. TPU v5e는 INT8, BF16 데이터 타입 연산을 수행하며, INT8 타입에서 최대 393TOPS, BF16 타입에서 최대 197TFLOPS 성능을 보임. TPU v5e의 전력 관련 지표는 공개되지 않음
- Meta는 2023년에 MTIA (Meta Training and Inference Accelerator)를 발표함. MTIA는 ASIC으로 설계되었으며, AI 워크로드 중 DLRM (Deep Learning Recommendation Model)을 가속화하는데 사용됨. MTIA는 INT8, FP16 데이터 타입 연산을 수행하며, INT8 타입에서 102.4TOPS, FP16 타입에서 51.2TFLOPS 성능을 보임. MTIA의 TDP는 25W임
- Tesla는 2021년에 자체 개발한 슈퍼컴퓨터 Dojo와 Dojo에 탑재되는 D1 칩을 발표함. Dojo는 차량에서 생성되는 데이터를 이용하여 AI 학습을 수행하는데 활용할 예정임. D1은 CFP8, CFP16, BF16, FP16, FP32 데이터 타입 연산을 수행하며, CFP8 및 BF16 타입에서 362TFLOPS, FP32에서 22TFLOPS 성능을 보임. D1의 TDP는 400W임

2) 국내.외 표준화 현황(또는 향후 기술 발전 추세)

○ 데이터 처리 인공지능 컴퓨팅 기술 표준화

[국내・외 표준화]

- (컴퓨팅 시스템 구조 표준화) Gen-Z 공개 표준으로 Intel의 QPI 규격에 의존하는 한계를 탈피하려는 시도가 진행되고 있으며, 50개 이상의 기술 대기업들이 CCIX, OpenCAPI, Gen-Z, CXL 컨소시움에서 힘을 합치고 있음

※ 2016년 10월, 30개 이상의 IT회사가 연합하여 Gen-Z 컨소시움을 구성하고 인터커넥트 표준화를 진행하고 있음. 이 컨소시움은 2017년 7월에 Gen-Z Draft Core Specification과 Scalable Connector Specification v1.0을 발표

※ 2016년 5월, AMD, ARM, 화웨이, IBM, Mellanox, 퀄컴, Xilinx가 연합하여 캐쉬 일관성 패브릭 인터페이스인 CCIX 표준을 만들기 시작했고 2017년 11월 현재 CCIX 표준을 멤버 회사에 제공

※ 2016년 10월, AMD, Dell EMC, Google, HPE, IBM Mellanox, Micron, NVIDIA, Xilinx가 연합하여 데이터센터 서버 성능을 10배 향상시킬 수 있는 새로운 공개 표준인 OpenCAPI를 발표함. OpenCAPI는 현재 Transaction Layer Specification v3.1과 Data Link Layer Specification v3.0을 제공

※ Intel은 Compute Express Link (CXL) 인터커넥트 표준을 통해 프로세서 내의 캐쉬 일관성을 유지하는 메모리 sementic을 가질 수 있는 주변 장치의 구현을 추진 중. 이러한 구조는 기존의 페이지기반으로 관리, 구현되고 있는 분리형 메모리 기술의 제약을 근본적으로 극복할 수 있는 기술로 관심을 받고 있음

- (비휘발성 메모리 규격 표준화) JEDEC Hybrid Memory Task Group과 SNIA(Storage Networking Industry Association) NVDIMM SIG에 의해서 DIMM Form Factor의 NVRAM은 NVDIMM-F와 NVDIMM-N이 표준화되어 있고 NVDIMM-P의 표준화가 2018년 완료될 예정

※ ACPI 6.2, UEFI Specification 2.7, JESD245A에 의해서 산업 명세화가 되어 있으며 윈도우, VMware, 그리고 리눅스 등의 운영체제에 의해서 지원

○ 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 기술

[국내 표준화]

- 빅데이터 유통 플랫폼에서 데이터 거래 시 일정한 기준을 달성할 수 있도록 데이터 추출, 데이터 변환, 데이터 정제라는 세 가지 주제를 기반으로 기능별 정의를 진행함
- TTA TC4 PG 417(지능형 반도체 프로젝트 그룹)에서는 인공지능 알고리즘을 효율적으로 수행하기 위한 지능형반도체 분야 표준화를 추진하고 있음
- 반도체공학회에서는 지능형반도체 포럼을 운영하고 있으며, 프로세서 구조, 인터페이스, 스파이킹 뉴럴넷 하드웨어 분야의 지능화반도체 기술 표준화와 신경망 압축의 지능형반도체 신경망 기술 표준화를 추진하고 있음

[국외 표준화]

- IEC 61507에서는 전자부품의 기능 안정성을 정의하고 있으며, IEC TC 47에서는 전자부품 및 반도체와 관련한 표준을 제정하고 있음
- 유럽통신표준기구의 MEC 표준화 단체인 ETSI에서는 MEC의 개념과 목적을 정의하고 시나리오에 따른 요구사항과 해결과제를 제시하는 규격 작업이 진행되고 있으며, MEC의 High-level 아키텍처 청사진을 제시하고 있으며 Active Device Location Tracking, Augmented Reality Content Delivery, Video Analytics 등의 유즈케이스 및 서비스 시나리오 별 개체(Entity)간 인터페이스 및 요구사항을 정의함

○ 엣지 사이버물리시스템을 위한 서버자원융합형 대규모 디바이스 가상화 기술 연구

[국내 표준화]

- 국내 TTA CPS PG609에서는 CPS 표준 제정을 주도하고 있으며, 스마트 제조 및 디지털트윈에 대한 국내 표준화 작업을 진행하고 있음
- 한국(ETRI)은 '17.11월 ISO/TC184/SC4 총회에서 디지털트윈 표준에 관한 New Project 4건을 제출하였고, '19.9월 Committee Draft가 통과하여, 국내 표준 제정을 주도적으로 수행하고 있음

- TTA 사물인터넷 네트워킹 프로젝트그룹(PG1002)은 다양한 이기종 기기들을 상호 연동을 위해 네트워크 단위로 통합시키기 위한 네트워크 기술 및 품질 측정에 관한 표준 제정을 진행하고 있음

[국외 표준화]

- ISO 23247에서는 제조를 위한 디지털트윈 프레임워크 제정 논의 진행 중이며, ISO 23247-1(개요 및 일반 원칙), ISO 23247-2(참조 아키텍처), ISO 23247-3(디지털 표현), ISO 23247-4(정보 교환)에 대한 세부 표준 제정 활동을 진행중
- ISO/TC 184/SC 4(산업데이터)는 디지털트윈에 대한 표준화 작업을 진행하고 있으며, WG 15에서 디지털 제조 표준화 담당하고 있음
- ISO/IEC JTC1: 디지털 트윈 기술 동향 보고서를 발표함, IEEE P2806: 팩토리 환경에서 물리적 객체에 대한 디지털 표현 시스템 아키텍처를 정의하는 것을 목표로 하는 프로젝트 시작
- ISO/IEC JTC1/SC 29 에서는 오디오, 비디오와 같은 멀티미디어 및 하이퍼미디어 정보에 대한 데이터 포맷 정의와 더불어 가상 세계에서 정보를 주고받기 위한 표준 제정을 진행하고 있음
- ISO/IEC JTC1/SC 35 에서는 높은 접근성으로 모든 사용자에게 감각(시각, 청각, 촉각 및 기타 감각)를 포함한 정보 전달을 위해 정보통신 기술환경에서 사용자-시스템간 인터페이스 제공을 위한 표준화를 진행하고 있음

○ 장치 중심 컴퓨팅 기술 표준화

- (SmartNIC/DPU) SmartNICs Summit은 데이터 센터에서 수행되는 워크로드의 상당 부분을 오프로드하는 SmartNIC 및 DPU 기술 관련 업계 전문가, 공급 및 소비 업체가 모여 모범 사례, 현재 동향, SmartNIC 기술이 하이퍼스케일 데이터 센터, 기업, 엣지 시설에 배포되는 방법을 공유하고 있으나 표준화는 하지 않고 있음. 업체는 Intel, AMD, NVIDIA, Arm, Canonical, F5, Achronix, Keysightm, Juniper Networks, Dell'Oro Group 등이 참여함.

※ 2022년 4월, SmartNICs Summit 2022 개최

※ 2023년 6월, SmartNICs Summit 2023 개최

- (InfiniBand 표준화) IBTA(InfiniBand Trade Association)는 Future I/O와 Next Generation I/O(NGIO) 기술이 통합되어 1999년에 설립됨. InfiniBand 아키텍처 사양을 유지하고 발전시키는 일을 담당하고 있음. 기술은 InfiniBand 뿐만 아니라 RDMA 소프트웨어 아키텍처, RoCE(RDMA over Converged Ethernet)를 다룸. 운영 위원회는 HPE, IBM, Intel, NVIDIA 등이 포함됨. InfiniBand 사양 중 Volume 1은 InfiniBand 아키텍처의 핵심으로 구성되어 InfiniBand 스위치, 라우터, 호스트 채널 어댑터 및 I/O 장치용 대상 채널 어댑터, InfiniBand 패브릭 관리를 위한 전체 아키텍처 등에 대한 요구사항이 작성됨. Volume 2는 아키텍처의 물리적 측면이 작성됨.

※ 2001년 10월, IBTA는 Volume 1 Release 1.0, Volume 2 Release 1.0을 발표. SDR(Single Data Rate)에 대해 정의.

※ 2015년 3월, IBTA는 Volume 1 release 1.3을 발표. 컴퓨터 시스템이 고성능 컴퓨팅과 상용 엔터프라이즈 데이터 센터 모두에 대한 높은 컴퓨팅 효율성 및 고가용성과 함께 증가된 확장성과 대역폭에 대한 요구사항을 충족할 수 있도록 하는 새로운 기능을 정의.

※ 2021년 8월, IBTA는 Volume 1 release 1.5를 발표. MPE(Memory Placement Extensions)에 대한 지원이 강조된 새 릴리스는 고성능, 엔터프라이즈 및 클라우드 데이터 센터에서 사용되는 PMEM(Persistent Memory) 애플리케이션의 대기 시간을 개선.

※ 2023년 10월, IBTA는 Volume 1 release 1.7, Volume 2 release 1.5를 발표. XDR(Extreme Data Rate)의 초기 사양이 포함되어 있으며 IBTA 로드맵에서 차세대 속도의 기반을 마련하여 AI 및 HPC 데이터센터의 고속 성능 및 확장성 요구사항을 충족하기 위해 이전 세대보다 두 배의 대역폭을 제공.

- (Ethernet 표준화) IEEE 802.3은 유선 이더넷의 물리 계층과 데이터 링크 계층의 MAC(Media Access Control)을 정의하는 작업 그룹이자 표준임. 일반적으로 LAN(Local Area Network) 기술을 다루며 일부 WAN(Wide Area Network) 기술도 다룸. 또한, IEEE 802.1 네트워크 아키텍처를 지원하며, CSMA/CD를 사용하여 LAN 액세스 방법을 정의함. 물리적 연결은 다양한 유형의 구리 또는 광섬유 케이블을 통해 노드 및 인프라 장치(허브, 스위치, 라우터) 간에 이루어짐.

※ 1983년, IEEE 802.3은 동축케이블 기반 이더넷 표준화 발표

※ 1995년, IEEE 802.3u는 100Mb/s 이더넷 표준화 발표

※ 1998년, IEEE 802.3z는 1Gb/s 이더넷 표준화 발표

※ 2002년, IEEE 802.3ae는 광케이블 기반 10Gb/s 이더넷 표준화 발표

※ 2010년, IEEE 802.3ba는 40Gb/s, 100Gb/s 이더넷 표준화 발표

※ 2017년, IEEE 802.3bs는 200Gb/s, 400Gb/s 이더넷 표준화 발표

※ 2022년, IEEE 802.3ck는 100Gb/s 레인을 사용하여 100Gb/s, 200Gb/s, 400Gb/s 이더넷 표준화 발표

※ 2024년, IEEE 802.3df는 100Gb/s 레인을 사용하여 200Gb/s, 400Gb/s, 800Gb/s 이더넷 표준화 발표 예정

※ 2026년, IEEE 802.3dj은 200Gb/s 레인을 사용하여 200Gb/s, 400Gb/s, 800Gb/s, 1.6Tb/s 이더넷 표준화 발표 예정

3) 동일, 유사내용에 대하여 국내.외 관련자들의 수행내용

○ 대용량 데이터 처리 고성능 컴퓨팅 구조 연구

- (HP Enterprise) 2017년 공개한 The Machine 프로토타입은 다수의 노드가 패브릭으로 연결된 대용량의 NVRAM을 공유하는 구조로, 40개의 물리 노드를 이용하여 1,280 코어들과 160TB 메모리를 제공

※ 패브릭: 노드 및 메모리를 연결하기 위한 인터커넥트를 의미하며, The Machine 프로토타입의 경우에는 FPGA 기반 Gen-Z 버스를 사용

※ 기존 리눅스 운영체제의 확장 및 개선 프로젝트(L4TM, Linux For The Machine)를 진행하고 있으며, FAM(Fabric Attached Memory)의 Non-Coherent 접근 지원 라이브러리, FAM 지원 공유 파일시스템 및 라이브러리 등을 개발 중

- (Numascale) NumaConnect는 통합서버의 모든 프로세서가 CPU 캐쉬일관성을 유지하면서 모든 메모리와 I/O 장치에 접근 가능하도록 지원

※ NumaConnect 어댑터와 Numa-Chip으로 구성되어 최대 4,096 노드까지 공유 메모리 및 캐쉬 일관성 지원을 통한 싱글 시스템 구성 가능

※ HW 기반으로 싱글 시스템 구성에 필요한 기술을 지원하므로 리눅스, 솔라리스, 윈도우 서버 등 기존 운영체제 운용 가능

- (Academia) 학계에서는 분리형 컴퓨팅 구조 연구에 대한 많은 연구가 진행

※ LegoOS 논문에서 메모리 서버의 장애에 대해 일부 다루고 있으나 단순한 chain replication에 불과하고 재현이 불가능한 문제점이 지적되고 있음

※ InfiniSwap (NSDI'18)에서 원격 메모리 서버의 장애에 대응하기 위해 로컬의 스토리지를 활용하는 방안을 제안하고 있으나 로컬 스토리지의 성능이 전체 시스템 성능의 병목으로 작용하고 있음

※ FastSwap (EuroSys'20)은 분리형 메모리 시스템에서 원격 메모리의 정보를 asynchronous하게 처리하여 원격 메모리 접근 속도를 최적화 하는 방안을 제안하고 있으나 메모리 서버의 장애에는 적절히 대응하지 못하고 있음

※ Leap (USENIX ATC'20)은 분리형 메모리 시스템에서 prefetch의 정확성을 개선하여 데이터 접근 효율을 개선시키는 방안을 제시함. 하지만 여러 threads/context가 동작하는 환경에서 target 페이지를 제대로 판별하지 못하는 문제점이 지적되고 있으며, 또한 메모리 서버의 장애에 대해 대응하지 못함

※ DiLOS (ApSys'21)은 리눅스 커널의 스왑 시스템의 페이지폴트를 최적화하여 응용이 분리 메모리를 높은 성능으로 활용할 수 있는 프레임워크 개발

※ Clover (USENIX ATC'20)은 계산 노드와 스토리지 노드를 분리하고 계산노드가 스토리지 노드에 직접 접근하여 관리할 수 있으며, 중앙 조정자를 통해서 계산 노드와 비휘발성 메모리 노드간 통신을 조정하여 데이터센터 구축 및 연산 비용을 줄임

- (Academia) 인공지능 데이터 파이프라인 가속화 기술에 대한 연구가 활발히 진행 중

※ TrainBox (Micro'20)은 FPGA를 활용하여 데이터 증강 기법을 CPU대신 처리하도록 하여 성능 향상 제공

※ Data Refurbishing (ATC'21)은 인공지능 데이터 로딩 파이프라인을 가속화하기 위해 최적화 알고리즘 제공

※ Gandiva(OSDI'18) 하나의 GPU자원을 공유하여 여러 인공지능 모델을 학습하는 연구 수행

○ 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 기술

- IBM과 오스트리아 빈 공과대학교는 서버리스 컴퓨팅의 패러다임을 엣지 환경으로 확장한 디바이스리스 엣지 컴퓨팅을 통해 가상화된 공간에서 리소스 풀 구성과 자원 공유의 탄력성을 제공하여 IoT와 엣지 디바이스를 통합 관리하는 방식을 제안

- 이는 임의의 센서로부터 획득된 데이터를 다른 디바이스나 서비스에서 활용하기 위한 Sensor-Cloud Infrastructure 연구와 네트워크로 연결된 디바이스들의 센서와 액추에이터를 가상 공간에서 관리해 서비스로 제공(sensing and actuating as a service)하는 OpenIoT framework의 개념을 구체화

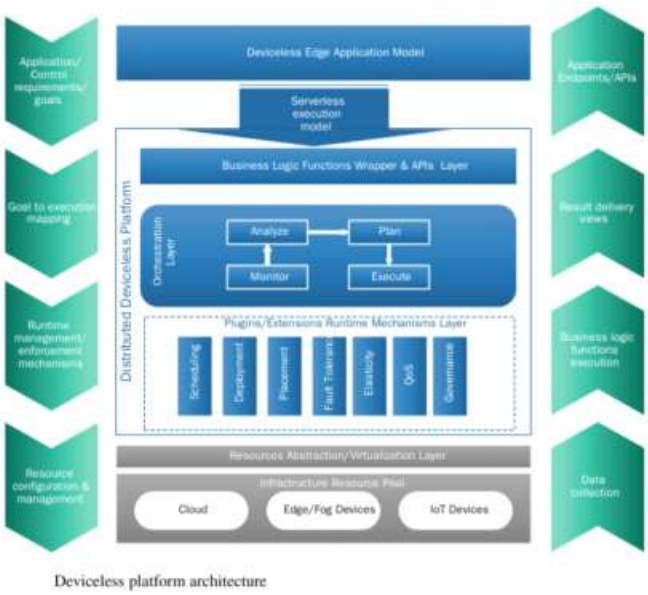


그림 5. Deviceless 플랫폼 아키텍처

- 오토모티브 엣지컴퓨팅 컨소시움(AECC, Automotive Edge Computing Consortium)은 2017년 8월, 인텔, 덴소, 에릭슨, NTT, NTTDoCoMo(도코모), 도요타 정보기술센터 및 도요타 자동차 등으로 결성된 조직으로 커넥티드카 생태계를 구축하기 위해 클라우드 컴퓨팅 기반 운전 지원, 실시간 데이터 지도 작성 등의 서비스 지원을 목표로 개발 중임

- 엣지 컴퓨팅 컨소시엄 (ECC, Edge Computing Consortium)는 154개사에서 참여하여 스마트 제조, 시티, 에너지&전력, ICT산업 부문에서 글로벌 기업 및 관련 분야의 연구소와 대학까지 포함하여 현재까지 21건의 테스트베드 솔루션을 완성함

표 3. 엣지 컴퓨팅 컨소시엄

구분	주요 내용
스마트 공장	사전 오류를 탐지, 불량품으로 인한 처리 비용 최소화 및 재고 저감을 목표로 개발 중
	엣지 컴퓨팅 도입을 통한 유지보수 및 트래픽 비용 절감 기대
스마트 빌딩	IoT 디바이스에서 전압, 전류 등의 데이터를 분석하고, 클라우드 데이터센터에서 전체를 통합하는 스마트 빌딩 또는 빌딩관리시스템 개발을 목표로 설정
스마트홈	서비스와 시스템관리가 엣지 OS에서 처리되는 형태로 스마트홈 시스템 개발 중
스마트 시티	도시에서 발생하는 방대한 데이터를 엣지 컴퓨팅을 통해 해결하고, 실시간 서비스를 위해 저지연 특성을 가지도록 설계

※ 출처: 중소기업 기술로드맵(2018-2020) 자료를 기반으로 재정리

○ 엣지 사이버물리시스템을 위한 서버자원융합형 대규모 디바이스 가상화 기술 연구

- 부산시에서는 '에코델타시티'라는 디지털트윈 활용 3D 설계 기술로 가상 도시 구축을 통해 시민·전문가가 가상공간에서 미리 체험하여 의견 교환, 향후 도시통합운영시스템과 연계하여 도시 관리에 활용 예정
- 전주 시에서는 디지털트윈으로 현실과 동일한 가상환경을 구축해 다양한 모의시험을 진행할 수 있는 '버추얼 전주' 플랫폼을 구축할 예정임
- 서울시는 국내 기업 솔루션과 데이터를 이용하여 3D 기반의 디지털트윈 '버추얼 서울'을 구축해 서울시 정책 결정 지원 및 다양한 실험으로 도시 문제 해결의 지렛대로 적극 활용 계획
- 세종특별자차시는 한국전자통신연구원과 함께 도시 행정 디지털트윈을 구축하여, 안전·환경·복지·교통 등 다양한 분야에 활용되고, 지능정보기술을 융합한 새로운 서비스를 개발해 제공할 예정

○ 인공지능 시스템 연결망 기술

- (CXL) 프로세서 중심 컴퓨팅에서 데이터 중심 컴퓨팅으로 변화하기 위해 CPU, 가속기(GPGPU, FPGA, ASIC 등), 메모리(DRAM, SCM, NVM 등)를 고속으로 연결하기 위한 차세대 인터커넥트 기술인 CXL이 활발히 개발되고 있음
- (CXL 표준 동향) 2019년, Intel을 중심으로 한 CXL 컨소시엄에서는 CPU, 메모리, 그리고 가속기를 캐시 일관성을 보장하며 고속으로 연결하는 기능을 제공하는 CXL 1.0 표준을 시작으로, 멀티 레벨 스위칭 및 패브릭 기능을 제공하는 CXL 3.1 표준까지 공개하였음
- (CXL HW 동향-호스트 프로세서) Intel에서는 2023년 CXL 1.1을 지원하는 CPU인 Xeon 계열 Sapphire Rapids를 공개하였으며, AMD에서는 CXL 1.1을 지원하는 CPU인 EPYC 계열 Genoa를 2022년 공개하였음. 각 CPU 벤더에서는 CXL 버전별로 이를 지

원하는 CPU 개발 로드맵을 공개하고, 이를 기반으로 개발을 진행 중임

- (CXL HW 동향-CXL 디바이스-메모리 확장장치) 대표적인 메모리 벤더인 삼성전자, SK Hynix, 그리고 Micron에서는 자사 메모리 모듈을 탑재한 CXL 2.0 기반 메모리 확장장치를 공개하였으며, 제품에 따라 96GB에서 512GB의 확장 메모리를 지원함
- (CXL HW 동향-CXL 디바이스-CXL 스위치) 미국의 XConn 및 Enfabrica에서는 CXL 2.0 및 3.0 표준의 주요 특징인 리소스 풀링을 지원하기 위한 CXL 스위치를 개발하였음. XConn 사의 CXL 스위치인 Apollo는 CXL 2.0을 지원하는 스위치로서 256개의 lane과 32개의 포트를 제공함
- (CXL SW 동향-삼성/SK Hynix/Memverge) 삼성전자 및 SK Hynix에서는 자사 메모리 확장장치를 지원하기 위한 SDK를 개발하고 이를 github에 공개하여 유저가 자사 제품을 효율적으로 이용할 수 있는 환경을 제공함. 미국의 Memverge에서는 CXL 기반 공유 메모리 SW 아키텍처인 GISMO를 통해 분산 환경에서 멀티 노드 간 실시간 데이터 공유를 제공하여, 네트워크 IO를 제거하고 이에 따른 데이터 전송 지연 시간을 줄임
- (InfiniBand) NVIDIA는 최근에 400Gbps 속도를 지원하는 InfiniBand 네트워크 기술인 NVIDIA Quantum-2를 출시하고 800Gbps급 기술을 개발하고 있음
- NVIDIA는 ConnectX-7 25/50/100/200/400G Ethernet NIC을 출시하였으며, 400Gbps 32포트를 지원하는 SN4700 이더넷 스위치를 제공하고 있음. Cisco는 최대 400Gbps 2포트를 제공하는 Nexus 9232E 32-port Switch를 발표함. UfiSpace는 32x400G QSFP-DD Switch인 S9310-32D 모델을 출시함
- NVIDIA는 Spectrum-4 SN5600 Series 스위치 제품을 통해 최대 800Gbps를 64 포트 제공하는 제품을 조만간 출시 예정임

4) 동일, 유사내용과 관련하여 제안자가 이미 수행한 사업 또는 연구개발과제

○ 대용량 패브릭 메모리에 대한 실현 가능성, 클라우드 컴퓨팅 환경에서의 요구사항 개발, 및 IP우선확보를 위해 다음 과제의 세부 기술 연구로서 진행

- 차세대 메모리 기술 기반 초저전력 컴퓨팅 시스템과 페타바이트급 메모리 클라우드 핵심 기술 개발 (2017.1.1.~2018.12.31.)

5) 국내.외 경쟁기관 현황

기술구분	경쟁기관 현황
빅데이터 관리 기술	- (Amazon AWS) · 빅데이터 저장, 전처리, 분석, 실시간 스트리밍 데이터 분석과 배치 데이터 분석을 지원 · Amazon 솔루션들(S3, Dynamo DB. Redshift, Kinesis 등)과 공개 SW들(Hadoop, SPARK, Mahout, Hive 등)로 구축 - (Google Cloud Platform) · Hadoop과 SPARK 기반의 빅데이터 분석 - (IBM Watson Data Platform)

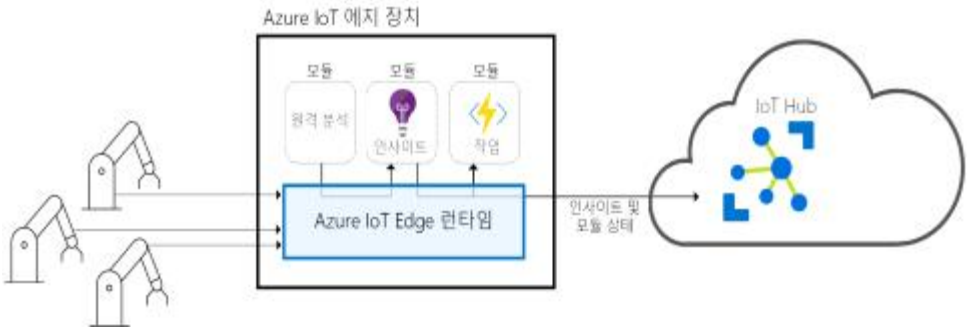
기술구분	경쟁기관 현황
	· Hadoop과 SPARK 기반의 빅데이터 분석 - (Microsoft Azure) · 일반 데이터 소스, 응용, 센서 및 디바이스들로부터 수집된 데이터를 HDFS 기반의 데이터 레이크에 저장 · Apache 빅데이터 프로젝트들(Hadoop, SPARK, Pig, Hive, HBase 등) 기반으로 데이터 분석 - (GridGain Ignite) · 스케일러블 인메모리 컴퓨팅 플랫폼으로 OLTP, OLAP, HTAP 응용들과 스트리밍 분석 및 머신러닝 응용들을 지원
지능형 분석 기술	- (Amazon ML Cloud & AWS Deep Learning AMI(Amazon Machine Image)) · 기계학습 및 딥러닝 관련 작업 환경 제공 · Python 환경을 포함한 Conda 기반 AMI 제공 · 다양한 딥러닝 프레임워크 지원 : Apache MXNet, TensorFlow, Microsoft Cognitive Toolkit(CNTK), Caffe, Caffe2, Theano, Torch 및 Keras - (Google Cloud Machine Learning Engine) · 딥러닝 엔진인 TensorFlow를 'Google Cloud Machine Learning'이라는 이름으로 제공하며 텐서플로 전용 CPU인 TPU(Tensor Processing Unit)를 이용하여 실행됨 · Cloud Vision API, Cloud Speech API, and Google Cloud Translation API 제공 - (IBM Watson Machine Learning) · 다양한 딥러닝 프레임워크 지원 : TensorFlow, Theano, Keras, Lasagne · 다양한 기계학습 응용 개발 환경 제공 : Spark ML, Python, Scala - (Microsoft Azure Machine Learning Studio) · 다양한 딥러닝 프레임워크 지원: TensorFlow, Keras, Caffe, Caffe2, Chainer, Deep Water, MXNet, NVIDIA DIGITS, Theano, Torch · 다양한 기계학습 응용 개발 환경 제공 : R, Python, Julia, C#, Java, node.js, F#
패브릭 인터커넥션 기술	- (CXL) · 최근 Intel이 발표한 인터커넥터로 CPU와 메모리, 주변장치를 연결 · 표준 PCIe를 기반으로 하면서도 128GB/s 대역폭을 지원 - (Gen-Z) · 여러 노드의 다양한 컴포넌트를 고속으로 연결하는 확장성 있는 인터커넥트 · 400GB/s 이상의 넓은 대역폭을 목표로 설계되었으며 바이트 수준 접근, 메시징, 그리고 블록 수준의 접근을 지원 · 캐쉬 일관성 보장에 필요한 프로토콜을 정의하고 있음 - (CCIX) · CPU와 가속기를 연결하는 PCIe 기반 인터커넥터 · 최대 대역폭이 50GB/s로 제한 - (OpenCAPI) · IBM이 개발한 인터커넥트 기술로 응용프로그램이 GPU, FPGA와 같은 가속기에 접근할 때 단일 주소공간과 캐시 일관성을 지원

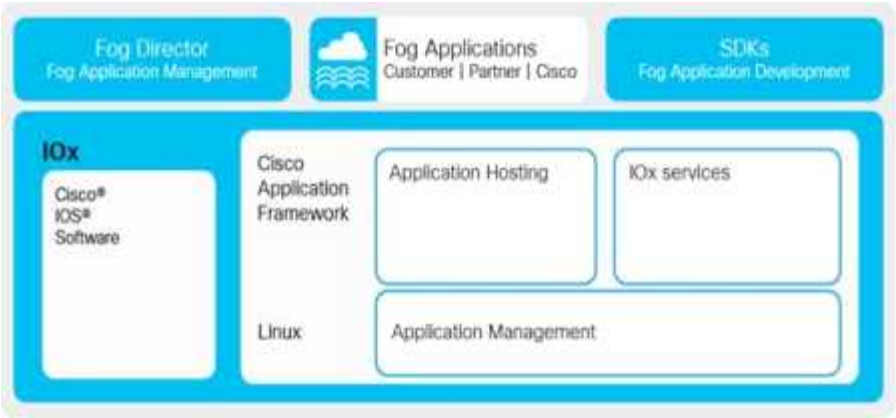
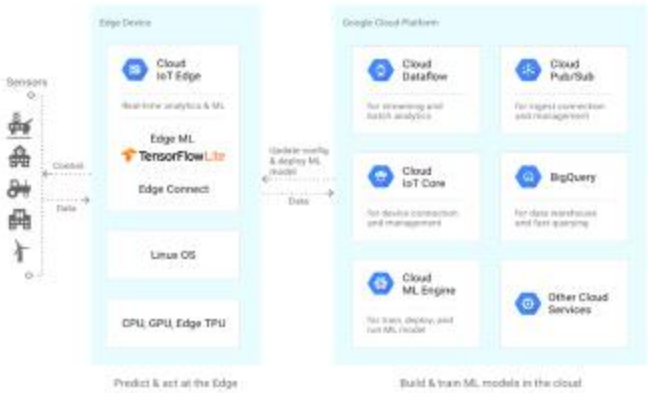
기술구분	경쟁기관 현황
	· IBM Bluelink를 기반으로 하고 Power9 프로세서만 지원하는 등 호환성에 제약이 있음. 최대 25GB/s 대역폭을 지원 - (InfiniBand) · 고성능 컴퓨팅과 기업용 데이터센터에서 널리 사용되는 스위치 방식의 통신 연결 방식 · 일반적으로 사용되는 4 링크 설정시 단방향으로 EDR, HDR, NDR이 각각 100Gbps, 200Gbps, 400Gbps를 지원하고 있음 - NVIDIA(NVLink): CPU와 GPU, 또는 GPU 간의 초고속 통신 지원 - Numascale(NumaConnect): 클러스터 기반 싱글시스템 이미지 제공 - SGI(NUMALink): 단일 시스템 기준 2,048 코어와 64TB 메모리까지 확장 가능
비휘발성 메모리	- Intel(3D XPoint): DDR4와 전기/물리적 호환성 제공, DRAM 대비 4배 용량 지원 - 삼성, Agiga, Diablo, Viking Technology, SK하이닉스: NVDIMM-F/N 발표, NVDIMM-P 표준화 참여 - Diablo Technologies(Memory1) : 플래시 기반으로 기존 DRAM과 병행하여 사용, DIMM 규격
메모리 컴퓨팅 운영체제 및 시스템	- HPE(L4TM) : The Machine을 위한 기존 리눅스 운영체제의 확장 및 개선 - Diablo Technologies(DMX) : Memory1 지원SW, 응용프로그램의 지능적인 메모리 접근 및 관리 기능 제공 - Plexistor(SDM) : 다수 노드의 메모리를 통합하여 응용에게 투명하게 제공하는 SW 기술로서 파일시스템 인터페이스를 제공함 - ScaleMP: 비휘발성 메모리를 활용한 메모리 확장 기술 및 다수 노드의 연산장치 및 메모리를 통합하는 가상화 기술 제공 - TidalScale: 다수 노드의 연산장치 및 메모리를 통합하는 가상화 기술 제공 - MemVerge: 인텔 Optane DCPMM 기반 메모리 확장 가상화 시스템 기술 제공
AI 가속기	- NVidia: NVIDIA는 2080억개 트랜지스터, 4bit 부동소수점 AI추론 기능, LLM 추론시간 5배 개선한 B200 GPU를 발표(3/18)하고, 2개의 B200 GPU와 그레이스 CPU를 통합한 GB200 슈퍼칩 발표하였음, GB200 슈퍼칩은 1.4EF AI성능, H100기반 GH200대비 LLM추론에서 30배 성능향상이 가능함 - NVIDIA는 1Rack에 36개의 GB200 슈퍼칩이 탑재(36개의 엔비디아 그레이스 CPU와 72개의 엔비디아 B200 GPU)되고 5세대 NV 링크로 연결되는 단일 슈퍼컴퓨터(FP8 720PF 지원) 또한 발표함 - AMD: AMD는 2023년에 생성형 AI 워크로드 및 HPC 어플리케이션에서 성능을 향상시킨 MI300 가속기를 출시함. MI300은 AMD CDNA3, AMD Infinity 아키텍처, AMD ROCm 기술을 지원하고, 데이터 타입 연산은 INT8, FP8, FP16, BF16, TF32, FP32, FP64를 수행함. INT8 타입에서 최대 5.22POPS, FP8 타입에서 최대 5.22PFLOPS, FP16 및 BF16 타입에서 최대 2.61PFLOPS, TF32 타입에서 최대 1.3PFLOPS, FP32 및 FP64에서 최대 163.4TFLOPS 성능을 보임. MI300의 TBP(Typical Board Power)

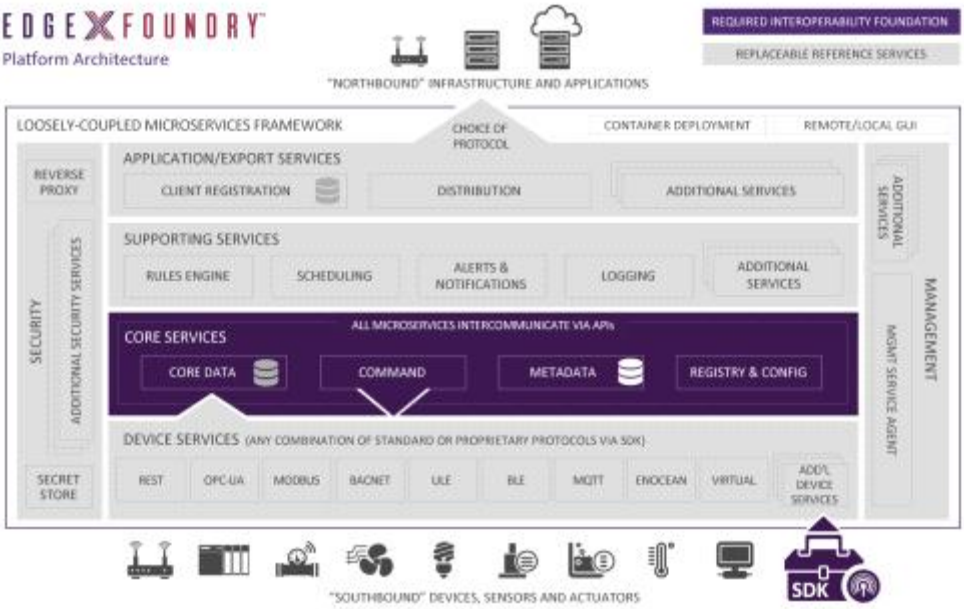
기술구분	경쟁기관 현황
	는 최대 750W 임. - Google: Google은 2016년 TPU v1부터 2023년 TPUv5e까지 공개함. TPU(Tensor Processing Unit)는 ASIC(Application Specific Integrated Circuits)으로 설계되었으며, 기계학습 워크로드를 가속화하는데 사용됨. 2023년에 발표한 TPU v5e는 1개의 텐서 코어로 구성되며, 각 텐서 코어는 4개의 MXU, 1개의 벡터 유닛, 1개의 스칼라 유닛으로 구성됨. TPU v5e는 INT8, BF16 데이터 타입 연산을 수행하며, INT8 타입에서 최대 393TOPS, BF16 타입에서 최대 197TFLOPS 성능을 보임. TPU v5e의 전력 관련 지표는 공개되지 않음. - FuriosaAI는 2024년에는 48GB HBM3를 탑재하여 GPT 수준의 초거대 인공신경망 모델에 대한 추론이 가능한 RNGD NPU를 발표함. RNGD는 TDP 150W 수준으로 최대 512 TF(FP8)을 지원하는 고성능 NPU임 - Rebellions은 2023년에 엣지 컴퓨팅 시스템과 클라우드 컴퓨팅 시스템 환경에서 CNN(Convolutional Neural Network), LSTM(Long Short-Term Memory), BERT(Bidirectional Encoder Representations from Transformers) 및 최신 트랜스포머 네트워크 (T5, GPT 등) 등 다양한 컴퓨터 비전, 언어 처리를 가속화하는 ATOM을 출시함. ATOM은 INT8, FP16 데이터 타입 연산을 수행하며, INT8 타입에서 128TOPS, FP16 타입에서 32TFLOPS 성능을 보임. ATOM의 TDP는 최대 150W임.
데이터 처리장치 기술	- NVIDIA BlueField DPU · 고급 네트워킹, 보안, 스토리지 기능을 제공하여 데이터센터 서버에서 CPU가 수행하는 여러 작업을 오프로드 함 · BlueField-2는 프로그래밍 가능한 멀티코어 ARM CPU, 200G/100G SmartNIC 네트워킹, 고성능 PCIe 인터페이스 및 강력한 네트워킹, 스토리지 및 보안 가속기를 결합 - Fungible DPU · 데이터 센터의 스케일 아웃 아키텍처에 최적화되어 있으며, 스토리지, 네트워크, 보안 작업을 가속화 함 · F1 DPU는 800 Gbps, S1 DPU는 200 Gbps의 네트워크 속도를 제공함 - Marvell OCTEON DPU · 네트워크 및 보안 기능에 중점을 둠 · OCTEON 10 DPU는 5nm 공정을 사용하고 Arm Neoverse N2 코어를 탑재하여 성능 향상과 전력 절감을 이루었으며, AI/ML 하드웨어 가속 기능을 내장함 - Xilinx SmartNIC 및 DPU · 데이터 센터에서 네트워크 및 컴퓨팅 작업 가속화 지원 · 프로그래밍 가능한 하드웨어에 강점을 가짐
데이터 처리 가속 SW	- Intel · (DPDK) Data Plane Development Kit. 고성능 데이터 패킷 처리를 위한 소프트웨어 라이브러리와 드라이버를 제공하는 오픈소스

기술구분	경쟁기관 현황
	· (SPDK) Storage Performance Development Kit. 고성능 스토리지 애플리케이션을 개발하기 위해 드라이버, 라이브러리, 애플리케이션을 모아 놓은 오픈소스 개발 키트 · (IPDK) CPU, IPU, DPU 또는 스위치에서 실행되는 인프라 오프로드 및 관리를 위한 드라이버 및 API로 구성된 오픈 소스, 벤더 독립적 프레임워크 - NVIDIA DOCA · NVIDIA BlueField DPU를 위한 SW 프레임워크로 네트워크, 스토리지, 보안 서비스를 오프로드, 가속화, 격리하는 기능 제공 · API와 드라이버, 라이브러리, 샘플 코드, 다큐멘테이션, 서비스, 사전 패키지형 컨테이너를 한데 모아 간소화하고 애플리케이션의 개발과 배포를 가속화 함
시스템 인터커넥션 기술	- Intel (CXL) · Intel를 주축으로 한 CXL 컨소시엄에서 제안된 인터커넥트로, CPU와 메모리, 가속기를 비롯한 주변장치를 캐시 일관성을 유지하며 연결하기 위한 인터커넥트 · PCIe의 물리 계층을 기반으로 하며, CPU와 메모리/주변장치 간 유연한 메모리 접근 메커니즘을 제공하여 컴퓨팅 노드의 메모리 용량 및 대역폭을 효율적으로 확장할 수 있도록 함. CXL 3.0 버전을 기준으로 최대 256GB/s의 넓은 대역폭을 제공하며 PCIe 대비 낮은 통신 지연 시간을 보장함 - ARM (CCIX) · CPU와 가속기를 고속으로 연결하기 위해 ARM이 주도하는 CCIX 컨소시엄에서 제안된 PCIe 표준 기반 인터커넥트로, 최대 128GB/s의 대역폭을 제공함 · 2020년 2.0 버전 업데이트를 이후로 개발이 중단됨 - HPE (Gen-Z) · 여러 노드의 다양한 컴포넌트를 고속으로 연결하는 확장성을 중시한 인터커넥트 · 토폴러지 구성에 따라 32GB/s에서 400GB/s에 이르는 대역폭을 제공함 · 바이트 수준 접근, 메시징, 그리고 블록 수준의 접근을 지원하며, p2p 구조에 한해 캐쉬 일관성을 보장함 · 2021년 Gen-Z 표준은 CXL 표준으로 통합됨 - Intel (PCIe) · CPU와 주변장치를 고속으로 연결하기 위해 2004년 Intel을 필두로 한 PCI-SIG에서 PCI의 후속으로 공개된 인터커넥트 · 현재 PCIe 6.0 표준까지 공개된 상태이며, PCIe 7.0 표준화 작업을 진행 중임 · PCIe 6.0을 기준으로 최대 128GB/s의 대역폭을 제공하며, 메모리 시멘틱 접근 및 캐시 일관성 유지 관련 기능은 지원하지 않음 - (InfiniBand) · 고성능 컴퓨팅 (High-Performance Computing, HPC) 및 데이터 센터 환경에서 사용되는 컴퓨터 네트워킹 및 입력/출력 기술의 표준으로 일반적으로 사용되는 4 링크 설정시 단방향으로 EDR, HDR, NDR, XDR

기술구분	경쟁기관 현황
	이 각각 100Gbps, 200Gbps, 400Gbps, 800Gbps를 지원함. 최근 InfiniBand® Trade Association (IBTA)에 의해 800Gpbs 규격의 표준화가 완료됨. · NVIDIA는 최근에 400Gbps 속도를 지원하는 InfiniBand 네트워크 기술인 NVIDIA Quantum-2를 출시하고 800Gbps급 기술을 개발하고 있음 - NVIDIA(NVLink): NVIDIA가 개발한 고성능 데이터 전송 규격으로 CPU와 GPU, 또는 GPU 간의 초고속 통신 지원함 · NVLink 1.0은 초당 최대 20Gbps(/lane)의 전송속도를 제공하며 최대 4lane을 지원 양방향 160Gbps를 지원함 · NVLink 2.0은 초당 최대 25Gbps(/lane)의 전송속도를 제공하며 최대 6 lane을 지원 양방향 300Gbps를 지원함 · NVLink 3.0은 초당 최대 50Gbps(/lane)의 전송속도를 제공하며 최대 12 lane을 지원 양방향 1200Gbps를 지원함
저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 기술	- SKT에서는 2018년 독일 도이치 텔레콤의 자회사인 모바일엣지X와 파트너십을 맺고 5G 시스템에 모바일 엣지 컴퓨팅(MEC) 기술을 도입하여 MEC 표준이 없는 상황에서 양사의 플랫폼을 통합하는 것이며, SKT의 MEC 기술은 사용자의 스마트폰에서 데이터센터까지 데이터가 최소 4단계 과정으로 전송되는 과정을 2단계로 줄여 5G 통신지연시간을 획기적으로 감소시킴 - 삼성전자에서는 신경망 처리 장치(NPU)를 탑재한 엑시노스 시리즈를 개발하고 있으며, 2019년 10월 자체 NPU 코어 2개와 DSP를 탑재한 엑시노스990을 공개하였으며, 통합 IoT 솔루션인 스마트 싱스의 SmartThings Hub가 엣지 컴퓨팅 장치로서의 역할을 수행하며, 스마트 폰 앱으로 스마트홈 구축 및 원격으로 스마트 디바이스와 가전들을 관리할 수 있는 솔루션을 개발하였음 - 삼성SDS에서는 한국 HPE와의 협력을 통해 반도체, 디스플레이 계열사 제조공장에 엣지 컴퓨팅 기반의 불량 검출 시스템을 구축하였음 - 통신장비 전문업체인 다산네트웍스솔루션에서는 MEC를 위한 HW 플랫폼과 초저지연 TSN스위치, 소프트웨어 정의 네트워크(SDN) 등 5G 서비스 및 엣지 컴퓨팅을 위한 핵심 모듈을 집중 개발 하고 있음 - 빅데이터 분석 전문업체인 위즈온텍에서는 컨테이너화된 애플리케이션 관리시스템 기술을 보유하고 있으며 GPU, 네트워크 가속화, 컴퓨팅 및 스토리지 프로비젼 관리를 할 수 있는 SKT의 스마트팩토리 서버 디바이스 관리 및 응용 SW 배포 기술을 개발 중 - 실시간 운영체제 전문 기업인 알티스트에서는 실시간 운영체제 최적화 기술을 보유하고 있으며 MEC 서버의 시스템 자원, 네트워크 상황 및 장애 상황 검출을 지원하는 실시간 모니터링 및 복구 기술을 개발하였음 - KAIST에서는 MEC 지원을 위한 FPGA 하드웨어 로직 분할 기술 및 FPGA 기반의 응용 고속처리 기술 개발하였음 - UNIST에서는 초성능 컴퓨팅 환경 구축을 위해 차세대 반도체 기술 기반 컴퓨팅 가속 하드웨어 시스템과 시스템 소프트웨어(OS,

기술구분	경쟁기관 현황
	가상화, 스토리지, 분산·병렬처리 등) 원천기술에 대한 연구·개발을 진행하고 있음 - 인텔에서는 모비디우스(Movidius), 너바나(Nirvana), 알테라(Altera), 모빌아이(Mobileye)에 이어 이스라엘 AI 반도체 스타트업 '하바나랩스(Habana Labs)'를 20억 달러에 인수하는 등 AI 반도체 시장 확장을 가속화 하고 있음 - 하바나랩스에서는 엔비디아보다 3배 뛰어난 추론 성능을 가진 '고야(Goya)'와 데이터 센터 작용을 위한 '가우디(Gaudi)'를 개발하였음 - 페이스북에서는 AI 반도체 개발 조직을 2019년 4월에 설립하여 AI 프로그램을 지원하기 위한 자체 SoC, ASIC 개발에 착수하였음  그림 6. Azure IoT Edge 개념 - 마이크로소프트에서는 자사의 클라우드 플랫폼인 Azure을 이용하여 클라우드 서비스를 엣지 기기로 전송하는 소프트웨어 플랫폼인 'Azure IoT Edge'서비스를 제공하고 있으며 스마트팩토리, 커넥티드카, 비행기 엔진관리 등 여러 산업에서 자사의 머신러닝, 분석 솔루션 서비스를 제공 중임 - 휴렛 팩커드 엔터프라이즈(HPE)에서는 슈나이더 일렉트릭과 공동으로 HPE 마이크로 데이터센터 솔루션을 출시하여 별도로 데이터센터를 구축하기 힘든 상황에서 엣지컴퓨팅 환경을 구축하고 군부대, 연구소, 공장, 사무실 등에 솔루션을 제공하고 있음 - 제네럴 일렉트릭스(GE)에서는 자사의 IoT 플랫폼인 프레딕스(Predix)에 엣지 컴퓨팅을 적용한 '프레딕스 엣지'는 모든 데이터를 클라우드에 전송하는 대신 엣지 자체 또는 시스템에서 직접 분석하고, 모든 것을 기업들이 활용할 수 있도록 공개함과 동시에 프레딕스용 애플리케이션 개발을 할 수 있도록 교육을 제공하고 있음 - Amazon에서는 자사의 AWS 플랫폼을 이용하여 IoT 그린그래스라는 엣지컴퓨팅 솔루션을 제공하고, AWS 클라우드 서비스의 일부 기능을 로컬 디바이스로 옮겨 코어 기기에서 클라우드와의 연결이 끊어지더라도 그린그래스 그룹에 속한 기기를 통한 실시간 운용환경 기반 AI서비스를 제공하고 있음 - Cisco에서는 포그 컴퓨팅을 위한 IOx아키텍처를 개발하여 시스코 네트워크 장비용 운영체제인 iOS에 Linux를 통합한 IoT 플랫폼을 제공함으로써, 많은 양의 데이터를 먼 곳에 있는 대용량 데이터 서버에 저장하지 않고, 데이터 발생 지점 근처에서의 연산 처리를 지원하고 있음

기술구분	경쟁기관 현황
	 그림 7. Cisco IOx 아키텍처 - 구글에서는 자사에서 개발한 AI 프레임워크인 텐서플로를 엣지에서 수행할 수 있도록 하는 Cloud IoT Edge를 통해 머신러닝을 엣지 시스템에서 수행할 수 있도록 서비스를 제공하고 있음 How Cloud IoT Edge works  그림 8. Google Cloud IoT Edge 개념 - 리눅스 재단에서 추진중인 EdgeX Foundry 프로젝트는 엣지 컴퓨팅을 위한 솔루션들의 상호운용성을 위한 조직인 LF Edge의 마이크로서비스아키텍처(MSA) 기반 산업용 IoT 엣지 컴퓨팅 플랫폼을 개발하여, 이기종 IoT 디바이스들과의 데이터 수집 및 처리를 위한 코어 서비스 제공하고 있음

기술구분	경쟁기관 현황
	 그림 9. EdgeX Foundry "EdgeX" 아키텍처 - Nvidia에서는 전 세계 GPU 시장의 70~80%를 차지하고 있으며, 고도화된 컴퓨팅 능력을 데이터센터가 아닌 반도체에 집약함으로써, 분산형 디바이스 엣지 컴퓨팅 시장을 주도하고 있으며, 자율주행 반도체 패키지 NVIDIA DRIVE PX2(주) 등을 내세워 자율주행 자동차 사업 전략을 추진 중임 ※ 자동차 제조사와 주요 공급업체가 자동화 및 자율주행차량 생산을 가속화할 수 있는 오픈 AI 자동차 컴퓨팅 플랫폼 개발 - Dell에서는 산업현장의 다양한 기기에서 발생하는 데이터를 수집 및 전송할 수 있는 엣지 컴퓨팅 솔루션인 '엣지 게이트웨이' 서비스를 제공하고, 다양한 환경을 지원하는 HPE와 달리 특정 환경인 산업용, 운송 및 물류, 디지털 신호체계에서 사용할 수 있는 3가지 모델을 출시하였음 그림 10. NVIDIA의 자동차용 디바이스 엣지와 Dell EMC의 엣지용 서버
엣지 사이버물리시스템을 위한 서버자원융합형 대규모 디바이스 가상화	- 삼성 SDS에서는 머신러닝 기법과 통계적 기법을 이용한 자동 특징 선택(Auto Feature Selection), 회귀, 분류를 통한 데이터 변형을 적용한 자동 데이터 정리 (Auto Data Cleansing)를 제공하는 플랫폼 개발(Brightics AI) - 서울대학교에서는 ROS(Robot Operating System)기반의 CPS를 통해

기술구분	경쟁기관 현황
기술	자동차 시스템의 시뮬레이션 성능과 실제 성능 사이의 차이를 줄이기 위한 정확한 출력 생성 시뮬레이터 알고리즘을 개발하는 중임(ROS2 based CPS simulator) - 성균관대학교에서는 5G-IoT 환경에서 대용량·비정형·이기종 데이터를 실시간으로 안전하게 처리할 수 있는 시스템을 개발하는 과제를 진행 중 - 한국전자통신연구원에서 5G의 초고속, 초저지연 전송기술을 활용해 도시 곳곳을 정밀하게 관제할 수 있는 스마트시트 구현을 위한 기술개발을 진행 하고 있음 - 포항공과대학교에서 가상기지국의 에지 시스템 성능의 두 가지 문제점인 Practal Frequency Reuse의 낮은 시스템 스펙트럼 효율과 Contined MultiPoint의 클러스터간 간섭을 해결하기 위해 공동 가상 클러스터링과 주파수 할당 체계를 제안함 - KT에서는 ’19.12월 KT의 디지털트윈인 AI 기가트윈을 개발하고, 도시 인프라를 모니터링하고 데이터를 통해 예측하는 서비스 제공할 예정임 - SKT는 ‘19년 MWC19에서 디지털트윈을 활용하여 현실과 가상공간을 넘나드는 ’5G 하이퍼스페이스 플랫폼‘ 최초로 공개하여, 인테리어, 교육, 엔터테인먼트, 커머스 등 많은 영역에서 활용할 예정임 - SK C&C의 스마트팩토리 솔루션 SCALA는 디지털트윈을 통해 스마트공장의 효과를 산출해 내고, 도입 시 예상되는 위험까지 사전 검증하여 생산 공정의 최적화 확보 - 삼성 SDS는 ‘인텔리전트한 全 제조 현장 관리’를 모토로 ‘Samsung Nexplant’를 개발, 플랫폼 기반의 제조 지능화를 구현 중, 인공지능 기반 분석 플랫폼인 브라이틱스(Brightics)를 적용하여 ‘Nexplant’를 고도화 중에 있으며, 브라이틱스(Brightics)는 방대한 제조 데이터를 신속하게 분석·시각화하여 솔루션을 제시하는 처방형 알고리즘의 AI분석 플랫폼임 - LG CNS는 제조 全 과정에서 표준화된 개발과 운영이 가능한 스마트 팩토리 플랫폼 ‘Factova’를 출시하였으며, “INFioT” IoT 플랫폼을 통해 센서와 디바이스를 등록 관리하고, 데이터를 산업별로 미리 정의된 표준에 맞게 변환하여 센싱 데이터를 수집 관리하며 스마트 팩토리 플랫폼인 “Factova”에 적용 및 연동함. 현재 LG 전자의 북미 세탁기 공장 및 LG디스플레이 파주 OLED 공장, LG화학 폴란드 전지 공장 등에 도입 추진 중임 - 포스코건설은 디지털 트윈 기반의 스마트 팩토리를 구현하여 시공 통합 시스템 구축, 공정 최적화 및 안전강화를 위한 디지털 시뮬레이션을 구현함 - GE에서는 2016년 GE는 기계에서 발생하는 대규모의 데이터를 분석 수집하고 사물인터넷으로 연결해 디지털트윈을 구현해주는 통합 IIoT(Industrial Internet of Things) 플랫폼 프레딕스(Predix) 공개하고 미국 네바다주에 위치한 자사 공장의 프로세스와 공급사슬의 디지털트윈 모델을 생성해 경영진이 데이터에 기반을 둔 수천 건의 의사결정에 도움이 됨을 보였음 - 지멘스는 공장 자동화 IoT 플랫폼인 마인드스피어를 구축하여 공장

기술구분	경쟁기관 현황
	내 각 설비에 부착된 센서를 통해 데이터를 받아들이고 공장의 설비를 실시간으로 디지털트윈과 연결을 통한 생산성 향상을 제공함 - IBM에서는 Watson IoT 플랫폼의 강력한 데이터 모델링 기능을 통해 장치(Device) 트윈과 자산(Asset) 트윈을 사용하여 데이터를 디지털 스레드의 근간을 구성할 수 있는 모델로 통합하였음 - SAP의 Leonardo IoT는 디지털트윈 모델 기반 서비스 제공, 규칙 및 이벤트 관리, 액션 중심 비즈니스 통합, IoT 강화 분석, IoT 데이터 액세스 제어, IoT 응용프로그램 개발 등 지원하고 있음 - 다쏘시스템의 단일 플랫폼 기반의 '3D익스피리언스 트윈(3DEXPERIENCE Twin)'은 현실 세계에서 존재하거나 존재할 수 있는 제품, 시스템, 시설 또는 환경을 표현하며, 제품의 생애주기의 모든 단계에서 동적 3D 모델로 제품과 프로세스, 공장·설비 운영을 복제를 제공하고 있음 - PTC에서는 기업의 산업 혁신을 추진하는 데 필요한 기능, 유연성, 확장성을 제공하는 산업용 IoT 플랫폼 Thingworx를 개발하고 3D CAD 소프트웨어 크레오를 사용하여 디지털트윈 구축을 통해 물리적 세계와 디지털 세계를 연결을 제공함 - NVIDIA에서는 게임, 제조, 미디어 분야 등을 포함한 그래픽 작업이 필요한 전반적인 영역에서 디지털 트윈을 위한 3D 모델을 쉽게 모델링하고 다른 사용자들과 실시간 협업이 가능한 'Omniverse' 플랫폼을 출시해 디지털 트윈 플랫폼 시장 선점을 시도 중

6) 국내.외 지식재산권 현황

○ [메모리 중심] 관련 기술에 대한 국내외 특허 현황

- ETRI는 국내 특허 11건, 미국 특허 2건을 출원하여 관련된 지식재산권을 확보하고 있음
  - HPE, IBM, Intel, VMware, Redhat, 삼성전자, SK하이닉스 등 다수의 기관들이 메모리 관련 기술에 대한 IPR 확보 중
  - 패브릭 메모리 관련 특허는 HPE, VMware, ETRI 가 선점하고 있으며 지속적인 IPR 확보가 요구됨

○ [EdgeCPS] 본 과제에서 제시하는 Massive Things의 초근접 서비스를 위한 지능형 EdgeCPS 플랫폼은 5G/6G 시대를 맞이하여 최근 새롭게 등장하는 기술로 정확하게 일치하는 기술이 없기 때문에 기반 기술이 될 수 있는 관련 기술 위주로 분석을 진행

- 디바이스리스 컴퓨팅 기술은 저자원 다바이스 성능 증강을 위해 엣지 시스템 자원 관리 기술을 통해 실현될 수 있으므로 아래와 같은 검색식을 사용하여 검색
  - (국내) 총 44건, 2015년 이전에 출원이 없다가, 2015년부터 출원 수가 증가하는 양상을 보임
    . 삼성전자, KAIST, 한국전자통신연구원 등 다수의 기관들이 엣지 컴퓨팅 기술에 대한 IPR 확보 중

표 4. 특허 분석 분야 및 검색에 사용된 키워드 (국내)

지역	검색식
국내	((AB=[("에지 컴퓨팅"+"엣지 컴퓨팅"+"edge computing")*(시스템+system)])+(TL=[(("에지 컴퓨팅"+"엣지 컴퓨팅"+"edge computing"))*(시스템+system)])) ((AB=[("에지 컴퓨팅"+"엣지 컴퓨팅"+"edge computing")*(플랫폼+platform)])+(TL=[(("에지 컴퓨팅"+"엣지 컴퓨팅"+"edge computing"))*(플랫폼+platform)])) ((AB=[("에지 컴퓨팅"+"엣지 컴퓨팅"+"edge computing")*((리소스+자원)*(관리+할당))])+(TL=[(("에지 컴퓨팅"+"엣지 컴퓨팅"+"edge computing"))*((리소스+자원)*(관리+할당))]))

- (미국) 총 268건, 통신사(Verizon, AT&T)와 IBM, Intel, Honeywell 등을 중심으로 IPR 확보 중
  . 전세계 엣지 컴퓨팅 시장에 대한 영향력만큼 IPR 확보 수도 크게 증가하는 추세
  . Verizon, AT&T, IBM, Intel 등을 중심으로 IPR 확보 중
- (유럽) 총 81건, 전체적으로 꾸준히 증가하다가 2019년에 감소한 추세를 보임
  . GE, Boeing, Honeywell, Saguna Networks 등을 중심으로 IPR 확보 중
- (중국) 총 1,065건, 2015년부터 전체적으로 꾸준히 증가하는 추세를 보임
  . 화웨이와 같은 글로벌 기업과 다수의 대학(베이징대, 난징대, 상하이대 등)을 중심으로 IPR 확보 수가 증가하는 추세

표 5. 특허 분석 분야 및 검색에 사용된 키워드 (국외)

지역	검색식
해외 (미국/유럽/중국)	((AB=[("edge computing")*(system)])+(TL=[("edge computing")*(system)])) ((AB=[("edge computing")*(platform)])+(TL=[("edge computing")*(platform)])) ((AB=[("edge computing")*(resource*manage)])+(TL=[("edge computing")*(resource*manage)]))

- 엣지 사이버물리시스템을 위한 대규모 디바이스 가상화 기술은 단순 모니터링 및 제조 분야에 한정된 특허가 대부분으로 본 기술과 연계성이 있는 특허만 분석
  - (국내) 총 18건, 2018년부터 출원이 시작되었으며, 꾸준히 증가하는 양상을 보임
    · 한국국토정보공사, 한국수력원자력 등의 공공기관 및 KT 등의 기업이 특허 기술 확보

표 6. 특허 분석 분야 및 검색에 사용된 키워드 (국내)

지역	검색식
국내	(AB=[("디지털 트윈"+"디지털트윈"+"digital twin")*("디바이스"+"device")])+(TL=[("디지털 트윈"+"디지털트윈"+"digital twin")*("디바이스"+"device")])

- (미국) 총 66건, 2016년부터 출원이 시작되었으며, 꾸준히 증가하는 양상을 보임
  · GE, Honeywell, Microsoft, Siemens 등의 기업들이 주도적으로 특허 기술 확보
- (유럽) 총 13건, 2016년부터 출원이 시작되었으며, 꾸준히 증가하는 양상을 보임
  · GE, Microsoft, Siemens 등의 기업들이 주도적으로 특허 기술 확보
- (중국) 총 206건, 2017년부터 출원이 시작되었으며, 꾸준히 증가하는 양상을 보임
  · 베이항대, 베이징대, 광동대, 광저우대, 난징대, 상하이대 등의 대학교들이 주도적으로 특허 기술 확보

표 7. 특허 분석 분야 및 검색에 사용된 키워드 (국외)

지역	검색식
해외 (미국/유럽/중국)	(AB=[("digital twin")*("device")])+(TL=[("digital twin")*("device")])

○ [이종가속기AI컴퓨팅] AI 병렬컴퓨팅 관련 기술에 대한 국내외 특허 현황

- (국내) AI 병렬컴퓨팅 관련 기술의 국내 특허 출원 건수가 2021년에 12건, 2022년에 24건, 2023년에 32건으로 최근 3년간 증가하는 추세임

표 8. AI 병렬컴퓨팅 관련 국내 주요 특허

문헌번호	발명의 명칭	주요 내용	출원인
KR 10-2020-0070088	GEMM 데이터플로우 가속기 반도체 회로	스마트 3D 적층 DRAM 구조를 포함하는 GEMM (general matrix-matrix multiplication) 데이터플로우 가속기 회로를 기술함	삼성전자
KR 10-2022-0170428	이기종 프로세서 기반 엣지 시스템에서 SLO 달성을 위한 인공지능 추론 스케쥴러	이기종 프로세서 기반 엣지 시스템에서 SLO (Service-Level Objective) 달성을 위한 인공지능 추론 스케쥴러 기술을 기술함	한국과학기술원
KR 10-2548582	뉴럴 프로세서 및 이의 명령어 페치 방법	뉴럴 프로세서 및 이의 명령어 페치 방법을 기술함	리벨리온

- (국외) AI 병렬컴퓨팅 관련 기술의 미국 특허 출원 건수는 2021년에 1,146건, 2022년에 1,911건, 2023년에 2,971건, 유럽 특허 출원 건수는 2021년에 109건, 2022년에

188건, 2023년에 325건으로 최근 3년간 증가하는 추세임

표 9. AI 병렬컴퓨팅 관련 국외 주요 특허

문헌번호	발명의 명칭	주요 내용	출원인
US 20230297406	CONFIDENTIAL COMPUTING USING MULTI-INSTANCING OF PARALLEL PROCESSORS	TEE (Trusted Execution Environment) 내 하나 이상의 PPU (Parallel Processing Unit)를 사용하여 가속화된 컨피덴셜 컴퓨팅을 제공함	NVIDIA
US 20230308270	PERFORMING SCRAMBLING AND/OR DESCRAMBLING ON PARALLEL COMPUTING ARCHITECTURES	무선 통신 및 프로세싱에서 병렬 컴퓨팅 아키텍처를 사용하여 통신 데이터의 스크램블링 및 디스크램블링을 수행하는 방법을 기술함	NVIDIA
US 20230004430	ESTIMATION OF POWER PROFILES FOR NEURAL NETWORK MODELS RUNNING ON AI ACCELERATORS	토폴로지를 고려하여 다른 하드웨어의 신경 네트워크에 대한 전력 프로파일을 추정하기 위한 기술을 제공함	Intel

나. 핵심요소 및 접근방법

○ Peta-Scale 메모리 중심 컴퓨팅 구조 기술

- 기존 프로세서 중심 컴퓨팅 환경에서는 CPU가 독립적으로 메모리를 가지고 연산을 수행하며 네트워크를 통해 데이터 교환 및 통신이 이루어짐
- 데이터는 CPU에 종속되어 이동이 빈번하게 발생하며, 이로 인해 성능 저하 및 에너지 낭비가 야기됨. 이는 빅데이터 분석 및 처리가 일반화된 근래에는 더욱 심각한 문제로 자리 잡음

→ (핵심 요소기술) 프로세서 중심에서 메모리 중심으로의 컴퓨팅 구조 혁신

- 메모리 중심 컴퓨팅은 대용량·공유 메모리 풀이 고속 인터커넥트를 통해 다수의 연산장치(CPU, GPU 등)들과 연결된 새로운 컴퓨팅 구조
- 대용량·공유 메모리 풀을 통해 거대 데이터에 대한 빠른 접근이 가능하며, 다수의 연산장치가 거대 데이터를 빠른 속도로 공유할 수 있음

○ 메모리 중심 컴퓨팅 기반 지능정보 데이터 고속 처리 기술

- 기존 지능정보 데이터 처리 기술은 기존의 컴퓨팅 구조에 최적화되어 있음. 기존 시스템 수준의 메모리 용량만을 활용하여 데이터 처리 속도를 최적화하며, 빠른 휘발성 메모리와 느린 저장장치만으로 데이터 처리 속도를 높이면서 데이터 영속성을 보장하기 위한 여러 기법들을 개발함
- 메모리 중심 컴퓨팅 구조는 대용량·공유 메모리 풀을 제공할 뿐만 아니라 비휘발성 메모리 기반으로 구축되므로, 이같은 하드웨어를 적극 활용하면 지능정보 데이터 처리가 비약적으로 빨라질 수 있음

→ (핵심 요소기술) 메모리 중심 컴퓨팅 기반으로 지능정보 데이터 처리 속도 혁신

- 대용량 메모리를 활용하는 메모이제이션(memoization) 기법 활용 및 응용 알고리즘 재설계
- 비휘발성 메모리를 활용하여 데이터 영속성을 보장하면서도 빠른 인메모리 데이터 구조 및 저장소 기술
- 공유 메모리와 연결된 연산장치들을 효율적으로 활용하는 스케일러블 소프트웨어 플랫폼 기술

○ 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 기술

- 기존 연구에서는 가상화된 공간에서 단순하게 리소스의 가상화를 통해 IoT와 엣지 디바이스를 통합 관리하는 방식을 제안함
- 디바이스 별로 창작되어 있는 자원(CPU, 메모리)을 가상화하여 관리하는 인프라를 제공할 수 있지만, 각 디바이스 단위의 가상화로 인해 학습/추론 등 AI 연산을 위해 필수적인 이종 가속기 자원에 대한 공유는 불가능

→ (핵심 요소기술) 이종 가속기의 공유 모델을 통해 디바이스 가상화 환경에서 자원의 공유 및 활용이 가능해져 디바이스 간 연산 능력 차이 극복이 가능

- 디바이스-엣지-클라우드 등에 산재한 이종 가속기(GPU, FPGA, NPU, TPU)를 가상화하고, 디바이스 가상화 환경에서 저성능 디바이스에 서비스 형태로 제공 가능한 구조를 제시
- 저자원 디바이스가 가속기와 같은 고가의 디바이스 없이(Deviceless) 학습/추론 등 AI 연산을 사용할 수 있는 구조임
- 엣지 사이버물리시스템에서 발생하는 이벤트와 실시간 자원 상황에 기반해 가용 가속기 자원에 대한 수요-공급 논리와 우선순위에 따른 최적 스케줄링 수행

○ 엣지 사이버물리시스템을 위한 서버자원융합형 대규모 디바이스 가상화 기술

- 엣지 사이버물리시스템은 센서와 액추에이터를 가진 초대규모의 디바이스들을 5G/6G 기반의 초연결성을 기반으로 공통의 목적에 따라 자율/군집 제어되는 시스템
- 기존에 사이버물리시스템의 모니터링과 제어를 위해 디지털트윈 개념이 활용되고 있으나, 디바이스의 수 폭증에 따라 연산의 복잡도도 증가하여 이를 지원하기 위한 초성능 컴퓨팅 기술이 요구됨
- 기존 디지털트윈 기술은 디바이스의 정보를 모니터링하고 시뮬레이션하는데 머물러 있어 가상 자원 공유, 인공지능 서비스 지원, 대규모 디바이스 간의 지식 공유 등 초대규모 군집디바이스의 인공지능 서비스를 지원하기 어려운 상황임

→ (핵심 요소기술) 초대규모의 실제 단말과 디바이스트윈 간의 실시간 동기화에 기반한 대규모 객체 관리와 초성능 워크로드 처리

- 단일 서버나 클라우드에서의 컴퓨팅 기술이 아닌 고성능 디바이스, 엣지 게이트웨이, 엣지 서버, 클라우드 등 이종 플랫폼이 형성하는 계층적 구조 상에서 100만개의 가상화된 자원을 안정적으로 관리할 수 있는 구조
- 다양한 디바이스의 속성에 대한 효과적인 모델 명세 및 디바이스 간 관계 모델링 기술을 통해 초대규모 디바이스트윈을 구현할 수 있는 이론적 근간 마련

- 가상화된 엣지 서버의 자원을 상황에 따라 최적화된 방식으로 디바이스트윈에 분할하여 할당할 수 있는 자원 동적 관리 오케스트레이션 기술

○ 이종가속기 기반 AI 컴퓨팅 기술

- 기존 고성능 컴퓨팅 분야에서는 스케일-아웃 구조에 기반하여 저비용으로 고성능 시스템을 구축해 왔으나, GPU를 포함한 다양한 가속기들의 등장으로 연산 자원이 노드 및 장치 단위로 분절되는 파편화가 심화됨
- AI 모델의 대형화·복잡화로 인해 단일 가속기 기반 컴퓨팅의 한계가 명확해지고 있으며, AI 반도체 기술의 급격한 발전으로 인해 동종 가속기 간 세대 교체 주기가 짧아지면서 서로 다른 세대와 특성을 지닌 이종 가속기가 혼재된 환경이 일반화되고 있음
- 이에 따라 다중 노드 환경에서 다양한 이종 가속기를 효율적으로 통합하여 활용할 수 있는 AI 병렬컴퓨팅 시스템 기술이 요구됨

→ (핵심 요소기술) 다중 노드에 분산된 이종 자원(CPU, GPU, NPU 등)들을 하나의 고성능·고효율 AI 병렬 시스템으로 통합

- 초거대화되는 LLM의 학습/추론에 요구되는 연산량이 기하급수적으로 증가함에 따라 이를 안정적이고 효율적으로 지원할 수 있는 초고성능 컴퓨팅 시스템 구조 기술
- 이종 가속기 자원 중에 해당 워크로드 특성에 적합한 가속기를 활용한 고효율 AI 병렬 컴퓨팅 시스템 기술

다. 혁신성과 독창성

○ 메모리 중심 컴퓨팅 구조 기술의 혁신성과 독창성

- 혁신성
  - 다수의 프로세서/노드가 메모리 풀을 공유하는 형태로, 차세대 패브릭 인터커넥트를 통한 공유 메모리로의 접근이 가능한 메모리 중심 컴퓨팅 구조 혁신
  - 고집적화가 용이하고 전원이 차단되어도 데이터가 보존되고 Byte-Addressing이 가능한 차세대 스토리지 클래스 메모리(Storage Class Memory, SCM)를 이용한 대용량 메모리 구조로 혁신
  - 메모리 중심 컴퓨팅 구조의 대용량·공유 메모리를 활용하는 알고리즘 개발로 지능정보 데이터 처리 속도 혁신
- 독창성
  - 경쟁기관의 패브릭 인터커넥션 기술과 비휘발성 메모리 기술은 요소 기술로써, 본 과제는 이들을 통합한 새로운 컴퓨팅 시스템 구조를 연구
  - 경쟁기관의 빅데이터 관리 기술, 지능형 분석 기술은 기존 컴퓨팅 구조에 최적화된 지능정보 데이터 처리 기술들이며, 본 과제는 컴퓨팅 구조 수준에서부터 혁신을 도모
  - HPE는 The Machine 프로젝트를 통해 메모리 중심 컴퓨팅 구조의 예시를 보였으나, 특수 하드웨어를 사용했다는 점과 비휘발성 메모리에 대해서는 에뮬레이션에 그쳤다는 점에서 한계를 보임. 본 과제는 기존의 일반적인 하드웨어 및 새로 출시된 비휘발성 메모리를 사용해 메모리 중심 컴퓨팅 구조를 실현하며, 향후 연구개발되는

하드웨어 기술들도 접목 가능

○ 메모리 중심 시스템SW 기술의 혁신성과 독창성

- 혁신성
  - 기존 하드웨어의 활용하는 시스템SW를 통해 메모리 중심 구조를 실현시키는 SW만의 성능 혁신
  - 새로운 메모리 및 인터커넥트 등 하드웨어 도입을 통해 소프트웨어+하드웨어 하이브리드 기술을 통한 성능 혁신
  - 최종성능 SLO를 고려하여 이종·혼성 메모리로 구성된 공유 메모리를 관리하는 시스템SW를 통해 메모리 중심 구조의 유용성 혁신
- 독창성
  - HPE에서 The Machine 프로젝트를 위해 개발한 L4TM은 The Machine 프로토타입 시스템만을 위한 것이며, 본 과제는 현재의 상용 하드웨어를 활용할 수 있는 시스템SW를 개발
  - Diablo Technologies, Plexistor 등은 자사의 비휘발성메모리 제품 상에서 응용프로그램 성능을 도모하는 시스템SW를 제공하지만, 본 과제는 일반적인 상용 비휘발성 메모리를 포함하여 원격 휘발성 메모리까지 지원하는 메모리 확장 시스템SW를 개발
  - TidalScale, ScaleMP 등은 동종 물리머신만을 활용하는 시스템 확장 기술을 개발했으나, 본 과제는 메모리 전용 노드도 활용 가능한 메모리 확장 기술을 개발하고, 여러 가상머신이 확장된 메모리를 효율적으로 공유할 수 있는 탄력적 메모리 공유 기술도 개발

○ 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 기술 연구의 혁신성과 독창성

- 혁신성
  - 저자원 디바이스가 디바이스트윈을 통한 자원 대여를 통해 기존 하드웨어의 한계를 극복하는 SW 중심의 성능 혁신
  - AI 연산에 필수적인 가속기 공유모델에 집중하고, 디바이스 간 학습/추론 결과를 공유 및 활용할 수 있는 미들웨어 기술과의 연계를 제공해 기술적 유용성 혁신 추구
- 독창성
  - IBM에서 가상화된 공간에서 리소스 풀 구성과 자원 공유의 탄력성을 제공하여 IoT와 엣지 디바이스를 통합 관리하는 디바이스리스 엣지 컴퓨팅을 제안했지만 이는 단순히 디바이스의 다양한 자원(CPU, 메모리)을 가상화하여 관리하는 인프라를 제공
  - 디바이스 간 지식 공유가 가능한 구조와 AI 연산을 위해 필수적인 이종 가속기 자원에 대한 공유와 이를 통해 가속기 디바이스가 없는 단말에서도 효과적인 추론이 가능한 방법을 제공

○ 엣지 사이버물리시스템을 위한 서버자원융합형 대규모 디바이스 가상화 기술의 혁신성과 독창성

- 혁신성
  - 대규모 엣지 디바이스가 보유한 다수의 자원을 가상화하고, 이를 공유 가능한 리소스베이스로 제공하는 형태로, 저자원 디바이스와 고성능 디바이스의 자원 매쉬업이 가능한 혁신적인 디바이스 가상화 구조 제시
  - 사이버물리시스템의 워크로드를 디바이스-엣지-클라우드 연계형 구조를 통해 협업/분업 처리하는 방법을 통해 자원의 효율적 활용이 가능
  - 초대규모의 디바이스 간 성능 평준화가 용이하고, 디바이스 간 지식 공유가 가능한 지식 공유 미들웨어를 통해 사이버물리시스템 단위의 지식 증분이 가능한 구조로 혁신

- 독창성
  - 경쟁기관인 GE의 대규모 디지털트윈 실행 기술은 엣지 컴퓨팅 기술로써, 본 과제는 성능 한계 극복을 위해 디바이스-엣지-클라우드 연계형의 새로운 컴퓨팅 시스템 구조를 연구
  - 경쟁기관의 디지털트윈, 빅데이터 분석, 이벤트 처리 기술은 Cloud to the Edge 구조에 최적화된 지능정보 데이터 처리 기술들이며, 본 과제는 Device to the Edge 수준으로 새로운 혁신을 도모

○ 이종가속기 기반 AI 컴퓨팅 기술의 혁신성과 독창성

- 혁신성
  - 다양한 이종 가속기를 통합하여 대규모 AI 모델의 분산 병렬 추론을 비용 효율적으로 제공하는 기술 개발을 통해 특정 가속기 벤터 락인을 해소하고 국내외 NPU 기업들이 AI 컴퓨팅 인프라 시장이 진입할 수 있는 기술 제시
- 독창성
  - 경쟁기관들은 동종 가속기 자원의 최적 활용에만 초점을 두고 있지만, 본 과제에서는 이종 가속기를 통합/연계하여 활용하는 데 중점을 둠
  - 점진적으로 구축된 다양한 이종 성능 가속기들을 통합하여 신규 투자 없이도 보유 가속기의 활용을 극대화하여 비용 효율적으로 대규모 모델을 분산 학습하는 기술 제시

2 과제목표 및 수행과정

1. 과제목표

구분			주요 내용
연구개발 목표 및 내용	최종 목표		ㅇ Trillion급 대규모 생성형 인공지능 모델의 연산 한계를 극복하는 이종 가속기 기반 대규모 AI 병렬컴퓨팅 기술 개발
	1단계	목표	ㅇ 인공지능 서비스와 데이터 처리를 가속하는 메모리 중심 컴퓨팅 기술 개발 및 저지연·실시간 인프라 구현을 위한 지능형 EdgeCPS 플랫폼 구조 설계
		내용	ㅇ 고성능·고효율 메모리 중심 컴퓨팅 핵심 기술 연구 ㅇ 메모리 중심 컴퓨팅 기반 초고속 데이터 분석 기술 연구 ㅇ 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 구조 설계 ㅇ 엣지 사이버물리시스템을 위한 대규모 디바이스 가상화 기술 설계
	2단계	목표	ㅇ 계산 성능한계를 극복하기 위한 AI 병렬컴퓨팅 구조 연구
		내용	ㅇ Peta-Scale 메모리 중심 컴퓨팅 핵심 기술 연구 ㅇ 메모리 중심 컴퓨팅 기반 스케일러블 인메모리 저장소 연구 ㅇ 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 핵심 알고리즘 개발 ㅇ 엣지 사이버물리시스템을 위한 대규모 디바이스 가상화 플랫폼 핵심 알고리즘 개발 ㅇ AI 병렬컴퓨팅 구조 연구
	3단계	목표	ㅇ 이종 인공지능 프로세서 가속기 기반 Trillion급 초성능 AI 컴퓨팅 시스템 연구
		내용	ㅇ 클러스터 레벨 대규모 생성형 AI 고성능 분산/병렬 컴퓨팅 기술 연구 ㅇ T급 생성형 모델의 이종 가속기 통합 AI 병렬 컴퓨팅 기술 연구 ㅇ 이종 가속기간 고성능 집합통신 기술 연구

2. 과제 연차별 수행과정 및 내용

(1) (1단계) 1차년도(2020년도) 개발 내용 및 범위

[메모리 중심]

수행기관	개발 목표	개발내용 및 범위
ETRI (주관)	① 이종 메모리를 고려한 메모리 중심 컴퓨팅 시스템 연구	**• 원격 메모리의 접근 대역폭을 고려한 서브-블록 관리 기법 개발** - 원격 메모리 접근 대역폭을 고려한 서브-블록 관리 기법의 필요성 · 본 과제에서는 ETRI에서 연구/개발한 대용량 메모리 관리 시스템인 Elastic Memory Platform (EMP)을 추가 개발하여 대규모 인-메모리 데이터 처리의 성능을 향상시키는 기술을 개발함 · 다른 노드의 메모리 (원격 메모리)에 빠르게 접근하기 위해서는 PCI-e나 RDMA와 같은 프로토콜을 통해서 Direct Memory Access (DMA)를 사용하게 되는데, 이때 통신 대역폭을 최대한 활용하기 위해서 많은 양의 데이터를 한번에 이동시키는 것이 중요하며, 이렇게 많은 데이터를 이동시킴으로써 프리페치의 효과를 얻을 수 있는 장점이 있음 - 서브-블록 관리 기법 연구 · 그림 11과 같이 한번에 가지고 올수 있는 연속된 페이지들을 서브-블록이라고 정의하고, 원격 메모리 접근 시 해당 서브-블록의 크기만큼 통신하도록 설정하였으며, 서브-블록의 크기를 조정할 수 있도록 구현함. 이를 통해서 원격 메모리의 접근 대역폭에 따라서 원하는 서브-블록의 크기로 조정할 수 있으며, 원격 메모리의 종류와 통신 대역폭 및 데이터 사용 패턴에 따라서 서브-블록의 크기를 정할 수 있도록 함 그림 11. 블록과 서브블록 **• 필요한 서브-블록을 먼저 처리하는 기술 개발** - 필요한 서브-블록을 먼저 처리하는 기술 개발의 필요성 · EMP 환경에서는 직접메모리에 데이터가 없는 경우 발생하는 페이지 폴트 과정에서 원격의 메모리에 저장되어있는 데이터를 직접메모리로 가지고 오는 작업을 수행함. 이때, 블록 단위로 데이터를 가지고 와서 직접메모리에 저장하며, 필요 없다고 여겨지는 데이터를 원격의 메모리에 내보내는 작업을 수행함 - 필요한 서브-블록을 먼저 처리하는 기술 연구 (CSF) · 블록의 크기를 크게 설정하는 경우 많은 데이터의 이동을 기다린 후 페이지 폴트를 처리하기 때문에 데이터의 이동이 크리티컬 패스상에 존재하게 되어 성능의 하락이 발생함. 따라서, 해당 성능 하락을 줄이기 위해서 페이지 폴트 처리 시 블록 크기만큼 원격 메모리로 DMA 읽기 요청은 보내지만 해당 폴트 처리에 필요한 서브-블록만 기다린 후 페이지 폴트를 처리하는 방식을 구현함 · 필요한 서브-블록에 대한 DMA 읽기 요청을 먼저 보내고, 나머지 서브-블록에 대한 DMA 요청은 이후에 보냄으로써 필요한 서브-블록이 빠르게 fetch될 수 있도록 구현함. 해당 기술을 Critical Sub-block First (CSF)로 명명함 · 그림 12과 같이 폴트가 나는 페이지를 포함하는 서브-블록만 기다린 후 폴트 처리를 수행하게 되면 하나의 블록 내의 다른 서브-블록들을 기다리는 시간이 제거되어 빠르게 폴트가 처리할 수 있으며, 이후에 블록내의 다른 페이지에 대해서 페이지 폴트가 나게 되면 나머지 서브-블록들에 대한 DMA 요청의 완료를 진행하도록 구현함. 만약 필요한 서브-블록만 처리한 블록이 원격 메모리로 evict되는 경우에는 해당 블록의 모든 서브-블록에 대한 DMA 요청을 완료시키고 evict되도록 개발함

수행기관	개발 목표	개발내용 및 범위

그림 12. 먼저 처리되어야할 서브-블록

• 희소 데이터 처리를 통한 네트워크 자원 소모 감소 기술 개발

- 희소 데이터 처리를 통한 네트워크 자원 소모 감소 기술 개발의 필요성
  · 통상적인 페이지 수준의 메모리 일관성 모델은 메모리의 접근 패턴 중 희소 (sparse) 접근이 발생할 경우, 높은 비효율성을 갖게 됨. 즉, 임의의 4 KiloByte (KB) 페이지에 대해서 응용의 메모리 접근이 8Byte만 이루어질 경우, 페이지 수준의 메모리 일관성 모델에서는 4KB 페이지 모두를 지역 메모리로 가져오고 이중 8Byte의 데이터에 대한 접근을 수행함. 이와 같은 데이터의 접근은 실제로 응용이 필요한 메모리 공간의 크기는 8Byte 였음에도 불구하고, 4KB를 가져오고 이에 대한 접근을 지원하기 위해서 4KB를 사용함으로써 지역 메모리를 낭비하는 결과를 초래함
  · 그림 13와 같은 희소 데이터 접근은 세 가지 비효율성을 초래함. 첫째, 의미 없는 데이터를 위치시킴으로써 직접메모리를 낭비시키게 됨. 둘째, 요구하지 않은 데이터를 적재함으로 인한 지연시간을 초래함. 셋째, 페이지 수준의 접근 추적은 4KB 중 8Byte만 갱신했음에도 불구하고 4KB페이지 전체가 변경되었음을 알려주어 네트워크 대역폭을 낭비하게 됨

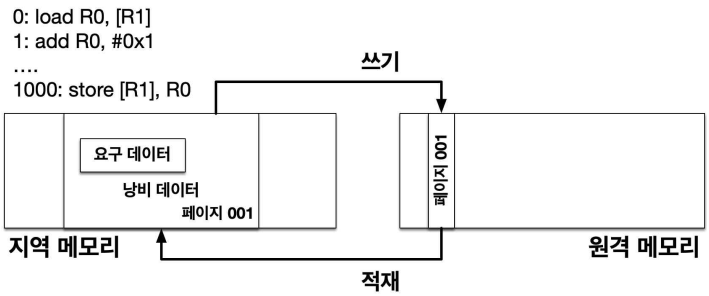


그림 13. 메모리 페이지에 대한 희소 접근으로 인한 비효율성

- 희소 데이터 접근의 비효율성 제거 연구
  · 본 과제에서는 이러한 희소 데이터 접근에 따른 비효율성을 제거하기 위하여 희소 버퍼 관리기를 구현함. 그림 14은 메모리 접근 제어에 따른 희소 버퍼 관리기의 실행 순서도를 나타냄
  · 원격 메모리 접근으로 인한 페이지 폴트 (fault)를 처리하는 도중에 실행되며 먼저 해당 접근이 원격 메모리로 인한 접근이었는지를 확인. 이때, 원격 메모리 접근이 아닌 지역 메모리에 대한 접근이었다면, 해당 명령어는 “읽기-전용”으로 설정된 페이지에 대한 접근임을 알 수 있음. 이때, 희소 버퍼에 관련된 페이지였다면 (주소:버퍼) 테이블로터 검색하여 해당 내용을 삭제하고 희소 버퍼를 삭제하며 지역 메모리 페이지에 대한 쓰기를 허용하도록 페이지 테이블을 조정하고 지역 메모리가 변경되어 추후 지역 메모리로부터 evict 될 때, 해당 페이지가 원격 메모리에 쓰여야함으로 표기함
  · 만약 원격 메모리 접근에 대한 페이지 폴트였다면 해당 폴트를 발생시킨 명령어를 디코드하고, 읽기 명령어였다면 (주소:버퍼) 테이블을 검색하여 저장된 희소 버퍼가 존재하는지를 확인하고 있다면 원격 메모리로부터 요구 페이지를 지역 메모리를 적재한 후 앞서 검색된 희소 버퍼 내용들을 해당 페이지에 적용한 후, 페이지를 “읽기-전용” 페이지로 주소

수행기관	개발 목표	개발내용 및 범위

공간에 사상함

· 만약 읽기 명령어가 아닌 쓰기 명령어였다면 디코드한 명령어로부터 주소와 값을 획득하여 희소 버퍼를 할당하고 (주소오프셋:값)을 할당받은 버퍼에 저장함. 이후 주소를 키로 사용하고 버퍼를 값으로 (주소:버퍼) 테이블에 삽입하고, 쓰기 명령어가 다시 실행되는 것을 방지하기 위해서 명령어 포인터 (IP)를 다음 명령어를 가르키도록 갱신함

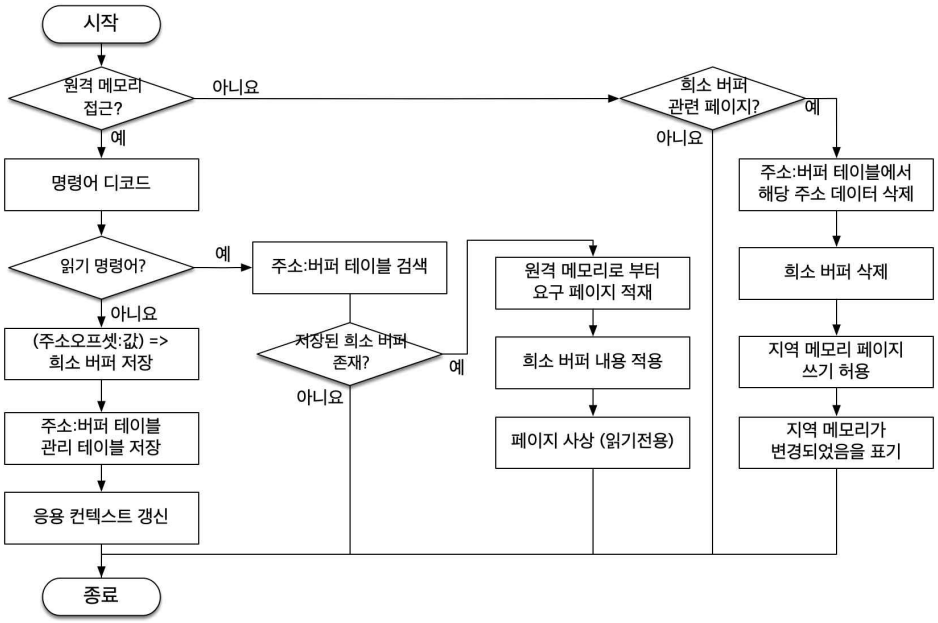


그림 14. 희소 버퍼 관리기 실행 순서도

- 희소 버퍼 관리기의 효과

· 그림 15와 같이 희소 버퍼 관리기를 통해서 작은 크기의 변경데이터만을 직접메모리에 저장하여 직접메모리 사용률을 낮추게 됨. 직접메모리는 메모리 분리 시스템에서 성능에 민감한 데이터의 저장을 제공하는 컴포넌트이고, 이는 저장 가능한 유효 데이터 블록의 수에 의해 결정됨. 즉, 페이지 수준 메모리 일관성 모델은 지역 메모리에 희소 데이터 메모리 접근 패턴에 대해서 낭비 데이터를 저장하게 되고, 필요 없는 원격 메모리와의 데이터 전송을 발생시키는 문제를 갖게 됨

· 하지만 희소 버퍼 관리기를 통해 제어되는 서브페이지 메모리 관리는 동일한 용량의 지역 DRAM을 활용하여 더 많은 데이터를 저장하고, 원격 메모리와의 요구되는 네트워크 전송량을 줄일 수 있음. 또한, 원격 메모리에 위치하고 있는 데이터에 대한 희소 쓰기를 수행함에 있어서 기존과 달리 원격 메모리를 지역 메모리로의 적재 없이 지역 메모리에 쓰기를 수행할 수 있기 때문에 원격 메모리에 대한 적재 비용 또한 줄일 수 있음

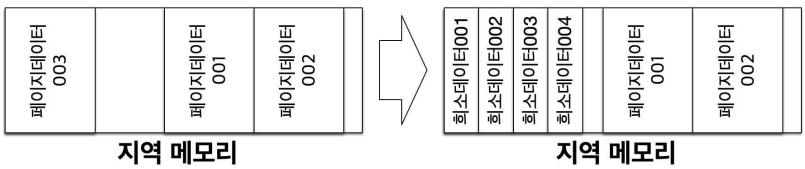
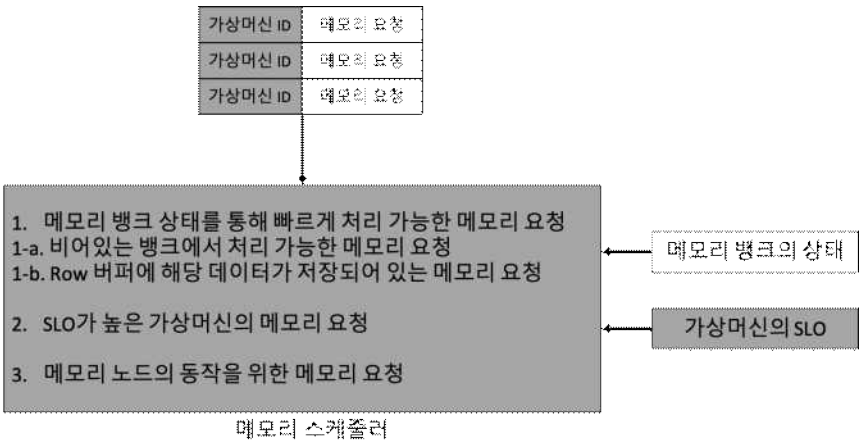
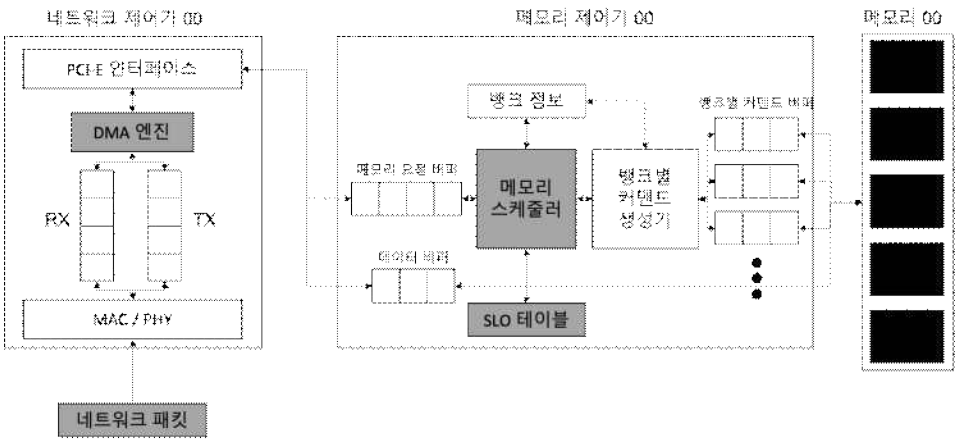


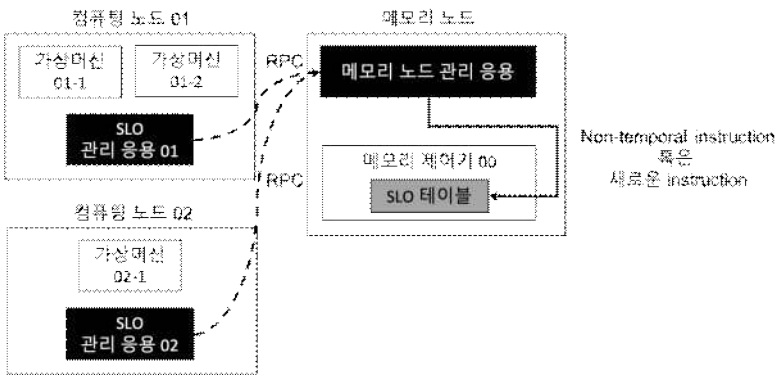
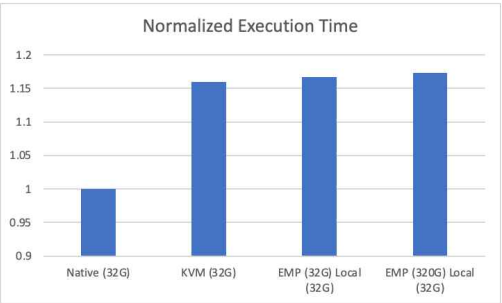
그림 15. 희소 버퍼 관리기를 통한 직접메모리 효율성

• QoS를 고려한 메모리 전용 노드의 메모리 컨트롤러 구조 및 스케줄링 방법 설계

- QoS를 고려한 메모리 전용 노드의 메모리 컨트롤러 구조 및 스케줄링 방법의 필요성

· 클러스터 환경에서 원격 메모리에 대한 접근 시간이 전체 클러스터 성능에 큰 영향을 주게 됨. 따라서, 각 컴퓨팅 노드의 가상머신이나 프로세스의 Quality of Service (QoS)를 보장하고 클러스터 전체의 성능을 향상시키기 위해서 메모리 노드의 메모리 컨트롤러 구

수행기관	개발 목표	개발내용 및 범위
		성의 변화가 필요함 - QoS를 고려한 메모리 컨트롤러 구조 및 스케줄링 방법 설계 · 컴퓨팅 노드에서 동작하는 프로세스의 QoS를 보장하기 위해서 메모리 전용 노드의 메모리 스케줄러는 그림 16와 같은 메모리 요청 스케줄링 우선순위를 가지고 처리를 진행하도록 설계함. 이는 메모리 노드의 메모리 요청이 burst하게 들어온다는 사실과, 각 프로세스에 대해서 Service-Level Objective (SLO)가 다르다는 것을 반영하여 메모리 요청을 처리하는 방법임  그림 16. 메모리 스케줄러의 처리 우선순위 · 메모리 노드에서 메모리 요청에 대한 스케줄링은 소프트웨어로도 처리할 수 있지만, 다수의 메모리 노드 요청은 one-sided RDMA를 사용하여 메모리 노드의 CPU 개입이 없이 데이터를 전송하는 방법을 사용하기 때문에 하드웨어를 수정하는 방법으로 설계함. · 그림 17은 메모리 노드에서 하드웨어 수정 부분을 음영으로 표현한 그림으로, 네트워크 패킷의 경우 각 가상머신의 ID를 포함하도록 수정하였으며, DMA엔진은 각 가상머신의 네트워크 패킷을 통해서 어떤 가상머신의 메모리 요청인지를 확인 가능함. 또한 메모리 스케줄러는 SLO 테이블에 저장된 컴퓨팅 노드의 프로세스나 가상머신의 SLO와 그림 16와 같은 우선순위를 바탕으로 메모리 요청을 처리함  그림 17. 메모리 노드의 하드웨어 수정사항 · 또한, 원격 메모리노드의 메모리 스케줄러는 각 가상머신이나 프로세스의 SLO를 알아야 함. 이를 위해서 클러스터 내부의 각 가상머신에 대한 SLO를 메모리 제어기00으로 전달하는 프로토콜이 필요함. 그림 18과 같이 전체 클러스터 관리 응용이나 각 컴퓨팅 노드에

수행기관	개발 목표	개발내용 및 범위
		서 해당 노드에서 동작하는 가상머신의 QoS를 보장하기 위한 SLO 관리응용이 동작함 · 이 관리응용이 주기적으로 Remote Procedure Call (RPC)를 통해서 메모리 노드 관리응용으로 각 가상머신의 SLO를 전달하고, 원격 메모리 노드의 관리응용은 해당 SLO를 메모리 제어기에 있는 레지스터를 활용하여 SLO 테이블의 형태로 업데이트 함  그림 18. 클러스터 내 가상머신의 SLO 전달방법 **• 메모리 확장을 통한 대규모 디지털포렌식 지원 검증** - EMP환경에서 디지털포렌식 지원검증 · EMP를 통한 메모리 확장을 사용하여 ㈜한컴위드에서 개발한 디지털포렌식의 동작 검증 · 대규모 디지털 포렌식의 경우 인-메모리 컴퓨팅을 필요로 하지만 메모리양이 적은 경우 디스크를 사용할 수밖에 없었으나, 메모리 확장을 통해서 인-메모리 컴퓨팅으로 수행할 수 있다는 장점이 있음 · 본 검증에서는 대규모의 디지털포렌식을 수행하지 않고, 크기가 적은 디지털포렌식을 통해서 기존 시스템대비 EMP를 적용하게 되면 발생하는 오버헤드에 대한 분석을 진행함  그림 19. EMP환경에서 디지털포렌식 수행 결과 성능평가 - EMP환경에서 디지털포렌식 지원 수행결과 성능평가 · 그림 19과 같이 디지털포렌식을 수행함에 있어 EMP환경 위에서도 잘 동작하는 것을 확인할 수 있었으며, 기존 시스템대비 15%의 성능 오버헤드는 가지고 있지만 대규모의 디지털포렌식을 수행할 수 있다는 것을 확인할 수 있음. 또한, 대부분의 성능하락은 가상화에 따른 오버헤드라는 것을 확인할 수 있었는데, 가상화된 환경과의 비교에서는 2%내의 오버헤드를 통해서 대규모 메모리 확장을 가능하게 함을 확인 **• 가상화 기반 메모리 확장 시스템을 위한 네트워크 성능 최적화 기법 연구** - 가상화 기반 메모리 확장 시스템을 위한 네트워크 성능 최적화의 필요성 · 가상머신은 물리노드의 운영체제를 통해서 네트워크 장치를 이용해야 하므로 네트워크 성능이 느릴 수밖에 없음 · 이를 극복하기 위해 가상머신에게 네트워크 장치를 물리적으로 할당하여 직접 사용하게

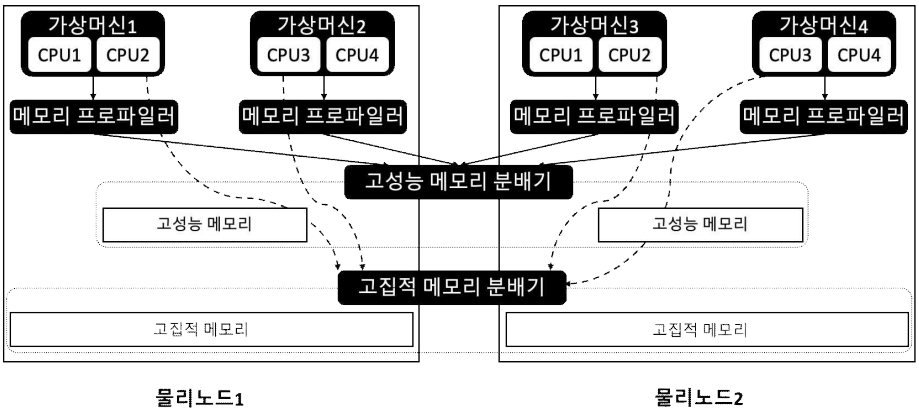
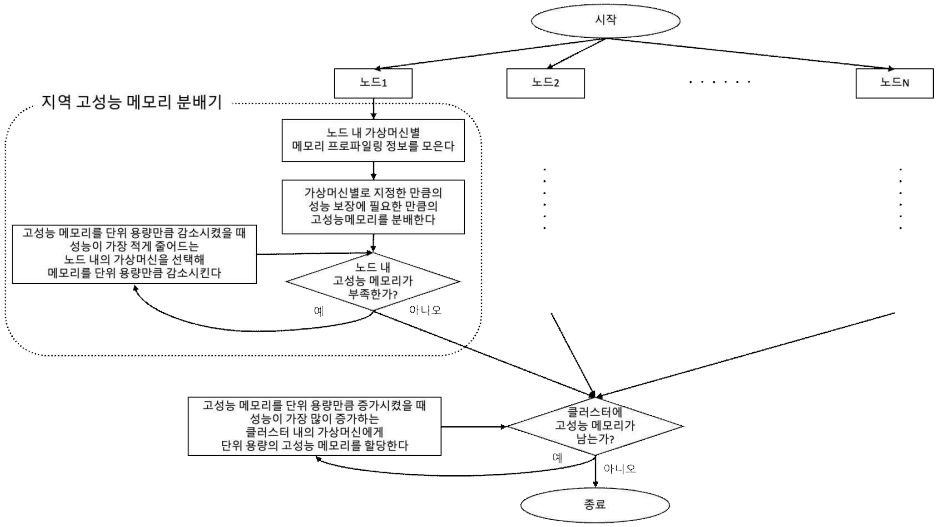
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>
하는 pass-through 기법이 제안됨. 하지만 해당 방법은 가상머신의 모든 메모리 영역이 물리노드의 DRAM에 할당되어 있다는 것을 가정하고 설계되어 있으므로 원격의 메모리나 고성능SSD도 메모리로 활용하는 메모리 확장 시스템에 적용 불가능
· 따라서 인메모리 키-밸류 저장소 등 대용량 메모리를 필요로 하면서도 네트워크 중심의 응용을 지원하기 위해서는, 메모리 확장 시스템을 위한 네트워크 성능 최적화가 필요

- 가상화 기반 메모리 확장 시스템을 위한 네트워크 성능 최적화 기법 설계
· 개념: 가상머신 내에서 네트워크를 위한 메모리 영역을 제한하고, 하이퍼바이저가 이를 고려하여 해당 메모리 영역을 물리노드의 DRAM에 고정시켜서, 메모리 확장 시스템 하에서도 pass-through 기법을 통해 네트워크 장치를 이용할 수 있게 함
· 구현 방법
1) 가상머신 내의 운영체제를 수정하여, 네트워크 장치를 위한 socket buffer가 4GB 이하의 메모리 영역만 사용하도록 바꿈. 리눅스의 ZONE_DMA32를 사용하여 구현
2) 하이퍼바이저를 수정하여, 가상머신의 메모리 중 4GB 이하의 영역은 항상 물리노드의 DRAM에 할당되도록 함. QEMU를 수정하여 구현

- 구현 및 실험 결과
· 가상머신 운영체제는 Linux 5.3에 구현. 하이퍼바이저는 QEMU 2.8에 구현
· 인메모리 키-밸류 저장소인 redis를 대상으로 성능 평가. 단, 네트워크 성능 차이에 집중하기 위해, 가상머신이 사용하는 지역 메모리는 350GB로 설정하고, 응용의 전체 데이터 양은 273GB로 설정하여 메모리 확장에 따른 성능 저하가 없도록 함
· 아래 표는 memtier benchmark를 통해 평가한 redis의 성능을 평균 지연 시간으로 나타낸 것. 본 과제를 통해 개발한 메모리 확장 시스템인 EMP에 pass-through 기법을 적용하면(EMP passthru), 리눅스의 기본 하이퍼바이저에 pass-through 기법을 적용한 것(KVM passthru)와 성능이 유사해지며, pass-through 기법을 적용하지 않을 때(KVM vhost)와 비교하면 2배 이상 성능이 향상됨

표 10. 가상화 기반 메모리 확장 시스템 성능 최적화 기법을 적용한 redis 성능 평가
<table>
<tr><th></th><th>KVM passthru</th><th>EMP passthru</th><th>KVM vhost</th></tr>
<tr><td>[insert]</td><td>0.27</td><td>0.30</td><td>0.78</td></tr>
<tr><td>[update_random]</td><td>0.25</td><td>0.30</td><td>0.77</td></tr>
<tr><td>[update_longtail]</td><td>0.25</td><td>0.28</td><td>0.77</td></tr>
<tr><td>[read_random]</td><td>0.26</td><td>0.29</td><td>0.76</td></tr>
<tr><td>[read_longtail]</td><td>0.26</td><td>0.29</td><td>0.76</td></tr>
<tr><td>[mix_random]</td><td>0.26</td><td>0.28</td><td>0.77</td></tr>
<tr><td>[mix_longtail]</td><td>0.26</td><td>0.29</td><td>0.76</td></tr>
</table>
</td></tr>
<tr><td></td><td>② 다중 응용을 고려한 공유 메모리 관리 기술 연구</td><td>
• 메모리 중심 컴퓨팅 시스템 기반 클러스터 메모리 통합 관리 도구 설계
- 클러스터 메모리 통합 관리 도구의 필요성
· 메모리 확장에 기반한 메모리 중심 컴퓨팅 시스템은, 대규모 메모리를 가지고 있기 때문에 클라우드컴퓨팅서비스 등 다중 응용을 실행하기에 적합
· 하지만, 대규모 메모리를 구현하기 위해 인피니밴드 등 고속네트워크로 연결된 다른 서버의 메모리나 고성능SSD 등을 쓰기 때문에 지역메모리만을 사용하는 시스템에 비해 성능이 느려지는 것이 단점. 특히 다중 응용이 실행되는 경우 I/O 대역폭 공유에 따른 성능 간섭이 일어나 성능이 불안정해지는 것이 단점으로 부각됨
· 따라서, 다중 응용을 사용하는 메모리 중심 컴퓨팅 시스템의 성능을 최적화하면서도 개별 응용의 성능을 보장해주는 기법이 요구됨

- 클러스터 메모리 통합 관리 도구 설계
</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		· 개념: 클러스터 메모리 통합 관리 도구는 다수의 서버로 구성된 클러스터의 모든 고성능 메모리(DRAM 등)와 고집적메모리(고성능 SSD 등) 통합 관리하여, 각 가상머신의 성능을 일정 수준(고성능 메모리만 사용했을 때에 비교한 비율로 나타냄) 이상으로 보장하면서 전체적인 성능을 최적화 · 구조: 아래 그림 20는 클러스터 메모리 통합 관리 도구의 구조를 나타냄. 두 개 이상의 물리노드에 각각 하나 또는 여러 개의 가상머신이 실행 중인 상황에서, 각 가상머신 별로 "메모리 프로파일러"가 실행됨. 그리고 클러스터 전체의 고성능 메모리 통합 관리를 담당하는 "고성능 메모리 분배기"와 클러스터 전체의 고집적 메모리 분배기를 담당하는 "고집적 메모리 분배기"로 구성됨  그림 20. 클러스터 메모리 통합 관리 도구 구조도 - 메모리 프로파일러: 메모리 프로파일러는 이전 과제의 연구 결과물로서 ETRI에서 보유 중인 기술을 을 활용 · 국내특허: "분할 메모리 관리장치 및 방법" (출원번호: 2018-0122434) · 논문: Kwangwon Koh, Kangho Kim, Changdae Kim, and Jaehyuk Huh, "End Performance SLA Support for Disaggregated Memory", The 1st Workshop on Resource Disaggregation (WORD 2019) - 고성능 메모리 분배기 설계  그림 21. 고성능 메모리 분배기 · 개념: 고성능 메모리 분배기는 클러스터 시스템 내의 고성능 메모리들을 통합 관리하여 각 가상머신의 성능을 보장하면서도 전체 가상머신의 평균 처리량을 가장 높일 수 있는

수행기관	개발 목표	개발내용 및 범위
		방법으로 고성능 메모리를 분배. 각 노드별로 동작하는 지역 고성능 메모리 분배기와 클러스터 단위로 동작하는 전역 고성능 메모리 분배기로 구분할 수 있음 · 고성능 메모리 분배 알고리즘 1) 가상머신 별 메모리 프로파일러를 통해 메모리 확장을 사용한 가상머신의 성능 추정치 파악 2) [지역 고성능 메모리 분배기] 가상머신 별로 지정된 만큼의 성능을 보장하기 위해 필요한 고성능 메모리 용량을 분배. 단, 노드 내 고성능 메모리 용량이 성능 보장에 필요한 양보다 적을 경우, 단위 용량만큼 고성능 메모리 용량을 감소시켰을 때 가장 성능이 적게 감소하는 가상머신의 고성능 메모리 용량을 감소시키는 것을 반복하여, 노드 내 가상머신들에게 분배된 고성능 메모리가 노드 내 고성능 메모리 용량을 초과하지 않게 함 3) [전역 고성능 메모리 분배기] 클러스터 내에 분배되지 않고 남아있는 고성능 메모리가 있는지 확인. 남는 고성능 메모리를 단위 용량 단위로, 단위 용량의 고성능 메모리가 추가로 주어졌을 때 성능이 가장 많이 증가하는 가상머신에게 배분. 이를 클러스터 내의 모든 고성능 메모리가 분배될 때까지 반복 - 고집적 메모리 분배기 설계 · 개념: 고집적 메모리 분배기는 클러스터 시스템 내의 고집적 메모리들을 통합 관리하여 각 가상머신의 전체 메모리 용량을 보장하면서도 전체 가상머신의 평균 처리량을 가장 높일 수 있는 방법으로 고집적 메모리를 분배. 가상머신이 새로 생성될 때마다 실행 · 고집적 메모리 분배 알고리즘

수행기관	개발 목표	개발내용 및 범위

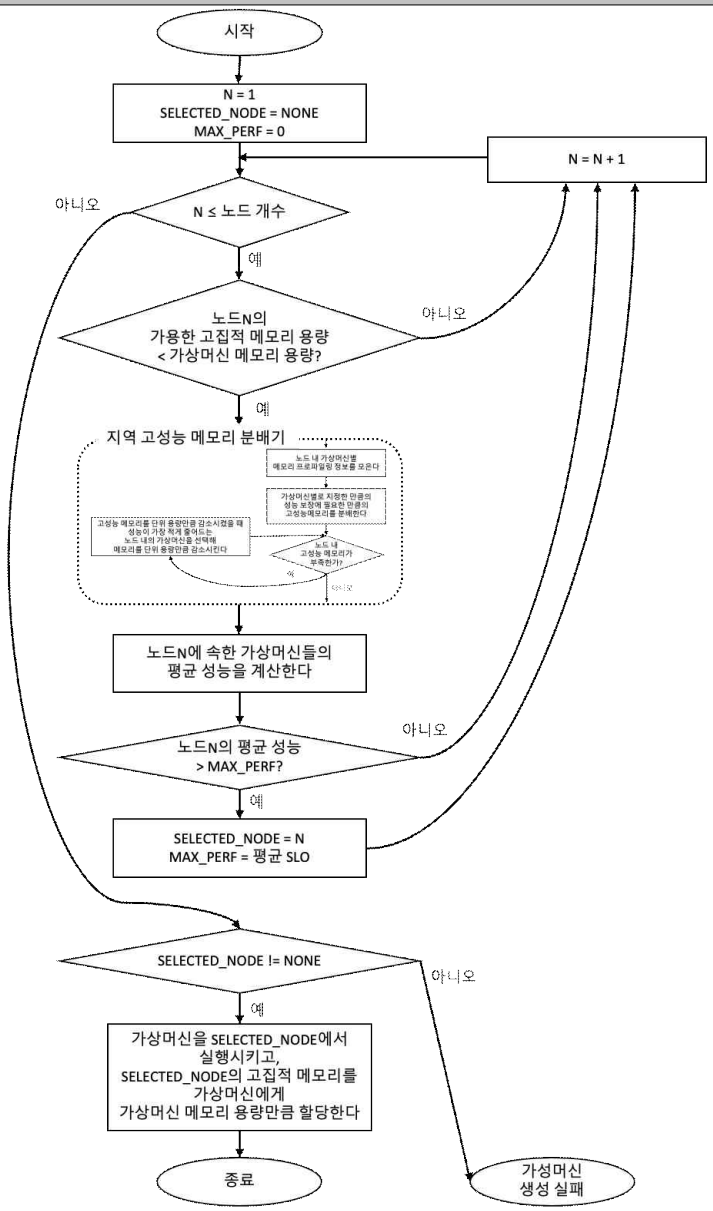
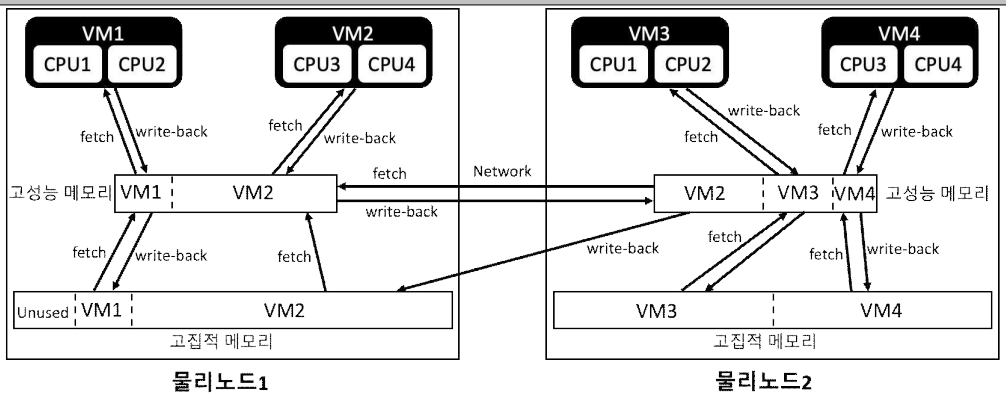


그림 22. 고집적 메모리 분배기

1) 노드의 고집적 메모리가 가상머신의 전체 메모리양보다 적을 경우 후보 노드에서 제외. 고집적 메모리는 가상머신의 전체 메모리 용량을 보장하기 위한 용도로 사용되기 때문
2) 남은 후보 노드 중 가상머신의 평균 성능이 가장 높은 노드를 가상머신이 고집적 메모리를 분배 받을 대상 노드로 선정. 평균 성능이 가장 높은 노드에서 실행되어야 클러스터 시스템 전체에서 실행 중인 가상머신들의 평균 성능에 가장 적은 영향을 끼칠 것이기 때문

- 시스템 소프트웨어 구현 방식 설계
  · 아래 그림은 클러스터 시스템 메모리 통합 관리 도구에 있어, 각 가상머신이 메모리를 접근하고 관리하는 기법을 나타냄

수행기관	개발 목표	개발내용 및 범위
		 그림 23. 클러스터 메모리 통합 관리 시스템에서 가상머신의 메모리 관리 기법 · 각 가상머신은 가장 자주 사용하는 데이터를 지역 고성능 메모리에, 그 다음으로 자주 사용하는 데이터를 원격 고성능 메모리에, 가장 가끔 사용하는 데이터를 고집적 메모리에 배치 · 각 가상머신이 메모리에 접근할 때, 해당 데이터가 지역 고성능 메모리에 없으면, 지역 고성능 메모리에 있는 데이터 중 가장 가끔 사용하는 데이터를 다음 단계의 메모리(원격 고성능 메모리를 배분 받았다면 원격 고성능 메모리, 아니라면 고집적 메모리)에 옮겨놓고, 접근하려는 데이터를 지역 고성능 메모리에 가져온 다음 접근 · 일반적으로 원격 고성능 메모리에 접근하는 시간이 노드 내의 고집적 메모리에 접근하는 시간보다 짧기 때문에 위와 같은 구조로 설계 노드1에서 실행 중인 가상머신1의 메모리 영역 (숫자는 가장 최근에 접근한 순서) 19 10 22 16 12 30 3 7 2 5 18 15 26 11 23 14 6 28 32 4 13 21 1 25 9 8 31 20 29 24 17 27 고성능 메모리(노드1): 1 2 3 4 5 6 고성능 메모리(노드2): 가상머신2 7 9 고성능 메모리(노드3): 가상머신3 8 10 11 고집적 메모리 (음영 표시된 영역은 업데이트 되지 않은 데이터를 포함할 수 있음) 19 10′ 22 16 12 30 3′ 7′ 2′ 5′ 18 15 26 11′ 23 14 6′ 28 32 4′ 13 21 1′ 25 9′ 8′ 31 20 29 24 17 27 그림 24. 가상머신 메모리 배치 방법 예시 · 위 그림은 노드1에서 실행 중인 가상머신에 대한 메모리 배치의 예시를 나타냄. 고집적 메모리는 가상머신의 메모리 용량을 보장해주며, 가상머신의 전체 메모리 영역에 대한 데이터를 가짐. 단, 지역 혹은 원격의 고성능 메모리에 배치된 데이터의 경우, 고집적 메모리에 있는 데이터가 최신의 데이터가 아닐 수 있음. 따라서 메모리 접근시 항상 고성능 메모리에 있는 데이터를 우선적으로 사용
	③ 데이터 분석 응용의 메모리 사용 패턴 연구	• 유전체 서열 정렬(Genome Sequence Alignment) 프로그램 분석 - 유전체 서열 정렬: 인간 혹은 생물체로부터 읽어낸 짧은 염기서열이 인간 혹은 생물체 전체의 유전체로 구성된 참조 유전체(Reference Genome)의 어느 곳에 위치하는지 결정하는 것. 모든 유전체에는 차이가 있고, 염기서열을 읽어내는 것에도 오류가 발생할 수 있기 때문에, 염기서열의 추가(insert), 삭제(delete), 변형(mutation)을 고려하여 참조 유전체에서 가장 비슷한 염기서열이 있는 곳을 찾아 위치를 결정 - BWA-MEM2 분석 · BWA-MEM2는 Intel에서 개발한 유전체 서열 정렬 프로그램으로 가장 널리 쓰이는 프로그램 중 하나. Borrwos-Wheeler Transform (BWT)과 Ferragina Manzini (FM)

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>
index, Banded Smith-Waterman Alignment 알고리즘을 사용

표 11. BWA-MEM2의 프로그램 단계들

<table>
<tr><th>단계 이름</th><th>설명</th></tr>
<tr><td>SMEM</td><td>FM-index를 사용해 염기서열의 일부분과 가장 크게 일치하는(Super Maximal Exact Matching) 참조 유전체의 부분에 대한 정보 탐색</td></tr>
<tr><td>SAL</td><td>SMEM 단계의 결과를 가지고 Suffix Array 색인을 탐색하여(Suffix Array Lookup)을 통해 해당 참조 유전체 부분의 정확한 위치를 얻어냄</td></tr>
<tr><td>CHAIN</td><td>SAL 단계를 결과를 보고 이어질 수 있는(possible to chain) 일치 정합(Exact Matching) 부분들을 찾아내어 정렬된 위치에 대한 후보군을 줄임</td></tr>
<tr><td>BSW</td><td>Banded Smith-Waterman 알고리즘을 이용하여 염기서열을 후보군들에게 정렬시켜 본 뒤, 가장 일치하는 정렬 위치를 찾아냄</td></tr>
<tr><td>SAM-FORM</td><td>위 과정을 통해 찾아낸 결과를 SAM 포맷으로 출력</td></tr>
</table>

· 아래는 각 단계에 대한 실행 시간을 분석한 것. SMEM, BSW에 대부분의 시간을 소모

표 12. BWA-MEM2 프로그램 단계별 수행 시간

<table>
<tr><th></th><th>실행 시간(초)</th><th>비율</th></tr>
<tr><td>Index load</td><td>41.69</td><td>4.8%</td></tr>
<tr><td>SMEM</td><td>292.22</td><td>33.4%</td></tr>
<tr><td>SAL (+CHAIN)</td><td>50.77</td><td>5.8%</td></tr>
<tr><td>BSW</td><td>228.08</td><td>26.1%</td></tr>
<tr><td>SAM</td><td>46.47</td><td>5.3%</td></tr>
<tr><td>IO wait (fastq, sam)</td><td>4.33</td><td>0.5%</td></tr>
<tr><td>misc(memory alloc, etc)</td><td>211.71</td><td>24.2%</td></tr>
<tr><td>total</td><td>875.27</td><td></td></tr>
</table>

· 가장 많은 실행 시간을 차지하는 SMEM의 경우, FM-index에 대한 random memory access가 빈번한 것을 확인

• BWA-MEM-LARGE 설계: 대규모 메모리 기반 BWA-MEM2 가속 프로그램
- 개념: BWA-MEM2에 대규모 메모리를 사용한 가속 기법 2가지를 개발하여 적용하고, 대규모 인덱스를 로딩하는 시간을 절약하기 위한 기법을 개발하여, BWA-MEM2를 가속
- Perfect Match 기법: 참조유전체와 일치정합(exact match)하는 염기서열(read)들에 대해 컴퓨팅, 특히 SMEM, SAL, BSW 단계를 생략하는 기법
</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

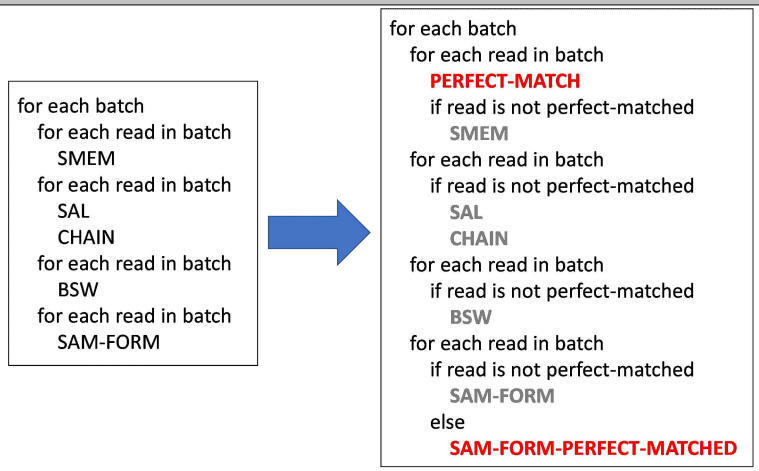


그림 25. Perfect Match 기법을 적용한 BWA-MEM-LARGE의 프로그램 흐름

· 유전체 서열 정렬은 염기서열에 대해 참조유전체 상의 가장 유사한 위치를 결정하는 것인데, 같은 종(species)의 유전체는 유사성이 상당히 높으므로, 짧은 염기서열들의 상당 수가 참조유전체의 부분과 일치 정합함.
· 실제 데이터셋에 대한 분석 결과 61~76%의 염기서열들이 참조유전체와 일치정합하는 위치가 존재함

표 13. 염기서열 데이터셋과 참조유전체 사이의 일치정합 비율 분석 결과

Data Set	BP len	#file	#read	%matched	%rc_matched	%not_matched	%N포함
HG001	148	871	2987589518	**73.78%**	36.25%	26.03%	0.19%
HG002	148	935	3078710427	**72.53%**	35.62%	27.31%	0.17%
HG003	148	1005	3188842617	**71.57%**	35.16%	28.27%	0.16%
HG004	148	1024	3505883476	**71.36%**	35.09%	28.50%	0.15%
PRJEB20654	150	300	121354031174	**61.31%**	30.11%	37.39%	1.30%
BWA-MEM2.D3	76	6	109203301	**64.21%**	30.95%	33.51%	2.28%
BWA-MEM2.D4	101	1	61005824	**74.32%**	36.32%	23.72%	1.96%
BWA-MEM2.D5	101	1	189323264	**75.92%**	36.89%	22.66%	1.41%

· 알고리즘: 참조 유전체를 특정 염기서열 길이로 자른 seed를 엔트리로 하는 해쉬 테이블을 구성. 해쉬 값이 같은 염기서열들은 엔트리에 left, right 항목을 추가하여 binary search tree를 구성. 염기서열(read)가 들어오면 해쉬 값을 계산한 뒤 해쉬 테이블 상의 엔트리를 찾고, 엔트리가 있을 경우 binary search를 사용해 해당 염기서열의 참조유전체 상의 위치를 읽어냄

수행기관	개발 목표	개발내용 및 범위

```
perfect_match(read):
    index = hash value of read
    entry = seed table entry with index = index
    return match_read(entry, read)

match_read(entry, read):
    if entry→valid == FALSE:
        return NOT_FOUND
    if read == entry→seed:
        return entry
    else if read < entry→seed:
        if entry→left == NOT_EXIST:
            return NOT_FOUND
        else:
            return match_read(entry→left, read)
    else if read > entry→seed:
        if entry→right == NOT_EXIST:
            return NOT_FOUND
        else:
            return match_read(entry→right, read)
```

그림 26. Perfect Match 기법의 알고리즘

- FM-index Accelerator (FMA) 기법: FM-index에 대한 random memory access가 일어나는 SMEM 단계에서, 각 염기서열에 대해 초반부에 접근하는 FM-index 부분의 유사도를 활용하여 SMEM 단계를 가속하는 기법
  · 알고리즘: 아래 그림은 BWA-MEM2의 SMEM 단계와 BWA-MEM-LARGE의 SMEM 단계의 알고리즘을 나타냄. BWA-MEM-LARGE에서는 각 염기서열의 처음 15글자(bp)를 FM-index가 아니라 FM-index Acclerator를 통해 계산

```
for each read in batch
  for each position in read
    a = read[position]
    smem = D[a + 1] – D[a]
    for p in position to length of read:
      a = read[p]
      extend smem with a
      if extension failed:
        break
```

```
for each read in batch
  for each position in read
    table = FMA root table
    p = position
    for level in 1 to 5:
      idx = read[ p : p + 3 ]
      smem = table[idx] // cached smem
      if extension failed:
        break
      p = p + 3
      table = table[idx]->next
    for p in (position+15) to length of read:
      a = read[p]
      extend smem with a
      if extension failed:
        break
```

그림 27. FM-index Accelerator 알고리즘

  · 효과: FM-index Accelerator는 1~2개의 cacheline만 접근하여도 15글자에 대한 BWT 알고리즘을 수행할 수 있음. 반면 기존 방식은 최소 15개의 cacheline을 접근하여야 함
- In-memory Index Store 기법: process shared memory 기법을 사용해 BWA-MEM-LARGE에서 사용되는 인덱스들을 한 번의 로딩 후에 반복사용하는 기법. 일반적으로 유전체 서열 정렬시 많은 수의 입력파일들을 하나의 참조유전체에 정렬하게 되므로, 참조유전체 관련 인덱스들은 여러 실행에 대해 반복사용하게 됨
  · 아래 표는 BWA-MEM2와 BWA-MEM-LARGE에 대한 인덱스 로딩 시간을 분석. BWA-MEM-LARGE가 대규모 인덱스를 사용하면서 인덱스 로딩에 훨씬 많은 시간이

수행기관	개발 목표	개발내용 및 범위

필요. In-memory Index Store는 해당 시간을 최초 1회로 한정시켜 주며, 인덱스 영역에 대해 HugeTLB도 사용하게 하여 실행 시간을 더욱 단축시킴

표 14. BWA-MEM2와 BWA-MEM-LARGE의 인덱스 로딩 시간

	Capacity	BWA-MEM2	PERFECT	FMA	PERFECT & FMA
FM-index	35.06GB	O	O	O	O
PAC (2-bit packed ref)	0.73GB	O	O	O	O
Ref. string (1-byte packed ref)	5.84GB	O	O	O	O
Perfect Table	224.99GB		O		O
FM-index Accelerator (FMA)	64.86GB			O	O
Loading Time (s)		40.1	261.6	95.3	328.7

- 실험 결과: BWA-MEM2 논문에 제시된 입력파일 중 D5 파일에 대해 성능을 측정. 그결과 최대 2.31배의 성능 향상을 얻음
  · org: BWA-MEM2
  · shm: In-memory Index Store
  · perfect: perfect-match
  · accel: FM-index Accelerator
  · 4kb, 1gb: Index에 대해 사용한 메모리 페이지 크기 (1GB는 HugeTLB 적용)

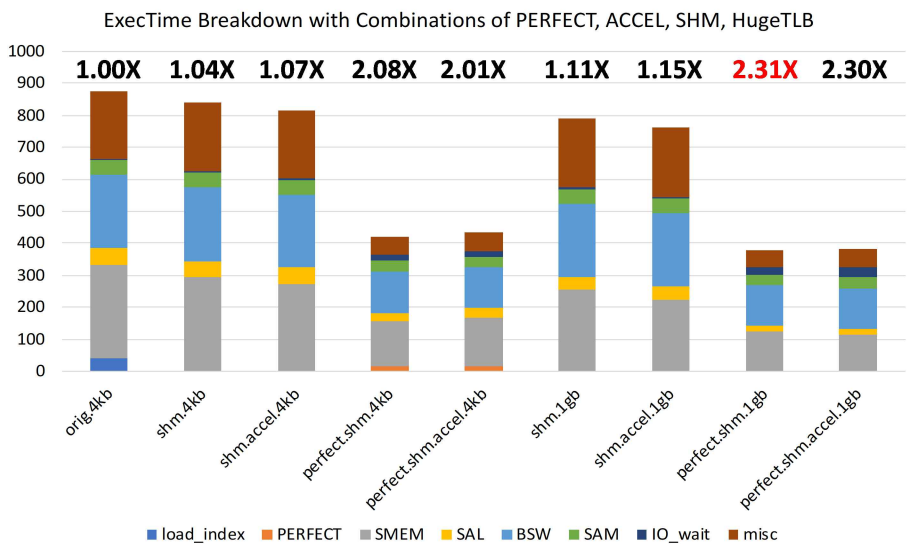


그림 28. BWA-MEM-LARGE의 기법들에 대한 성능 분석

• 데이터센터의 메모리 사용패턴 분석
- 개념: 많은 컴퓨팅 및 데이터 분석이 이루어지는 클라우드 서비스 및 데이터센터의 트레이스를 분석하여, 클라우드 상의 서버들의 메모리 사용패턴을 분석. 차후 연구에 활용하고자 함
- 분석한 트레이스
  · Google Trace: 2011년 5월에 구글 클라우드에서 수집된 트레이스
  · Bitbrains tace: 2013년 8~9월에 비트브레인즈 데이터센터에서 수집된 트레이스
  · Alibaba Trace: 2018년에 알리바바 클라우드에서 수집된 트레이스
- 데이터센터 메모리 사용패턴 분석
  · 평균 할당량은 28~88%. 특히 최근 트레이스인 Alibaba trace에서는 88%를

수행기관	개발 목표	개발내용 및 범위

나타냄으로써 메모리 용량 문제로 다른 자원의 활용률이 떨어질 것으로 예상

표 15. 데이터센터 평균 메모리 할당량 분석 결과

Traces	Allocated Memory
Google trace	70%
Alibaba trace	88%
Bitbrains trace	28%

· 아래 그림의 누적분포도를 보면, Google trace와 Bitbrains는 노드 간 큰 격차를 보이고 Alibaba trace에서는 대부분의 노드가 평균에 근접한 수치를 보임. 메모리 관리를 적극적으로 하는 것으로 보임

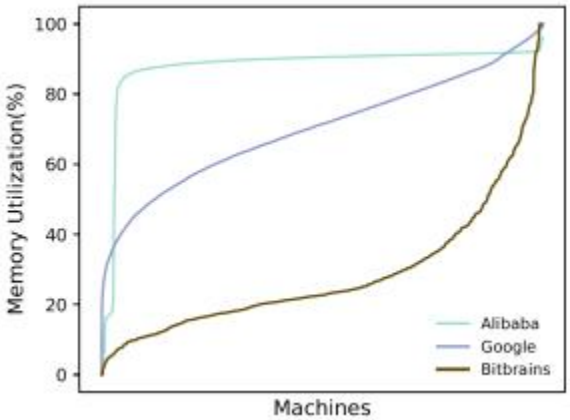
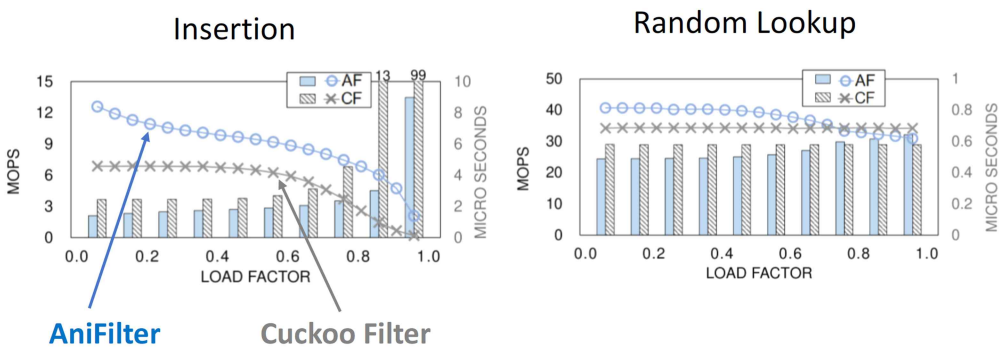


그림 29. 데이터센터 노드들에 대한 메모리 활용률 누적분포도

· 각 노드의 메모리 활용률이 100%가 되면 기존의 시스템에서는 프로그램이 실행이 중단되므로 알리바바처럼 메모리 활용률 관리를 적극적으로 하는 경우에도 노드 내 메모리의 10% 이상을 유휴 자원으로 남겨두게 됨. 이를 메모리 확장 등에 사용하면 시스템 전체 자원 활용율이 높아질 것으로 예상

• 비휘발성 메모리 상의 Approximate Membership Queries에 대한 메모리 사용패턴 분석
- Approximate Set Data Structure (AMQ): 주어진 x가 set에 포함되어 있는 지를 파악할 수 있는 데이터 구조로, 약간의 false-positive (x가 set에 포함되어 있지 않지만 포함되어 있다고 판단)를 허용하는 대신 빠르게 판단함. 다음 세 가지 operation으로 구성됨
  · Insert(x): x를 set에 추가
  · Lookup(x): x가 set에 있는지를 판단하여 true 혹은 false를 반환
  · Delete(x): (optional) x를 set에서 제거
- 비휘발성 메모리 상에서 기존 AMQ 구조의 문제점
  · 대규모 메모리 시스템 구성에 필수적인 비휘발성 메모리는 DRAM에 비해 read latency가 2~3배 정도 느리고, bandwidth는 1/10 수준. 따라서 DRAM 버전의 AMQ를 비휘발성 메모리 상에서 실행하면 성능이 나빠짐
  · AMQ는 약간의 false-positive를 허용하는 대신 매우 짧은 지연시간을 가짐. 따라서 복잡한 최적화 기법은 적용할 수 없음
  · 최신 기술인 Cuckoo filter의 경우, 버킷당 4개 슬롯을 가지고 12bit fingerprint를 사용하여 버킷 크기가 6바이트가 되는데, NVM의 원자적 쓰기 단위는 8바이트. 이 미스매치로 인해 결함의 원자성(failure-atomicity)이 보장 안 됨
- 비휘발성 메모리 상에서 AMQ 구조를 최적화하기 위한 기법 제안
  · Spillable Buckets: 기존 기법인 Cuckoo filter는 두 가지 해쉬를 사용해서 키 당 두 개의 버킷을 사용하게 함. Spillable Bucket을 이 두 가지 버킷에 더해, 두 버킷의 바로

수행기관	개발 목표	개발내용 및 범위
		다음 버킷들의 첫 번째 slot이 비어있으면 그곳을 이용할 수 있도록 함. 버킷들이 메모리 상에 연속된 위치에 존재하므로 splillable bucket에 대한 접근은 cache hit일 가능성이 높음 · Lookahead eviction: 각 버킷의 슬롯이 전부 할당되어 있는지를 1비트로 표시하는 occupancy flag 테이블을 둠. 그리고 insert 때 두 버킷이 모두 차 있어서 eviction이 필요한 경우, occupancy flag 테이블을 참고하여 추가적인 eviction이 일어나지 않을 버킷에 대해 우선적으로 eviction 수행 · Bucket Primary: 두 해쉬 함수에 대해 하나를 primary, 다른 하나를 secondary로 두고, 각 버킷이 overflow 되었는지를 slot 내의 fingerprint 값 순서로 표시하여, primary 해쉬 값에 해당하는 버킷이 overflow 되지 않았을 경우 secondary 해쉬 값에 해당하는 버킷에 대한 lookup을 수행하지 않음 - 실험 결과 · 제안된 기법을 적용한 AniFilter와 최신 기술을 Cuckoo Filter를 비교. Insertion에 대해서는 평균 2.6배의 성능 향상을 보이며, Random Loockup에 대해서도 대부분의 load factor에 대해 더 좋은 성능을 보임  그림 30. 비휘발성 메모리 기반 Approximate Membership Query 최적화 기법 성능 평가
아주대학교 (공동1)	메모리 중심 컴퓨팅 시스템 기반 쓰레드 병렬화 기술 연구	• 스레드 이주 기술 연구 - 연구 개요 · 기존의 단일 기기 기반의 데이터처리 프로그래밍 패러다임은 노드의 코어 수에 맞춰 스레드를 만들어 task를 할당, 병렬화하는 형태임. 하지만 메모리 중심 컴퓨팅으로 패러다임의 전환에 따라 기기가 사용하는 메모리의 물리적인 위치의 제한이 없어지므로 프로세스와 스레드가 실행되는 물리적인 위치의 제한 또한 해결해야 할 필요성이 제기됨 · 기존의 Single-system Image (SSI) 시스템 연구들은 다수의 노드를 논리적으로 하나의 시스템으로 묶어 사용할 수 있는 기술을 제공하지만 대부분 프로세스 수준에서 시작 노드의 조절이나 프로세스 전체의 재배치 기능만 제공하여 프로세스보다 작은 스레드 수준에서의 병렬화는 불가능한 한계가 있음 · 병렬 프로그래밍 패러다임은 응용의 확장성 있는 실행을 보장하는 큰 장점이 있으나 단일 노드를 기반으로 작성된 코드를 프레임웍이 제공하는 API와 기능을 통해 전체적으로 새롭게 다시 작성해야 하는 부담이 크게 작용함 · 본 연구에서는 기존에 작성된 응용의 수정을 최소화하며 다수의 노드에 스레드를 자유롭게 이주, 병렬화시킬 수 있는 리눅스 기반의 시스템인 DEX (Distributed thread EXecution) 기술을 구현하였음 - 연구 개발 내용

수행기관	개발 목표	개발내용 및 범위

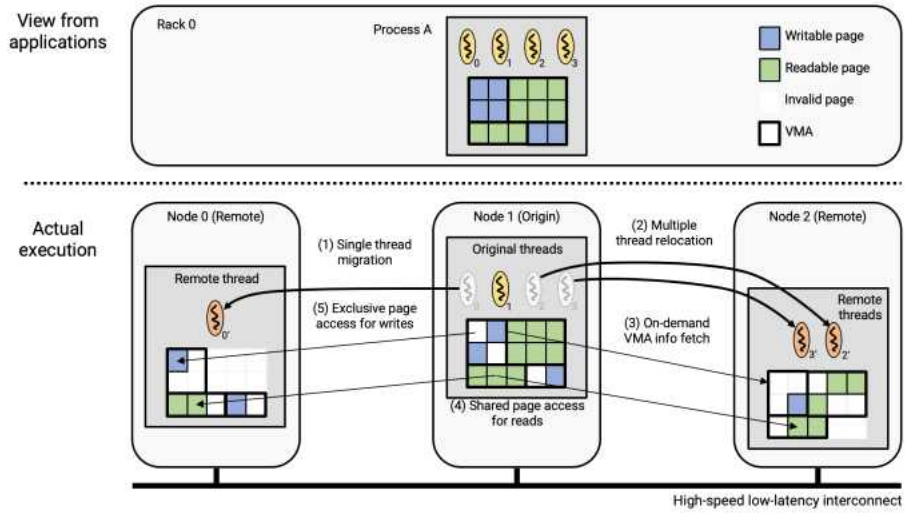
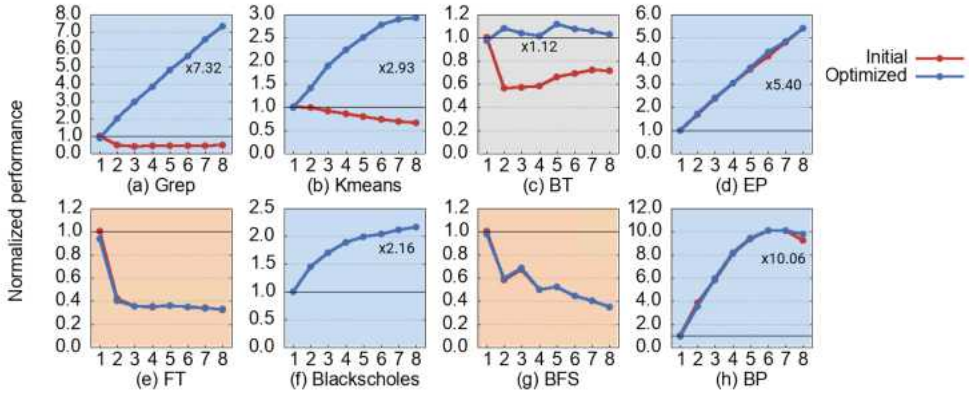


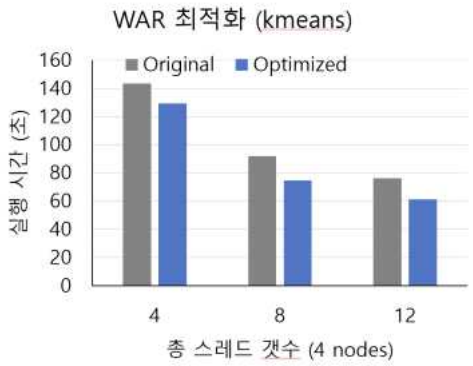
그림 31. DEX 시스템 개요도

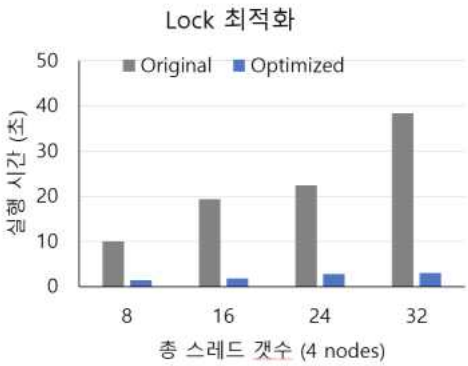
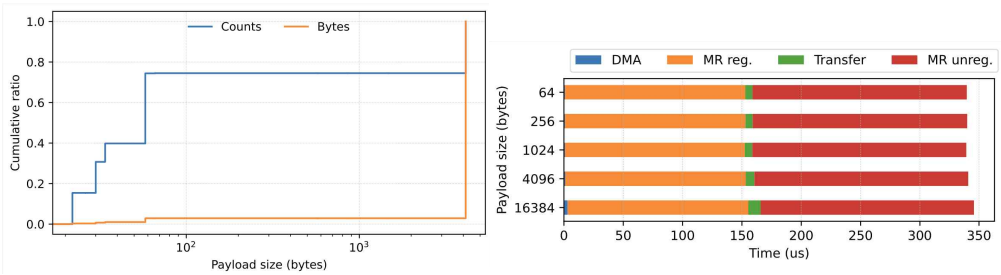
· DEX는 기존의 단일 노드 기반으로 작성된 프로그램을 그대로 사용할 수 있으며, 같은 프로세스에 속한 스레드이더라도 서로 다른 노드로 분산, 이주(migration)하여 실행하는 것을 지원함 ((1) 및 (2) 기능)
· 이러한 스레드 이주 기능은 운영체제의 context switching 기능을 확장하여 구현하였음. 즉, 다른 노드로 이주하려는 스레드의 실행을 정지시킨 후 해당 스레드의 context를 수집하고 이를 원격 노드에 전송, 동일한 context를 가지는 스레드를 만드는 방식으로 스레드를 복제하였음
· 원래 노드에서 동작 중인 스레드는 실행을 중지하고 원격 노드의 스레드가 실행을 대신함. 하지만 스레드들끼리 동기화를 하거나 VMA를 변경하는 등 프로세스 수준에서의 처리가 필요한 경우, 요청을 원래 노드로 요청을 포워딩하여 처리하는 work delegation 구조를 통해 처리함
· 한 프로세스에 속한 스레드는 모두 동일한 메모리 데이터를 공유해야 함. DEX 에서는 스레드들이 서로 다른 노드에서 분산 실행되는데, 이들 간의 메모리 정합성을 보장하기 위해 페이지 수준에서 동작하는 read-replicate write-invalidate 프로토콜을 가상 메모리 시스템에 구현

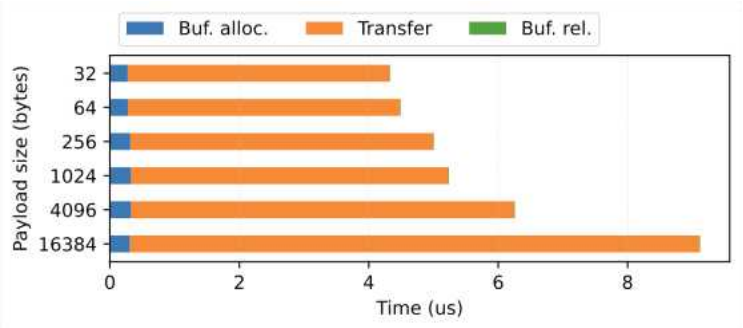
· 프로세스의 주소 공간은 페이지 단위로 소유권(ownership)을 다르게 가질 수 있음. 프로그램이 최초 실행되는 노드를 origin 노드라고 할 때, 최초에는 origin 노드가 모든 페이지의 소유권을 가지고 있음
· 원격 노드에서 읽기를 위해 페이지가 필요한 경우 해당 페이지를 가지고 있는 노드에 접근, 복제해서 가져가며 소유권을 공유하게 됨. 원격 노드에서 쓰기를 위해 접근하는 경우 다른 노드의 소유권은 모두 폐지(revoke)하여 독점적인 소유권을 확보한 후 쓰기를 처리하여 데이터의 정합성을 보장함
· 이러한 과정을 통해 각 페이지 별로 ownership이 조정되며 노드 간 페이지 데이터가 공유 및 분산되게 됨

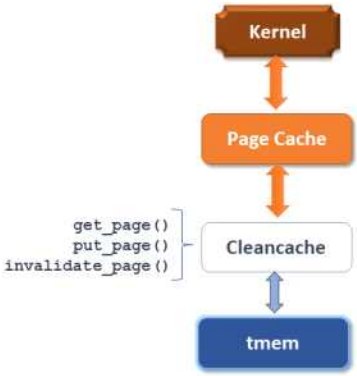
- 성능 평가
· 기존의 단일 노드 기반으로 작성된 대표적인 데이터 처리 응용 8종을 DEX를 통해 여러 노드에서 실행되도록 개선하였음
· 프로그램을 변경하는데 몇 줄 수정 수준으로 병렬화가 가능하였으며, 이를 통해 8개의 노드에서 기존 단일 노드에서의 동작 성능 대비 최대 10.1배의 성능을 달성하였음. 이를 통해 DEX가 제공하는 기능으로 기존에 작성된 응용의 성능을 획기적으로 개선할 수 있음을 검증하였음

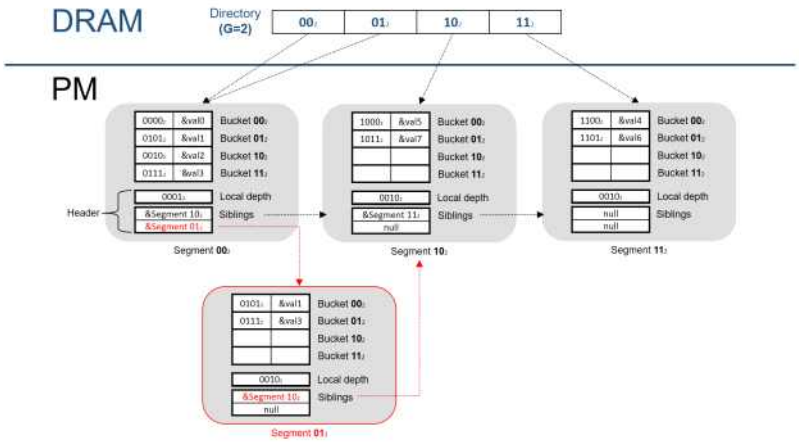
수행기관	개발 목표	개발내용 및 범위
		· BT, FT, BFS 등 병렬화 및 성능 개선이 잘 이루어지지 않는 일부 응용이 있음을 확인하였고, 이들 응용과 성능이 잘 개선되는 응용의 차이를 파악함으로써 추가적인 성능 개선 연구를 수행하고자 함  그림 32. DEX를 활용한 분산 병렬 프로그램 성능 평가(1~8노드) • 분산 스레드 간 메모리 정합성 보장 기술 연구 - 연구 개요 · 앞서 설명한 DEX 시스템은 대규모의 데이터를 여러 노드가 나누어 처리하는 형태의 클라우드 수준의 데이터 프로세싱을 고려해서 읽기가 많은 워크로드에 적합한 read-replicate write-invalidate 메모리 정합성 프로토콜을 기본으로 구현하였음. 하지만 일부 응용에서 계산 중간 결과 저장을 위해 많은 쓰기가 유발되고, 이로 인해 응용의 성능이 많이 저하되는 것을 발견 · DEX를 통해 성능이 잘 개선되지 않는 응용의 경우 공유 데이터에 대한 contention이 심하게 발생하며 성능이 저하되는 것을 확인하였음. 이러한 응용은 여러 노드에서 연산에 필요한 데이터가 동시에 업데이트되며 정합성을 맞추기 위해 잦은 데이터 전송이 발생하는 것이 성능 저하의 근본적인 원인으로 파악됨 · 프로그램의 수정을 통해 응용이 발생시키는 메모리 참조 패턴을 바꾸어 성능을 크게 개선할 수 있을 것으로 기대되지만, 이 경우 기존에 작성된 프로그램의 대대적인 수정이 필요하게 되며, 이는 본 연구가 지향하는 바와 맞지 않음 · 따라서 응용의 메모리 접근 패턴을 직접 변경하지는 않으면서 쓰기로 인해 발생하는 많은 contention을 효과적으로 제어할 수 있는 메모리 정합성 보장 기술의 개선이 요구됨 - Write-after-Read (WAR) 연구 개발 내용 · 성능 저하가 발견된 응용 들의 쓰기 접근의 특이성을 파악함. 즉 자주 업데이트되는 데이터의 경우 단순히 데이터를 쓰는 동작이 주를 이루기 보다는 기존의 값을 먼저 읽은 이후 여기에 적절한 연산을 적용하여 쓰는 형태의 동작이 많이 관찰됨 · 이러한 메모리 접근 패턴에서는 기존 값을 읽는 과정에서 페이지가 복제되고, 이후 값을 업데이트하는 과정에서 복제가 제거되는, 두 번의 메모리 정합성 프로토콜의 관여가 일어나게 됨 · 기존의 메모리 정합성 프로토콜에서 복제가 일어난 이후 독점적인 소유권을 보장하기 위해서는 해당 노드를 가지고 있는 모든 노드에게 페이지 맵핑 변경을 요청해야 하며, 각 노드에서는 페이지 맵핑 변경 이후 TLB shootdown을 발생하며 오버헤드가 크게 발생함 · 이러한 문제를 해결하기 위해 읽기 이후 짧은 시간 내에 쓰기가 발생하는 페이지를 탐지하고, 이러한 페이지는 읽기 시점에 페이지의 독점적인 소유권을 미리 보장함으로써 데이터 정합성 보장을 위해 발생하는 오버헤드를 줄이고자 함 · 이를 구현하기 위해 origin 노드에서 원격 메모리 접근을 유발하는 instruction의 주소와

수행기관	개발 목표	개발내용 및 범위
		접근하는 페이지의 주소를 수집함. DEX 시스템에서 수 십~수 백의 스레드가 동시에 원격 메모리 접근을 요청할 수 있으므로 이를 효과적으로 수집하기 위해 해시 기반의 인덱스를 구축, 관리하였음. 해당 인덱스를 통해 읽기를 유발한 이후 1 jiffie 이내에 같은 페이지에 쓰기를 발생시키는 원격 노드를 트래킹하여 처리하였음 · 이러한 Write-After-Read (WAR) 접근 최적화를 통해 기존 응용 대비 최대 25%의 성능을 개선할 수 있음을 확인하였음. 특히 스레드가 노드 당 스레드가 많아져서 contention이 심할수록 제안하는 방법을 통해 성능 개선 효과가 커지는 것을 확인하였음  그림 33. Write-After-Read 최적화 성능 평가 - 분산 스레드 환경에서 동기화 매커니즘 최적화 연구 개발 내용 · 스레드 간 동기화(synchronization)을 위해 다양한 메커니즘들이 존재하는데, 근래의 멀티스레드 런타임들은 본질적으로 futex (Fast User-level Mutex)를 통해 운영체제의 도움을 받으며 구현되어 있음. · Futex는 특정 메모리 word의 값에 따라 atomic하게 프로세스를 blocking할 수 있는 매커니즘임. 동기화가 필요한 스레드들은 해당 word에 경쟁적으로 접근하게 됨. 따라서 단일 노드 실행 환경에서 스레드 간의 동기화를 위해서는 해당 word에 대한 간단한 연산으로 동기화가 구현되고 관리됨 · DEX를 통해 스레드들이 서로 다른 노드에서 실행하게 됨에 따라 스레드 간 동기화를 위해 해당 futex의 값을 저장하는 페이지가 매우 빈번하게 접근하게 되고 이로 인해 매우 많은 메모리 정합성 프로토콜을 위한 데이터 전송을 유발함. 이는 특히 동기화로 보호해야 하는 구간이 짧을수록 큰 오버헤드를 발생하게 됨 · 이를 개선하기 위해 동기화 자료구조를 분산 스레드 환경에 적합하도록 재구성하고 MCS lock 형태로 구현을 변경하여 동기화 성능을 개선함 · MCS lock은 contending되는 경우 공유되는 데이터에 대한 한 번의 메모리 접근과 스레드마다 가지고 있는 변수로 계층화되어 접근하게 됨. 따라서 공유 데이터로의 접근 횟수가 bound되어 높은 contending과 동기화 오버헤드가 발생하는 분산환경에서의 동기화 객체를 구현하기에 적합한 것으로 판단됨 · 이에 따라 런타임에서 기존의 futex기반으로 구현된 spinlock을 MCS lock 형태로 구현하였고, 변경된 동기화 객체를 사용하여 응용의 동기화 성능을 6배 이상 향상시킴

수행기관	개발 목표	개발내용 및 범위
		 그림 34. Lock 최적화 성능 평가 • RDMA 기반 고속 통신 체계 연구 - 연구 개요 · 메모리 중심 컴퓨팅과 분리형 메모리 시스템 구현을 실현하기 위해서는 운영체제 수준에서 지원하는 고성능 통신 계층이 필수적임. 이를 위해 기존의 Ethernet과 TCP/IP 기반으로 작성된 통신 계층을 InfiniBand와 RDMA를 통해 개선하고자 하는 연구가 활발하게 진행되고 있음 · RDMA를 사용하여 데이터에 접근할 경우 원격 노드의 CPU의 영향을 받지 않고 빠른 접근이 가능한 장점이 있지만 RDMA를 통해 데이터를 전송하기 위해서는 DMA를 위한 메모리 영역을 확보, 관리해야 하고, 이를 위해 많은 시간이 소모됨 · 또한 control-path의 메시지들은 처리의 시급성은 높으나 그 크기가 작고 대부분 처리 결과를 동기화하여 반환하는 구조로 인해 RDMA를 통한 처리가 적합하지 않음 · 이에 DEX를 구현하는데 사용된 전송 계층을 분석하여 메모리 중심 컴퓨팅 시스템에서 자주 발생하는 메시지 전송 특성을 파악하고 이를 바탕으로 통신 계층을 최적화함 - 연구 개발 내용  그림 35. 분산 응용체제에서 통신 계층의 메시지 특성 분석 · 왼쪽 그림은 특정 응용을 DEX 상에서 실행하면서 발생한 메시지의 크기에 대한 분포의 분석 결과로, 전체 전송되는 메시지 중 개수로 약 76%는 60 바이트 수준으로 매우 작음. 하지만 전송되는 데이터 양의 97% 이상은 4 KB 페이지 전송으로 발생. 이는 분산형 운영체제의 통신 계층은 매우 많은 수의 작은 메시지와 큰 메시지 전송에 최적화할 필요가 있음을 나타냄 · 오른쪽 그림은 전송하려는 데이터 크기에 따라 처리에 필요한 시간의 분석 결과임. 분산형 운영체제가 자주 사용하는 메시지 크기와 분포 규모에서 실제 데이터 전송에 소모되는 시간(Transfer)은 매우 작으며, 대부분 전송 버퍼를 준비, 관리(MR reg, MR unreg)하는데 소비되는 것을 확인 · 이러한 결과를 바탕으로 전송 계층의 버퍼를 미리 할당하여 재사용하고, 또한 메시지 전송 완료와 연관된 path의 최적화를 수행하였음

수행기관	개발 목표	개발내용 및 범위
		· 작은 크기의 컨트롤 메시지는 원격 노드에서 요청을 처리하고 그 결과를 반환해야 하는 형태임. 따라서 한쪽 노드에서 직접 데이터를 쓰거나 읽는 RDMA를 통한 처리로는 성능상 이득을 보기 힘듬. 따라서 컨트롤 메시지는 기존의 RPC 형태를 통해 처리함 · InfiniBand에서 데이터 전송은 DMA가 가능한 메모리 영역에서만 이루어지기 때문에 데이터를 전송하기 위해서는 디바이스의 DMA 영역에 해당 메모리 영역을 등록, 해제해야 함. 하지만 작은 크기의 데이터에 대해서는 데이터 복사가 맵핑 관리보다 더 높은 효율이 보이며, 이를 활용하여 시스템 초기화 과정에서 메모리의 일부를 DMA가 가능하도록 미리 맵핑시키고, 전송하는 데이터의 크기가 작은 경우에는 이 전송 영역에 데이터를 복사하여 전송 메시지를 구성하는 형태로 개선함. · InfiniBand는 메시지 전송 완료 시점에 completion queue를 통해 이벤트를 전달하는 형태로 통신이 이루어짐. 따라서 메시지 전송 완료 시점을 빨리 파악하는 것은 컨트롤 메시지의 처리 성능에 크게 영향을 줌. 이를 위해 여러 스레드가 동시에 메시지를 전송하는 상황에서 completion 메시지를 스레드끼리 협력하여 공통으로 처리하는 형태로 개선함 · 이러한 개선을 통해 메시지 전송 시 버퍼 관리로 인한 소모 시간을 최소화시킴  그림 36. 개선된 전송 계층에서의 메시지 전송 시간 분석
성균관대 학교 (위탁1)	비휘발성 메모리 기반 데이터 구조 연구	**• 비휘발성 메모리 기반 원격 파일 시스템 캐싱 시스템의 개념** - 개념: 비휘발성 메모리와 고속 네트워크를 이용하여 Disaggregation된 노드의 대용량 메모리를 원격 파일 시스템 캐시로 활용할 수 있는 시스템 소프트웨어 - 효과: 파일 시스템 캐시에서 교체되는 페이지들을 원격 캐시 서버로 보내는 커널 모듈을 구현함으로써, 사용자 응용 프로그램의 코드 수정 없이 메모리 집약적인 응용들이 기존 클러스터에서 원격 노드의 대용량 메모리를 활용할 수 있게 되고 수행 시간이 단축될 수 있게 함 **• 비휘발성 메모리 기반 원격 파일 시스템 캐싱 시스템 디자인** - RDMA 네트워크로 연결된 원격 서버의 비휘발성 메모리(Persistent Memory)를 로컬 서버의 디스크 캐시로 이용하는 시스템인 PMDFC (Persistent Memory Disaggregation File Cache)를 디자인하고 구현함 - PMDFC는 용량이 작은 메모리를 가진 서버들이 원격 PM 서버의 대용량 비휘발성 메모리를 자신의 메모리처럼 사용할 수 있게 함 - PMDFC는 CleanCache 모듈을 원격 RDMA 통신으로 확장한 것으로, CleanCache 는 DRAM 용량이 부족할 때 Base Page 중 수정되지 않은 Clean Page 들을 버리는 대신, 다른 저장소에 보관할 수 있게 하는 기능을 제공함 - 리눅스 커널의 page frame replacement algorithm (PFRA)에 의해 퇴출되는 페이지들이 퇴출되기 전에 cleancache에 의해 다른 저장 장치로 저장하는 것을 시도할 수 있다. 즉, Page Cache 내의 clean 페이지가 퇴출될 때, Clean Cache의 put_page() 함수를 불러 Transcendnet memory에 저장한다. 이후 page fault가 발생할 때, Transcendent memory에서 get_page()함수를 호출하여 clean 페이지를 읽어오게 하며, 이는 메모리 과부하시 느린 저장장치로의 접근을 줄여주는 역할을 함

수행기관	개발 목표	개발내용 및 범위
		 그림 37. Clean Cache 구조 **• 비휘발성 메모리 기반 원격 파일 시스템 캐싱 시스템 클라이언트 구현** - PMDFC의 클라이언트는 메모리가 부족하여 백엔드 원격 PM 서버로 페이지를 보내는 모듈 - 클라이언트는 페이지가 퇴출될 때는 cleancache_put_page()를 통하여 퇴출이 일어나는 페이지를 원격 PM 서버로 전송하고, page fault가 일어났을 때는 cleancache_get_page()를 통해서 해당페이지를 원격 PM 서버에서 찾아서 리턴함 - PUT 기능 구현 · 리눅스 커널은 메모리 할당에 실패하거나 메모리의 유휴 공간이 적어지면 kswapd를 통해서 메모리에서 페이지들을 퇴출시키는 작업을 수행함. 퇴출의 타겟이 된 페이지들은 shrink_page_list() 함수에 리스트 형태로 전달되고 Dirty page의 경우 디스크에 다시 쓰이고 clean page의 경우에는 remove_mapping() 함수를 호출해 메모리에서 삭제함. 이 remove_mapping() 함수 내부에서 cleancache_put_page() 함수가 해당 페이지에 대하여 불리게 되며 이 함수가 PMDFC의 put() 함수를 호출하여 해당 페이지를 원격 PM 서버로 보내는 작업을 함. 이 때 remove_mapping() 함수는 spinlock을 통해 보호되는 영역이기 때문에 cleancache_put_page() 내부에서 sleep할 수 있는 함수를 호출할 수 없고 빠르게 반환해야 한다는 제약이 있음. 이를 해결하기 위해서 미리 할당해 놓은 공간에 페이지를 복사하고 바로 반환하도록 구현하였고 따라서 리눅스의 메모리 관리에 무시할 만한 오버헤드만을 부과함. 백그라운드 스레드는 복사된 페이지들을 네트워크를 통해 원격 PM 서버로 전송함 · 백그라운드 스레드에 의해 보내지기 전의 페이지들을 저장하기 위해서 미리 할당해 놓은 공간은 8개의 클러스터로 구성되어 있음. 하나의 클러스터는 4096개의 페이지를 저장할 수 있도록 4MB의 공간이 미리 할당되어 있음. PUT요청이 발생하면 8개의 클러스터 중 여유 공간이 있는 클러스터부터 차례대로 페이지가 채워지고, 클러스터가 가득차면 네트워크 스레드가 해당 클러스터를 원격 PM 서버로 보내는 작업을 수행한다. 동시에 다른 PUT요청이 발생하여도 비어있는 클러스터에 페이지를 복사하고 빠르게 반환할 수 있음. - GET 기능 구현 · 리눅스 메모리 접근에서 페이지 폴트가 발생하게 되면 요청된 페이지를 디스크에 찾아서 메모리에 불러옴. 리눅스 커널의 mpage_readahead() 함수는 읽어야 하는 페이지들을 모아서 Block I/O (bio)를 호출하는 함수임. 이 함수는 해당 페이지를 bio로 보내기 전에 cleancache_get_page() 함수를 호출하여 PMDFC 모듈의 get() 함수를 통해 원격 PM 서버에 저장되어 있던 페이지를 가져옴 · GET은 메모리에 존재하지 않는 모든 페이지에 대하여 발생하기 때문에 원격 PM 서버에 저장되지 않은 수많은 페이지에 대하여 통신이 발생하는 문제가 발생함. 이를 해결하기위해서 확률적 필터인 블룸 필터를 도입하였음. 메모리가 부족한 상황에서 사용되는 모듈이기 때문에 메모리의 사용량을 최소로 하기 위하여 모든 페이지가 필터링 되는 것을 포기하고 블룸 필터를 사용하였음. 이를 통해 백엔드에 저장되지 않은 페이지들을 클라이언트에서 판단하여 빠르게 반환할 수 있도록 하였으나, 블룸필터는 삭제를 지원하지 않기 때문에 추후에 카운팅을 적용하여 삭제를 지원하는 Counting

수행기관	개발 목표	개발내용 및 범위
		Quotient Filter 로 변경할 계획임. · 클라이언트가 서버로부터 페이지를 정상적으로 전달 받았다면, 이를 활용하여 데이터를 처리함. 만약 요청한 페이지가 서버에 존재하지 않다면, 기존과 같은 방식으로 디스크로부터 페이지를 읽어 처리함. **• 비휘발성 메모리 기반 원격 파일 시스템 캐싱 시스템 서버 구현** - PMDFC의 서버는 클라이언트로부터 요청을 받아서 비휘발성 메모리에 페이지를 저장하고 반환하는 서버를 의미함. PMDFS 서버는 가능한 모든 컴퓨팅 자원을 활용하여 Key-Value 연산을 빠르게 수행해야 함. 따라서, PMDFS 서버는 멀티스레딩을 통해 다수의 클라이언트를 지원하게 설계하였음 - NUMA-aware Balanced Cacheline-Conscious Extendible Hashing · Optane DIMM은 DRAM과 마찬가지로 각 CPU에 인접한 메모리 슬롯에 장착되어 원격 노드 접근에 대한 지연 차이가 발생하는데, 원격 노드 접근에 대한 지연은 DRAM보다 PM에서 더 크게 발생함 · 본 연구에서 개발한 NUMA-balacned CCEH는 Segment Split 시에 노드의 부하를 파악하여 부하가 적은 쪽 노드에 새로운 Segment를 할당함  그림 38. DRAM+PM Hybrid CCEH · NUMA-aware Balanced CCEH는 위 그림과 같이 디렉토리를 PM이 아닌 DRAM에 위치시켜 hybrid 자료구조로 변형시킴. 본 시스템은 각 NUMA 노드 별로 서로 다른 PM pool을 따로 가지게 하였으며, 디렉토리는 16-byte Persistent 포인터를 갖기 때문에 아래 식과 같이 디렉토리 lookup 만으로 타겟 세그먼트가 어느 노드의 PM pool에 위치해 있는지 판별할 수 있음 CONST addr = D_RO(dir->segment[x]) - dir->segment[x].oid.off; CONST nodeId = NODEID WHERE addr == PMEMOBJPOOL[NODEID] · 판별 후 적절한 노드의 스레드에 수행해야 하는 연산을 전달함으로써 실제 PM에는 로컬 접근을 수행하도록 함. 디렉토리 lookup을 할 때의 원격 접근은 피할 수 없겠지만 이를 상대적으로 지연이 낮은 DRAM에서 수행함으로써 NUMA 효과를 최소화할 수 있음 - NUMA-aware Key-Value Store · NUMA-aware Balanced CCEH를 사용하여, 비휘발성 메모리의 원격 접근의수를 줄이고 같은 노드 접근의 수를 늘리기 위해 NUMA aware KV store를 구현하였음.

수행기관	개발 목표	개발내용 및 범위

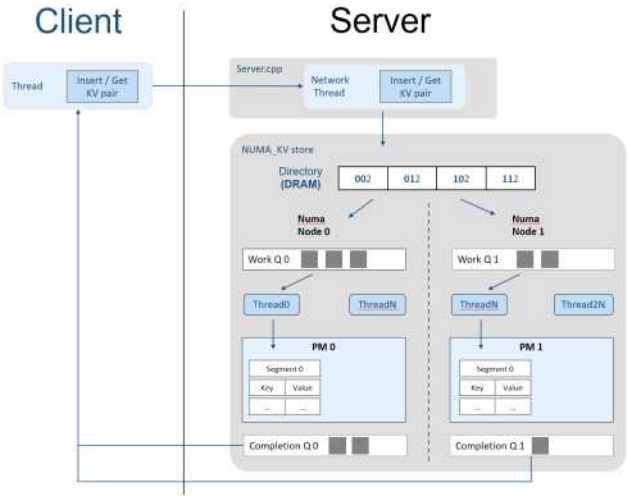


그림 39. NUMA-aware Key-Value Store

· KV store는 비휘발성 메모리에 직접 접근하는 CCEH위에서 동작하여 DRAM에서 Directory를 먼저 탐색하여 Node ID에 해당하는 스레드에게 일을 할당하는 역할을 함
· 먼저 클라이언트로부터 넘어온 쿼리들은 응용수준에서 개발한 서버의 네트워크 스레드가 처리함. 네트워크 스레드는 이 쿼리들의 key값과 식1을 이용해 해당 쿼리가 처리되어야하는 노드 ID를 구함. Directory는 Hybrid CCEH 방식에 의해 DRAM에 상주하기 때문에 이를 통해 노드 ID를 구하는 과정은 원격 접근이 일어나더라도 성능에 큰 피해를 주지 않음. 이 정보를 활용하여 네트워크 스레드는 쿼리를 누마 노드별로 존재하는 원형 큐(작업 큐)에 배분함. 이를 처리하는 소비자 스레드는 KV store가 초기화될때 생성되는데 각 스레드는 자신이 속한 누마 노드에 바인딩되어서 동작함. 소비자 스레드들은 자신의 노드의 작업 큐를 참조하여 쿼리들을 실제로 처리하기 때문에 비휘발성 메모리로의 모든 접근은 지역접근이 됨.
· GET의 경우에는 GET 쿼리의 반환값을 호출자에게 반환하여야 함. 하지만 이를 과정까지 네트워크 스레드가 처리하게 되면 네트워크가 바틀넥이 되므로 이를 해결하기 위해서 완료 큐를 두고, 전송 스레드가 이를 반환하는 형태로 구현하였음.
· 모든 큐는 락프리 큐를 사용하였음. 락프리 큐는 많은 스레드가 락 없이 동시에 데이터를 추가 혹은 제거할 수 있는 자료구조임. 이는 클라이언트들로부터 많은 쿼리가 들어오더라도 여러 스레드들이 동시에 작업 큐로부터 쿼리를 받고 처리할 수 있도록 함.

• 비휘발성 메모리 기반 원격 파일 시스템 캐싱 시스템 성능 평가

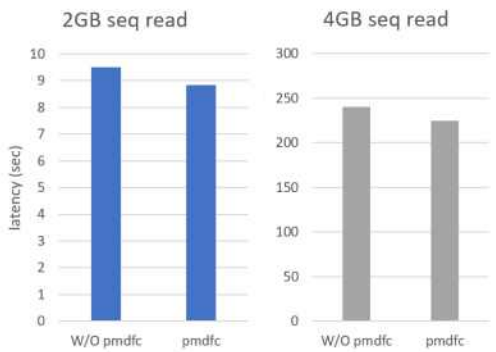
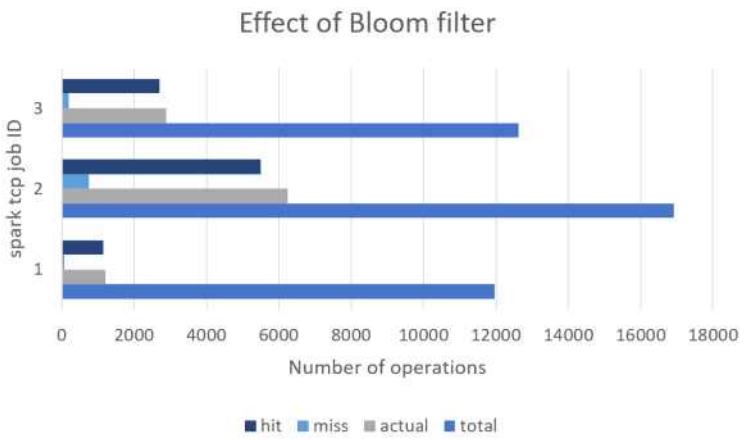
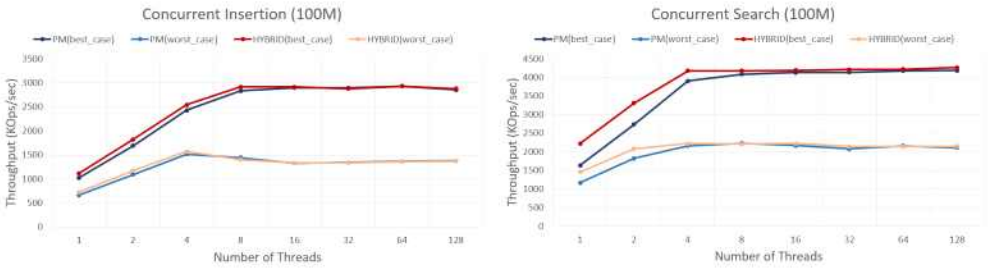
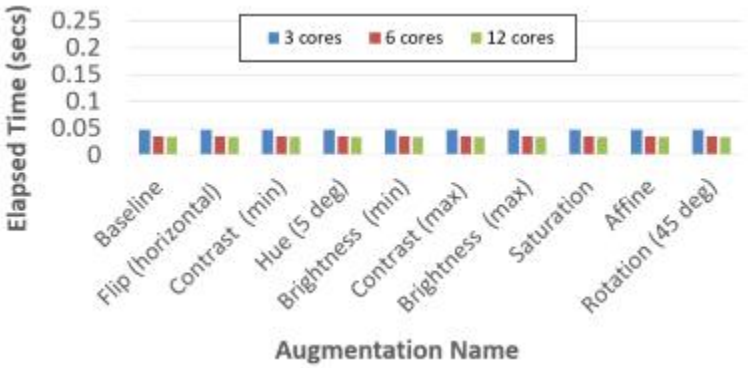
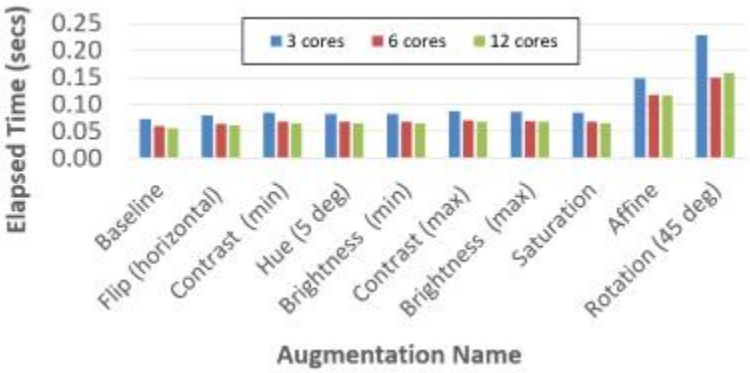
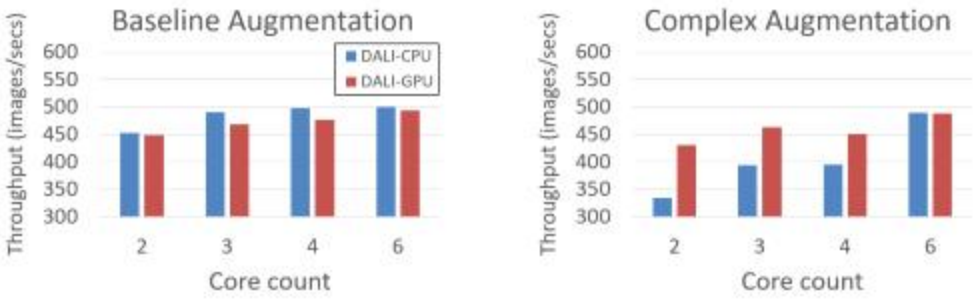
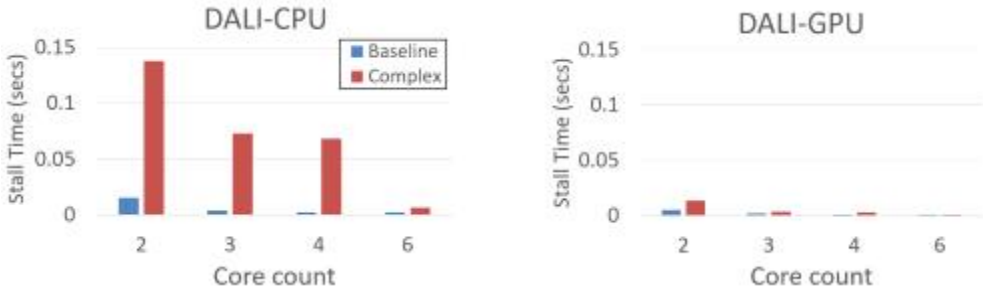


그림 40. PMDFC를 활용한 캐시 교체 성능 향상

- 위 그림의 실험에서는 PMDFC 모듈을 삽입하고 file read를 2G, 4G 파일에 대하여 진행하였음. 커널 패러미터 조정으로 메모리사이즈는 8G로 고정함. RDMA 통신은 버그가 있어 RDMA가 아닌 TCP/IP를 이용함. 또한, 안정성이 떨어져 작은 파일을 읽는 것으로 실험을 진행하여 GET 오퍼레이션이 1000번 미만으로 호출되어 성능 차이가 크게

수행기관	개발 목표	개발내용 및 범위
		나타나지는 않았으나 유의미한 성능 향상을 얻을 수 있었음  그림 41. Bloom Filter의 효과 실험 - 위 그림의 실험은 Bloom filter가 통신의 수를 줄여주는가에 대한 실험임. 해당 그래프는 PMDFC 모듈을 삽입하고 spark tpc benchmark job 1, 2, 3번을 수행한 결과임. 블룸 필터의 bits 수는 10000000, 해시 함수는 md5, sha1, sha224 3개를 사용하였음. 아래의 Total은 cleancache에 의해 불린 GET 명령의 수이고, actual은 그 중 실제로 서버로 전송된 GET의 수임. hit은 actual 중 서버에 저장되어 있던 페이지의 수이고 miss는 나머지임. Miss의 관해서는 불필요한 서버와의 통신비용을 가지지만 대부분의 저장되어 있지 않은 페이지는 바로 반환할 수 있음.  그림 42. Concurrent Search 성능 - 위 그래프는 CCEH Hybrid의 효과와 비휘발성 메모리의 NUMA에 따른 성능 차이를 나타낸 그래프임. 그래프에서 PM은 디렉토리와 세그먼트 모두 PM에 저장하는 방식이고 Hybrid는 디렉토리를 DRAM에 저장하고 복구 가능하도록 디자인한 방식임. Best case는 모든 오퍼레이션이 로컬 접근으로 이루어지는 경우이고, Worst case는 모든 접근이 원격으로 이루어지는 경우임. 위 그래프에서 best와 worst의 차이로 비휘발성 메모리의 원격 접근의 비용이 매우 크다는 것을 알 수 있음. Hybrid의 처리량차이가 미세한 이유는 DRAM과 PM의 속도 차이로 인해 얻을 수 있는 이점이 PM방식에서의 캐싱에 의해 가려지기 때문이라고 생각됨
UNIST (위탁2)	메모리 중심 컴퓨팅 시스템을 위한 가속기 기술 선행 연구	**• CPU와 가속기(GPU) 사용에 따른 인공지능 데이터 증강(Augmentation) 성능 분석** - 인공지능 학습에서 학습 데이터 로딩은 CPU를 통해 주로 수행되며, CPU 자원이 많을수록 빠른 학습 데이터 로딩이 가능해짐. 하지만 GPU 성능이 증가하면서 CPU 자원만으로 학습 데이터 로딩을 수행할 경우 병목 현상이 생김 - 특히 데이터 증강(Augmentation) 기법이 널리 사용되면서 학습 데이터 로딩 과정에 필요한 연산량이 더욱 많아지고 있으며, 이에 따라 CPU에 의한 병목 현상은 더욱 두드러짐 - NVIDIA에서 데이터 증강 기법에 GPU를 적용하는 플랫폼을 발표하였으며, 본 연구에서는

수행기관	개발 목표	개발내용 및 범위
		이를 이용해 데이터 증강 기법에 GPU를 활용할 경우 성능 분석을 수행 · 코어 개수 및 데이터 증강 기법의 복잡도에 따라 CPU와 GPU 데이터 로딩의 차이 파악 · 데이터 증강 기법에 따른 처리량 및 처리 시간 확인 - 분석에 사용한 데이터 증강 기법(Baseline Augmentation) · 기본 데이터 증강 기법(Baseline Augmentation): ResNet 모델의 데이터 증강 기법들 · 복잡한 데이터 증강 기법(Complex Augmentation): 기본 데이터 증강 기법에 Saturation, Horizontal flip, Rotate, Contrast, Horizontal flips를 추가 · 대중적인 인공지능 플랫폼인 PyTorch에서 기본으로 주어지는 개별 데이터 증강 기법이 성능에 어떠한 영향을 주는지도 분석 - 실험 환경 · CPU: 24코어 Intel Xeon Gold 6226 · 메모리: 128GB · GPU: GeForce 2080 Ti 11GB RAM · 운영체제: Ubuntu 16.04 · 인공지능 플랫폼: PyTorch 0.4, DALI 0.20 (Docker 환경) - 실험 내용: 학습에서 데이터 로딩 파이프라인만 활성화한 채로 CPU 코어를 3, 6, 그리고 12개로 제한하여, CPU 코어에 따른 기본 데이터 증강 기법별 처리량을 비교 분석  그림 43. GPU에서 데이터 증강 기법별 처리 시간  그림 44. CPU에서 데이터 증강 기법별 처리 시간 - CPU 및 GPU 상에서 뎅터 증강 기법별 처리 시간 분석 · 위 그림에서 X축은 선행 연구에서 선별한 데이터 증강 기법들이며, Y축은 처리 시간. CPU 3코어를 기준으로 데이터 증강 기법별 처리 시간을 우측으로 갈수록 오래 걸린 순으로 정렬 · 코어별 데이터 로딩 기법에서 CPU 6코어일 때 CPU와 GPU 모두 감소하였고, 그 이후 12코어에서는 약 0.001초 차이. 행렬 연산이 대부분인 아핀(Affine)과 회전(Rotation) 기법을 적용했을 때는 CPU에서 가장 비효율적. GPU 상의 성능을 보면 행렬 연산에 최적화되어있는 GPU에서는 거의 모든 기본 데이터 증강 기법에 대한 처리 시간이 유사.

수행기관	개발 목표	개발내용 및 범위
		Baseline의 경우, GPU 활용 가능한 자원이 충분한 상황에서는 GPU 데이터 로딩 시간이 CPU 데이터 로딩 시간보다 오래 걸림  그림 45. CPU와 GPU에서 데이터 증강 기법 복잡도별 처리량 (Throughput) 비교  그림 46. CPU와 GPU에서 데이터 증강 기법 복잡도별 지연 시간 (Stall time) 비교 - 기본 데이터 증강 기법과 복잡한 데이터 증강 기법 성능 분석 · 위 그림의 그래프 X축은 CPU 코어 개수이며, Y축은 각각 학습 데이터 처리량 학습 지연 시간(Stall time) · 기본 데이터 증강 기법을 사용할 때, 대부분의 CPU에서의 학습 지연 시간은 GPU와 비슷한 성능을 보임. 그러나 복잡한 데이터 증강 기법을 사용하면 CPU에서 지연 시간이 GPU보다 1.2배 정도 크게 나타남. 학습 데이터 처리량 또한 지연 시간과 같은 경향성을 보임. CPU 코어 개수가 증가한 경우에 이전과 마찬가지로 지연 시간이 줄었고 처리량이 증가 · 충분한 CPU 코어가 있다면 CPU도 데이터 증강 기법을 GPU와 비슷하게 처리할 수 있음을 알 수 있음. 그러나 데이터 증강 기법이 복잡하다면 GPU 데이터 로딩이 우위 **• SmartSSD: 데이터 증강 기법에 대한 가속기 기술 제안** - 현재 많이 사용되는 시스템의 경향을 보면, 최신 인공지능 학습 서버 중 하나인 NVIDIA DGX-2 기준으로 GPU당 CPU 코어 개수는 3개 정도이고, DGX-2 기준의 NVIDIA Tesla V100 GPU와 위 실험에서 사용한 NVIDIa Geforce 2080 Ti GPU 성능을 비교하면 TensorFlow FP16 학습 성능에서 1.81배 정도 성능 향상이 있음. 따라서 충분한 CPU 코어가 있다고 보기 어려우며, GPU 성능 발전에 따라 CPU의 성능 부족 현상은 더욱 심화될 것 - 데이터 증강 기법을 데이터가 위치한 저장장치 상의 가속기에서 처리한다면, CPU 자원이 부족한 상황에서도 최적화된 성능과 높은 대역폭을 활용하여 병목 현상이 해결할 수 있을 것으로 예상 **• 가속기에 포함할 데이터 증강 가속 데이터 증강 기법 선별** - Convolutional Neural Network (CNN)에서 주로 사용되는 증강 기법 분석 · TensorFlow 벤치마크에서 9개의 모델 중 5개의 모델이 Inception 모델 기반 데이터 로딩 파이프라인을, 2개의 모델이 VGG 모델 기반 데이터 로딩 파이프라인을 사용 · Inception 모델 기반 데이터 로딩 파이프라인은 정규화(Normalization)와 50% 좌우

수행기관	개발 목표	개발내용 및 범위

대칭(Horizontal flips) 및 무작위 색조 변경을 포함하고 있었으며, VGG도 비슷한 파이프라인을 가짐

· PyTorch는 Inception 기반 데이터 로딩 파이프라인을 벤치마크에 사용. VGG와 Inception 전처리는 내부적으로 동일

· 이밖에 최신 데이터 증강 기법은 CutOut을 기반으로 한 것이 3가지가 있었으며, 이외에도 모델의 견고함을 위한 데이터 증강 기법인 AugMix이 사용됨

Model	Preprocessing
Cifarnet	CifarNet preprocessing
Inception (v1 ~ v4)	Inception preprocessing
Inception ResNet v2	Inception preprocessing
LeNet	LeNet preprocessing
MobileNet (v1 ~ v3)	Inception preprocessing
NASNet	Inception preprocessing
PNASNet	Inception preprocessing
ResNet(v1~v2)	VGG preprocessing
VGG	VGG preprocessing

그림 47. 모델별 전처리 방법

- 9가지 데이터 증강 기법 선별 내용 및 근거
  · ① RandomResizedCrop, ② RandomHorizontalFlip, ③ Normalize: 기존 모델의 학습에 포함되어 있고 모델 및 플랫폼에 귀속되어 있으며 정확도에 필수적인 기법
  · ④ ColorJitter: 기존 모델의 논문에 포함되어 있는 기법
  · ⑤ RandomRotation: 2019년도 CVPR에서 발표된 AutoAugment 및 2020년도 CVPR에서 발표된 RandAugment 연구에서 ImageNet 데이터셋을 사용했을 때 가장 큰 효과를 본 기법
  · ⑥ Erasing, ⑦ RandomErasing, ⑧ RandomAffine, ⑨ AugMix: 최근 논문에서 제시된 데이터 확장 기법으로 모델의 견고성 및 정확도 향상에 도움이 됨

• 데이터 로딩 가속기 디자인 설계 및 구현

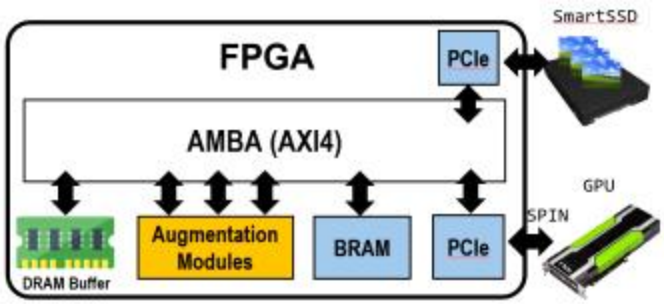
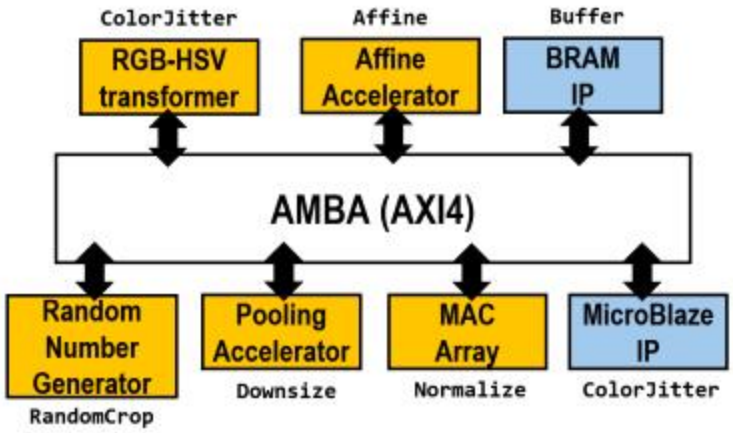
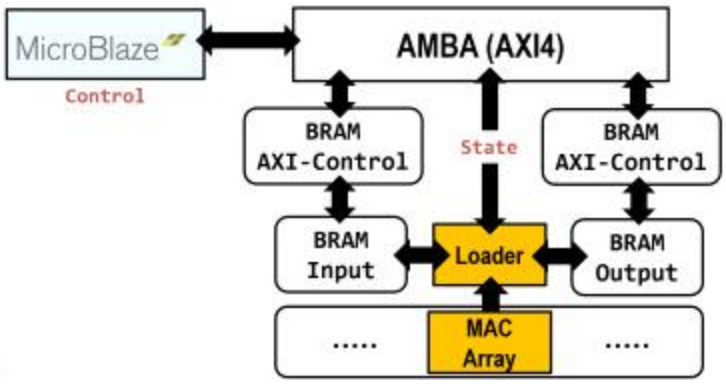
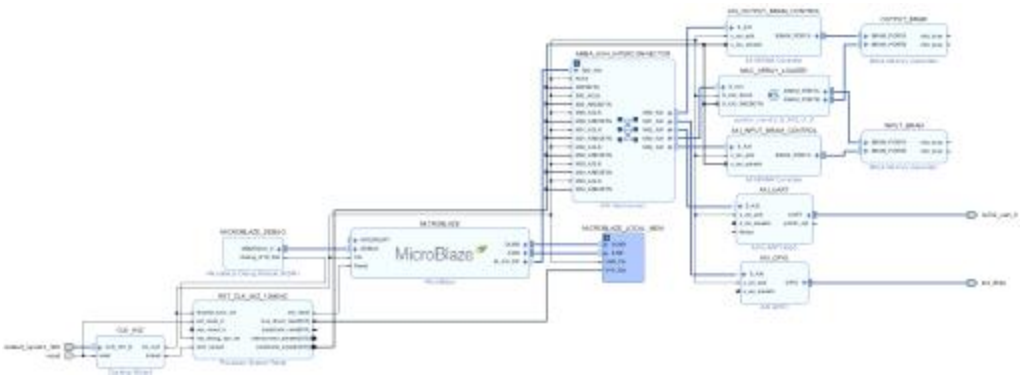
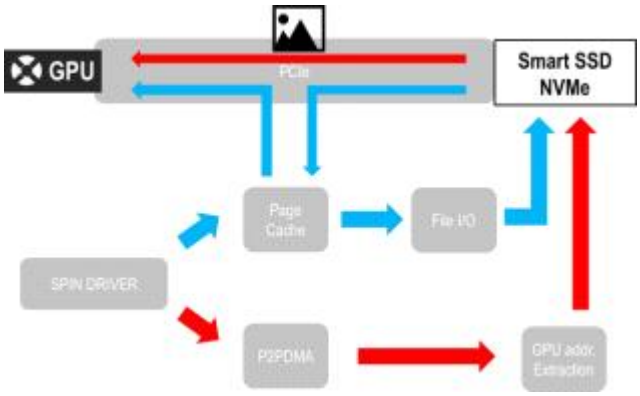
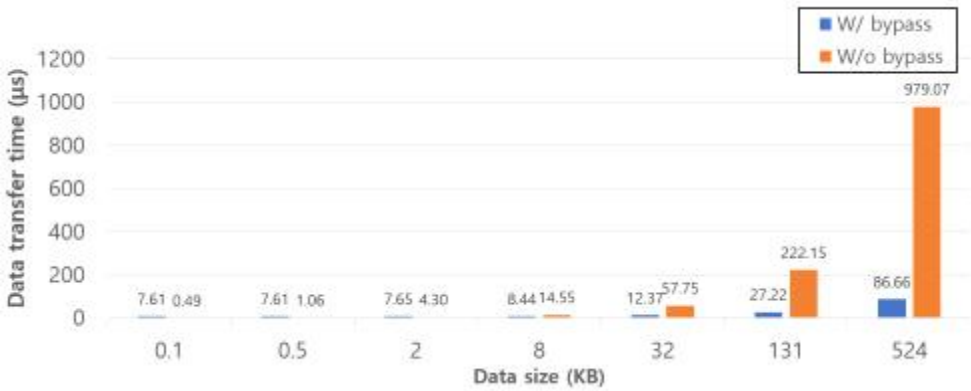


그림 48. 시스템 설계 개요

- 개념: 고성능SSD에 FPGA 보드가 포함되어 있는 SmartSSD을 기반으로 저장장치에서 데이터 증강 과정을 수행한 결과를 바로 GPU로 전송할 수 있도록 하는 데이터 로딩 가속기 디자인을 개발
- 개발 내용 요약
  · 데이터 로딩의 전처리 과정을 하드웨어 언어인 베릴로그(Verilog HDL)로 작성하였으며, 설계 일부를 FPGA 상에 구현
  · 데이터 로딩 파이프라인의 데이터 증강 기법 및 내부 함수를 개별적인 모듈로 구성하여 확장성(scalability)을 높임.

수행기관	개발 목표	개발내용 및 범위
		· 위에서 선별한 데이터 증강 기법 바탕으로 이를 처리해야할 모듈(module)들을 작성하였고, 작성된 모듈 간의 통신 프로토콜(protocol)과 데이터 동기화(synchronization)를 고려하여 CPU 및 GPU 기반 데이터 로딩보다 빠르게 동작할 수 있도록 설계 · 다양한 메모리 (DRAM, BRAM, 레지스터 메모리(Register Memory)) 구성은 사용자가 하드웨어 상에서 자유롭게 설정 가능하도록 하여, 읽기/쓰기 속도에 맞추어 최적화할 수 있도록 함 · 원활한 데이터 이동을 위해 하드웨어의 특성을 고려한 내부적인 데이터 송신 경로 개발  그림 49. 하드웨어 내부 데이터 증강 모듈 설계. - 데이터 증강 기법 모듈 개발 내용 · RGB-HSV 변환(transformer) 모듈: ColorJitter 내부의 여러 색조 변경 기능을 담당 · 아핀 가속장치(Affin Accelerator) 모듈: 아핀 기하학에 사용되는 회전(Rotation), 이미지 반전(Flip), 등의 행렬 연산을 수행 · 무작위 숫자 생성기(Random Number Generator) 모듈: 무작위 연산 무작위 잘라내기(RandomCrop) 및 무작위 이미지 반전(Random Flip) 등에 사용 · MAC array 모듈: 정규화(Normalize) 같은 커널(Kernel) 기반 행렬 곱 및 합을 수행 - 데이터 전송 및 저장 관련 모듈 개발 내용 · Advanced eXtensible Interface(AMBA) AXI4 버스 인터페이스: 모듈 간 데이터 전송을 담당하는 프로토콜. 병렬, 고성능, 동기식, 고주파수, 다중 마스터, 다중 슬레이브 통신 인터페이스를 지니고 있기에, 많은 모듈들이 다중 슬레이브와 마스터 구조를 통해 서로 통신 가능 · Microblaze: 다른 모듈과 함께 색조 변경을 처리하면서 슬레이브 마스터로써 모듈 간의 동기화 및 데이터 전송 부분을 담당 · BRAM 및 DRAM 모듈: 중간 데이터의 버퍼(Buffer)로 사용  그림 50. 모듈 내부 디자인 설계

수행기관	개발 목표	개발내용 및 범위
		- 데이터 로딩 가속기 구조 및 특징 · 위 그림은 데이터 로딩 가속기 내부 구조를 나타냄. BRAM을 외부뿐만 아니라 내부 및 AXI에 두어 저장하여 내부 연결에 병목현상이 없도록 설계. BRAM과 AXI를 연결하기 위한 컨트롤러가 존재하며, 내부에는 총 2개의 BRAM으로 이루어져 있는데 하나는 내부 입력을 저장하는 역할(Input BRAM) 다른 하나(Output BRAM)는 외부 입력을 저장하는 역할 · 각 모듈은 로더(Loader)를 통해 모듈과 BRAM 데이터 통신 그리고 모듈의 현재 상태(state)가 어떤지 MicroBlaze 모듈에게 알려주는 역할을 함. 모듈의 현재 상태는 유휴(idle)와 동작(busy) 2가지가 있음. · 가속기의 모듈들은 각 모듈의 현재 상태 기반으로 동작. 먼저, MicroBlaze에서 외부 입력(PCIe)를 통해 SSD에서 데이터를 전송받아 입력 BRAM에 입력. 모듈이 만일 유휴 상태라면 Loader는 입력 BRAM에서 읽기를 시작하며, 동작(Busy) 상태일 때는 읽기를 중단. 다시 유휴 상태로 전환되었을 때, 모듈이 유효한 출력인지 값을 내보내 주기에 출력 BRAM에 쓸 수 있음. 반대로 동작 상태라면, MicroBlaze는 모듈이 유휴 상태가 되기까지 기다리며 다음 데이터를 입력 BRAM에 입력. 이후 출력 데이터들은 외부 출력(PCIe)를 통해 학습을 수행하는 GPU로 전송됨. 입력 BRAM이 가득 차면 중단되며, 모듈에서 출력 데이터를 보유한 경우 출력 데이터의 같은 상대적 주소(Relative addresss)에 있는 입력 BRAM의 입력 데이터는 삭제되거나 다른 입력 데이터로 대체될 수 있음. 여러 가지 모듈의 진행 순서 또한 MicroBlaze에 의해 결정되며 호출됨. · 디코딩 중간 단계에서 중간 데이터를 또 다른 출력 BRAM으로 내보내어 재사용함으로써 불필요한 중복작업(redundancy)을 제거. - 아래 그림은 MAC Array 모듈과 함께 실제 구현한 디자인을 나타냄  그림 51. Vivado 2019.1을 통한 실제 구현 디자인 • **SmartSSD-GPU 데이터 송신 시스템 설계** - 개념: 데이터 로딩 가속기의 효율성 확보하기 위해 CPU 호스트(Host) 거치지 않고 바로 GPU로 데이터를 송신하는 기법이 필요. SSD와 GPU 간에 송신을 지원하는 선행연구인 SPIN(Seamless Operating System Integration of Peer-to-Peer DMA Between SSDs an GPUs)을 기반으로 해당 기법을 구현 - 개발 내용 · 기존 SPIN 라이브러리는 SmartSSD 드라이버와 호환되지 않기 때문에 Linux 4.4.120 버전에 맞추어서 재작성. GPU와 CPU 메모리로의 매핑(mapping)은 NVIDIA GDRCopy를 사용하였으며, 이를 SPIN 드라이버의 GPU 메모리의 커널 메모리 공간 매핑을 위한 API에 결합하여 선행 연구를 진행 · NVMe 모듈은 가상 파일 시스템(VFS) 우회 방식인 SPIN의 주소 터널링(address tunneling)을 지원하는 기능을 수행하며, 입출력 제어(IOCTL)는 SPIN 드라이버(spindrv)가 담당 · SPIN 드라이버는 Linux 커널 시스템 콜을 후킹(Hooking)하여 GPU 메모리에 대한

수행기관	개발 목표	개발내용 및 범위
		접근인지, CPU 메모리에 대한 접근인지를 판단. 이후 GPU와 SSD간의 메모리 직접 접근(DMA)을 수행할 것인지를 결정. 메모리 직접 접근의 경우 NVMe 모듈과 결합한 VFS 우회를 통해 SSD가 GPU에 메모리 직접 접근하도록 GPU 주소를 제공하여 PCI 버스로 통신할 수 있게 함  그림 52. SmartSSD-GPU 데이터 송신 시스템 설계. · 위 그림은 SPIN의 데이터 흐름을 나타냄. SPIN 드라이버에서 메모리 직접 접근으로 판단한 경우는 빨간 화살표의 이동 경로로 데이터가 전송되고, 아닌 경우 하늘색 화살표를 통해 데이터를 전송 - 성능 평가를 위해 사용한 실험 환경 · CPU: 24코어 Intel(R) Xeon Gold 6208U · 메모리: 128GB 메모리 · GPU: Tesla V100 16GB PCIe · 운영체제: Ubuntu 16.04 - 비교군 · CPU 사용(w/o bypass): GPU 메모리 주소 공간을 CPU 메모리 주소 공간에 매핑하여 페이지 캐시를 사용하는 방식 · CPU 우회(w/ bypass): GPU와 SSD의 메모리 직접 접근(DMA)으로만 데이터를 주고받는 방식  그림 53. CPU 사용(W/o bypass)와 CPU 우회(W/ bypass)의 데이터 전송 시간 비교 - 실험 결과분석 · 위 그림은 CPU 사용과 CPU 우회 사이의 데이터 전송 시간 비교 결과. X축은 전송한 데이터 크기를 나타내며, Y축은 데이터 전송 시간(초)을 나타냄 · 2KB 이하의 데이터 전송은 CPU 사용시 데이터를 더 빠르게 전송하였는데, 이는 CPU 사용이 페이지 캐시(page cache)를 효과적으로 사용할 수 있어 데이터 이동(data movement)이 적었기 때문. 하지만 8KB 이상에서는 반대로 CPU 우회가 데이터 전송 시간을 더 적게 소모. 해당 결과는 CPU 사용에서 데이터 크기에 비해 제한적인 4KB

수행기관	개발 목표	개발내용 및 범위
		페이지 캐시로 인하여 데이터 이동이 많았던 것이 원인 · 학습에 사용되는 이미지 한 장은 압축되지 않은 224x224 BMP RGB 3x8 bit/pixel 정도의 파일이고 이 파일 크기가 150KB. 즉, 데이터 로딩 이후 결과로 전송될 데이터 크기는 150 KB 이상. 따라서, 데이터 로딩에서 CPU 우회 방법을 사용하는 것이 데이터를 더 능률적으로 전송할 수 있음

ETRI (주관)

· 이종 메모리를 고려한 메모리 중심 컴퓨팅 시스템 연구
· 다중 응용을 고려한 공유 메모리 관리 기술 연구
· 데이터 분석 응용의 메모리 사용 패턴 연구

김강호

아주대학교 (공동1)

· 메모리 중심 컴퓨팅 시스템 기반 쓰레드 병렬화 기술 연구

성균관대학교 (위탁1)

· 비휘발성 메모리 기반 데이터 구조 연구

UNIST (위탁2)

· 메모리 중심 컴퓨팅 시스템을 위한 가속기 기술 선행 연구

(2) (1단계) 2차년도(2021년도) 개발 내용 및 범위

[메모리 중심]

수행기관	개발 목표	개발내용 및 범위
ETRI (주관)	① 메모리 중심 컴퓨팅 시스템 자원 스케줄링 기술 연구	**• 메모리 중심 컴퓨팅 시스템 기반 클러스터 메모리 관리 도구 개발** - 클러스터 메모리 관리 도구의 필요성 · 클러스터는 가상화라는 기술을 바탕으로 물리적인 컴퓨팅 자원을 여러 유저가 사용하는 환경에서 주로 사용되며, 이러한 환경에서 EMP와 같은 분리메모리를 사용하는 경우에는 물리머신에 장착되어있는 직접메모리의 자원이 중요한 역할을 함 · 하나의 컴퓨팅 노드를 여러 유저가 사용하게 되면 원격의 메모리가 있다고 하더라도 성능에 큰 영향을 주고 작은 크기의 직접메모리를 효과적으로 사용하는 기술이 필요함. 따라서 컴퓨팅 노드의 메모리를 효과적으로 분배해줄 수 있는 메모리 관리 도구가 필요함 - 클러스터 메모리 관리 도구 개발 · 본 연구는 컴퓨팅 노드에 있는 한정적인 직접메모리의 크기를 각 가상머신에 효과적으로 배분해 줄 수 있는 관리 도구를 개발함. 해당 관리 도구는 하나의 물리머신에 대해서 동작하며 원격 메모리에 대한 접근을 인지하고, 각 가상머신의 SLO에 따라서 메모리를 나누어줄 수 있는 도구임 · 클러스터 메모리 관리 도구는 각 가상머신의 원격 메모리 접근 패턴을 프로파일링하여 아래의 4가지 정책을 중 하나로 각 가상머신에 메모리를 배분함 1) 가상머신에 따라 같은 크기로 배분: 이 정책은 메모리 관리도구가 설정가능한 총 메모리의 양을 가상머신의 개수에 따라 똑같은 크기로 메모리를 배분하는 정책임 2) 메모리 페이지 폴트를 최소화 하는 메모리 크기로 배분: 각 가상머신의 원격메모리 접근 패턴을 사용하여 모든 가상머신의 총 메모리 페이지 폴트를 최소화 할 수 있는 방향으로 메모리를 배분하는 정책 3) 성능을 최대화하는 메모리 배분: 각 가상머신의 SLO와 상관없이 하나의 머신에 대해 성능을 최대화할 수 있는 방향으로 메모리를 배분하는 정책 4) SLA를 최대한 살리는 메모리 배분: 각 가상머신의 SLO를 최대한 맞추는 방향으로 메모리를 배분하는 정책 - 클러스터 메모리 관리 도구 사용검증 · 그림 54은 본 연구에서 개발한 클러스터 메모리 관리 도구의 사용 환경을 나타냄. 왼쪽의 두 개의 세션은 하나의 머신에서 동작하는 두 개의 서로 다른 가상머신의 사용을 나타내며, 오른쪽의 세션은 클러스터 메모리 관리 도구를 나타냄 · 관리 도구는 각 가상머신이 사용가능한 메모리의 양을 표현하고, 각 가상마신의 SLO값을 표현함. 또한, 정해진 간격으로 가상머신의 원격메모리 접근을 파악하고, 원격메모리 접근 패턴과 SLO, 그리고 해당 가상머신의 현재 메모리양을 바탕으로 다음 턴의 메모리 크기를 결정함 · 만약 하나의 가상머신이 새로운 원격메모리 접근 패턴을 가지게 된다면 패턴의 변화를 인지하고 프로파일링을 좀 더 수행하고 메모리의 크기를 결정하며, 클러스터 메모리 관리도구가 설정 가능한 총 메모리의 크기를 조절하여 물리머신의 총 메모리 크기를 넘지 않도록 구현함

수행기관	개발 목표	개발내용 및 범위

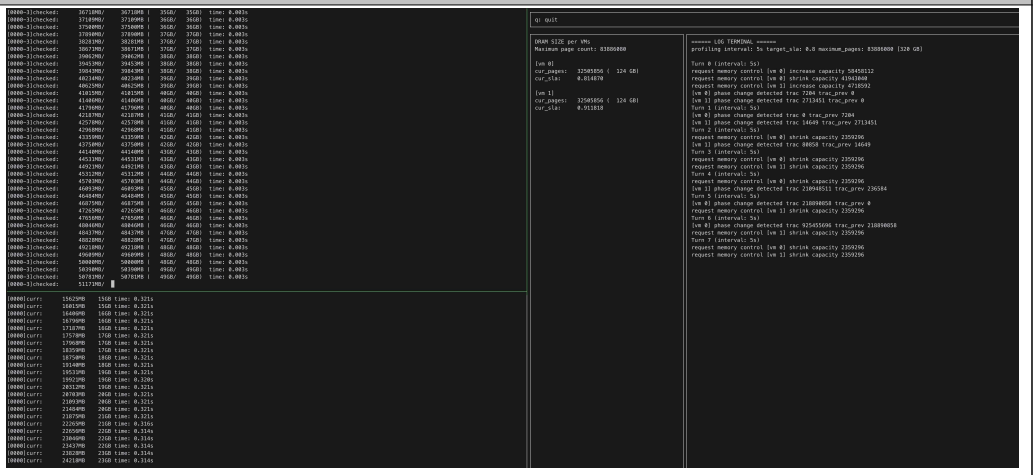


그림 54. 클러스터 메모리 관리도구

· 사용자를 위한 성능 최적화 도구 개발

- 쓸모없는 페이지(stale page) 정보 제공의 필요성
  - · EMP는 물리머신에서 동작하는 가상머신에 대해서 해당 물리머신이 가지고 있는 메모리보다 많은 양의 메모리를 제공해주는 플랫폼임. 가상머신 내에서 하나의 프로그램이 동작하여 원격의 메모리를 사용하다가 종료되는 경우 대부분의 응용이 원격의 메모리를 초기화하여 메모리 해제 (free) 시켜주는 작업을 진행함
  - · 하지만 응용이 자신이 사용한 원격의 메모리에 대한 해제를 시켜주지 않는다면, EMP는 해당 원격 메모리의 데이터를 계속 관리하여 초기화 되지 않은 상태로 유지되기 때문에 다른 응용이 해당 데이터의 위치에 접근하려고 하면 원격 메모리에 접근하여 해당 데이터를 직접메모리로 가지고와서 초기화를 시키게 됨. 이렇게 되면 쓸모없는 페이지(stale page)에 대한 접근이 필요해짐
- 원격메모리 접근 횟수를 줄여주는 성능 최적화 도구 연구
  - · 본 연구에서는 쓸모없는 페이지에 대한 접근을 줄이기 위해서 가상머신 내부에서 동작하여 쓸모없는 페이지를 EMP에게 알려주는 도구를 제공하며 이는 원격메모리에 대한 접근 횟수를 줄여주는 역할을 함. 해당 최적화 도구를 EMP 도구라 명명함
  - · EMP 도구는 가상머신 내에서 동작하며, 어떠한 프로세스가 종료된 후 동작하여 쓸모없는 메모리 페이지를 EMP에 알려주어 해당 메모리 페이지에 대해서 원격메모리 접근 없이 초기화 시킬 수 있도록 도움을 줌. 이에 대한 구현은 그림 55와 같이 가상머신 내부에서 메모리 해제할 메모리의 크기를 설정하여 해당 메모리 크기만큼 EMP에게 쓸모없는 페이지라고 가상머신의 모듈에게 전달하고, 이 모듈이 EMP에게 HyperCall을 보내주는 방식으로 구현함

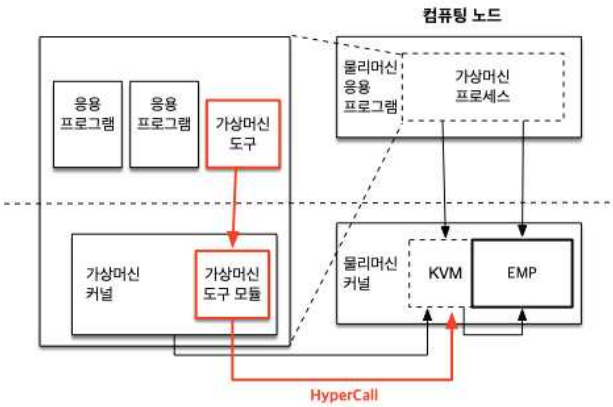
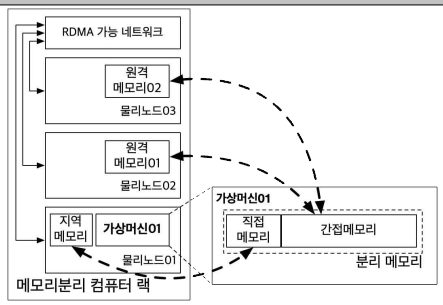


그림 55. 가상머신 도구 통신 흐름도

수행기관	개발 목표	개발내용 및 범위
		- 가상머신 도구에 대한 성능평가 방법 · 가상머신 도구에 대한 성능 평가를 위해서 EMP환경에서 직접메모리는 104GB, 원격메모리는 512GB를 사용하는 환경을 설정하고, MariaDB를 활용하여 250GB의 워크로드를 Insert하는 작업을 수행함 · 총 3번의 Insert작업을 수행하되 가상머신 도구를 사용한 결과에서는 Insert작업 이후 500GB에 대한 메모리 영역을 쓸모없는 페이지 영역이라고 EMP에 알려주는 작업을 추가함 - 가상머신 도구에 대한 성능평가 결과 · 그림 56는 EMP환경에서 응용의 실행이후 가상머신 도구를 수행시켰을 때의 성능을 그렇지 않았을 때의 성능과 비교하는 그래프임. x축은 첫 번째, 두 번째, 세 번째 수행과 평균 수행시간을 의미하며, y축은 수행시간으로 높을수록 성능이 떨어짐 · 가상머신 도구가 없이 수행시키는 경우 (파란색 바) 첫 번째와 두 번째 수행의 성능차이가 큰 것을 확인할 수 있음. 이는 쓸모없는 페이지에 대해서 초기화를 수행할 때 원격메모리로의 접근이 일어나기 때문에 발생하는 오버헤드임. 하지만 가상머신 도구를 사용하는 경우 (주황색 바), 첫 번째, 두 번째, 세 번째 수행의 결과가 거의 유사함을 확인 가능 · 평균적인 수행시간도 가상머신 도구를 사용하지 않을 때에 대비해서 성능 향상이 있음을 확인 가능 그림 56. 가상머신 도구에 대한 성능평가 **• 메모리 중심 컴퓨팅 시스템 기반 가상머신 이주 기반 메모리 접근 대응 기술 개발** - 분리 메모리를 활용하는 가상머신의 다양한 메모리 접근으로 인한 자원관리 필요성 · 메모리 분리 시스템에서 빠른 메모리 접근이 요구되는 데이터는 가상머신이 동작하는 호스트에 존재하는 지역 DRAM에 위치하고, 그 외의 데이터를 원격 메모리 등에 위치시킴으로써 높은 메모리 접근 성능과 컴퓨터 메모리 서브시스템에 의해 제어되는 DRAM보다 대용량의 메모리를 제공할 수 있음 · 빠르게 접근해야하는 메모리의 양이 지역 DRAM보다 커지는 경우, 동작하는 가상머신은 지속적으로 원격 메모리에 접근하게 됨으로 인해 성능 저하가 심각해지므로, 이에 대응하는 자원 관리 기술이 요구되며 이는 메모리 사용 패턴을 고려하여 가상머신을 이주시킴으로써 대응할 수 있음 - 다양한 메모리 접근 패턴을 고려한 가상 머신 이주 기법 연구

수행기관	개발 목표	개발내용 및 범위



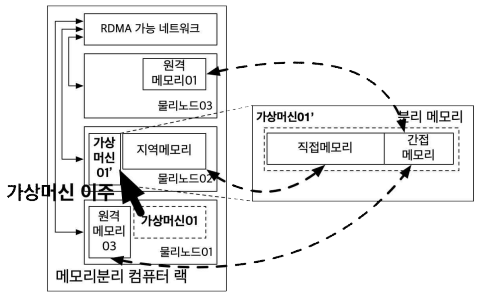


그림 57. 분리 메모리 시스템 - 메모리 할당 그림 58. 가상머신 이주로 인한 메모리 할당 변화

· 위의 그림 57은 통상적인 분리 메모리 시스템에서 사용되는 메모리 자원 할당을 나타내며, 이에서 나타난 바와 같이 가상머신01은 물리노드01에서 동작하며 직접메모리를 위해서 물리노드01에 위치하는 지역메모리를 사용하고, 간접메모리로써 물리노드02와 물리노드03에 위치하는 원격으로 접근 가능한 메모리를 사용함. 이처럼 여러 노드의 메모리 자원을 활용하여 동작하는 하나의 가상머신을 생성함으로써 기존의 하나의 물리노드에 있는 메모리 자원만을 활용하는 가상머신 기술들과 비교하여 높은 유연성 (Flexibility)와 높은 자원 활용율 (Utilization)을 기대할 수 있음

· 위의 그림 58은 가상머신01이 물리노드01에서 활용 가능한 지역메모리의 한계로 인해서 직접메모리로써 사용가능한 메모리가 더 많은 물리노드02로의 가상머신 이주를 나타내는 것으로. 물리노드01에 위치할 때보다 더 많은 지역메모리를 할당함으로써 더 많은 성능-민감 데이터를 지역메모리에 위치시켜서 분리메모리로 인한 성능 하락을 막을 수 있음

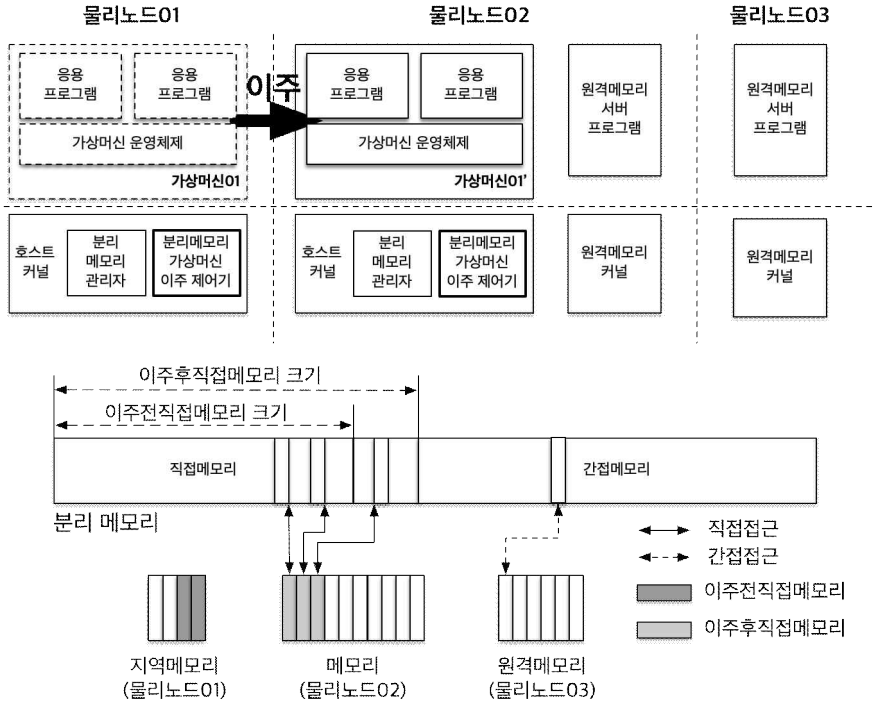
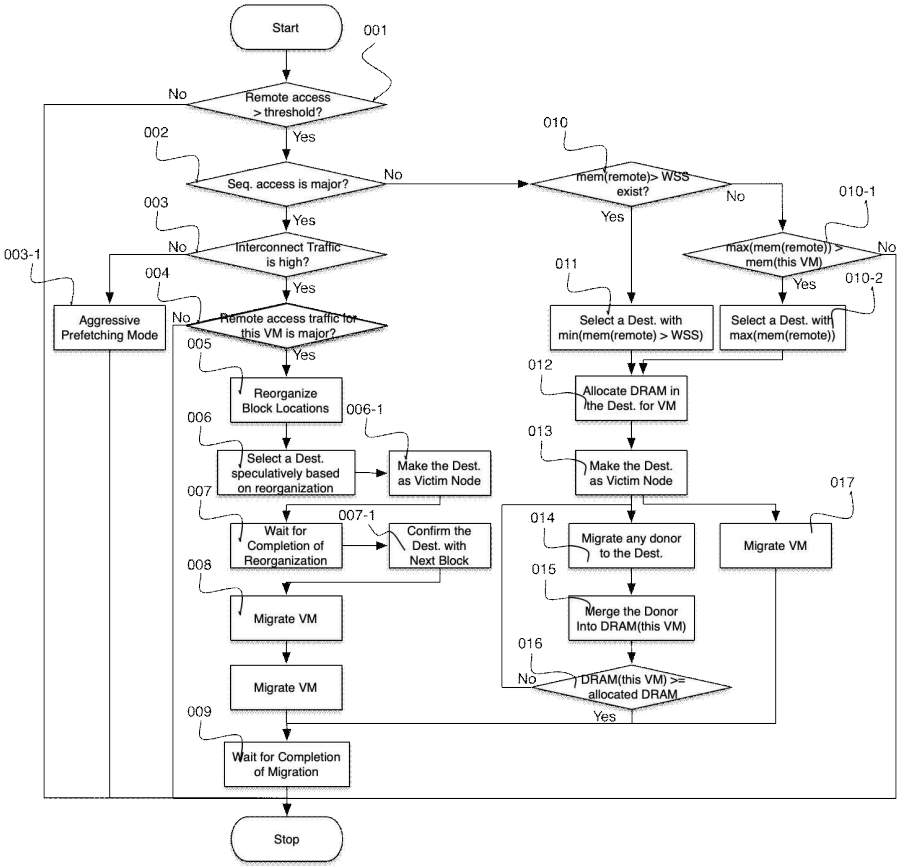


그림 59. 가상머신 이주에 따른 직접메모리 크기 변화

· 위의 그림은 가상머신의 이주에 따른 직접 메모리 크기 변화를 보다 구체적으로 나타낸 것으로. 물리노드01에서 동작하고 있는 가상머신01은 물리노드02로 이주함으로 인해서 더 많은 메모리를 할당받음으로 인해 직접메모리 크기를 증가시킬 수 있고, 이로 인해서 더 많은 성능-민감 데이터를 직접 메모리에 위치시킬 수 있음. 이는 분리메모리 가상머신 이주 제어기에 의해서 제어되며, 이는 분리메모리 관리자가 원격 메모리 접근의 수행에 추가적으로 원격 메모리에 대한 접근 패턴 분석, 원격 메모리 접근을 통해 요구되는

수행기관	개발 목표	개발내용 및 범위
		Working Set Size (WSS) 분석, 원격 메모리 서버들에 대한 이동 제어, 가상머신 이주시 발생하는 eviction을 위한 원격 메모리 제어를 이용하여 현재의 원격 메모리 접근 패턴을 고려하고, 노드간 연결된 인터커넥트 비용을 줄임으로 효과적인 가상머신 이주를 수행함 - 효과적인 분리메모리 기반 가상머신 이주 방법  그림 60. 메모리 분리 컴퓨터 구조에서 가상 머신 이주 방법에 대한 순서도 · 001: 원격 메모리 접근이 사용자 혹은 관리자가 설정한 임계점을 넘는 경우, 성능을 고려한 가상머신 이주를 수행하게 됨. 이때 원격 메모리 접근 횟수와 평균 원격 메모리 접근으로 인한 페널티를 통하여 현재의 응용의 성능 저하를 계산할 수 있으며, 평균 원격 메모리 접근 시간을 가정하면 원격 메모리 접근 횟수 모니터링으로 성능 저하 예측이 가능 · 002: 이후 원격 메모리 접근 시 해당 메모리에 대한 가상머신 물리 주소 (GPA)를 알 수 있고, 이에 대한 추적으로 통해 메모리 접근이 순차적으로 발생하는지 아닌지를 확인가능 · 003: 원격 메모리 접근이 순차적으로 발생하고 있는 것을 확인할 경우, 노드간 연결을 제공하고 또한 원격 메모리 접근에 사용되는 인터커넥트 상에서 많은 트래픽이 발생하는지를 판단. 이는 동일한 물리 노드에 위치하는 다수의 분리 메모리를 사용하는 가상머신들이 많은 원격 메모리 접근을 발생시킬 경우 해당 노드에서 인터커넥트 대역폭을 많이 사용함으로 해당 노드에 동작하는 다른 가상머신들에 서로 영향을 끼치게 됨 · 003-1: 인터커넥트 트래픽이 적은 경우, 순차적 메모리 접근에 대응하기 위해서 가상머신 이주를 선택하는 것이 아닌 적극적인 페이지 프리페칭을 통해서 원격 메모리 접근에 대응 · 004: 인터커넥트 트래픽이 많은 경우, 현재 가상머신이 물리 노드에 존재하는 분리 메모리 가상머신 중 가장 많은 트래픽을 만들어 내는지 확인하여 이주 대상인지를 확인. 현재의 원격 메모리 접근을 발생시킨 가상머신이 이주 대상이 아니라면, 다른 가상머신이 이주될 것을 가정하고 이주하지 않음 · 005: 해당 가상머신이 이주 대상이라면, 해당 가상머신이 사용하는 다른 메모리 노드에 존재하는 블록들이 순차 접근에 대해서 최적화되도록 재구성 요청 · 006: 원격 메모리 서버 재구성 진행률을 이용하여 재구성 완료 시점과 예측되는 메모리

수행기관	개발 목표	개발내용 및 범위
		노드를 선정 · 006-1: 앞 단계에서 예측한 메모리 노드를 eviction의 대상으로 선정하여 현재부터 발생하는 eviction을 모두 대상 노드로 전송되도록 설정함 · 007,007-1: 재구성의 완료를 기다리고 이후 앞 단계에서 예측한 메모리 노드가 다음 원격 메모리 접근을 발생시킨 노드인지를 확인. 해당 예측이 맞는 경우와 맞지 않는 경우에 따라서 통상적인 가상머신 이주 기법 중 프리마이그레이션 (pre-migration), 라이브마이그레이션 (live-migration) 등을 선택하여 이주를 진행 · 008,009: 해당 노드로 가상머신을 이주시킴. 이때, 메모리를 제외한 스토리지, 네트워크 등에 대한 처리는 통상적인 가상머신 이주에 사용되는 것과 동일하며 가상머신 이주 제어기는 가상머신 이주의 완료를 기다림 · 010: 원격 메모리 접근을 확인, 가상머신의 메모리 요구량 (WSS)을 추정하고 해당 용량의 메모리가 가용한 원격 노드가 있는지를 확인하며, 또한 이주 전 노드에서는 더 이상의 지역 DRAM을 이주 중인 가상머신에게 할당하지 않도록 설정 · 010-1,010-2: 해당 노드가 없다면 원격 메모리를 제공하는 노드 중 가용한 메모리가 가장 큰 노드가 현재 가상머신이 사용하는 지역 DRAM 용량보다 작으면 가상머신 이주를 하지 않고 종료시키고, 현재 가상머신이 사용하는 지역 DRAM 용량 보다 크다면 가용한 메모리가 가장 큰 노드를 대상노드로 선정함 · 011: 가상머신 메모리 요구량 이상의 메모리가 가용한 노드 중, 가용한 메모리 용량이 가장 작은 노드를 대상 노드로 선정 · 012: 선정한 대상 노드에서 이주 후 동작할 가상머신을 위한 지역 DRAM을 할당하고 이를 가상머신 이주 동안 원격 접근이 가능하도록 설정 · 013: 현재 노드에서 동작 중에 메모리 압력으로 발생하는 eviction의 대상을 앞 단계에서 선정한 대상 노드로 설정하여 현재부터 발생하는 eviction을 모두 대상 노드로 전송되도록 설정 · 014-016: 가상머신의 메모리 제공자들을 중 하나를 임의로 선정하여 대상 노드로 이주시키고, 주시킨 메모리 제공자에 해당하는 영역을 가상머신의 012에서 할당받은 지역 DRAM 영역에 합치며, 이주한 가상머신이 대상 노드에서 할당받은 지역 DRAM의 용량을 전부 채울 때까지 앞 단계들을 반복함 · 017 앞 단계와 동시에 현재의 가상머신은 이주 전 노드에 있는 지역 DRAM에 위치한 메모리들의 대상 노드로 전송시키면서 가상 머신 이주를 진행함으로 분리 메모리 환경에서의 가상머신 이주를 메모리 접근 패턴을 고려하여 효과적으로 수행할 수 있음 - 메모리 접근 패턴을 고려한 가상머신 이주로 인한 효과 · 기존 가상머신 이주 기술은 가상머신이 사용하는 메모리를 가상머신이 동작하고 있는 메모리에 위치시키는 것을 가정하고 있으나, 본 연구는 이와 달리 완전히 다른 분리메모리 환경에서 가상머신 이주 기술에 대한 효과적인 방법을 제공함으로써 각 분리 메모리 가상머신이 할당받은 직접 메모리 양과 원격 메모리 접근 패턴 및 네트워크 사용량을 고려하여 분리메모리를 활용하는 가상머신의 성능을 관리하고 가상머신 이주 비용을 낮춤으로써 다수의 노드를 대상으로 분리 메모리가 적용되는 클라우드 인프라 환경의 인프라 운영 비용을 효과적으로 절감할 수 있음 **· 메모리 중심 컴퓨팅 시스템 기반 가상머신 네트워크 가속 기법 개발** - 가상머신 네트워크 가속 기법 개발의 필요성 · 클라우드 인프라에서 사용되는 대표적인 서비스 중 하나인 메모리 캐시 서비스는 네트워크를 통해서 메모리에 기록되어 있는 정보를 클라이언트 요청에 따라서 전송하는 것으로, 클라이언트의 데이터 요청 패턴에 따라서 지역성 (Locality)가 강하게 나타나는 경우도 있으나, 그 성격이 거의 없는 경우도 존재함 · 요청 데이터가 지역 DRAM에 위치할 경우 빠르게 네트워크를 통해서 해당 데이터를 전송할 수 있으나, 요청 데이터가 원격 메모리에 위치할 경우, 가상머신 노드는 원격 메모리에 접근하게 되고 이를 분리 메모리 관리자에 의해 원격 메모리로부터 지역 메모리로 적재한 후, 이를 네트워크를 통해서 데이터를 전송해야 함. 이는 네트워크로 송신해야하는 데이터

수행기관	개발 목표	개발내용 및 범위

를 포함하는 메모리 블록의 적재 요청, 원격 메모리 서버로부터 지역 DRAM으로의 적재를 수반하기 때문이 네트워크 요청에 대한 지연시간이 증가하고, 네트워크에 대한 대역폭을 충분히 사용하기 어려움

· 이처럼 메모리 캐시 서비스와 같이 낮은 지역성을 갖는 네트워크 기반 응용 서비스를 원격 메모리 기반 가상머신 시스템에서 지원하기 위해서는 분리 메모리 시스템이 이를 효과적으로 지원하기 위한 방법이 요구됨

- 기존 가상머신 네트워크 기술 분석

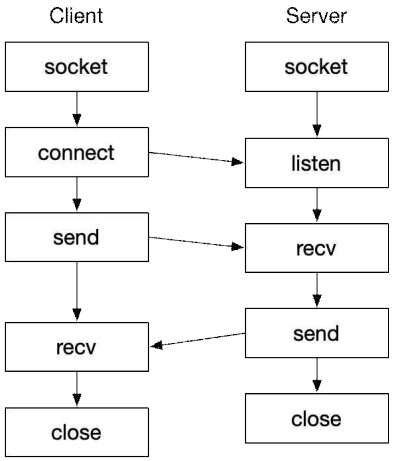


그림 61. socket을 이용한 통신 순서

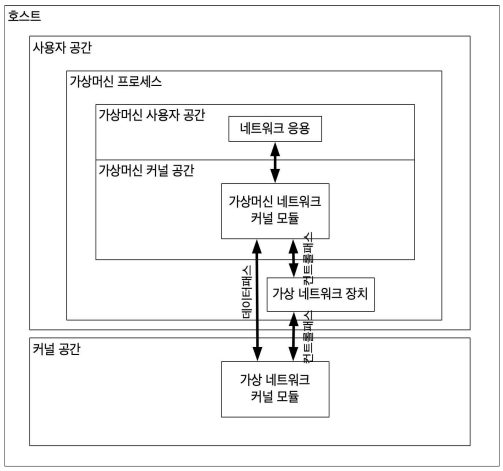


그림 62. 가상머신의 네트워크 전송 구조

· 위 그림 61은 네트워크를 사용하여 통신하는 응용프로그램 client와 server의 통신 순서를 나타냄. connect 함수를 통해서 client는 listen을 수행하여 기다리고 있는 server와 연결된고, 이후 client와 server는 send/recv 함수를 이용하여 필요한 정보를 서로 주고받게 됨

· 위 그림 62은 가상머신의 네트워크 전송 구조를 나타냄. 네트워크 응용에서 수행되는 send 함수는 시스템 콜을 이용하여 가상머신 네트워크 커널 모듈로 요청하고, 네트워크 응용에 요청한 데이터를 네트워크 커널 모듈이 시스템으로부터 할당받은 버퍼에 복사함. 이렇게 할당 받은 버퍼에 저장되어 있는 데이터는 더 이상의 복사 없이 가상머신 네트워크 장치로 전달되고 이를 다시 가상 네트워크 커널 모듈로 전달되어 최종적으로 네트워크 장치를 통해서 네트워크로 송신됨

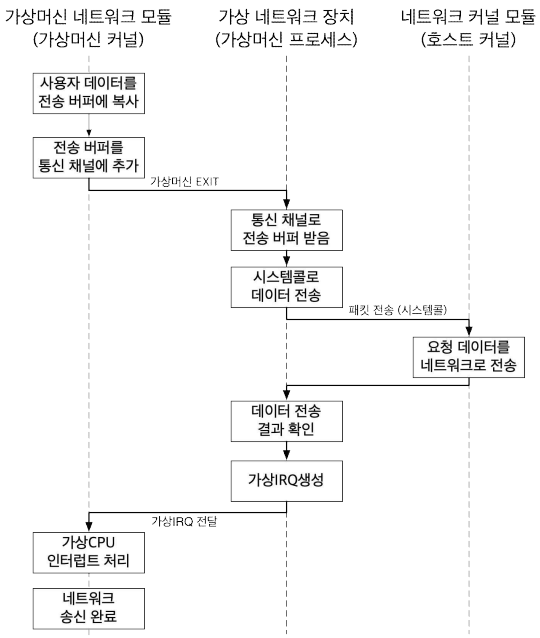
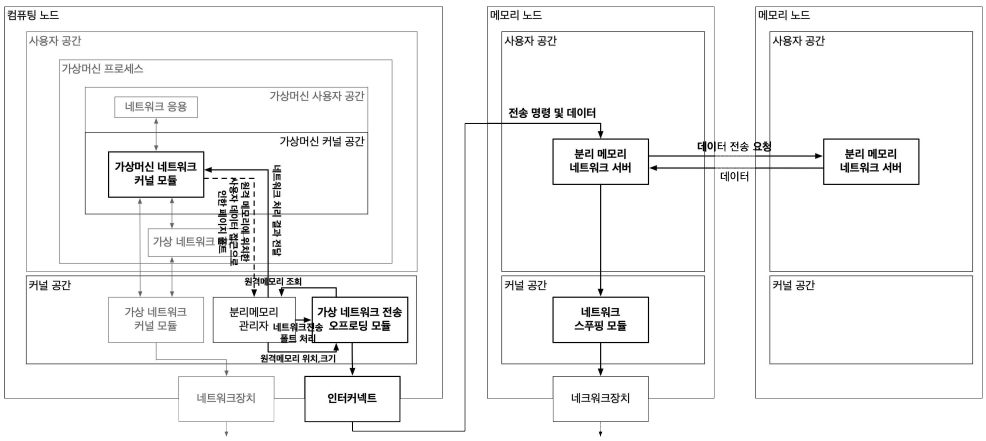
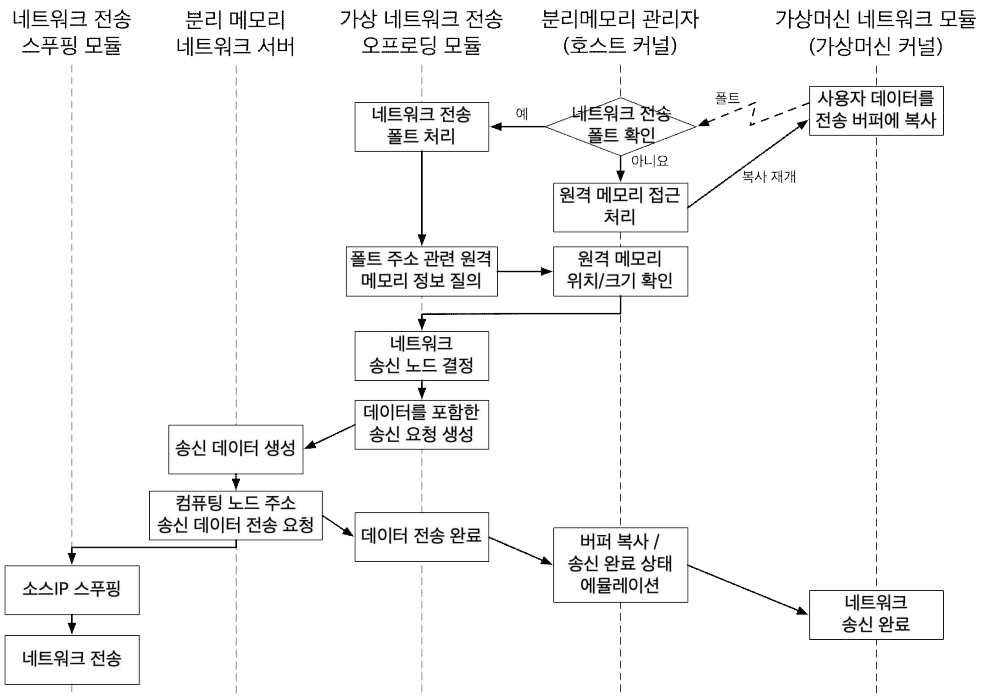
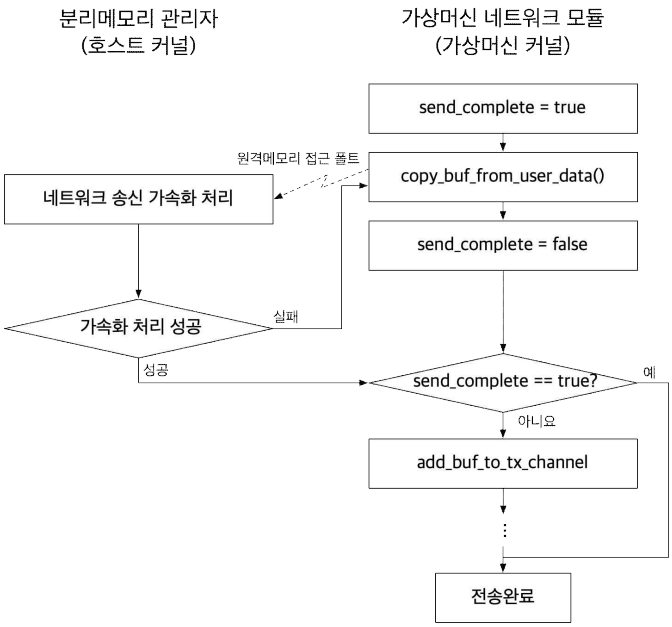
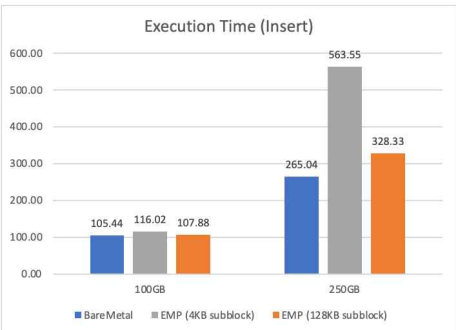
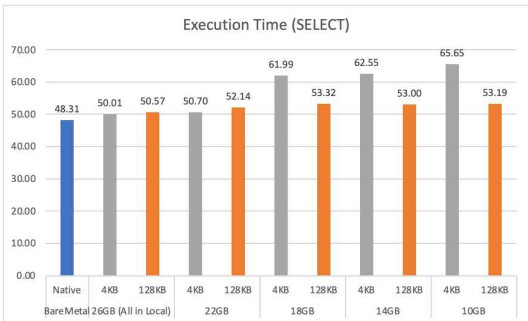
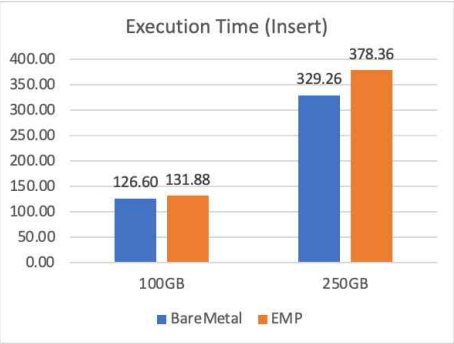


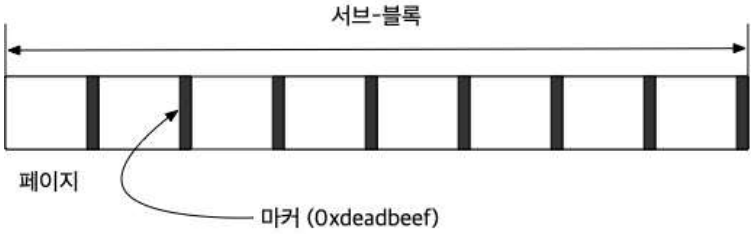
그림 63. 가상머신 네트워크 전송의 데이터패스 흐름도

수행기관	개발 목표	개발내용 및 범위
		· 위 그림은 가상머신 네트워크 전송에 대한 데이터 경로의 흐름을 나타내는 것으로, 사용자가 요청한 데이터는 가상머신 네트워크 모듈에서 전송 버퍼에 복사되고 이를 가상 네트워크 장치를 거쳐서 네트워크 커널 모듈에 전달하여 요청 데이터를 네트워크로 전송하게 됨. 해당 결과는 가상IRQ 등을 통해서 가상CPU에게 전달되어 네트워트 송신이 완료됨 - 분리 메모리 시스템에서의 가상머신 네트워크 가속화 기술  그림 64. 분리 메모리 가상머신에서의 네트워크 전송 가속화를 위한 컴포넌트 구조 · 위 그림은 분리 메모리 가상머신에서의 네트워크 전송 가속화를 위한 컴포넌트들의 구조를 나타내는 것으로, 가상머신 네트워크 커널 모듈에서 사용자 송신 요청 데이터의 복제 시 해당 데이터가 원격 메모리에 위치하면 분리메모리 관리자에게 제어가 넘어가고, 이를 가상 네트워크 전송 오프로딩 모듈을 통해 네트워크 송신 노드를 선정함. 선정된 송신 노드는 송신에 필요한 데이터를 원격 메모리 서버들과 컴퓨팅 노드로부터 전송받고 이를 합쳐서 네트워크 스푸핑 모듈을 통해 가상머신의 IP로 해당 송신을 수행함  그림 65. 네트워크 전송 가속화 기술에서 송신 데이터패스 흐름도 · 위 그림은 분리메모리 가상머신에서의 네트워크 전송 가속화 기술에서 데이터 송신을 위한 각 컴포넌트 별 수행 흐름도를 나타내는 것으로, 먼저 가상머신 네트워크 모듈에서 사

수행기관	개발 목표	개발내용 및 범위
		용자 프로그램의 송신 데이터가 전송 버퍼로 복사됨. 이때, 송신 데이터가 원격 메모리에 위치하고 있으며 원격 메모리 접근이 발생하며, 이로 인해 페이지 폴트가 발생하고 제어가 분리메모리 관리자에게 전달됨. 이와 달리, 송신 데이터가 모두 지역 DRAM에 위치해 있을 경우 폴트가 발생하지 않고 기존 가상머신과 동일하게 전송됨. 또한 네트워크 전송 폴트로 인지하지 못하는 경우, 기존 원격 메모리 접근과 같이 처리됨 · 네트워크 전송에 대한 원격 메모리 접근 폴트가 확인되면, 가상 네트워크 전송 오프로딩 모듈이 분리메모리 관리자에게 폴트 주소에 관련된 원격 메모리 정보를 질의하게 됨. 사용자가 송신하고자 하는 데이터의 메모리 위치 정보를 확인하여 컴퓨팅 노드 혹은 원격 메모리 노드들 중 가장 많은 데이터가 위치한 노드를 선정함. 아래 그림에서 나타난 바와 같이 데이터 구성정보와 컴퓨팅 노드 IP 주소, 컴퓨팅 노드상에 위치하는 데이터를 이용하여 분리 메모리 네트워크 서버로의 송신을 요청하게 되고, 이후 분리 메모리 네트워크 서버는 송신요청 정보 상의 데이터 구성 정보를 확인하여, 필요할 경우 다른 원격 메모리 서버들에 필요한 데이터를 요청/수신하고 해당 정보를 모아서 완전한 송신 데이터를 생성함. 완전한 송신 데이터를 이용하여 컴퓨팅 노드 주소를 이용해서 네트워크 전송을 수행하고, 이는 가상 네트워크 전송 오프로딩 모듈에게 전달됨 데이터 구성 정보 (메모리서버 정보, 길이) | … | 컴퓨팅 노드 IP 주소 | 데이터(컴퓨팅노드) 그림 66. 분리 메모리 네트워크 서버로의 송신요청 정보  그림 67. 분리 메모리 관리자의 네트워크 가속화 처리에 따른가상머신 제어 전달 경로 · 분리메모리 관리자는 가상머신 상에서 기존에 중단되었던 사용자 데이터의 복사 작업이 완료되었고, 또한 네트워크 송신이 끝났음을 알리도록 가상머신의 컨텍스트를 수정하는 에뮬레이션 작업을 수행함 · 이후 가상머신 네트워크 모듈로 제어가 전달되며, 사용자 데이터 버퍼 복사와 네트워크 전송이 끝났음을 알고 사용자 프로그램에 네트워크 송신이 끝났음을 알리게 됨. 위 그림에서 나타난 바와 같이, 가속화 처리가 끝나면, 처리 성공에 따라 성공하면 가상머신 네트워크 모듈은 데이터 송신이 성공했음을 확인하게 되고, 가속화 처리가 실패하면 사용자 데이터를 버퍼로 복사하는 작업을 재개함으로써 네트워크 송신을 가속화함 - 가상머신 네트워크 가속 기법의 효과 · 클라우드 데이터 센터에서 수행되는 다수의 응용들이 사용자 혹은 응용 로직 서버에게 네

수행기관	개발 목표	개발내용 및 범위
		트워크를 통해서 서비스를 제공하고 있고, 이와 같은 용용들은 요청으로부터 응답이 오기까지의 서비스 응답시간 (Service Latency)의 평균도 중요하지만, 또 다른 주요한 성능 요인은 꼬리 응답시간 (Tail Latency)임 · 가장 오랜 걸린 응답시간 군이 주요한 성능이 되는 환경에서 분리 메모리 시스템은 메모리 블록이 원격 메모리에 위치함으로 이를 컴퓨팅 노드로 적재 후 서비스하고, 그 지역성이 낮을 경우, 원격 메모리 접근이 잦고, 결과적으로 높은 꼬리 응답시간을 갖게 되어 그 적용 시나리오가 한정됨 · 본 연구는 네트워크 서비스를 수행함에 있어서 메모리 분리 시스템에 네트워크로 송신해야하는 데이터에 대해서 그 메모리 서버의 위치를 고려하고 해당 서버로 네트워크 송신을 오프로딩 함으로써 가속화함. 이는 다수의 요청된 서비스의 응답을 다수의 메모리 노드가 직접 응답하게 됨으로써 하나의 컴퓨팅 노드에서만 서비스하는 경우 보다 높은 네트워크 대역폭을 제공할 수 있고, 이로 인해 분리 메모리 환경에서 집중된 노드가 아닌 다수의 메모리 서버가 네트워크에서 직접 서비스함으로써 꼬리 응답시간이 증가를 방지함
	② 혼성 메모리 기반 메모리 중심 컴퓨팅 시스템 기술 연구	**• EMP에 대한 MariaDB 성능 평가** - EMP의 MariaDB 성능 평가 방법 · 혼성 메모리를 사용하는 메모리 확장 플랫폼인 EMP에 대한 성능을 평가함. 성능은 SSD를 메모리 확장에 사용하는 경우와 다른 노드의 메모리를 메모리 확장에 사용하는 경우 두 가지로 나누어서 성능을 평가함. 성능 평가는 MariaDB를 사용하여 Insert 작업의 수행 성능 결과와 Scan 작업의 수행 성능 결과를 기존 컴퓨팅 환경과 비교하는 방식으로 평가함 - EMP의 MariaDB 성능 평가 결과 · 첫째, 그림 68은 SSD를 이용해서 메모리를 확장하는 경우의 Insert 작업 성능 평가 결과임. EMP환경은 직접메모리로 104GB를 사용하고 원격메모리로 512GB를 사용하는 환경으로 설정함. x축은 워크로드의 크기를 나타내고, y축은 MariaDB에서 Insert 작업시 걸리는 시간을 나타냄. 따라서 y축 값이 클수록 성능의 하락이 크다는 것을 나타냄 · 100GB의 워크로드의 경우 EMP환경에서 직접메모리만을 사용하기 때문에 기존 시스템의 성능과 거의 유사한 성능을 나타냄. 하지만 250GB 워크로드에 대해서 Insert 작업을 수행하는 경우, SSD로의 접근이 자주 발생하기 때문에 EMP환경에서 성능의 하락이 발생하는 것을 볼 수 있음. 하지만 서브-블록의 크기를 128KB로 설정한 경우 성능의 하락이 약 19%정도만 발생하는 것을 확인함. 즉, 100GB의 메모리만 가지고 있을 경우 250GB의 메모리를 사용하는 워크로드를 수행시 약 19%의 성능 하락만으로 해당 워크로드를 수행시킬 수 있음을 확인함 · 그림 69은 MariaDB의 Select작업에 대한 성능 평가를 나타냄. 워크로드는 20GB를 사용하였고, EMP환경은 직접메모리로 26GB 부터 10GB까지 워크로드의 크기에 대한 비율을 줄여가며 조정하였고, 원격메모리로 512GB를 사용하는 환경을 설정함. EMP의 경우 워크로드의 20%에 해당하는 직접메모리만을 가지고 있는 상황에서 전체를 직접메모리로 가지고 있는 상황의 Select 결과대비 10% 이내의 성능하락만을 가지는 것을 확인 가능함  그림 68. SSD를 이용한 메모리 확장의 Insert 작업 성능평가  그림 69. SSD를 이용한 메모리 확장의 Select 작업 성능평가

수행기관	개발 목표	개발내용 및 범위
		· 둘째로, 그림 70는 다른 노드에 있는 메모리를 이용한 메모리 확장의 경우에 MariaDB Insert작업 성능 결과임. SSD를 이용한 메모리 확장과 동일한 설정으로 실험하였으며, 250GB워크로드의 경우 기존 컴퓨팅 성능대비 약 13%의 성능하락만이 발생하는 것을 확인 가능함 · 그림 71은 MariaDB Select작업의 성능 결과를 나타내는데, 이 실험도 SSD를 이용한 확장과 동일한 설정으로 진행하였고, 워크로드 크기의 20%의 직접메모리만을 가지고 있는 환경에서도 10%이내의 성능 하락만으로 SELECT작업을 수행할 수 있음을 확인함  그림 70. 원격DRAM을 이용한 메모리 확장의 Insert 작업 성능평가 그림 71. 원격DRAM을 이용한 메모리 확장의 Select 작업 성능평가 **• 필요한 페이지를 먼저 처리하는 기술 개발** - 필요한 페이지를 먼저 처리하는 기술 개발의 필요성 · 블록이나 서브-블록과 같은 큰 크기의 데이터 단위를 사용하게 되면 프리페치의 효과를 얻을 수 있고, 통신 대역폭을 효과적으로 사용할 수 있는 장점이 있으나, 블록이나 서브-블록의 데이터를 기다리는 시간으로 인해서 해당 페이지에 대한 페이지폴트 처리시간이 길어지는 단점이 발생함 · 또한, 블록이나 서브-블록의 페이지에 대해서 더 이상 접근이 되지 않는 경우에는 필요없는 페이지를 fetch하게 되는 것이라 통신 대역폭의 낭비가 발생하게 됨 - 필요한 페이지를 먼저 처리하는 기술 연구 (CPF) · 기존에 위와 같은 문제를 해결하기 위해서 Critical Sub-block First (CSF)를 개발하였으나, 서브-블록의 크기가 큰 경우 하나의 서브-블록을 기다리는 시간이 커지게 되므로 서브-블록 내부에서도 페이지 폴트가 일어난 페이지만 먼저 처리하는 기술을 구현함. 해당 기술을 Critical Page First (CPF)로 명명함 · 필요한 페이지를 먼저 처리하는 기술을 위해서는 서브-블록 단위 DMA 읽기 요청 중에서 폴트가 난 페이지에 대해서 직접메모리로 fetch가 완료되었다는 것을 확인할 수 있어야 함. 기존의 서브-블록의 완료의 경우 DMA에 대한 완료 요청처리만으로 해당 서브-블록의 fetch가 완료되었다는 것을 확인할 수 있었지만, 페이지에 대한 fetch 완료는 다른 메커니즘이 요구됨 · 그림 72과 같이 각 메모리 페이지 (4KB)를 할당할 때마다 페이지의 마지막 8Byte에 마커를 설정하고, 해당 페이지가 쓰여지지 않았다는 것을 확인할 수 있도록 처리함. 따라서, 원격메모리로부터 fetch가 된 페이지에 대해서는 해당 마커가 바뀌었다는 것으로 확인이 가능하게 구현함

수행기관	개발 목표	개발내용 및 범위
		 그림 72. 페이지 fetch완료 확인을 위한 마커 · 페이지 폴트가 발생한 페이지의 fetch가 완료되면 바로 폴트의 처리를 수행하고, 블록의 나머지 서브-블록 및 페이지들은 다음 해당 블록에 폴트가 발생하면 DMA 읽기 요청 완료를 진행함. 만약 해당 페이지에 대한 폴트 처리가 완료되었을 때, 서브-블록에 대한 DMA 읽기 요청 완료가 가능하면 해당 서브-블록에 대한 매핑도 진행하는 옵션을 추가하여 페이지 테이블 매핑이 가능한 경우에는 바로바로 매핑을 수행하는 방식을 적용함 - CPF의 성능평가 방법 · CPF의 성능을 평가하기 위해서 16GB의 직접메모리를 사용하고 512GB의 SSD를 이용한 메모리 확장을 사용하는 EMP환경을 구축함. Memcached를 이용하여 성능을 평가하였는데, 워크로드로는 Facebook ETC 워크로드를 사용하여 95%정도의 읽기와 5%정도의 쓰기가 공존하는 인풋의 키-밸류 스토어에 대한 성능을 평가함. 총 워크로드의 사이즈는 22GB로 70% 정도의 데이터가 직접메모리에 저장될 수 있음. 실험의 편의를 위해서 블록의 크기는 2MB로 고정함 - CPF의 성능평가 결과 · 그림 73는 위의 실험 환경에서 EMP와 EMP에 CPF를 적용시킨 결과에 대한 성능 그래프임. x축은 서브-블록의 크기를 나타내며, y축은 초당 처리량을 나타내어 성능을 표현함. 회색 바는 EMP의 성능을 나타내고 노란색 바는 EMP에 CPF를 적용한 성능을 나타냄. 그래프를 보면 파란색 선과 같이 서브-블록의 크기가 커질수록 EMP의 성능 향상이 있다는 것을 확인할 수 있음 · 128KB의 서브-블록을 사용 시 4KB 페이지 단위 fetch에 대비하여 68%정도의 성능향상이 발생한 것을 확인 가능함. 또한 CPF를 적용하는 경우 주황색 선과 같이 EMP를 그냥 적용한 결과에 비해서 성능향상이 나타났으며, 128KB 서브-블록을 사용하는 경우 92%정도의 성능향상이 발생함 · 그림 74은 그림 73와 같은 결과에서 EMP 성능 대비 CPF를 적용하였을 때의 성능향상을 %로 나타낸 결과이며, 각 결과에서 CPF와 CSF의 비율과, CPF/CSF가 잘 활용되었는지에 대한 지표를 나타낸 그래프임. 앞서 설명한대로, CPF를 적용하였을 때, 해당 페이지에 대한 폴트 처리가 완료되었을 때, 서브-블록에 대한 DMA 읽기 완료가 처리가능하게 되면 해당 서브-블록에 대해서도 매핑 가능하도록 적용하였음. 따라서, CSF 비율은 해당 페이지가 폴트 처리이후 바로 서브-블록에 대해서도 처리가 가능했음을 나타내는 지표임. 주황색 선을 보면 서브-블록의 크기가 작을수록 CSF의 비율이 큰 것을 확인할 수 있는데, 서브-블록의 크기가 작으면 하나의 페이지의 fetch가 완료되면 서브-블록 내부의 다른 페이지들도 빠른 시간 내에 fetch가 완료된다는 것을 확인 가능함 · 회색 선은 CSF나 CPF가 쓸모가 있었음을 나타내는 지표로 CPF나 CSF로 하나의 페이지나 서브-블록만 fetch한 상태에서 해당 블록이 직접메모리에서 evict될 때까지 다시 접근되지 않은 경우를 표현하고 있음. evict될 때까지 다시 접근되지 않은 경우에는 하나의 페이지나 서브-블록만 fetch완료된 상황이기 때문에 블록의 다른 부분에 대한 처리를 진행하지 않아서 CPF나 CSF의 효과를 더 얻을 수 있는 장점이 있음. 회색 선을 보면 서브-블록의 크기와 상관없이 약 12-13%정도의 블록에 대해서 CPF나 CSF의 효과를 크게 얻을 수 있다는 것이 확인됨 · 성능을 보면 서브-블록의 크기가 커질수록 CPF나 CSF의 효과가 크다는 것을 확인할 수 있고, 128KB 서브-블록의 경우 CPF의 성능 향상이 떨어지는 것을 확인할 수 있는데, 이는 CPF에서 CSF로 변환시도 시 오버헤드가 발생하기 때문임. 따라서, 128KB 서브-블록

수행기관	개발 목표	개발내용 및 범위

에 CPF 단독으로 사용하는 남색 바를 보면 EMP 대비 성능의 향상이 약 21%정도 발생하는 것을 확인할 수 있음

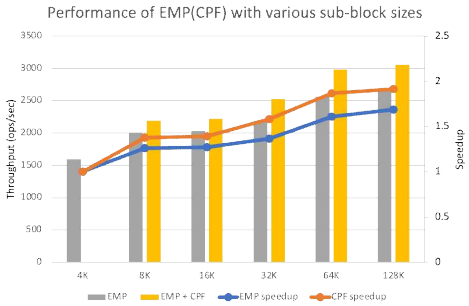


그림 73. EMP (CPF) 성능

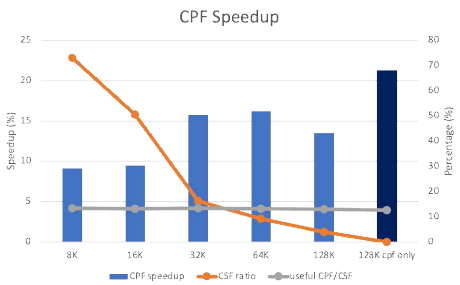


그림 74. CPF의 성능 향상 비율

• 원격메모리에 대한 접근 최적화를 위한 기술 개발

- 원격메모리에 대한 접근 최적화 기술 개발의 필요성
  · 원격메모리로 다른 노드의 메모리를 사용하는 경우 RDMA를 통해서 데이터를 전송하며, 직접메모리에서 evict되는 경우 원격메모리로 블록의 크기만큼의 데이터가 이동함. 이때, 블록의 크기만큼의 데이터를 보내는 요청의 경우 4KB 페이지 단위로 쪼개져서 요청이 보내지는데, 이 DMA 쓰기 요청의 완료를 진행하기 위해서는 하나의 블록 내부의 각 페이지 쓰기 요청들에 대한 완료를 모두 진행시켜야 함. 즉, 2MB블록을 원격메모리에 쓴다고 가정하면, 512개의 쓰기 요청에 대해서 완료를 진행시켜야 하기 때문에 CPU에 많은 자원의 사용을 초래함
- 원격메모리 대한 접근 최적화 기술 연구 (Chained writeback)
  · DMA 쓰기 요청에 대한 각각의 처리비용을 줄이기 위해서 DMA 쓰기에 대해서 한 번의 쓰기 완료 요청으로 하나의 블록에 대한 모든 페이지의 쓰기 완료 요청이 진행될 수 있도록 구현함. 해당 최적화 기술은 chained writeback이라고 명명함.
  · 이는 블록의 첫 번째 페이지의 쓰기 요청 작업에 다음 모든 페이지의 쓰기 요청을 체인형식으로 묶어서 관리하는 방법을 사용하여 첫 번째 페이지의 쓰기 요청 작업이 완료되는 경우 해당 블록의 모든 페이지에 대한 쓰기 요청의 완료작업을 수행하도록 구현함
- Chained writeback의 성능평가 결과
  · 그림 75는 EMP와 EMP에 chained writeback기능을 적용하였을 때의 MariaDB Insert작업 성능을 나타내는 그래프임. 250GB의 워크로드 환경에서는 원격메모리에 대한 접근이 자주 발생하기 때문에 chained writeback기능의 효과가 두드러지게 나타나는 것을 볼 수 있음. chained writeback을 적용하였을 때, 약 6%의 성능향상이 발생하는 것을 확인할 수 있음. 이는 DMA 쓰기요청에 대해서 하나의 블록당 하나의 DMA 쓰기완료만 진행하면 되기 때문에 CPU로의 부하가 적게 걸리어 생기는 효과임

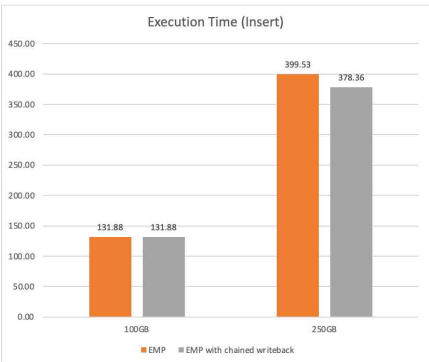
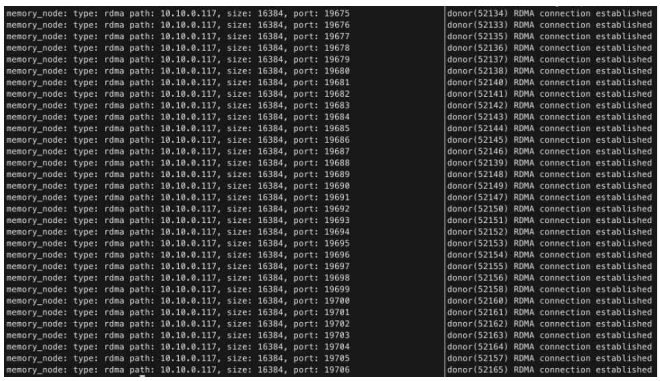
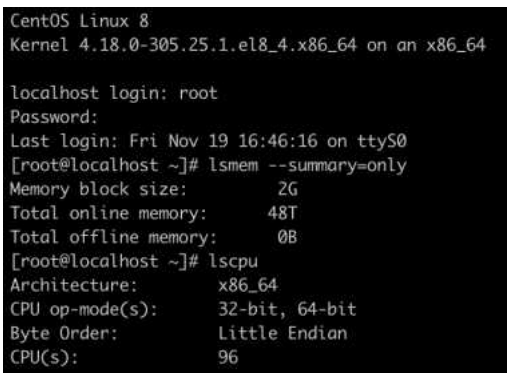
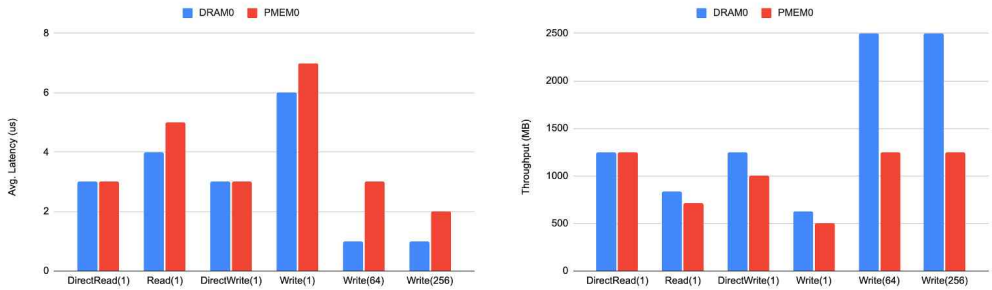


그림 75. chained writeback의 성능평가

• 원격 메모리 관리기와의 병렬적 메모리 연결 기능 구현

수행기관	개발 목표	개발내용 및 범위
		- 원격 메모리 관리기와의 병렬적 메모리 연결 구현 · 다른 노드의 메모리를 이용하는 메모리 확장의 경우 다른 노드에 원격 메모리 관리기라는 에이전트가 동작하여 컴퓨팅 노드에서 동작하는 EMP와의 RDMA 연결을 구성해주어야 함. 이때 원격 메모리 관리기와 컴퓨팅 노드에서 동작하는 EMP사이에 RDMA연결을 병렬적으로 수행하는 기능을 구현함 · EMP와의 병렬 RDMA연결을 위해서 원격 메모리 관리기에서 미리 RDMA로 사용될 메모리 영역을 초기화하고 등록하는 과정을 진행하고 EMP와의 연결을 기다림. 컴퓨팅 노드의 EMP가 연결을 요청하면 여러 개의 프로세스로 구성된 원격 메모리 관리기가 해당 EMP RDMA 연결 작업에 참여하여 병렬적으로 연결을 맺음. 또한 EMP와의 연결이 종료되면 다시 초기화 및 등록 없이 바로 연결이 해당 원격 메모리 관리기에 연결이 가능하도록 구현 - 병렬적 메모리 연결 검증 · 그림 76는 원격 메모리 관리기와 컴퓨팅 노드의 EMP사이의 병렬연결을 보여주는 스크린샷으로 왼쪽 세션이 컴퓨팅 노드의 EMP를 나타내며, 오른쪽 세션이 원격 메모리 관리기를 나타냄  그림 76. EMP와 원격 메모리관리기의 병렬연결 **• 수십TB 급 메모리 확장 기술 구현** - 수십TB 급 메모리 기반 가상머신을 구축하기 위해 Linux Kernel Virtual Machine (KVM)과 QEMU에 존재하는 한계를 극복 · Linux KVM은 메모리 디바이스 당 8TB를 최대 용량으로 함. 이를 극복하기 위해 여러 NUMA 노드를 가지는 가상머신을 생성 · QEMU에서 지원하는 BIOS는 물리머신의 운영체제에 따라 8TB~16TB까지의 메모리만 지원하도록 구현되어 있음. BIOS 자체의 메모리 버퍼 크기를 늘려줌으로써 48TB의 메모리를 갖는 가상머신 생성 성공  그림 77. 48TB의 메모리를 가지는 가상머신 생성 화면 - EMP 메타데이터의 경량화 기술의 필요성

수행기관	개발 목표	개발내용 및 범위
		· EMP가 관리하는 페이지에 대한 메타데이터를 관리하기 위해서 페이지 (4KB) 단위 당 메타데이터를 관리하게 되면, 48TB의 대규모 메모리 확장을 지원하고, 메타데이터의 크기로 64B를 사용하는 경우 768GB의 메모리를 메타데이터 관리에만 사용하는 상황이 발생하여 직접메모리에 대부분을 메타데이터 관리에만 사용하는 현상이 발생함. 따라서 이러한 메타데이터의 크기를 줄이거나 메타데이터가 관리하는 페이지의 크기를 늘리는 방법이 필요함 - EMP 메타데이터 경량화 기술 연구 · 본 과제에서는 메타데이터의 크기를 줄이기보다는 메타데이터가 관리하는 페이지의 크기를 늘리는 방법을 채택하고, 서브-블록 당 메타데이터를 관리하여 EMP가 관리하는 총 메타데이터의 크기를 줄임. 따라서 서브-블록 128KB로 설정하는 경우 24GB의 메모리만으로 메타데이터를 관리할 수 있도록 설정함 • **Active Remote Memory Server 설계** - 적극적인 원격메모리 관리에 대한 필요성 · 분리 메모리 (Disaggregated Memory) 환경에서 다른 노드의 메모리를 사용하는 기술들이 논문으로 많이 발표되고 있음. 하지만 DRAM의 용량은 한정적이고 Opatane DCPMM과 같은 비휘발성 메모리의 등장으로 큰 용량의 메모리를 제공해줄 수 있게 됨. 따라서, DRAM과 PMEM이 함께 존재하는 서버를 활용하는 것이 가능해졌고, 이 서버를 효율적으로 관리하기 위해서는 메모리를 제공해주는 서버에서도 자원 분배에 대한 처리를 진행하는 것이 중요함 · 기존 연구인 Infiniswap [NSDI'17]이나 FastSwap [EuroSys'20]의 경우 one-sided RDMA를 사용하여 빠르게 메모리노드에 접근하도록 설계함. 하지만 해당 논문들에서는 원격 메모리서버에 대한 제어가 불가능하고, 응용의 원격메모리 사용률이나 원격메모리 접근패턴에 대한 고려가 없었음. 또한, 혼성의 메모리로 구성된 메모리 노드의 경우 이종의 메모리에 대한 데이터 위치를 고려하여야 하기 때문에 이전의 기술들을 그대로 적용할 수 없음 · 그림 78과 같이 이종의 메모리에 접근할 때의 지연시간이나 처리량의 차이가 크기 때문에 각 메모리의 종류와 데이터의 접근패턴에 따라 데이터의 위치를 정하는 것이 성능에 큰 영향을 끼치게 됨. 그림에서 Direct라고 적혀있는 부분이 one-sided RDMA를 사용한 결과이며, 나머지는 RPC를 사용한 데이터 접근 성능을 나타냄 · DRAM의 경우 PMEM에 비해서 접근 성능은 더 좋으나 메모리 크기에 제약이 있기 때문에 자주 접근하는 데이터를 위치시키는 것이 좋고, PMEM의 경우 DRAM보다는 느리지만 메모리양이 많이 때문에 자주 접근하지 않는 데이터를 저장하는 것이 좋음. 또한, 적은 양의 데이터에 접근하는 경우 RPC에 비해 one-sided RDMA를 사용하는 것이 성능이 더 좋음을 확인함  그림 78. One-sided RDMA와 RPC 통신의 지연시간 및 처리량 성능평가 - 적극적인 원격메모리 관리 연구 (ARM 설계) · 본 과제에서는 메모리를 제공해주는 서버에서 적극적으로 메모리 관리를 수행하도록 구성하여 컴퓨팅 노드와의 데이터 접근 속도를 향상시키고, 혼성의 메모리를 사용하여 대규모의 메모리 확장이 효율적으로 가능하도록 메모리 노드 서버를 설계하고 이를 Active

수행기관	개발 목표	개발내용 및 범위

Remote Memory Server (ARM) 라고 명명함

· 그림 79은 ARM의 아키텍처를 표현한 그림임. 원격의 메모리노드에는 DRAM과 Optane DCPMM이 존재하여 이종의 메모리를 관리하여야 하며, 컴퓨팅 노드에서 메모리 요청이 들어온 경우 one-sided RDMA를 최대한 사용하여 빠르게 메모리 요청을 처리하지만 데이터의 사용패턴에 따라서 이종의 메모리에 적절히 위치시키는 작업을 ARM 백엔드에서 수행시킴. 이는 컴퓨팅 노드와의 통신을 통해서 해당 데이터에 대한 원격 메모리상의 위치를 조정하여 구현함. 컴퓨팅 노드는 기존 연구인 FastSwap을 이용해서 원격의 메모리와 통신을 진행하도록 구현함

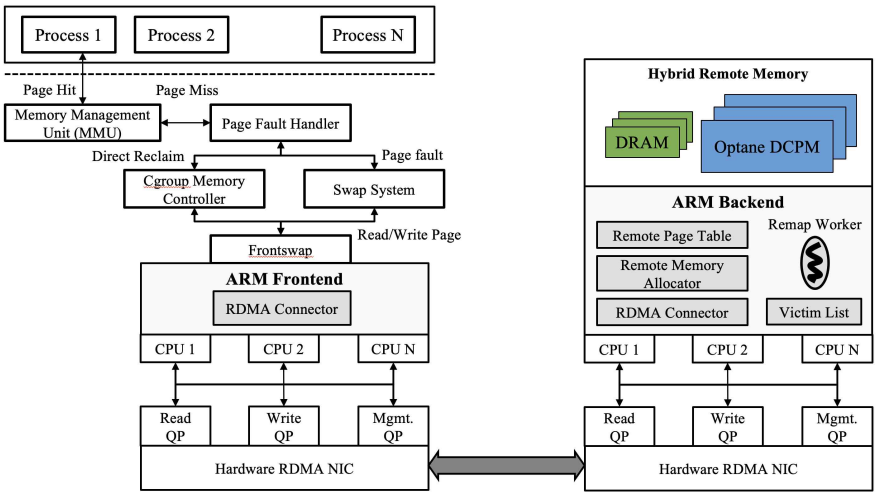


그림 79. Active Remote Memory (ARM) 아키텍처

· 그림 80는 ARM의 RDMA 프로토콜을 나타냄. 컴퓨팅 노드에서 원격 메모리에 대한 접근이 읽기인 경우에는 one-sided RDMA를 활용하여 빠르게 데이터에 접근하여 읽어갈 수 있도록 구현하고, 쓰기인 경우에는 two-sided RDMA를 사용해서 ARM 백엔드에 쓰기 명령을 보내는 형태를 사용함. 이는 데이터에 따라 서로 다른 메모리에 위치할 수 있도록 ARM 백엔드가 결정하고, 해당 명령이 처리되면 컴퓨팅 노드에 ack와 함께 정해진 데이터의 위치를 전송해 줌으로써 컴퓨팅 노드의 응용은 바로 해당 데이터의 위치에 다음 읽기 명령을 수행하도록 도와줌

· 또한 ARM에 원격 메모리 관리 명령을 보낼 수 있도록 구현함. 해당 명령에 따라서 현재 원격 메모리에 존재하는 데이터의 사용 패턴에 따라서 데이터의 위치를 옮기는 작업 (재배치 작업)을 수행할 수 있도록 설계함

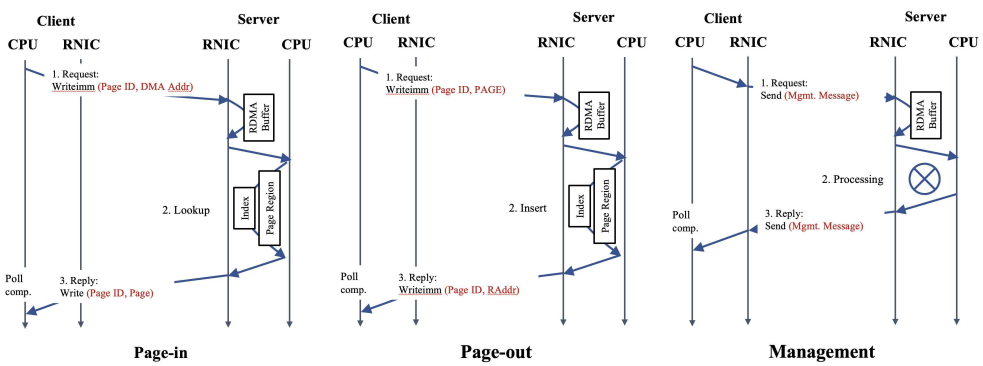
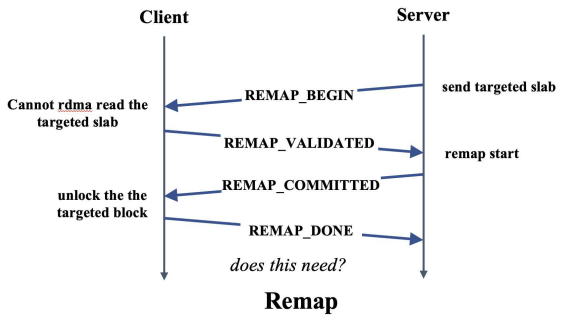
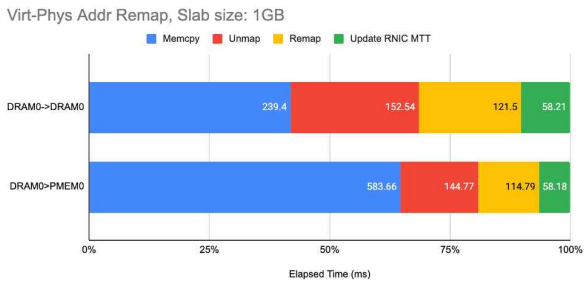


그림 80. ARM RDMA 프로토콜

· 원격메모리 관리 명령을 통해서 ARM이 원격 메모리에 데이터를 이주시키는 작업을 수행

수행기관	개발 목표	개발내용 및 범위
		하는 경우 컴퓨팅 노드의 데이터 접근 명령이 진행되는 것을 막아야함. 이는 그림 81의 프로토콜을 통해서 컴퓨팅 노드와 원격메모리 노드사이의 접근 제어를 수행함 · 먼저 ARM은 컴퓨팅 노드에 데이터 재배치가 수행되고 있으니 데이터 접근을 수행하지 말라는 명령을 보내고, 이를 바탕으로 컴퓨팅 노드는 재배치되고 있는 영역에 대해서 메모리 접근을 수행할 수 없음. 이후 재배치가 완료되었는지 원격메모리노드에 확인을 요청하면 ARM은 재배치가 끝난 후 컴퓨팅 노드에 재배치가 완료되었다는 메시지를 보내어 다음 데이터 접근을 계속하도록 만들어 줌  그림 81. ARM 재배치 프로토콜 - 적극적인 원격메모리 관리 중 데이터 재배치에 걸리시는 시간 측정 · 그림 82은 ARM의 데이터 재배치가 걸리는 시간을 나타내는 그래프임. 위쪽 그래프가 DRAM에서 DRAM으로의 재배치를 나타내는 그래프이고, 아래쪽 그래프가 DRAM에서 PMEM으로의 재배치를 나타냄 · 대부분의 시간이 memcpy()를 통해서 데이터가 이동하는데 걸리는 시간이고, 새로이 데이터를 저장하기 위해서 remap()을 수행하고 이전의 매핑을 지우는데 걸리는 시간이 필요함. 또한, one-sided RDMA 읽기를 계속적으로 지원해주기 위해서 NIC에 저장되어있는 Address Translation Page Table (MTT)를 업데이트 해주어야 함  그림 82. ARM 재배치의 성능평가
	③ 대규모 메모리 기반 데이터 분석 가속 알고리즘 연구	• 메모리 스케일러블 유전체 분석 가속 기법 설계 - 개념: 1차년도에 연구한 대규모 메모리 기반 유전체 분석 가속 기법 연구를 확장하여, 시스템의 유휴 메모리 양에 따라 유전체 분석을 가속하는 기법을 설계하고, 추가적인 가속 기법들을 연구 - 개요: BWA-MEM2에 아래와 같은 가속 기법들을 적용 · In-memory Index Store: 1차년도에 개발한 기법과 동일. 추가된 가속 기법에도 적용 · Selective Index loading: 시스템의 유휴 메모리 양에 따라 성능을 최적화할 수 있도록 인덱스를 선택적으로 로딩. Exact Match Filter의 경우에는 부분적으로 로딩 가능 · Exact Match Filter: 1차년도에 개발한 Perfect Match를 개선. 인덱스의 크기를 약 1/4로 줄임. 인덱스의 일부분만 로딩해서도 성능 가속 효과를 얻을 수 있도록 개선. 로딩된 인덱스의 시드 길이뿐만 아니라 시드 길이 이상의 모든 염기서열 리드에 대해 성능 가속

수행기관	개발 목표	개발내용 및 범위

효과를 얻을 수 있도록 개선

· FM-index Accelerator: 1차년도에 개발한 것과 동일한 것을 적용

· Enumerated Radix Tree (ERT): 미시건 대학에서 ISCA 2021년도에 발표한 논문의 내용을 적용. In-memory Index Store 및 selective index loading을 사용하도록 수정

· Pipeline Optimization: 프로그램 연산 과정이 빨라짐에 따라 새로운 병목이 된 입출력 부분에 대한 부분을 최적화. 입출력 관련 메모리 할당을 큰 단위로 하게 하고, 입력파일을 선행적으로 읽은 read-ahead 쓰레드를 구현. SAM 파일 포맷에 대한 쓰기를 배치로 처리. 파이프라인의 쓰레드 숫자를 조절하여 병목 현상 완화

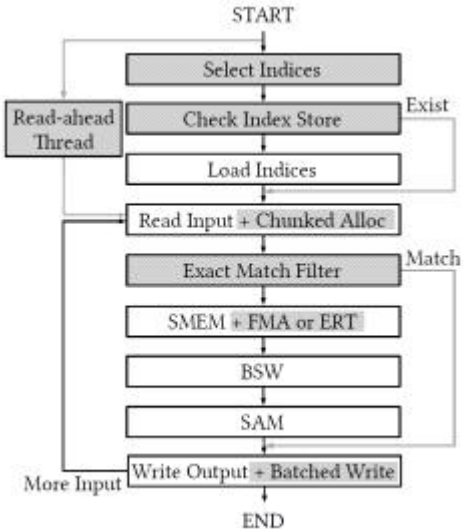


그림 83. BWA-MEM-SCALE의 흐름도

표 16 BWA-MEM-SCALE의 단계별 동작. 이탤릭체로 표시된 부분이 BWA-MEM2와의 차이.

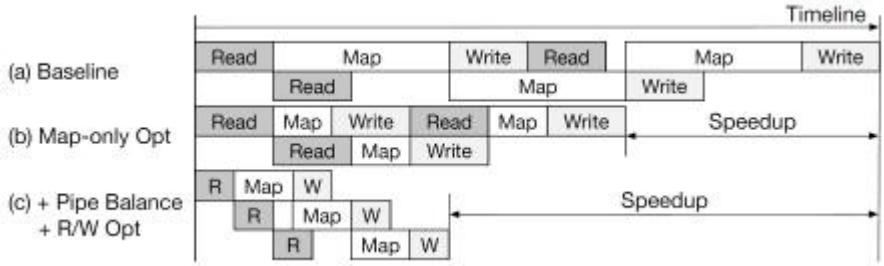
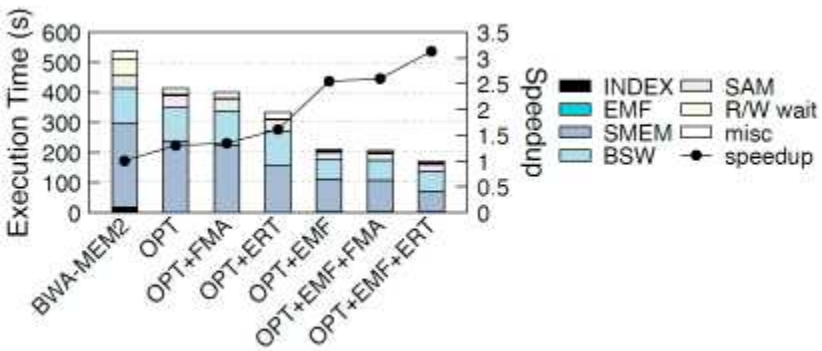
Stage	Step	Description
INDEX	*1) Selective Indices*	*For a given memory capacity, select best indices to load.*
	2) Check Index Store	*Check Index Store Check that selected indices exist on the in-memory index store. If exist, skip the step 3.*
	3) Load Indices	Load the *selected* indices.
READ	4) Read input *+ Chunked Alloc.* *+ Read-ahead Thread*	Read FASTQ file and parse the content *using chunked allocation for the buffer. A separate read-ahead thread prefetches the FASTQ file from the disk.*
MAP	*5) Exact Match Filter*	*Check that a read is exactly matched in its full length to the reference genome. If matched, skip the step 6~8 and directly generate SAM strings.*
	6) SMEM *+ FMA or ERT*	Find Super-Maximal-Exact-Matching (SMEM) parts using FM-index *or ERT [24]. When using FM-index, FM-index Accelerator (FMA) can be applied.*
	7) BSW	Score the possible mappings using Banded-Smith-Waterman (BSW) algorithm.
	8) SAM	SAM Generate the result string in SAM file format.
WRITE	9) Write Output *+ Batched Write*	Write SAM formatted strings to disk *in batches.*

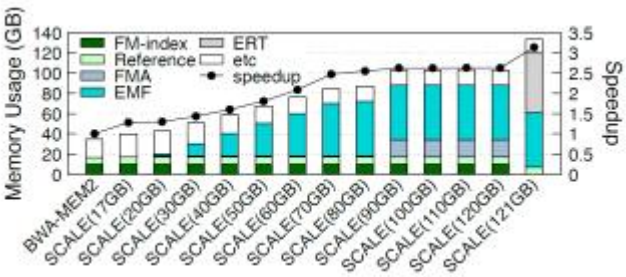
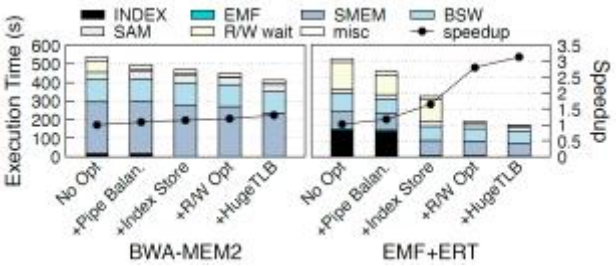
- Exact Match Filter: 1차년도에 개발한 Perfect Match를 개선. 기본 개념은 Perfect Match와 같음

· 인덱스 크기 감소: 시드를 인덱스 내에 저장하지 않고 다른 필수 인덱스에 포함된 참조

수행기관	개발 목표	개발내용 및 범위

		유전체를 참조하도록 함. 엔트리 내 변수들의 크기를 최소한의 필요 크기로 조절. 이를 통해 인덱스 크기를 약 1/4로 줄임. 225GB → 55GB
		· 부분 인덱스 로딩: 인덱스를 앞에서부터 로딩할 경우, 전체를 로딩하지 않고도 성능 가속 효과를 얻도록 함. 염기서열 리드로부터 해쉬값을 계산한 다음, 해쉬값에 해당하는 엔트리가 메모리에 로딩되어 있지 않은 경우, 기존 BWA-MEM2와 같은 방식으로 연산을 수행하도록 함. Exact Match Filter는 성능 가속 효과만 있을 뿐, 수행하지 않더라도 정확도에는 문제가 생기지 않음
		· 시드 길이 제한 삭제: 시드 길이 이상의 모든 염기서열 리드에 대해 성능 가속 효과를 얻을 수 있도록 개선. 염기서열 리드의 길이가 메모리에 로딩된 Exact Match Filter의 인덱스 시드 길이보다 긴 경우, 시드 길이까지만으로 Exact Match 여부를 판별한 후, 참조 유전체에서 해당 위치의 뒷부분을 비교하여 최종적인 Exact Match 여부를 판단함. 입력파일 내의 리드 길이가 다양하거나 quality-based trimming 등이 적용될 경우 유용한 기법
		- Selective Index Loading: 시스템의 유휴 메모리 양이 지정되면 그에 맞추어 성능을 최적화할 수 있도록 인덱스들을 선택해서 로딩. 프로그램 실행 중에는 로딩된 인덱스를 활용한 기법들만 추가될 수 있음. 인덱스들의 우선 순위는 아래와 같음
		1) Per-Base FM-index Accelerator (PB-FMA): FM-index를 사용하는 BWT 알고리즘을 11bp에 대해 중간과정까지 모두 기록해놓은 인덱스. 512MB의 인덱스로 전체 프로그램 실행을 약 2% 가속할 수 있으므로, 메모리 용량 당 가속이 가장 우수
		2) Exact Match Filter (EMF): 부분 로딩이 가능하며, 추가 1GB당 약 1.5% 정도의 성능 가속을 얻을 수 있음
		3) Final-Only FM-index Accelerator: FM-index를 사용하는 BWT 알고리즘을 15bp에 대해 최종 결과만을 기록해놓은 인덱스. 16GB의 인덱스로 전체 프로그램 실행을 약 2% 가속할 수 있음
		4) Enumerated Radix Tree (ERT): 60GB의 인덱스로 전체 프로그램 실행 속도 30% 향상 가능. 또한, ERT를 적용하면 SMEM 스텝이 완전히 대체되므로 PB-FMA와 FO-FMA는 로딩이 취소됨
		
		procedure SELECT_INDEX(*mem*)
		INCLUDE_INDEX(*FM-index,Reference*)
		mem ← *mem*− *size(FM-index,Reference)*
		if *mem* ≥ *size(PB-FMA)* **then**
		INCLUDE_INDEX(*PB-FMA*)
		mem ← *mem*− *size(PB-FMA)*
		end if
		if *mem* > *size(multi_location_table)* **then**
		INCLUDE_INDEX(*multi_location_table*)
		mem ← *mem*− *size(multi_location_table)*
		N ← *mem*/*size(EMF_hash_entry)*
		INCLUDE_INDEX_PART(*EMF_hash_entry, N*)
		mem ← *mem* − *N*×*sizeof(EMF_hash_entry)*
		end if
		if *mem* ≥ *sizeof(FO-FMA)* **then**
		INCLUDE_INDEX(*FO-FMA*)
		mem ← *mem*− *sizeof(FO-FMA)*
		end if
		if *mem*+*size(FM-index,PB-FMA,FO-FMA)* ≥ *size(ERT-index)* **then**
		EXCLUDE_INDEX(*FM-index,PB-FMA,FO-FMA*)
		INCLUDE_INDEX(*ERT-index*)
		end if
		end procedure
		
		그림 84. BWA-MEM-SCALE의 인덱스 로딩 알고리즘
		
		- Pipeline Optimization: FMA, EMT, ERT 등의 가속 기법이 적용됨에 따라 map stage가 빨라지면서 read, write stage가 병목이 됨. 다음의 기법들을 적용하여 read, write stage로 인한 병목 현상을 완화

수행기관	개발 목표	개발내용 및 범위
		· Read-ahead Thread: 입력 파일에 대한 디스크 I/O만 담당하는 별도의 쓰레드를 추가. 기존의 read stage에서는 parse 및 메모리 할당만 수행 · Chunked memory allocation: 기존에는 각 리드마다 별도로 메모리를 할당 받던 것을 배치 단위로 메모리를 할당 받도록 수정 · Batched write: 기존에는 각 리드마다 SAM 포맷의 문자열을 만들어 각각 디스크로 출력하던 것을, 배치 단위로 문자열을 만들어 한 번에 디스크로 출력하도록 수정 · Pipeline balancing: mapping을 하는 쓰레드 개수를 줄여서 read와 write stage가 항상 CPU를 사용할 수 있도록 함. CPU 자원 경쟁에 따른 병목 현상 완화 · 아래 그림은 Pipeline optimization에 의한 효과를 도식화한 것. Map stage만 최적화하는 것보다 큰 성능 향상을 얻을 수 있다는 것을 보임  그림 85. Read/write 최적화 및 Pipeline balancing의 효과 • 메모리 스케일러블 유전체 분석 가속 기법 성능 평가 - 실험 환경 · 시스템: Intel Xeon Gold 6230 CPU 2개, 384GB 메모리 · 데이터셋: BWA-MEM2 논문에서 사용된 입력파일 중 3가지와 Genome-in-a-Bottle consortium의 HG001~HG004 데이터셋을 사용 · 참조유전체: hg38의 최초 버전 사용 - 그래프 라벨 설명 · OPT: in-memory index store (1GB page), pipeline optimization · FMA: Per-Base FM-index Accelerator, Final-Only FM-index Accelerator · ERT: Enuemrated Radix Tree · EMF: Exact Match Filter - 제안된 가속 기법들의 성능 분석 · In-memory index store와 pipeline optimization만으로 약 30% 성능 향상 · FMA, ERT, EMF가 각각 4%, 31%, 125% 성능 향상. FMA와 ERT는 SMEM 단계만을 가속하는 것이어서 성능 향상이 제한적  그림 86. 제안된 가속 기법들의 성능 분석 - 유휴 메모리 용량을 활용하는 BWA-MEM-SCALE의 성능 향상 · 인덱스에 활용할 수 있는 시스템의 유휴 메모리 용량을 제시하면 selective index

수행기관	개발 목표	개발내용 및 범위
		loading 기법을 활용해 유전체 서열 정렬 성능을 최적화하는 방식의 성능 향상 분석 · 아래 그래프에서 막대 그래프는 각 인덱스들의 메모리 용량, 선 그래프는 BWA-MEM2 대비 성능 향상 폭을 나타냄. X축의 괄호 안의 용량은 사용자가 제시한 메모리 용량을 나타냄 · 메모리 용량에 따라 인덱스들이 선택되면서 메모리 용량과 비례하여 성능이 향상됨을 알 수 있음. 단, 90GB~120GB 구간은 ERT가 선택되기엔 용량이 부족하나 다른 가속 기법들을 모두 적용된 후여서, 메모리 사용량 및 성능이 변화가 없음  그림 87. 유휴 메모리를 활용하는 BWA-MEM-SCALE의 성능 향상 - Pipeline 최적화 기법들의 성능 분석 · 아래 그래프는 BWA-MEM2에 pipeline 최적화 기법들을 일부 혹은 전체 적용했을 때의 성능 변화와 EMF와 ERT가 적용된 상태에서 pipeline 최적화 기법들을 일부 혹은 전체 적용했을 때의 성능 변화를 나타냄 · BWA-MEM2에 기법들을 적용할 경우, pipeline balancing이 약 9%, in-memory index store가 5.7%, read/write 최적화가 4.8%, hugeTLB(1GB)가 10.8%의 성능 향상을 보임 · EMF와 ERT가 적용된 경우, mapping stage가 가속된 상태에서 다른 stage들이 가속되면서 더 큰 성능 향상을 보임. Pipeline balancing이 15.5%, read/write 최적화가 115%의 성능 향상을 보임. 또 대규모 인덱스를 사용하면서 in-memory index store만 적용해도 47%의 성능 향상을 보이고 hugeTLB도(1GB)은 33%의 성능 향상을 보임  그림 88. Pipeline 최적화 기법들의 성능 분석 - 개인 유전체 분석 소모 시간 분석 · 개인의 유전체 서열 정렬을 수행하기 위해서는 300~500GB에 달하는 시퀀싱 데이터를 처리해야 함. 해당 시나리오에 대한 성능을 분석하기 위해 Genome-in-a-Bottle의 HG001~HG004 데이터셋에 대한 성능 분석을 수행 · 아래 그래프는 HG001~HG004에 대한 서열 정렬을 수행할 때 BWA-MEM2 사용시 걸리는 시간화 BWA-MEM-SCALE을 사용했을 때 걸리는 시간을 비교 · BWA-MEM-SCALE(17GB)는 BWA-MEM2에 비해 추가적인 인덱스를 사용하지 않는데도 2배에 가까운 성능 향상 폭을 보임. 이는 데이터셋 내의 개별 FASTQ 파일 크기가 작아서 in-memory index store와 pipeline 최적화의 효과가 크게 나타나기 때문 · BWA-MEM-SCALE(121GB)는 3.15~3.34배의 성능 향상을 보임

수행기관	개발 목표	개발내용 및 범위

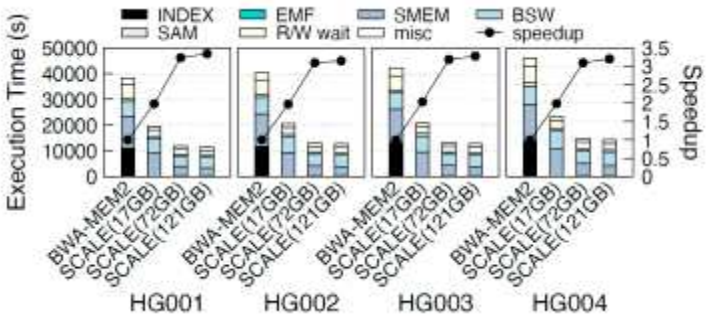


그림 89. BWA-MEM-SCALE 적용시 개인 유전체 분석 소모 시간

• 대규모 메모리 기반 양자 컴퓨팅 시뮬레이션 가속 기법

- 개념: TB급 메모리가 필요한 양자 컴퓨팅 시뮬레이션을 2단계 메모리 구성과 데이터 압축 기법을 통해, 적은 수의 노드만으로 빠른 속도로 수행
- 양자 컴퓨팅 시뮬레이션: 양자 게이트를 통과하는 양자들의 상태를 모사하는 것. 양자의 얽힘과 중첩 상태를 고려해야 하기 때문에 상태 모사에 필요한 데이터가 기하급수적으로 늘어남. n개의 큐빗을 시뮬레이션하기 위해서는 2^(n+4) 바이트의 데이터가 필요하며, 45큐빗 시뮬레이션을 위해서 0.5페타바이트가 필요
- 2단계 메모리 기반 양자 컴퓨팅 시뮬레이션: 대규모 메모리가 필요함에 따라 기존 논문에서는 45큐빗 시뮬레이션을 위해 슈퍼컴퓨터를 활용. 하지만 슈퍼컴퓨터를 계속 활용하기는 어려우므로, ETRI에서는 고성능SSD를 2단계 메모리로 하는 양자 컴퓨팅 시뮬레이션 프로그램인 양자가상머신 개발

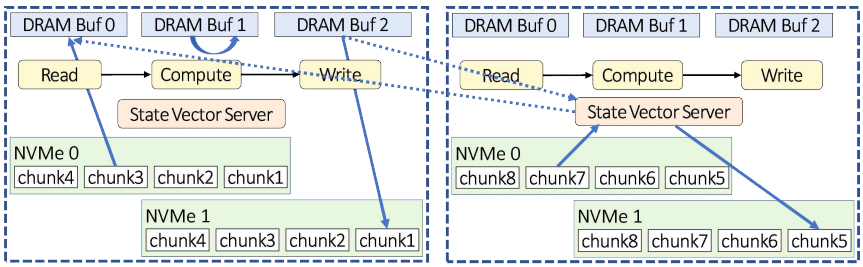
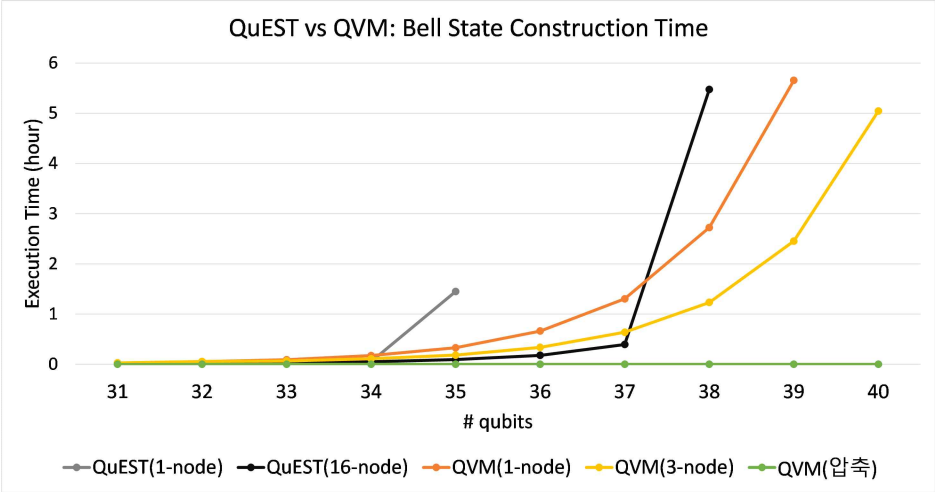


그림 90. 양자 가상머신 개념도

- 양자 상태 압축 기법: 본 과제에서는 양자 가상머신의 속도를 높이기 위해 양자 상태를 압축하는 기법을 개발
  · Run-length 기반 압축 기법 적용: 양자의 상태를 나타내는 상태 벡터의 초기 상태는 값의 종류가 1~2종류로 제한된 경우가 대부분. 그리고 양자 게이트를 통과할 때마다 값의 종류가 늘어남. 따라서 값과 해당 값의 갯수로 표현하는 run-length 기반 압축 기법만으로도 상당한 압축 효과를 얻을 수 있음
  · Chunk 단위 압축: 양자 가상머신은 chunk 단위로 2단계 메모리 관리와 연산을 수행하기 때문에, 압축도 chunk 단위로만 수행. 전체적인 압축율을 떨어뜨리기는 하지만, 2단계 메모리 관리와 연산의 최적화된 성능을 그대로 유지할 수 있다는 장점이 있음
  · 압축 상태로 연산 수행: 압축 저장된 양자 상태 벡터의 압축을 풀지 않고 압축된 상태로 연산하여 압축된 상태로 저장하는 기법을 구현. Run-length 기반의 단순한 압축 기법을 사용하고 chunk 단위의 압축을 사용했기 때문에 가능. 일반적인 압축 기법 사용시에 필요한 압축 해제 오버헤드도 없앨 뿐만아니라 필요한 연산량 자체를 줄이는 장점도 있음
- 성능 평가 환경
  · 20노드 클러스터: 56코어 10노드 + 80코어 10노드 (80코어 노드 우선 사용)

수행기관	개발 목표	개발내용 및 범위
		· 노드 당 메모리: 384GB · 상태 벡터 저장용 고성능SSD: 노드당 2TB 16개, 총 29.1TiB - 성능 평가 결과 · Bell State Construction 수행 시간 측정: InitZero 1번, Hadamard 1번, CNOT n-1번으로 구성 (n: 큐빗 개수) · 최신 양자 컴퓨팅 시뮬레이터인 QuEST와 성능 비교. QuEST는 MPI를 통해 분산 실행은 할 수 있지만 모든 상태벡터가 메모리에 올라와 있어야 하기 때문에 실행할 수 있는 큐빗 수가 제한됨 · 아래 그림을 보면, QuEST는 16개 노드를 사용하더라도 37개까지만 시뮬레이션 가능. 38개 큐빗의 경우에 swap을 써서 시뮬레이션을 하기 때문에 실행 시간이 대폭 증가. 반면, 양자가상머신은 1개 노드만으로도 39큐빗의 시뮬레이션 가능. · 본 과제에서 개발한 압축기법을 적용하면 QuEST에 비해 훨씬 짧은 수행 시간을 보임. 35큐빗의 경우 QuEST 16노드에 비해 310배 성능 향상  그림 91. Bell State Construction Time 비교 : QuEST .vs. QVM • 대규모 그래프 분석 연구 - 개념: 1 trillion edge 규모의 그래프 분석 응용에 대한 연구 - 시스템 구성: 1 trillion edge는 데이터 규모가 크기 때문에 고성능의 장치 및 많은 양의 메모리를 갖춘 시스템 구성이 필요 · CPU: Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz X 2개 (총 112코어) · DRAM: 1024GB · 그래프 저장용 SSD: Intel SSD D7-P5510 (7.68TB) X 7개 · 그래프 분석용 SSD: Intel SSD D7-P5510 (7.68TB) X 16개 - 소프트웨어 구성 · 운영체제: CentOS 8.2, Linux 4.18.0-193.el8 · 그래프 생성 플랫폼: TrillionG (SIGMOD'17) · 그래프 분석 플랫폼: Graphene (FAST'17) - 개발 내용 · TrillionG를 사용해 Vertex 4,294,963,201개, Edge 976,767,207,659 개(약 1 trillion)의 그래프 생성 · Graphene은 효과적인 연산을 위해 데이터 전처리 과정을 요구. 그러나 전처리 과정이 큰 크기의 데이터에 대해 비효율적으로 구현되어 있어, 효율화된 파싱, 파일 단위의 병렬성 활용, 메모리 사용 효율화 등을 통해 성능을 개선 · Graphene을 통해 입출력 성능을 최대화하기 위한 파라미터를 사용할 경우 데드락으로 인해 프로그램이 진행되지 않는 문제점을 발견하여 수정 - 성능 실험: BFS 알고리즘 수행 결과, 1735.49초 소모

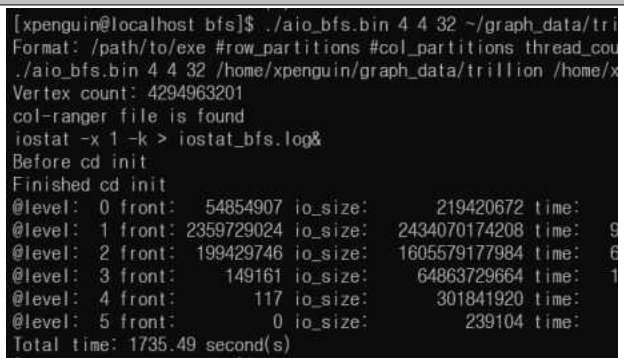
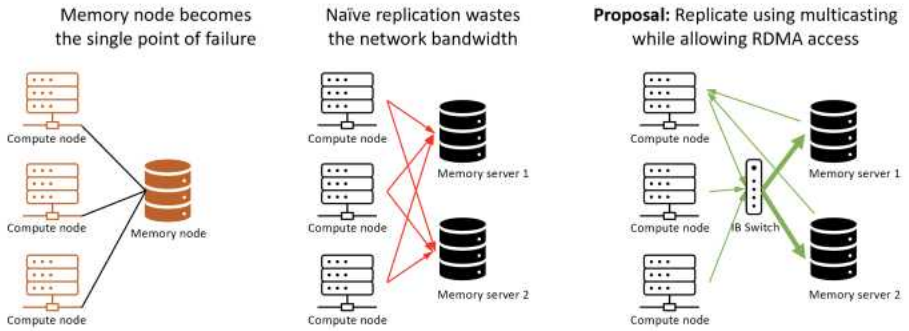
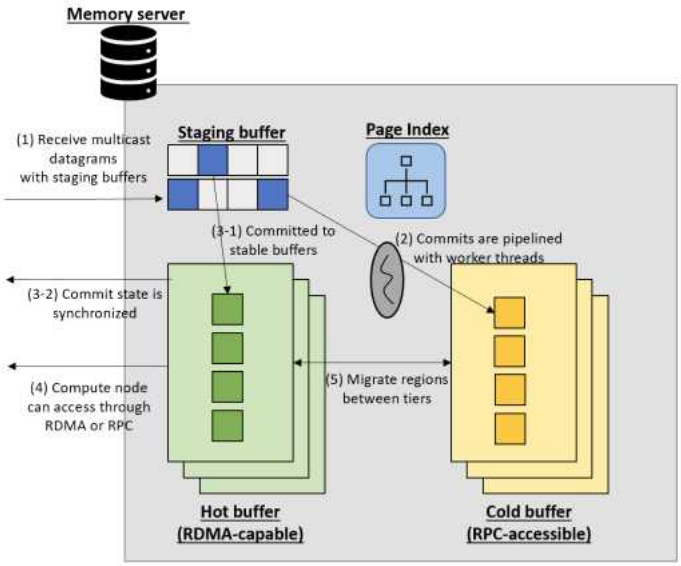
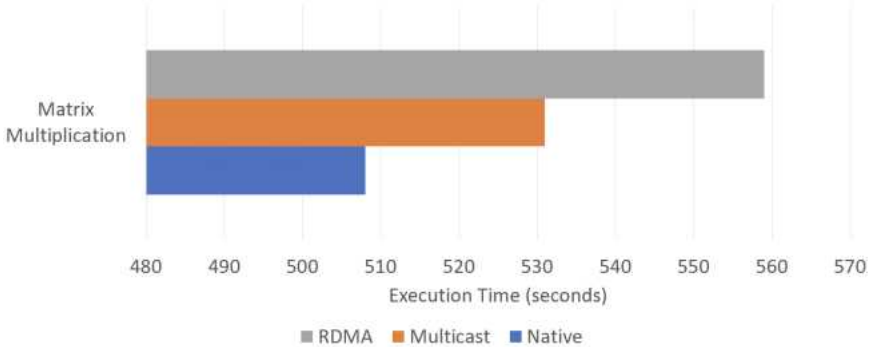
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>


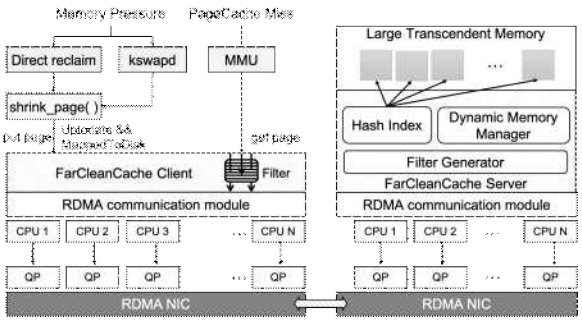
그림 92. 그래프 분석 성능 실험
</td></tr>
<tr><td>아주대학교
(공동1)</td><td>메모리 중심 컴퓨팅 시스템 기반 쓰레드 병렬화 기술 개발</td><td>
• 공유 데이터로의 접근 성능 개선 및 장애 대응을 위한 멀티캐스팅 활용 연구
- 연구 개요
· 메모리 중심 컴퓨팅 시스템의 주요 적용 대상인 클라우드 및 데이터센터 워크로드에서는 방대한 데이터를 많은 노드가 동시에 접근하는 형태가 빈번함. 따라서 메모리 중심 컴퓨팅 시스템을 통해 구축된 물리적인 제약이 없는 방대한 메모리 공간 내에서 데이터를 여러 노드에 복제하여 저장하고, 필요시 노드 근처에 위치한 데이터를 접근하는 방식으로 시스템의 성능 향상을 기대할 수 있음
· 또한 메모리 중심 컴퓨팅 시스템 환경에서 다수의 컴퓨팅 노드의 데이터를 저장하는 메모리 노드에 장애가 발생하는 경우 연관된 다수의 컴퓨팅 노드가 영향을 받게 되므로 메모리 노드의 장애에 대응하기 위해 데이터를 복제하여 저장해 둘 필요가 있음
· 기존의 연구들은 대부분 메모리 노드에 데이터를 저장하는 시점에 컴퓨팅 노드가 다수의 요청을 통해 여러 노드에 내용을 복제하는 형태인데, 이 경우 컴퓨팅 노드와 메모리 노드 사이의 데이터 경로에 많은 트래픽이 발생하고 데이터 이동 및 처리에 많은 오버헤드가 발생함
· 본 연구는 메모리 중심 컴퓨팅의 장점을 활용하고 메모리 노드의 장애 대응을 위해 InfiniBand의 multicasting 기능을 활용하여 메모리 데이터를 다수의 노드에 효과적으로 복제하고 활용하는 기술을 연구 개발하였음

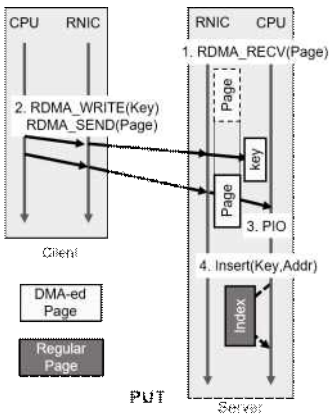
- 연구 개발 내용
· 기존의 메모리 중심 컴퓨팅을 구현하는 연구와 기술들은 낮은 지연시간과 높은 대역폭을 보장하는 InfiniBand의 RDMA 기술을 주로 사용하는데, 이 기술은 단일 대상 주소로의 데이터 전송만 지원하여 데이터 복제를 위해서는 하나의 복제당 한 번의 RDMA write가 필요하게 되며, 복제의 개수가 늘어날수록 컴퓨팅 노드의 outbound 네트워크에 큰 부하를 발생시키게 됨
· 이러한 outbound network traffic 문제를 해결하고자 LegoOS나 InfiniSwap들은 노드들을 순차적으로 연결하는 chain을 구성하여 앞의 노드가 수신한 데이터를 뒤의 노드에게 전달하며 복제를 수행하는 primary replication 방식을 사용함. 이 경우 outbound 네트워크의 트래픽은 줄일 수 있으나 복제 경로가 길어지고 각각의 전송에 한 번의 RDMA write가 필요함에 따라 원격 메모리로의 데이터 전송 성능이 크게 저하되는 문제점이 발생함. 따라서 높은 수준의 데이터 복제도를 유지하는데 어려움이 있음
· 한편 InfiniBand는 다수의 노드가 그룹을 형성하고 이 그룹을 목적지로 메시지를 전송하면 스위치 수준에서 그룹 멤버에게 데이터를 복제하여 전송하는 멀티캐스팅(multicasting) 기능을 제공함. 멀티캐스팅을 통해 데이터를 전송하는 경우 최초 발송 서버에서 스위치 사이에는 단 하나의 메시지만 전송되며, 이 메시지는 스위치에서 복제되어 다른 그룹 멤버에게 각각 전송됨. 따라서 복제를 위해 전송해야 하는 데이터의 양을 최소화할 수 있으며, InfiniBand의 link 수준에서 데이터가 복제되기 때문에 높은 성능을 유지할 수 있음
· 하지만 멀티캐스팅을 통해 데이터를 전송하기 위해서는 전송 도중 데이터가 손실될 수
</td></tr>
</table>

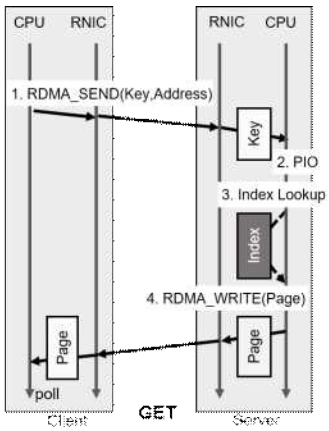
수행기관	개발 목표	개발내용 및 범위
		있는 패킷 단위의 전송 방식인 unreliable datagram (UD) 방식을 통한 연결만 사용할 수 있음. 기존의 RDMA를 통한 데이터 접근은 연결 기반의 전송 방식인 reliable connection (RC)을 통해서만 처리가 가능하며 UD 연결에는 사용할 수 없음 · RDMA를 통해 데이터를 읽고 쓰는 경우 통신 계층에서 추가적으로 발생하는 out-of-band 데이터가 없음. 따라서 데이터를 최종적으로 저장할 버퍼를 통신 계층의 데이터 수신용 버퍼로 활용하여 메모리 copy 횟수를 최소화 할 수 있음. 하지만 멀티캐스트를 통해 발신된 메시지를 수신하는 경우 해당 메시지의 전송 경로에 대한 정보 (routing information)이 당초 발신 메시지의 앞 부분에 헤더의 형태로 첨부되기 때문에 메모리 데이터를 저장할 버퍼를 데이터를 수신하는 버퍼로 바로 사용하는 구조를 가질 수 없음 · 이러한 접근 방법들의 장단점을 보완하기 위해 본 연구는 멀티캐스팅을 통해 데이터의 복제를 수행하고, 메모리 서버에서 수신한 데이터는 RDMA가 가능한 메모리 영역으로 비동기적으로 파이프라인 형태로 처리되며 이동되는 서버 구조 및 버퍼 관리 기술을 개발하였음  그림 93. 기존 메모리 중심 컴퓨팅 시스템의 한계와 이를 해결하기 위한 multicasting 적용 방안 · 메모리 노드는 시스템 초기화 과정에서 지정된 멀티캐스트 그룹에 join하고, 멀티캐스트 메시지를 수신하기 위한 staging buffer를 구성해 놓음. 또한 장기적으로 데이터를 저장하고 컴퓨팅 노드가 RDMA를 통해 직접 접근할 수 있도록 cold buffer를 초기화. 개략적으로 데이터는 staging buffer를 통해 수신되고, worker threads에 의해 cold buffer로 정리되는 형태로 처리됨 · 컴퓨팅 노드 또한 시스템 초기화 과정에서 지정된 멀티캐스트 그룹에 합류함. 컴퓨팅 노드는 메모리 노드에게 멀티캐스트를 통해 메모리 데이터를 전송함. 컴퓨팅 노드는 주기적으로 (현재는 1,024개의 멀티캐스트마다) 메모리 노드에게 지금까지 staging buffer에 수신된 내용을 cold buffer로 옮겨 commit이 되도록 요청함 · 컴퓨팅 노드에서 메모리 서버에 있는 데이터로의 접근은 commit이 일어나기 이전에 발생하는 접근과 commit 이후에 발생하는 접근으로 나뉘어짐. 데이터가 쓰인 이후 commit이 이루어진 경우 해당 데이터는 cold buffer에 존재하게 됨. 따라서 컴퓨팅 노드는 RDMA를 통해 cold buffer의 데이터를 직접 접근이 가능함 · 컴퓨팅 노드에서 commit이 일어나기 전에 해당 데이터가 다시 필요한 경우, 해당 데이터는 메모리 노드의 staging buffer에 있음. 하지만 staging buffer는 RDMA를 통해 접근할 수 없어 컴퓨팅 노드에서 직접 접근할 수 없음. 이러한 경우를 위해 컴퓨팅 노드에는 아직 commit이 이루어지지 않은 데이터를 별도의 버퍼에 보관하고 있다가 해당 데이터의 요청이 발생하는 경우 이를 반환하도록 구현 · 이러한 구현 방법을 통해 staging/commit 단계에서 발생할 수 있는 동기화 문제는 해결이 가능하지만, 드물게 발생하는 경우를 위해 모든 메모리 쓰기 요청에 대해 한 번의 메모리 카피와 버퍼링이 오버헤드로 발생할 수 있음. 이를 해결하고자 commit이 되지 않은 데이터를 요구하는 경우 메모리 노드에 즉시 commit을 요청하고 이후 일반적인 RDMA를 통해 접근하는 방식을 통해 일반적인 상황에서의 성능을 개선하고자 함 · 또한 UD를 통한 데이터 전송으로 인해 페이지 데이터나 commit 메시지가 유실될 수 있음. 데이터 페이지의 경우 commit을 요청하면서 처리해야 하는 페이지 리스트를

수행기관	개발 목표	개발내용 및 범위
		전달함으로써 처리 도중 누락된 페이지를 확인할 수 있으며, 이 경우 컴퓨팅 노드의 버퍼나 다른 메모리 노드에게 요청하는 방식을 통해 해결할 수 있음. Commit 메시지 자제가 유실되는 경우에는 일정 수준의 timeout을 통해 처리함. 하지만 장기간에 걸친 구현 및 실험 도중 UD를 통해 데이터 유실이 발생하는 경우를 단 한 번도 겪지 못하였음. 이는 FaSST 논문에서 이야기 하듯 InfiniBand의 UD에서의 데이터 유실은 매우 낮은 수준임을 의미함 · 한편 다수의 메모리 노드에 데이터가 복제되어 저장된 데이터를 활용하기 위해 컴퓨팅 노드는 데이터가 필요할 때 지정된 메모리 노드가 아니라 멀티캐스트에 속한 노드 중 임의의 노드로부터 데이터를 읽도록 구현. 이를 통해 메모리 노드로의 접근이 분산, 응용의 성능 향상을 도모하였음 · 복제된 데이터를 활용하여 메모리 노드의 장애에 대응하는 메모리 시스템 기술을 구현하였음. InfiniBand layer에서 네트워크 장애에 대한 이벤트가 보고되거나 메모리 노드로 발신한 요청에 대해 지정된 timeout 동안 응답이 오지 않는 경우 해당 메모리 노드에 문제가 발생한 것으로 판단할 수 있음. 이러한 방식으로 메모리 노드의 장애를 탐지하고, 장애 발생시 가용 메모리 서버로만 메모리 데이터 요청이 처리되도록 하여 메모리 서버의 장애에 투명하게 대응 · 메모리 노드가 장애로 인해 서비스를 제공할 수 없는 경우 복제의 개수가 줄어듦에 따라 추가적인 장애에 취약하게 되는데, 이에 대응하기 위해 장애 발생시 유휴 메모리 노드를 선택하고, 이들 노드에게 추가적인 복제를 수행하도록 구현 중  그림 94. Staging buffer를 활용한 계층형 원격 메모리 구현 기술 - 성능 평가 · 구현된 시스템의 성능을 평가하기 위해 Python의 numpy를 통해 40,000 x 50,000 크기의 integer matrix multiplication을 수행하는 microbenchmark를 작성하였음. 해당 벤치마크의 동작을 위해서는 최대 50 GB의 메모리가 필요함 · 32GB의 메모리를 탑재한 컴퓨팅 노드와 128GB의 메모리를 탑재한 두 대의 메모리 노드를 구성하고, 컴퓨팅 노드에서 벤치마크를 실행하며 동작 시간을 측정하였음. 또한 비교군으로 128GB의 메모리를 탑재한 컴퓨팅 노드를 활용하여 메모리 데이터의 이동이 전혀 발생하지 않는 경우의 성능을 측정. 이 경우 약 508초의 실행 시간을 보임 · 컴퓨팅 노드에서 메모리가 부족해짐에 따라 일부 데이터는 두 대의 메모리 노드에 복제되어 저장되고 사용됨. 기존의 시스템처럼 별도의 RDMA를 통해 메모리를 복제하는 경우 실행시간은 약 559초가 소요됨. 하지만 제안하는 multicasting을 이용하는 경우 531초의 실행시간을 보이고, 이를 통해 성능과 기능이 개선됨을 확인하였음

수행기관	개발 목표	개발내용 및 범위
		 그림 95. Multicasting을 활용한 분리형 메모리 시스템 성능 개선 실험 결과 • 계층형(hierarchical) 원격 메모리 구조에서 응용의 힌트에 기반하여 데이터의 관리 및 계층간 이동 최적화 기술 개발 - 연구 개요 · 기존의 분리형 메모리 시스템들은 전체 주소 공간이 단일한 특성을 가지는 원격 메모리를 가정하고 구성, 개발되어 옴. 하지만 DRAM만 사용하여 단일 노드에 대규모의 메모리를 구성하기에는 전력, 공간, 에너지, 비용 등의 난점이 발생하며, 이에 대한 해결책으로 Intel OptaneDC Persistent Memory, NVMe SSD 등의 높은 집적도를 가지는 매체를 활용하여 원격 메모리를 계층화하여 구성할 필요성이 대두됨 · 계층적 원격 메모리 구조에서는 데이터를 저장하는 매체에 따라 접근 성능이 크게 차이가 나며, 계층 간 데이터 이동시 메모리 노드와 컴퓨팅 노드 간의 동기화 문제로 인해 성능이 크게 저하되는 문제점이 발생 가능함 · 이에 응용이 접근할 데이터의 특성을 알려주고, 이를 바탕으로 메모리 노드 내에서 데이터가 저장될 위치를 판단하는 기술을 연구 - 연구 개발 내용 · 응용은 실행에 필요한 데이터와 그렇지 않은 데이터를 가장 정확히 알 수 있으며, 또한 언제 해당 데이터가 필요할지 파악할 수 있음. 따라서 이 정보를 활용하여 메모리 노드에서 최적의 데이터 배치를 결정하고 데이터가 접근되기 전에 미리 고속의 메모리 공간으로 데이터를 prefetching하는 형태의 최적화가 가능할 것으로 판단됨 · 앞선 연구에서 멀티캐스팅을 효과적으로 처리하기 위해 메모리 노드 내의 메모리를 staging buffer와 cold buffer로 나누어 구현, 관리하는 기술을 습득하였음. 이를 바탕으로 메모리 노드에서 제공하는 주소 공간을 RDMA를 통해 빠르게 접근할 수 있는 버퍼, RDMA를 통해 직접 접근을 할 수 없지만 대용량의 저장 공간을 가지는 NVMe 기반의 버퍼, NVMe 기반의 버퍼보다 더욱 큰 용량을 제공하지만 동작 성능은 더욱 느린 SATA 기반의 버퍼로 나누고, 각 버퍼 간 효과적인 데이터 이동을 지원하는 방안을 연구 중 · 앞선 연구에서 commit을 기점으로 staging buffer의 내용이 cold buffer로 이동하였음. 이와 유사한 방식을 통해, 컴퓨팅 노드가 접근에 필요한 주소/페이지번호를 메모리 노드에게 전달하면 해당 데이터를 RDMA를 통해 접근이 가능한 영역으로 옮기는 기능을 연구 중임. · 메모리 노드에서는 메모리 사용 상황과 저장된 데이터의 접근 특성을 바탕으로 데이터의 중요도를 판단하고, 중요도가 낮은 데이터는 저속/고용량의 원격 메모리 티어로 이동시키고, 중요도가 높은 데이터는 고속의 원격 메모리 티어로 이동시키는 기술을 구현 중
성균관대학교 (공동2)	비휘발성 메모리 기반 데이터 관리 기술 연구	• 비휘발성 메모리 기반 원격 파일 시스템 캐싱 시스템 개발 개요 - 개념: 원격 노드의 대용량 메모리를 원격 파일 시스템 캐시로 활용할 수 있는 시스템 소프트웨어를 개발 · Intel의 Optane DC PM을 사용하려면 최신 하드웨어 아키텍처가 필요하기 때문에, 기존 레거시 서버가 대용량 메모리를 활용할 수 없게 함. 만일 모든 서버에 고가의 대용량

수행기관	개발 목표	개발내용 및 범위
		메모리가 장착한다면 클러스터 전체의 메모리 사용률 불균형을 악화시킬 수 있음 · 1차년도에 구현한 원격 파일 시스템 캐시를 고속 네트워크로 연결된 클러스터의 여러 클라이언트가 공유하는 환경에서 메모리를 효율적으로 배분하고 빠르게 캐시 교체를 수행할 수 있는 정책을 고안함 - 기존 기술의 문제점 · RDMA 및 CXL과 같은 고속 상호 연결을 통해 메모리를 분해하고 사용되지 않는 원격 메모리 공간을 활용하기 위해 InfiniSwap 시스템이 NSDI '17에 발표되었으며, FastSwap 시스템이 EuroSys '20에 발표되었음. InfiniSwap 및 FastSwap은 원격 메모리 공간을 Anonymous 페이지들을 위한 스왑 공간으로 활용하는 시스템으로, 원격 메모리 서버는 메모리 공간만을 제공하고, 메모리 관리는 1-Sided RDMA를 사용하여 클라이언트들이 관리함 · 1-Sided RDMA는 원격 CPU를 우회하므로 2-Sided RDMA보다 대기 시간이 짧지만 1-Sided RDMA를 사용한 원격 메모리 액세스는 2-Sided RDMA에 비해 유연성이 떨어지며, 원격 서버가 메모리 관리에 개입하는 것을 허용하지 않음. 즉, 각 클라이언트에 할당된 메모리 공간의 크기. 스왑 공간의 수요가 고정적이지 않다는 점을 고려할 때 원격 메모리 공간의 정적 할당은 원격 메모리 공간의 활용도가 변동한다는 점을 반영할 수 없음 · 기존 InfiniSwap이나 FastSwap 시스템들이 타겟으로 삼은 Anymous 페이지와 달리 파일 시스템 페이지들은 클린(Clean)한 페이지들과 더티(Dirty)한 페이지들로 구분이 됨. 파일 시스템 공간이 부족할 때, 더티 페이지를 디스크로 플러시하는 대신 원격 메모리 서버에 캐싱하면 오버헤드가 큰 디스크 IO를 피할 수 있어 I/O 성능을 향상시키는 데 도움이 될 수 있지만, 더티 페이지는 결국 디스크에 기록되어야 한다는 문제점이 있음. 따라서 원격 메모리에 더티 페이지를 캐싱하면 이중 쓰기 문제가 발생함. - 더티 페이지와 비교하여 원격 메모리에 클린 페이지를 캐싱하면 많은 것이 간단해짐. 즉, 메모리가 작은 클라이언트 시스템의 클린 페이지는 종종 캐시에서 메모리에서 축출되어 버려짐. 버려지더라도 파일 시스템에 원본 페이지가 저장되어 있기 때문에 원격 메모리에 클린 페이지를 캐싱하는 경우, 필요에 따라 원격 메모리를 접근할 수도 있고, 디스크의 원본 페이지를 접근할 수 도 있어, 내결함성 문제가 발생하지 않고 파일 시스템 성능 향상에 도움이 되어 단점은 없고 이점만 있는 시스템이 됨. 이는 익명 페이지에 비해 상당한 이점이 됨. 즉, 익명 페이지가 원격 메모리에 캐시되면 클라이언트 서버와 원격 메모리 서버가 모두 가동되고 실행되어야 하기 때문에 애플리케이션에 문제가 생길 확률이 두 배 높아지는 반면, 클린 페이지는 원격 서버에 문제가 생겨도 애플리케이션에 문제가 생기지 않음 • Disaggregated Transcendent Memory (DTM) 개발 - 아키텍처 디자인  그림 96. Disaggregated Transcendent Memory (DTM) 구조도 · DTM (Disaggregated Transcendent Memory) 시스템은 위 그림과 같이 클라이언트 (클라이언트)와 백엔드 (메모리 서버)로 구성됨. 클라이언트는 clean cache를 위한

수행기관	개발 목표	개발내용 및 범위
		Transcendent Memory Driver임. 클라이언트는 커널의 PFRA (페이지 프레임 교체 알고리즘)에 의해 제거된 클린 페이지를 캡처하고 RDMA를 통해 원격 백엔드로 페이지를 전달함. 백엔드는 RDMA 통신 모듈, 동적 메모리 할당기, 해시 인덱스 및 AMQ(Approximate Membership Query) 필터 생성기로 구성됨 - Put 연산 · 클라이언트는 메모리가 부족할 때 Linux 커널은 비활성 LRU 리스트에서 축출할 희생 페이지를 선택함. 선택된 페이지가 더티 페이지인 경우 디스크에 다시 기록되지만 클린 페이지인 경우에는 remove_mapping() 함수가 호출되어 LRU 목록에서 해당 페이지를 제거하고 cleancache_put_page()를 호출하여 cleancache에 저장함 · DTM에서 cleancache_put_page()는 아래 그림과 같이 RDMA WRITE 및 RDMA SEND Verb를 통해 희생 페이지를 원격 메모리 백엔드 (서버)로 전송함. RDMA READ/WRITE Verb는 메모리 시멘틱을 사용하는 반면 RDMA SEND/RECV Verb는 통신 채널 시멘틱을 가짐. 즉, 페이로드는 사전 게시된 RDMA RECV에서 receiver가 지정한 원격 메모리 주소에 기록됨. 즉, 버퍼링되지 않은 소켓 구현에서와 같이 RDMA SEND/RECV는 receiver가 RDMA RECV 기능을 호출하여 페이로드를 쓸 수 있는 DMA 페이지를 준비하도록 요구함.  그림 97. PUT 연산을 위한 2-sided RDMA 통신 · 1-Sided RDMA READ/WRITE와 달리 2-Sided RDMA SEND/RECV는 receiver에서 CPU 오버헤드를 가지므로 대기 시간이 약간 길어지고 처리량이 낮아짐. 따라서 2-Sided RDMA Verb는 FaRM을 비롯한 여러 키-밸류 시스템에 적합하지 않은 것으로 간주되었음. 그러나 일방적인 RDMA READ/WRITE는 클라이언트가 요청을 보냈는지 확인하기 위해 서버가 클라이언트당 메모리 영역을 계속 폴링해야 함. 또한 1-sided RDMA READ는 요청이 서버에서 처리되었는지 확인하기 위해 여러 번 왕복해야 함. 그 결과 1-sided RDMA는 확장성이 좋지 않음. 따라서 HERD와 같이 DTM은 하이브리드 접근 방식을 취하고 1-sided 및 2-Sided RDMA 동사를 모두 사용하도록 설계함 · 클라이언트는 RDMA WRITE Verb를 이용하여 페이지 오프셋과 아이노드 번호를 키로 보냄. 키의 사이즈가 작기 때문에 키는 RDMA WQE(Work Queue Element)에 인라인으로 전송함. 그러나 페이지 크기는 최대 PIO 크기(256바이트)를 초과하므로 RDMA WQE에 인라인시킬 수 없어 DMA 작업이 필요함. 클라이언트는 RDMA SEND를 통해 희생된 클린 페이지를 전송함 · 백엔드 서버가 클라이언트로부터 키와 페이지를 받으면 위 그림과 같이 3단계로 동적 해시 테이블의 페이지를 인덱싱함. remove_mapping() 함수는 메모리가 부족한 경우 LRU 목록에서 여러 페이지를 제거함. 따라서 클라이언트는 각 개별 페이지를 보내는 대신 단일 RDMA 동사를 사용하여 지연 방식으로 여러 페이지를 한번에 백엔드에 전송함. 이러한 지연 제거 방식은 메모리 압력이 높은 클라이언트에서 제거된 페이지를 버퍼링하는 비용으로 네트워크 통신 오버헤드를 줄이는 데 도움이 됨

수행기관	개발 목표	개발내용 및 범위
		- Get 연산 · Transcendent 메모리는 커널에서 직접 액세스하거나 주소를 지정할 수 없음. 따라서 페이지 누락이 발생하면 클라이언트는 백엔드 Transcendent Memory에 요청된 페이지가 포함되어 있는지 확인해야 함. 백엔드의 CPU는 Transcendent Memory에서 페이지를 검색하고 페이지를 찾으면 클라이언트로 전송하여 클라이언트의 페이지 캐시에 로드함. zcache 또는 Xen과 같은 기존 Transcendent Memory는 로컬 시스템이기 때문에 cleancache를 검색하는 오버헤드는 무시할 수 있는 수준의 성능 저하를 초래함. 그러나 Disaggregated Memory를 사용하면 RDMA 통신 비용과 인덱스 검색 오버헤드로 인해 무시할 수 없는 수준의 오버헤드가 발생함. 백엔드에서 페이지를 찾으면 이러한 오버헤드보다 얻는 성능 상의 이득이 크지만, 만일 페이지가 존재하지 않으면 이 오버헤드는 성능을 크게 저하시킬 수 있음. · Negative 원격 인덱스 검색을 피하기 위해 DTM은 분산 카운팅 필터를 사용하고 RDMA 동사를 선택적으로 호출하여 원격 cleancache에 액세스하도록 설계함. 카운팅 필터는 원격으로 캐시된 페이지를 모른 채 클라이언트가 모든 페이지 캐시 누락에 대해 cleancache_get_page()를 호출하여 불필요한 상당한 통신 오버헤드를 초래하기 때문에 필요함. 페이지 캐시가 누락되면 VFS 계층의 mpage_readpages() 함수가 페이지가 액세스할 주소 공간을 채움. cleancache가 활성화되면 클라이언트는 블록 I/O를 내보내는 대신 cleancache_get_page()를 호출하며, 필터가 원격 백엔드에 페이지가 있음을 나타내는 경우 RDMA 동사를 호출함.  그림 98. GET을 위한 2-sided RDMA 통신 · 위 그림과 같이 클라이언트는 2-Sided RDMA SEND를 통해 페이지 오프셋과 inode 번호를 키로 전송함. 또한 백엔드가 요청된 페이지를 쓸 수 있는 DMA 페이지의 주소를 보냄. 2-Sided RDMA SEND를 사용하면 백엔드는 즉시 인덱스를 조회하여 요청된 페이지를 찾아 DMA 페이지에 씀. 백엔드에서 페이지가 새로 저장되거나 GET에 의해 삭제되면 이 정보는 클라이언트에 통지하여 클라이언트의 필터를 업데이트해야 함. 그래야, 불필요한 원격 메모리 접근을 줄일 수 있음. 필터가 원격 클린 캐시에 요청된 페이지가 없다고 표시하면 클라이언트는 원격 백엔드 서버를 접근하는 대신 파일 시스템을 통해 디스크에서 페이지를 읽음 · PUT 작업과 달리 GET 작업은 요청된 페이지를 페이지 캐시에서 사용할 수 있을 때까지 애플리케이션 프로세스를 차단함. 따라서 프런트엔드는 각 GET 요청을 버퍼링하지 않고 캐시 미스당 즉시 RDMA SEND를 호출함. 그러나 큰 블록 I/O로 인해 많은 수의 캐시 누락이 발생하면 이러한 개별 페이지 전송으로 인해 RDMA 호출에 상당한 오버헤드가 발생할 수 있으므로, 이 문제를 완화하기 위해 백엔드는 클라이언트별로 페이지 액세스 기록을 관리하고 순차 I/O가 감지되면 단일 RDMA 동사를 통해 여러 연속 페이지를 보냄

수행기관	개발 목표	개발내용 및 범위
		· 원격 대용량 메모리 서버는 Anonymous 페이지와 Transcendent Memory 페이지의 비율을 관리해야 함. 그렇지 않으면 스래싱이 발생하고 성능이 저하될 수 있음. 따라서 DTM의 백엔드는 동적 해시 테이블을 사용하여 제거된 clean 페이지를 관리하므로 작업 부하에 따라 clean 페이지 및 익명 페이지의 메모리 공간을 조정할 수 있음. · DTM의 백엔드는 여러 클라이언트를 지원함. 즉, 백엔드는 많은 클라이언트 노드에서 interleaving 요청을 수신하는 공유 캐시임. 따라서 백엔드는 캐시 상주 및 파괴적 간섭에 대한 경쟁을 고려해야 함. DTM에서 고려해야 할 두 가지 유형의 경쟁 중 하나는 익명 페이지와 파일 지원 페이지 간의 경쟁이고 다른 하나는 여러 클라이언트 노드의 파일 지원 페이지 간의 경쟁임 · 첫째, 익명 페이지와 파일 지원 페이지 간의 경쟁을 위해 Linux 커널이 로컬 메모리에 대해 수행하는 것처럼 파일 지원 페이지는 익명 페이지에 우선 순위를 부여해야 함. 즉, 많은 양의 익명 페이지가 메모리 서버로 전송되는 경우 익명 페이지를 위한 공간이 생성되도록 클린 페이지용 메모리 풀의 크기를 줄여야 함. 익명 페이지가 메모리 서버에서 클라이언트로 스왑인되면 클린 페이지용 메모리 풀이 더 많은 메모리 공간을 사용할 수 있어야 함. 이러한 동적 확장 및 축소를 지원하기 위해 DTM의 백엔드는 동적 해시 테이블을 사용함 · 둘째, 여러 클라이언트 노드의 파일 지원 페이지 간의 경쟁을 위해 백엔드는 캐시 교체 정책을 구현해야 함. 지난 수십 년 동안 ARC, LIRS, LeCaR, CACHEUS를 포함한 수많은 페이지 교체 알고리즘이 캐시 상주를 관리하기 위해 개발되었음. Linux 커널에서 VFS 계층은 LRU/2를 사용하는 고도로 최적화된 페이지 캐시를 가지고 있음. 그러나 Transcendent Memory가 한번 축출된 페이지를 저장하는 Secondary 캐시임을 고려할 때 메모리 참조의 시간적 Locality가 높지 않고 Primary 캐시에 비해 메모리 Locality를 활용하기 어려우므로, 엄격한 LRU 교체 정책은 DTM에서 효과적이지 않음. 클린 페이지 외에 익명 페이지에 대용량 메모리 서버를 사용하는 경우 클린 페이지를 위한 메모리 공간은 익명 페이지를 위한 메모리 공간과 별도로 관리해야 하며 두 메모리 공간의 크기를 동적으로 조정해야 하며, 이를 위해 DTM 백엔드 서버는 CCEH 동적 해시 테이블을 사용함 - Hash 기반 Secondary 캐시 관리 · 해시 기반 인덱스에서 가장 큰 성능의 문제는 해시 충돌을 해결하는 것임. 해시 충돌이 발생할 경우, 해시 테이블 크기를 키우는 rehashing, 다른 빈 버켓을 찾아가는 Linear Probing, Cuckooing, Chaining 등은 모두 많은 메모리 접근을 필요로 함. 특히 리해싱은 인덱싱된 레코드를 다른 버킷으로 이동해야 하고 작업 중에 해시 테이블에 액세스하는 것을 방지하기 때문에 쿼리의 꼬리 대기 시간을 크게 증가시키는 것으로 알려져 있음. CCEH와 같은 동적 해싱 기법은 해시 테이블의 일부분만 rehashing 함으로써 문제를 완화시킴 · DTM은 캐시 교체와 해시 충돌 처리를 함께 처리하도록 설계함. 즉, 해시 충돌이 발생하면 새 버킷을 할당하고 값비싼 rehashing을 수행하는 대신 충돌이 난 버켓에 저장되어 있던 이전 페이지를 새 페이지로 덮어쓰는 것임. 이 접근 방식은 LRU 목록을 사용하는 경우에 비해 해시 테이블 관리를 크게 단순화시킴. LRU 목록을 사용하면 가장 오래된 페이지가 목록에서 제거되는데, 삽입될 새 페이지는 해당 버킷이 아닌 다른 버킷을 필요로 할 수 있음 · 따라서, DTM의 백엔드는 페이지 액세스의 글로벌 시퀀스를 추적하기 위한 LRU 목록을 관리하지 않음. 대신 캐시 교체 정책을 해시 충돌 해결 정책과 통합함. 즉, 해시 충돌이 발생하면 일정 수의 다음 버킷을 조사하고 (즉, linear probing), 그 중에 새 페이지를 수용할 수 있는 버킷이 있는지 확인함. 또한, 이러한 버킷들을 조사하는 동안 가장 오래된 페이지를 추적함. 여유 버킷이 없으면 클린 캐시 크기를 키울 수 있는지 확인함. 만일 클린 캐시 크기를 키울 수 있다면, 이전 페이지를 교체하는 대신 새 세그먼트를 할당하고 세그먼트를 분할함. 그렇지 않으면 새 페이지로 다음 버킷을 조사하는 동안 찾은 가장 오래된 페이지를 덮어씀

수행기관	개발 목표	개발내용 및 범위

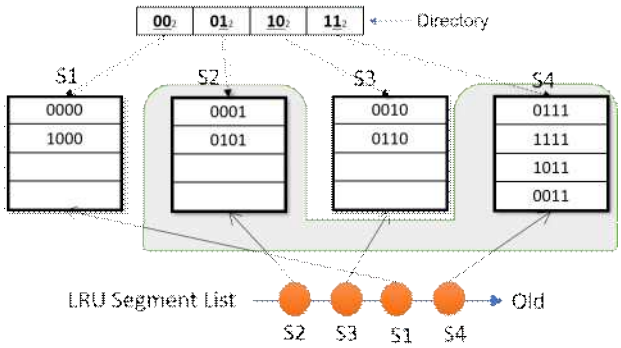


그림 99. Hash LRU

· 클라이언트가 클린 캐시에서 페이지를 읽으면 해시 테이블에서 페이지가 제거되고 활용도가 낮은 세그먼트가 이웃한 세그먼트와 병합됨. 이를 위해 백엔드는 위 그림과 같이 CCEH 세그먼트에 대한 LRU 목록을 관리함. 세그먼트에 새 페이지가 삽입되면 세그먼트의 수정된 시간을 업데이트하고 그에 따라 LRU 목록을 업데이트함. Transcendent Memory가 cleancache 크기를 줄이기로 결정하면 가장 오래된 세그먼트를 선택하고 병합함

· 두 세그먼트를 병합할 때 해시 충돌이 있는지 확인하고 일반적인 삽입 작업과 같이 처리, 즉, 오래된 페이지를 버림. 위 그림의 예에서는 가장 오래된 세그먼트 S4와 이웃인 세그먼트 S2를 서로 병합함.

- 필터 관리

· 클라이언트와 백엔드 사이의 불필요한 통신을 피하기 위해 DTM은 Distributed Quotient Filter를 사용함. Bloom 필터, Counting 필터, Quotient 필터, Cuckoo 필터를 포함한 다양한 AMQ 필터는 데이터베이스 및 스토리지 시스템에서 대략적인 멤버십 테스트 쿼리에 자주 사용됨. Bloom 필터는 가장 널리 사용되는 확률적 AMQ 중 하나이지만, 치명적인 단점은 삭제 작업을 반영하지 않는다는 것임. 즉, cleancache에서 페이지를 제거하더라도 Bloom 필터는 삭제를 반영하지 않고 클라이언트에 false positive 결과를 반환하므로 불필요한 통신과 인덱스 조회가 발생할 수 있음. 이 문제를 해결하기 위해 Counting Bloom filter 및 Counting Quotient Filter와 같은 Bloom 필터의 많은 변형이 개발되었음. 이러한 필터는 각 양수 해시 비트의 발생 횟수를 계산하므로 블룸 필터보다 더 많은 공간을 차지함. 그러나 삭제 작업은 cleancache에서 중요하므로 DTM의 백엔드는 필터가 캐시 교체를 반영할 수 있도록 Counting Quotient Filter를 사용함

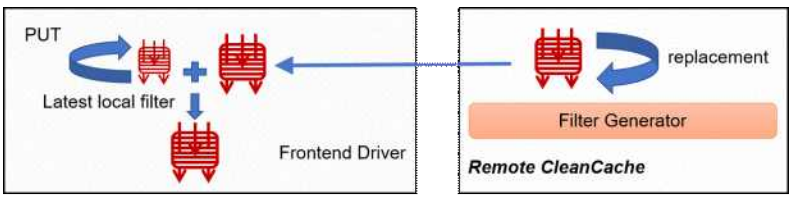
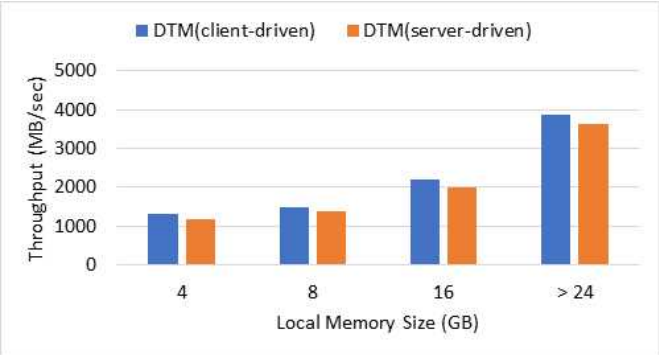
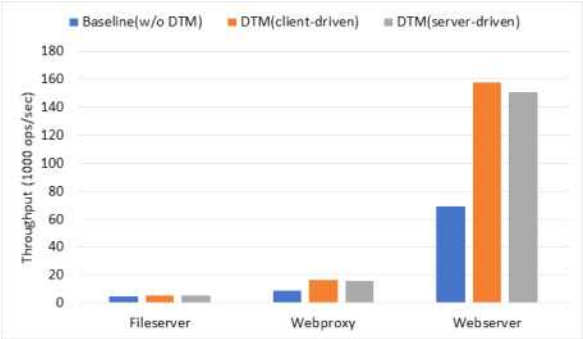
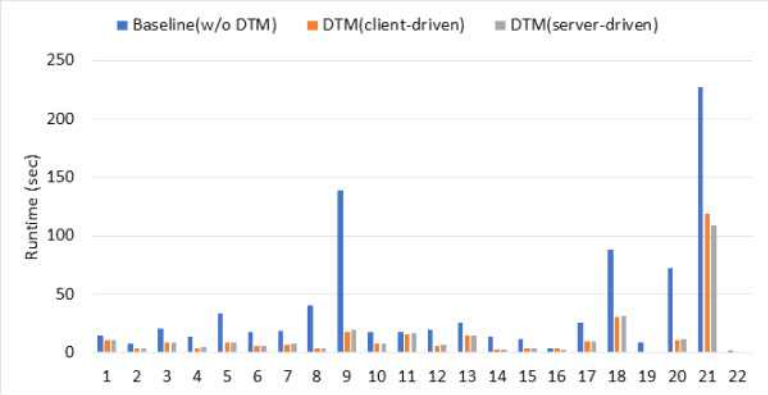
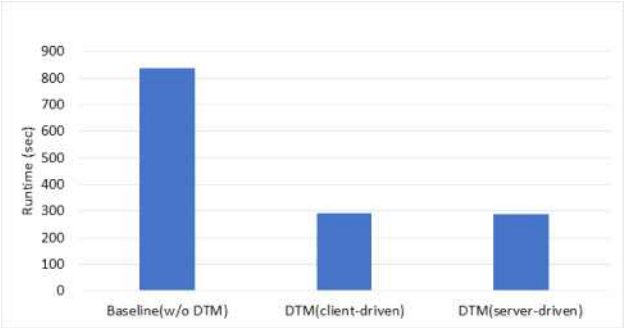


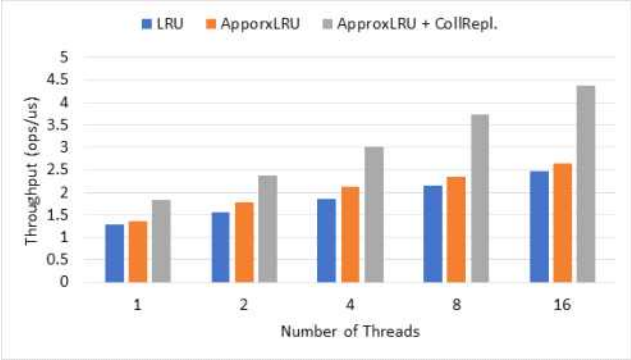
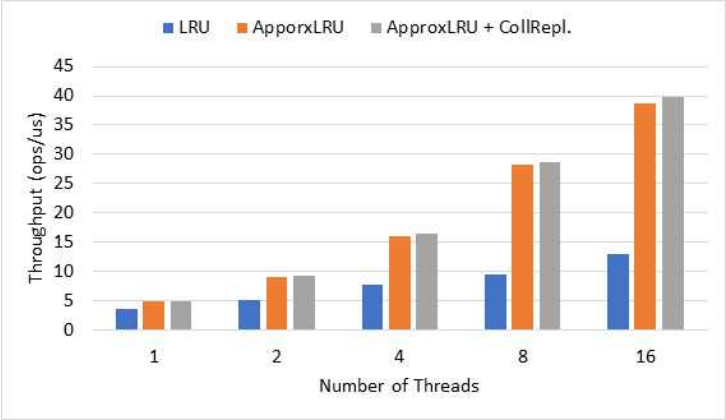
그림 100. DTM을 위한 분산 필터 관리

· 백엔드 서버는 주기적으로 필터를 클라이언트에 전송함. 클라이언트는 위 그림과 같이 백엔드 서버로부터 받은 필터와, 자신이 최근에 서버로 보낸 페이지들의 정보를 반영하는 로컬 필터를 합쳐서 사용함으로써, 서버의 필터에 미반영된 최근 페이지들 정보도 사용함.

• Disaggregated Transcendant Memory 시스템 성능 평가

수행기관	개발 목표	개발내용 및 범위
		- 본 과제가 제안하는 구조는 DTM(server-driend)으로 서버에서 원격 메모리를 관리하며 two-sided RDMA를 사용해 RPC 방식의 통신을 함. 비교군으로는 클라이언트에서 원격 메모리를 관리하면서 one-sided RDMA를 사용하는 DTM(client-driven)이 있고, Baseline(w/o DTM)은 DTM을 사용하지 않는 secondary 캐시가 없는 구조임 - FIO 성능  그림 101. FIO: 순차 읽기 · DTM의 기본적인 성능을 알아보기 위해서 디스크 벤치마크로 주로 사용되는 FIO를 사용해서 순차 읽기 성능을 측정하였음. IO 쓰레드는 8개, 파일 크기는 24GB이며 로컬 메모리 크기를 4GB부터 파일 크기 이상까지 설정하며 수행하였음. DTM의 캐시 효과를 측정하기 위해 먼저 디스크에서 파일을 읽고 DTM 서버로 clean page를 evict 시킴 · 그림 101을 보면, DTM(client-driven)과 DTM(sever-driven) 모두 로컬 메모리 크기가 증가함에 따라 읽기 처리량이 증가함. 로컬 메모리가 파일 크기에 비해 작은 4GB의 경우, DTM(server-driven)의 읽기 처리량은 1178 MB/sec인 반면 로컬 메모리가 파일 크기보다 큰 경우의 처리량은 3621 MB/sec 임. 즉, 로컬 메모리로 DTM 캐시에서 페이지를 읽음과 동시에 로컬 메모리에서 페이지가 DTM 서버로 축출되는 경우가 DTM 캐시에서 페이지를 읽기만 하는 경우에 비해 성능이 낮음. 읽어드릴 페이지를 위한 페이지 확보가 빠르게 되지 않는 것으로 추정되어 원인을 분석하고 개선할 예정임 - Filebench 성능  그림 102. Filebench 성능 · 다음은 파일 시스템과 스토리지 워크로드를 시뮬레이션 하는 Filebench를 사용해 DTM의 효과를 측정했다. Fileserver, Webproxy, Webserver 워크로드를 사용했고, 각 워크로드를 300초 동안 실행하여 데이터 셋에 대한 재참조가 발생했음 · Fileserver의 경우 Read:Write 비율이 1:2이 워크로드로서 DTM의 효과가 미비함. 이에 반해 Webproxy, Webserver는 읽기-집약적인 워크로드이어서 DTM의 캐싱 효과가 큼. Webproxy와 Webserver는 Baseline에 비해 DTM(client-driven)은 1.9배, 2.3배 DTM(server-driven)은 1.8배, 2.2 배의 처리량 향상이 있었음

수행기관	개발 목표	개발내용 및 범위
		- Postgress 성능  그림 103. Postgres (TPC-H 벤치마크) 성능  그림 104. TPC-H의 총 수행시간 · DTM의 실제 효과를 응용에서 측정하기 위해 페이지 캐시를 사용하는 데이터베이스 엔진인 Postgres에 OLAP의 성격의 워크로드인 TPC-H를 실행함. 사용한 데이터 셋은 scale factor가 10 (약 10GB)이고 다른 옵션은 기본값을 사용했음. 로컬 메모리 크기는 4GB로 설정했다. TPC-H 벤치마크는 22개의 쿼리를 수행하였고 각 쿼리에 대한 실행시간을 측정했음 · OLAP 특성상 데이터 셋에 대한 분석 및 처리를 하기 때문에 같은 데이터에 대한 읽기가 많이 발생함. 따라서 그림 103를 보면, 모든 쿼리에 대해서 DTM secondary 캐시가 있을 경우 Baseline에 비해 실행시간이 줄어듬 · 위 그림 104은 TPC-H의 총 수행시간을 나타내는 그래프인데 secondary 캐싱 효과로 인해 DTM(server-driven)이 Baseline에 비해 약 2.88배 수행시간이 줄었음을 확인할 수 있음 - Hash Replacement 성능 · 다음은 DTM 서버에서 사용하는 해시 기반의 페이지 교체 정책에 대한 성능 그래프임. ApporxLRU는 CCEH의 세그먼트에 접근 시간을 기록하여 대략적인 LRU 순서를 지켜서 교체하는 정책이고 +CollRepl. 은 최소 유휴 페이지 개수가 임계점 이하로 떨어졌을 때 빠른 페이지 회수를 위해 해시 충돌로 인한 스플릿 리해싱을 하지 않고 기존 페이지를 회수하는 방식임. 비교군인 LRU 리스트는 Intel TBB의 concurrent lru를 사용했다. 본 실험은 DTM 플랫폼 없이 인덱스 단독으로 성능을 평가했음. CCEH가 인덱싱하는 메모리 풀보다 두 배 많은 양의 데이터를 사용해서 페이지 교체 정책을 평가했음. 쓰레드 개수를 1개에서 16개로 증가시키며 처리량을 측정했음

수행기관	개발 목표	개발내용 및 범위
		 그림 105. Insert 성능 · 위 그림은 Insert 성능을 보여줌. LRU 리스트를 사용하면 매 insert마다 LRU 리스트에서 세그먼트의 위치를 이동시켜야하기 때문에 쓰레드가 증가함에 따라 동기화 오버헤드가 커져서 가장 성능이 낮은 것을 확인할 수 있음. ApproxLRU는 동기화 오버헤드가 없어 LRU 리스트에 비해 높은 성능을 보임. ApproxLRU + CollRepl. 의 경우 유휴 페이지가 극단적으로 적은 상황에서 해시 충돌에 대한 리해싱이 없고 사용 중이던 페이지를 회수하기 때문에 다른 두 개의 기법에 비해 모든 경우 처리량이 높으며 16스레드일 때 77%의 성능 향상을 보임  그림 106. Search 성능 · 위 그림은 Search 성능을 보여줌. LRU 리스트를 사용할 때는 Insert와 같은 이유로 처리량이 가장 낮고 ApporxLRU를 사용했을 때의 처리량이 높음. 특히, 16 쓰레드에서 ApproxLRU가 LRU 리스트에 비해 3배의 처리량이 높음. Search에서는 페이지 교체가 없기 때문에 CollRepl의 효과가 영향을 미치지 않음 · 향후 해시 기반의 교체 정책을 DTM 플랫폼에서 실험할 계획이고 ApproxLRU의 hit ratio를 LRU와 비교하여 평가할 예정임
UNIST (공동3)	인공지능 학습을 위한 인메모리 데이터 관리 연구	**• GPU 메모리 공유 지원 시스템의 요건** - 딥러닝 모델의 학습은 iteration의 반복으로 구성되며, iteration은 forward pass라 불리는 연산작업과 backward pass로 불리는 역연산 작업으로 구성. 각 forward pass에서는 연산을 수행한 뒤에 이후 역연산에 필요한 임시 텐서들을 보관하고, 이후 역연산이 이루어진 뒤에 임시 텐서들을 해제. 그로 인하여, 다수의 딥러닝 모델 학습 과정은 다음 그림과 같이 forward pass에서는 증가하고 backward pass에서는 감소하는 메모리 사용 패턴을 보임

수행기관	개발 목표	개발내용 및 범위

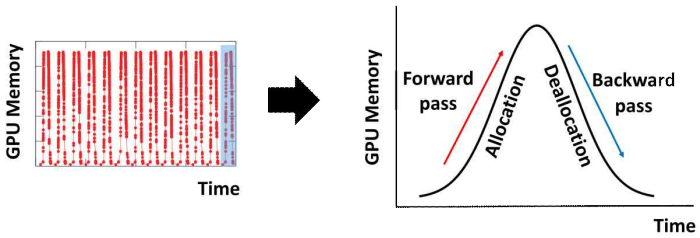


그림 107. 딥러닝 모델 학습 과정에서의 반복적인 GPU 메모리 사용량

- GPU의 자원효율을 최대한으로 올리면서 제한된 메모리 사용을 보장하기 위해서 해결해야 하는 문제점으로는 Memory Bloating 현상, 비동기적인 GPU의 연산작업, 딥러닝 모델에 따른 다양한 메모리 사용 패턴이 있음
- Memory Bloating 현상: GPU에서 사용하는 메모리를 최종 사용까지 유지함으로 인해서 첫 번째 접근과 마지막 접근 간의 시간차가 큰 데이터의 장시간 유지에 따른 메모리의 비생산적인 과용 현상
  · 아래 그래프는 딥러닝 모델 중 하나인 NASNet 학습에 따른 Feature 데이터(Sharable)와 다른 데이터(Non-Sharable)의 수명의 누적분포도. 모델의 학습과정에서 생성되는 Feature 데이터의 중위값과 평균값이 각각 243ms와 134ms로 다른 데이터들의 중위값과 평균값인 2ms와 18ms에 비해 월등히 높게 유지됨. 이와 같은 과도한 메모리 사용량은 공유될 수 있는 GPU 메모리 자원을 공유하지 못하게 만드는 장애물로 작용

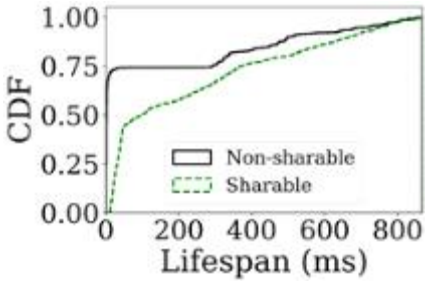


그림 108. NASNet 텐서 수명의 누적분포도

- 비동기적인 GPU의 연산작업: 기존의 GPU를 연산에 활용하는 방법은 연산을 위한 커널을 CPU가 대량으로 GPU로 보낸 뒤 GPU가 비동기적으로 처리하도록 하는 것. 이 방법은 다양의 연산을 효율적으로 처리하는데는 좋지만, CPU가 GPU 내의 자원 활용 상황을 알지 못하게 되므로 자원 관리 측면에서 맹점이 있음
  · 다음은 딥러닝 모델 중 하나인 Resnet 50을 학습시킬 때의 GPU의 메모리 사용량을 CPU의 관점으로 본 그래프. 실제로 GPU에서는 연산작업이 진행되고 있음에도 불구하고 CPU는 메모리 사용량이 없다고 판단하게 됨

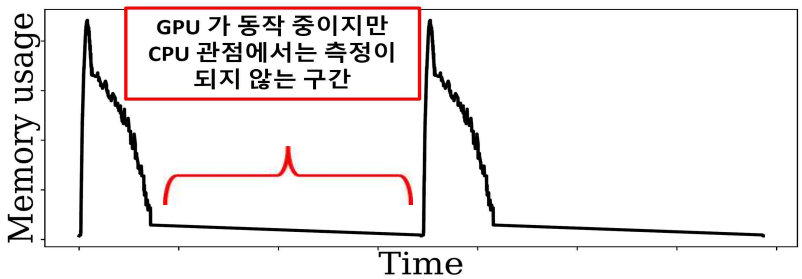


그림 109. CPU 관점에서 측정한 딥러닝 모델 학습 시 GPU 메모리 사용량

수행기관	개발 목표	개발내용 및 범위

· 메모리 공유 시스템에서 메모리 사용을 관리하려면 짧은 시간 단위로 메모리 사용량 확인이 가능해야 함

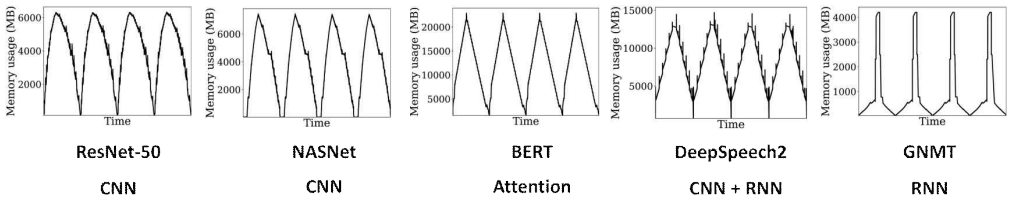


그림 110. 딥러닝 모델에 따른 메모리 사용량과 메모리 사용 패턴

- 딥러닝 모델에 따른 다양한 메모리 사용 패턴 문제: 다양한 딥러닝 모델이 개발됨에 따라, 모델에 따라 메모리 사용량 뿐만아니라 메모리 사용 패턴도 다양해짐. 이에 따라 성능 간섭을 최소화하는 GPU 자원 공유가 더욱 어려워짐

• **GPU 메모리 공유 시스템 설계**

- GPU 메모리 공유 시스템의 필수 요소
  · 높은 성능: 메모리를 공유하면서도 각 딥러닝 모델의 빠른 학습 시간을 보장하는가?
  · 범용성: 다양한 딥러닝 모델을 지원하는가?
  · 호환성: 현존하는 딥러닝 훈련 플랫폼(Tensorflow, PyTorch 등) 에서 사용 가능한가?
- 기존의 GPU 자원 공유 방법
  · Temporal Sharing: 시간을 나누어 GPU 자원을 공유하는 방법으로, 여러 학습 모델이 시간 순으로 실행됨. 널리 사용되고 있으나 GPU 자원 활용율은 낮음
  · Spatial Sharing: 여러 학습 모델을 GPU 자원의 양을 나누어서 실행시키는 방법. NVIDIA사의 MPS(Multi-Process Service)가 있음. 단일 GPU에서 다중 프로세스의 동시 실행을 지원하지만, 메모리를 정적으로 할당함으로써 메모리 자원의 초과(Memory Over-subscription)에 따른 성능 하락 문제가 심각

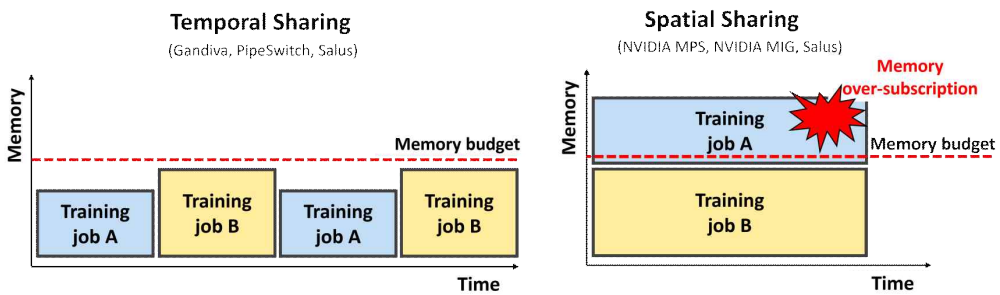


그림 111. 기존의 GPU 자원 공유 방법

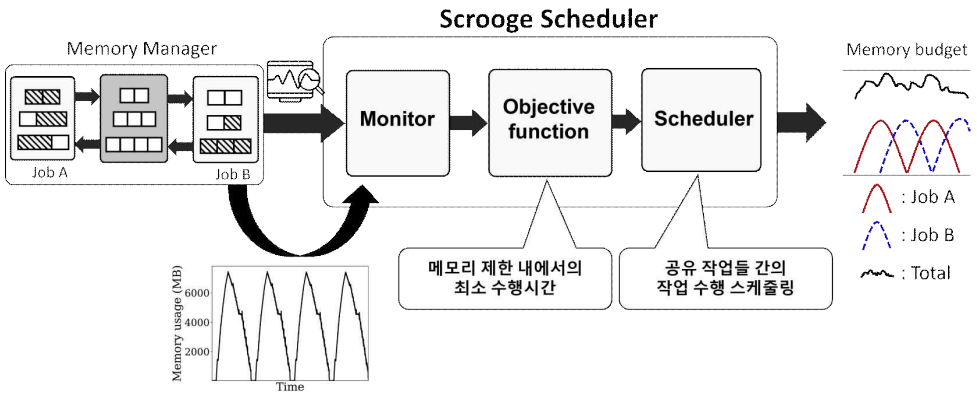
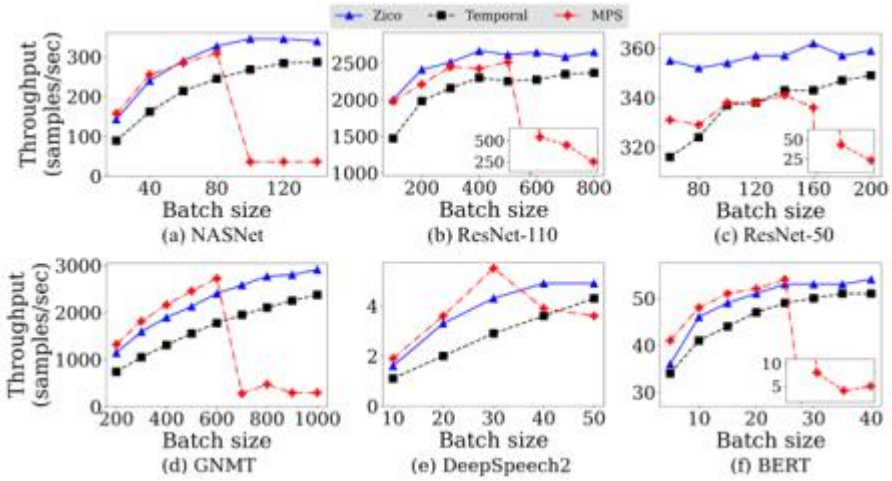


그림 112. ZICO 구조도

수행기관	개발 목표	개발내용 및 범위
		- ZICO 시스템: 높은 성능과 범용성, 호환성을 제공하는 GPU 메모리 공유 시스템. Scrooge 스케줄러와 메모리 매니저를 통해 요구 사항을 충족시킴 · 공유 작업들 간의 작업 수행 스케줄링: 스케줄러는 주기적인 CUDA EVENT(GPU 연산 stream에서 해당 지점까지에 연산의 완료를 확인할 수 있는 커널)의 활용을 통해서 연산이 완료되어서 더 이상 사용되지 않는 텐서를 안전하게 확인 후 공유자원으로 회수. 또한, 메모리 할당을 위한 커널이 CPU에서 과도하게 선제적으로 발생하지 않도록 GPU로의 연산 커널 전송 속도를 GPU의 처리 속도에 맞추어서 적절한 메모리 할당이 점진적으로 일어나게 함. 이 커널 전송 속도는 딥러닝 모델 학습의 초기 단계에서 프로파일링을 통해서 파악 가능 · 공유 자원 회수 기능: 텐서의 정보에 따라서 공유 자원으로의 회수가 가능한지를 구분해야 함. Tensorflow의 경우, 연산 텐서는 DAG(Directed Acyclic Graph, 유향 비순환 그래프)를 통해서 순서도가 미리 정의되어 있고, 각 텐서에 정보가 충분히 포함되어 있기 때문에, 이 정보를 이용하여 텐서 분류를 진행. PyTorch 또한 연산 그래프를 생성하여서 GPU로 연산을 보내는 특성이 있기에, 본 연구와 같은 효과를 기대할 수 있음 **· 메모리 제한 내에서의 최소 수행시간 모델링** - GPU 메모리 공유를 고려한 스케줄링 알고리즘은 메모리 소비의 패턴을 최소한의 오버헤드(overhead)를 이용하여서 파악해야 함. 메모리 소비의 모델링에서 과도하게 세밀한 라벨링은 최적화 연산에 많은 오버헤드를 발생시킬 수 있으나, 효율적인 GPU 메모리 프로비저닝을 위해서는 메모리 패턴을 파악하기 위한 단위를 잡아야 함. 본 연구에서는 메모리 패턴을 파악하기 위한 단위를 "리전(Region)"으로 설정하고, 이를 통해서 GPU 메모리에서 모델 간 메모리 할당과 해제를 관리. 모델 학습 과정에서 매 iteration마다 forward 연산에서 메모리 할당이, backward 연산에서 메모리 해제가 발생 - 이 관리 기법은 다음과 같이 도식화할 수 있음. 딥러닝 모델 트레이닝 과정에서 할당 순서에 따른 "리전"의 할당($R_i(a)$) 및 해제($R_i(d)$)가 M번 발생한다고 가정할 때, 각 "리전"의 수명을 측정할 수 있으며($Time(R_i(d)) - Time(R_i(a))$), 스케줄러의 할당된 forward 매개변수($TimeShift$)를 반영. 이렇게 측정된 전체 "리전"의 시간의 합이 최소가 된다면, 가용 메모리의 총량(C)을 넘지 않으면서 최소 시간 지연을 달성했다고 판단 가능. 이를 일반화된 수식을 통해서 표현하면 아래와 같음 $\underset{TimeShift}{\arg\min}\,(Time(R_1(d)) - Time(R_1(a)) + TimeShift)$ **subject to,** $\sum_{i=1}^{K} Size(R_i) \le C$ **Where,** $Size(R_i), 1 \le i \le K : the\ sizes\ of\ K\ active\ regions$ $timeshift : a\ control\ parameter\ of\ pausing\ the\ forward\ phase$ 그림 113. 메모리 제한 내의 최소 수행 시간 모델링 **· ZICO 시스템 성능 평가** - 비교군: Temporal Sharing, MPS(Spatial Sharing) - 성능 지표: 모델 학습 속도(초당 처리하는 샘플의 수) - 워크로드(6개 모델): NASNet, ResNet-110, ResNet-50, GNMT, DeepSpeech2, BERT · Batch Size를 다르게 한 두 모델을 동시에 한 GPU에 학습을 하는 방식 - 결과 분석 · 각 모델별로 메모리의 한계가 되는 Batch Size는 다름 · MPS의 경우 일정 batch size 이후에는 메모리 자원 초과에 따른 성능 하락을 보임

수행기관	개발 목표	개발내용 및 범위
		· Temporal Sharing의 경우는 메모리 자원 초과 문제는 발생하지 않았지만, GPU 자원을 두 학습 모델이 교차로 사용함으로 인한 overhead 발생 · ZICO는 GPU의 자원을 최대한으로 활용하면서 GPU의 메모리 자원을 초과하지 않는 스케줄링을 통해서 큰 batch size에서도 일정 수준의 성능을 확보할 수 있음을 보임  그림 114. ZICO 시스템 성능 평가 결과

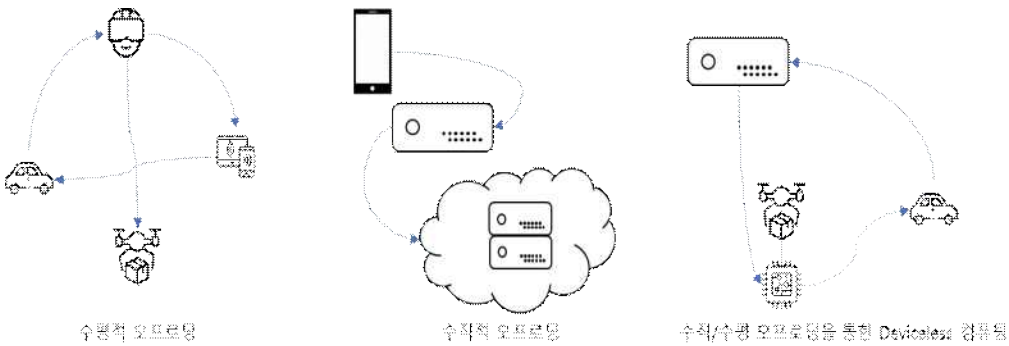
ETRI (주관)
· 메모리 중심 컴퓨팅 시스템 자원 스케줄링 기술 연구
· 혼성 메모리 기반 메모리 중심 컴퓨팅 시스템 기술 연구
· 대규모 메모리 기반 데이터 분석 가속 알고리즘 기술 연구
김강호

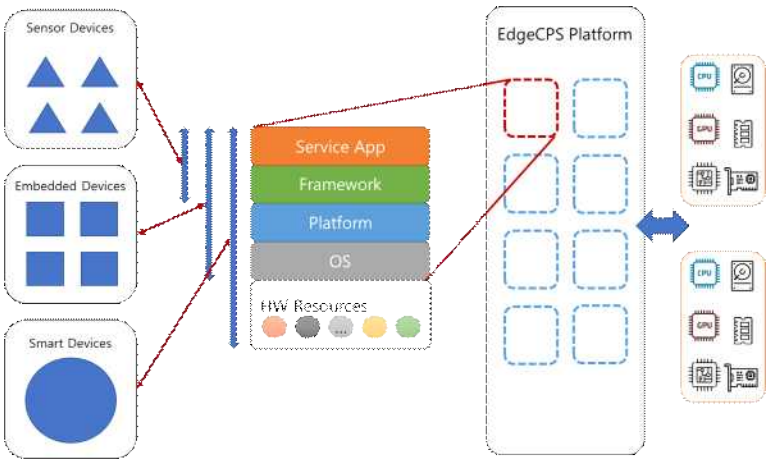
아주대학교 (공동1)
· 메모리 중심 컴퓨팅 시스템 기반 쓰레드 병렬화 기술 개발

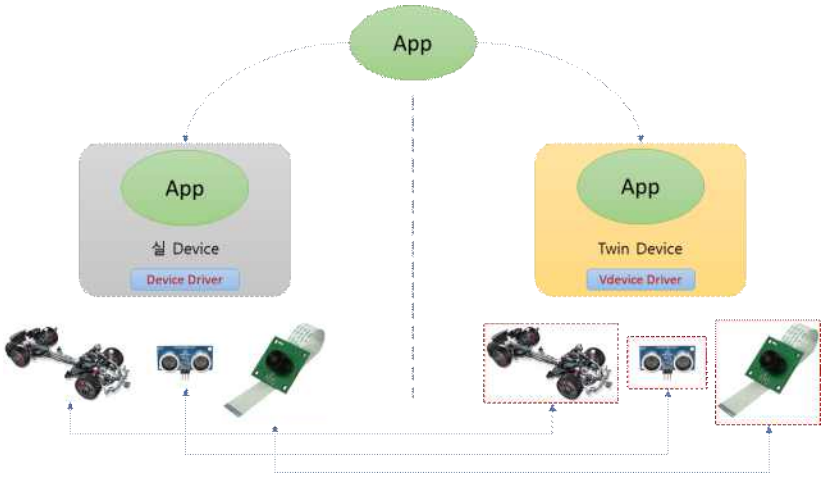
성균관대학교 (공동2)
· 비휘발성 메모리 기반 데이터 관리 기술 연구

UNIST (공동3)
· 인공지능 학습을 위한 인메모리 데이터 관리 연구

[EdgeCPS]

수행기관	개발 목표	개발내용 및 범위
ETRI (주관)	① 저자원 디바이스간 연산 능력 차이를 극복하는 디바이스리스 컴퓨팅 구조 설계	**• 성능 프로파일링을 통한 계층적 디바이스 관리 모델 설계** - 기술 개요 · 연구 정의 : 대규모 디바이스들에 대한 다양한 자원 증강의 실현을 위해 플랫폼 환경에서 가상화되는 디바이스 트윈의 동작 환경 구성을 위한 실 디바이스 분류와 가상화된 디바이스들의 증강 기술에 대한 설계를 위한 연구 수행 · 연구 목표 (1) 디바이스리스 컴퓨팅 연구 방향 수립 (2) 컴퓨팅 자원에 따른 실 디바이스의 계층 구조 연구 (3) 디바이스 분류에 따른 가상 환경의 Vertical SW 스택 정의 (4) 자원 증강을 위한 태스크/데이터 오프로딩 정의 (5) 오프로딩 관리를 위한 성능 지표 수립 연구 - 세부 개발 내용 · 디바이스리스 컴퓨팅 연구 방향 수립 (1) 디바이스리스 컴퓨팅 개념 . 중앙 집중화된 구조의 클라우드 컴퓨팅 기술은 대규모 디바이스에서 발생하는 대용량의 데이터와 네트워크 트래픽의 한계가 발생함 . 데이터가 발생하는 근처에서 컴퓨팅을 지원하는 에지 컴퓨팅 기반의 클라우드 연구가 확산하고 있음 (Cloud to the Edge) . 클라우드 서버에 비해 자원의 한계를 지니는 엣지 서버에서는 한정된 자원에 대한 효율적인 관리가 필요함 . 클라우드 환경에서 제안된 서버리스 컴퓨팅을 엣지 서버에 적용하여, 수평/수직 오프로딩 태스크/데이터 오프로딩을 활용하는 디바이스리스 컴퓨팅 기술에 대한 연구가 필요함  그림 115. 디바이스리스 컴퓨팅 실현을 위한 오프로딩 기술 (2) 디바이스리스 컴퓨팅 연구 방향 . 디바이스리스 컴퓨팅 실현을 위해서는 엣지 서버에서 가상화되는 디바이스 가상화 기술, 효율적인 오프로딩 지원을 위한 워크플로어 기반 SW 개발, 연산능력 차이 극복을 위한 서비스 기반 디바이스 가속 기술 개발이 필요함 . 1차년도는 디바이스 가상화를 위한 실 디바이스의 계층적인 구조 분석을 통해 가상화에 대한 모델을 정의 하고, 오프로딩 성능 연구를 통해 태스크 오프로딩과/데이터 오프로딩을 정의 하였음

수행기관	개발 목표	개발내용 및 범위
		· 컴퓨팅 자원에 따른 실 디바이스의 계층 구조 연구 (1) 동작 형태에 따른 디바이스 분류 . Actuators (작동기) : 특정 조건에 따라 물리적인 동작을 행하는 제어 장치 . Sensors (센서) : 환경의 변경 내용을 감지하여 전기적인 신호를 만들어 내는 장치 . Transducers (변환기) : 에너지를 하나의 형태에서 다른 형태로 변환하기 위한 장치로 IoT 환경에서는 데이터를 전송하는 내부 센서 및 작동기가 포함됨 . Microcontroller unit (MCU) : 마이크로칩에 CPU, RAM, ROM등을 포함한 장치로 간단한 작업을 실행할 수 있으나, MPU 보다 제한적인 컴퓨팅 파워를 지님 . Microprocessor unit (MPU) : MPU는 단일 또는 다중 집적 회로를 통해 CPU 기능을 수행하나, 추가의 주변 장치가 필요함 . SoC (System on Chip) : 하나의 칩에 시스템 자체가 포함된 장치로, 높은 컴퓨팅 자원과 다양한 주변기기를 포함하여 스마트 기기에서 주로 사용됨 (2) SW 스택 관점에서의 디바이스 분류 . Firmware : Binary, sensor, actuator . programmable mcu device : Boot Loader + Binary, aduino, simple IoT . low end device : single core . middle end device : multi core . common embedded device (3) EdgeCPS 디바이스 분류 . Senser Devices : 펌웨어 형태의 바이너리 형태 응용이 동작하는 디바이스로 정적으로 고정된 기능을 수행 할 수 있는 장치로, EdgeCPS 플랫폼에서는 응용 단위의 배포를 통해 증강되는 장치임. 기본적으로 부트로더 수준의 SW 제어 구조가 있고, 바이너리 업로드 형태로 응용을 배포할 수 있음 . Embedded Devices: 부트로더 혹은 운영체제를 통해 SW의 관기가 가능한 장치로 정의되며, 장치 특성에 의존적인 형태로, HW 특성에 민감한 장치임 . Smart Devices: 독립된 컴퓨팅 장치로 동적인 응용 수행이 가능하며, 하드웨어에 대한 의존도를 낮출 수 있는 Platform, Framework 등의 구조를 가지고 있음. Edge 서버로의 가상화를 통해 기능/성능/요구사항 증강이 가능함  그림 116. 디바이스 계층별 가상화 개념도 · 디바이스 분류에 따른 가상 환경의 Vertical SW 스택 정의 (실-가상 디바이스 연계를 위한 디바이스 가상화 단계) (1) Virtaul Machine 기반의 가상화 플랫폼 . IaaS 형태의 클라우드 컴퓨팅의 대표적인 구성 방법으로 소프트웨어 기반으로 하드웨어를 가상화하는 형태 . 대표적인 SW 플랫폼으로는 OpenStack, VMware 등이 존재하며, 하드웨어를

수행기관	개발 목표	개발내용 및 범위
		완전히 가상화하는 전가상화와 Host OS에서 Guest OS를 가상화 해주는 반가상화 구조로 분류됨 (2) 컨테이너 기반의 가상화 플랫폼 : . IaaS 형태의 클라우드 컴퓨팅의 대표적인 구성 방법으로 소프트웨어 기반으로 하드웨어를 가상화하는 형태 . 대표적인 SW 플랫폼으로는 OpenStack, VMware 등이 존재하며, 하드웨어를 완전히 가상화하는 전가상화와 Host OS에서 Guest OS를 가상화 해주는 반가상화 구조로 분류됨 (3) 마이크로서비스 기반의 서비스 기반 플랫폼 . IaaS 형태의 클라우드 컴퓨팅의 대표적인 구성 방법으로 소프트웨어 기반으로 하드웨어를 가상화하는 형태 . 대표적인 SW 플랫폼으로는 OpenStack, VMware 등이 존재하며, 하드웨어를 완전히 가상화하는 전가상화와 Host OS에서 Guest OS를 가상화 해주는 반가상화 구조로 분류됨 · 자원 증강을 위한 태스크/데이터 오프로딩 정의 (1) 태스크 오프로딩 . 태스크 오프로딩은 디바이스내에서 수행되는 SW(코드, 바이너리)를 Edge 서버 혹은 클라우드 서버에서 실행하는 동작을 의미함 . 태스크 오프로딩을 수준/단위/방법은 실 디바이스의 종류와 Edge 서버에서 가상화된 가상 디바이스 수준에 따라 다르게 진행 됨 . 1차년도에서는 가상화 컨테이너 환경인 kubernetes를 기반으로 디바이스와 서버를 정의하고 pod 형태로 제작된 응용의 migration 됨을 확인하였음 . 이후 연구에서는 workflow 형태로 구성된 App 혹은 서비스들에 대한 모델을 정희하고 관련된 오프로딩 규칙을 수립할 예정임  그림 117. 태스크 오프로딩 프로토타이핑 (2) 데이터 오프로딩 . 데이터 오프로딩은 네트워크로 연결된 두 개의 장치(디바이스-디바이스, 디바이스-서버, 서버-서버) 간의 물리적 데이터의 교환이 고려된 연산 행위를 의미함 . 1차년도에서는 데이터 오프로딩에 대한 시나리오를 정의하고, 데이터 관점에서 고려되어야 하는 성능 지표에 대한 정의를 수립하였음 . 이후 연구에서는 데이터의 타입과 용량에 따른 오프로딩 규칙을 정립하고, 함수 기반의 스토리지 기술에 대한 모델을 설계할 예정임

수행기관	개발 목표	개발내용 및 범위

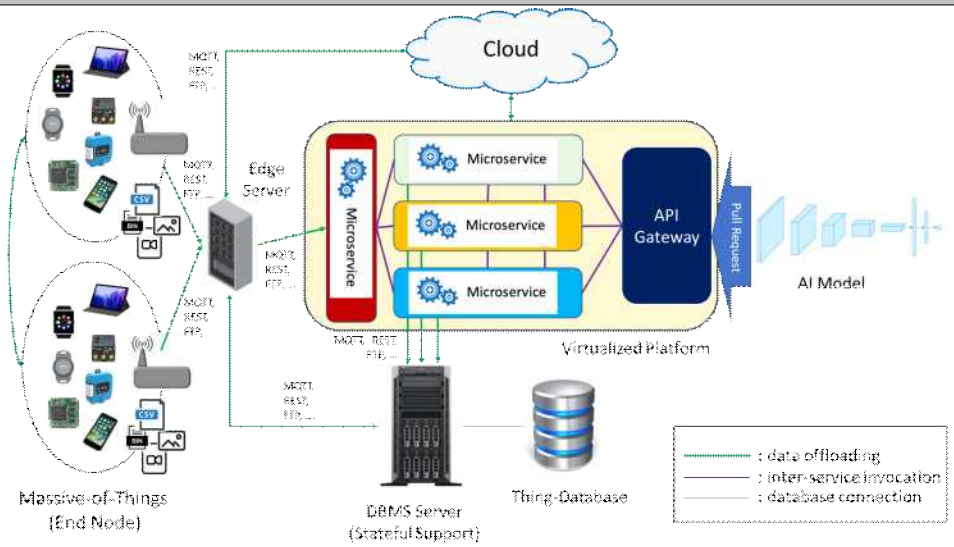


그림 118. 데이터 오프로딩 흐름도

· 오프로딩 관리를 위한 성능 지표 수립 연구
(1) 기존의 오프로딩에 관련된 논문 연구를 통해 성능 영향 인자를 정의함
(2) 영향인자에 따른 실험 데이터 수집

표 17. 오프로딩시 고려되는 성능 영향 인자

	성능 인자	정의
1	**네트워크 유형**	데이터 오프로딩에 사용되는 네트워크 유형
2	**통신 규약**	데이터 오프로딩에 사용되는 통신 규약
3	**서버 가상화 정도**	오프로딩 서버의 컨테이너 갯수
4	**데이터 압축**	오프로딩할 데이터의 압축 유무
5	**데이터포맷**	오프로딩할 데이터의 유형
6	**오프로딩 위치**	데이터 오프로딩 장소
7	**데이터 이질성**	오프로딩할 데이터 타입의 이질적인 정도
8	**디바이스 민감도**	실디바이스 장치 자원 차이
9	**(응용 실행을 위한)선택적 오프로딩 아키텍처**	실디바이스와 가상화 플랫폼 사이의 데이터 처리 비율
10	**데이터 규모**	오프로딩할 데이터의 크기

- 향후 개발 내용
  · 실디바이스 증강을 위한 인프라 모델 설계
  · 디바이스별 핵심 관리 요소 분석
  · 계층별 디바이스 관리 모델
  · 가상 디바이스 관리 인프라 구축 기술
  · 자원 증강지원을 위한 EdgeCPS 서비스/응용 분석
  · 디바이스-가상 디바이스 컴퓨팅 증강 기술
  · 디바이스-디바이스 컴퓨팅 증강 기술
  · 복합 오프로딩을 활용한 자원 증각 기술
  · QoE 보장 디바이스 최적 활용 동적 오프로딩 기술
  · QoE 지원 오프로딩 스케줄링 기술

수행기관	개발 목표	개발내용 및 범위

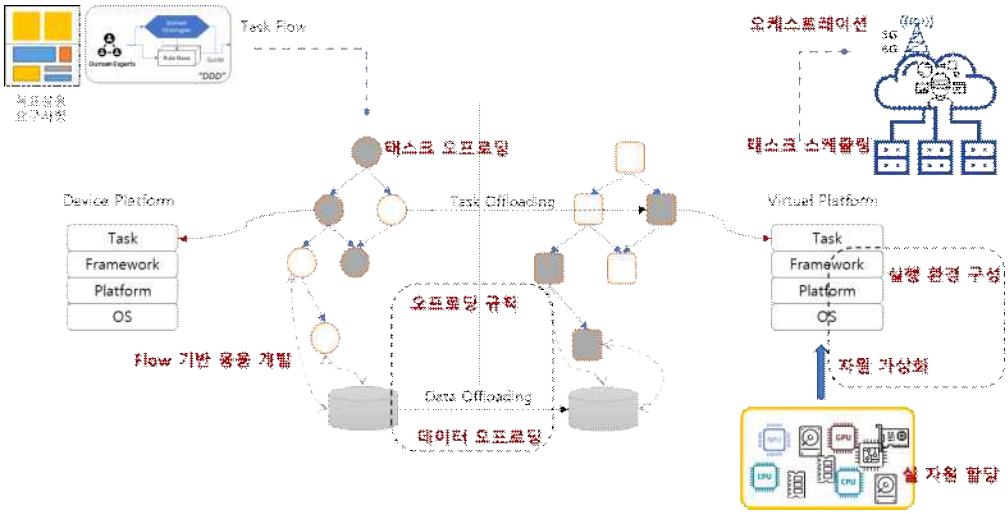


그림 119. Deviceless 컴퓨팅 추후 연구 계획

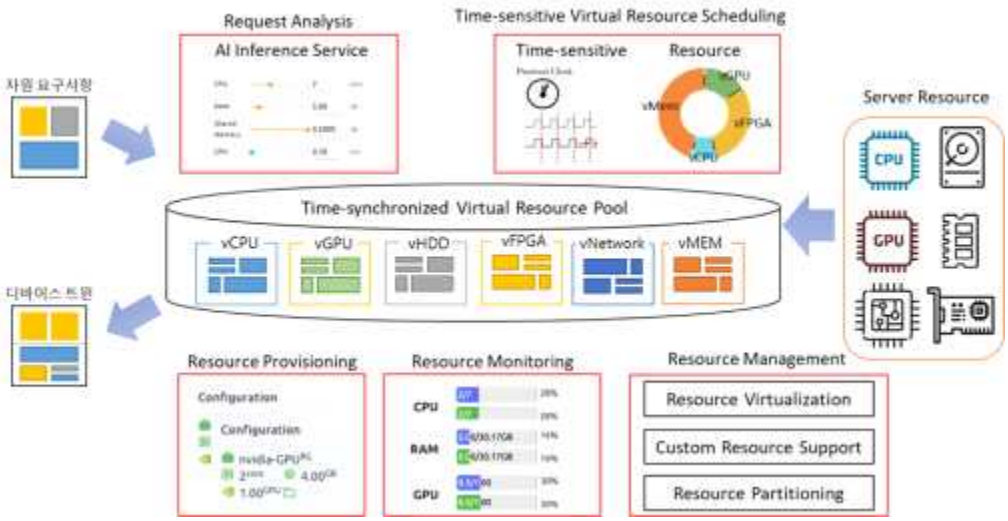
• 인공지능 연산 가속 지원 HW 자원 가상화 기술 정의 및 가상화 자원 공유 모델 설계

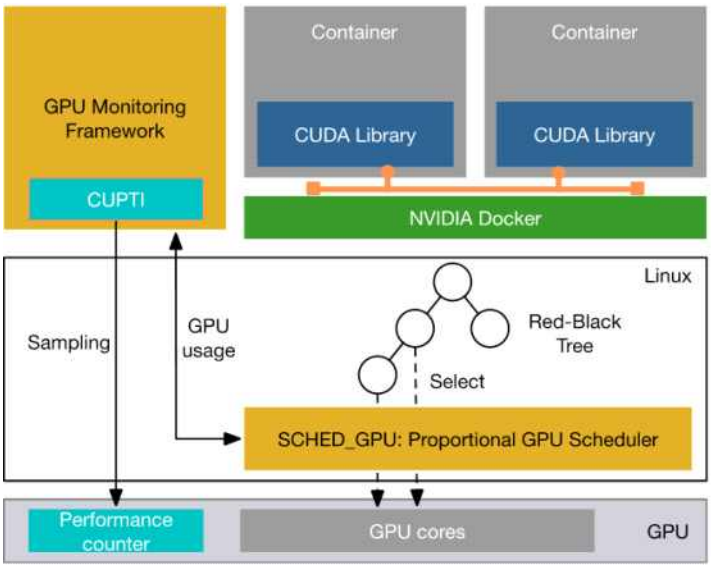
- 기술 개요
  · 연구 정의 : 엣지 서버에서 사용하는 HW 자원 가상화와 이종 가속기의 공유 모델을 통해 엣지 사이버물리시스템 환경에서 자원의 공유 및 활용이 가능해져 대규모 마이크로서비스의 수행이 가능하고, 디바이스 트윈와 연계하여 디바이스 간 연산 능력 차이를 극복하기 위한 서버자원 대여 기술에 대한 설계를 위한 연구 수행

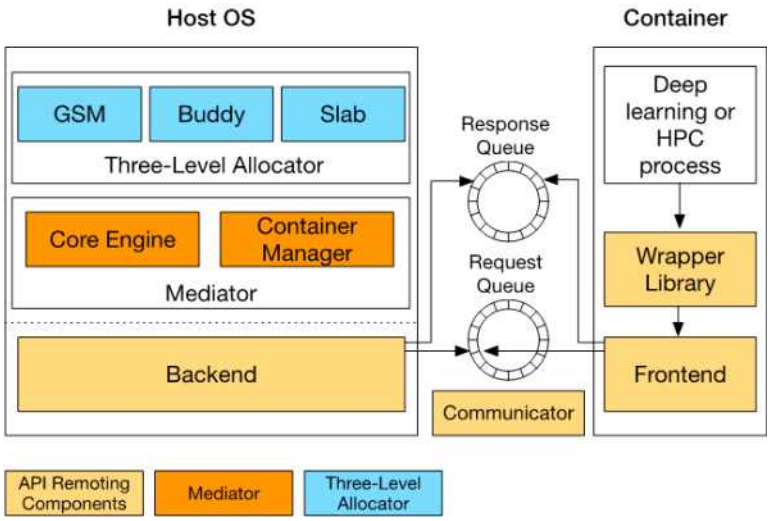
  · 연구 목표
    (1) 서버 자원 가상화 연구 방향 수립
    (2) 서버 자원 가상화 기술 정의
    (3) 서버 가상화 자원 관리 모델 연구
    (4) 1종의 HW 가속기 공유 모델 개발 (정량 목표)

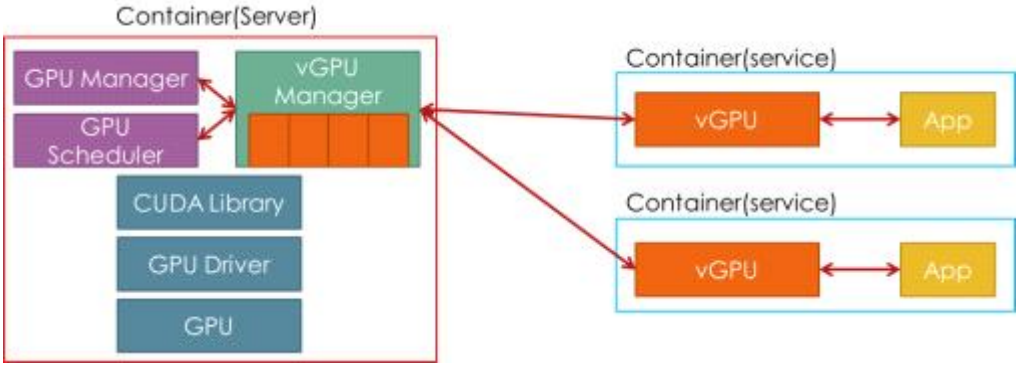
- 세부 개발 내용
  · 서버 자원 가상화 연구 방향 수립
    (1) 서버 자원 가상화 개념
      . 중앙 집중화된 구조의 클라우드 컴퓨팅 기술은 대규모 마이크로 서비스를 처리하기 위하여 발생하는 대용량의 데이터와 네트워크 트래픽의 한계가 발생하고 이를 줄이기 위하여 데이터가 발생하는 근처에서 컴퓨팅을 지원하는 엣지 컴퓨팅 기반의 클라우드 연구가 필요함
      . 엣지 컴퓨팅을 수행하는 엣지 컴퓨팅 서버에서 대규모 마이크로 서비스를 수행하기 위해서는 엣지 컴퓨팅 서버 HW 자원의 가상화가 필요함
      . 한정된 자원을 가진 엣지 컴퓨팅 서버에서 높은 수준의 자원 활용률과 분산 처리 능력을 제공하기 위하여 가상화된 자원을 실시간으로 파악하고 관리하는 연구가 필요함
      . 한정된 자원을 가진 엣지 컴퓨팅 서버에서 인공지능 연산과 관련된 마이크로 서비스를 수행하기 위하여 전용 컴퓨팅 연산기(GPU/FPGA)를 여러 마이크로 서비스들이 공유하여 수행할 수 있는 자원 공유/파티셔닝 기술에 대한 연구가 필요함

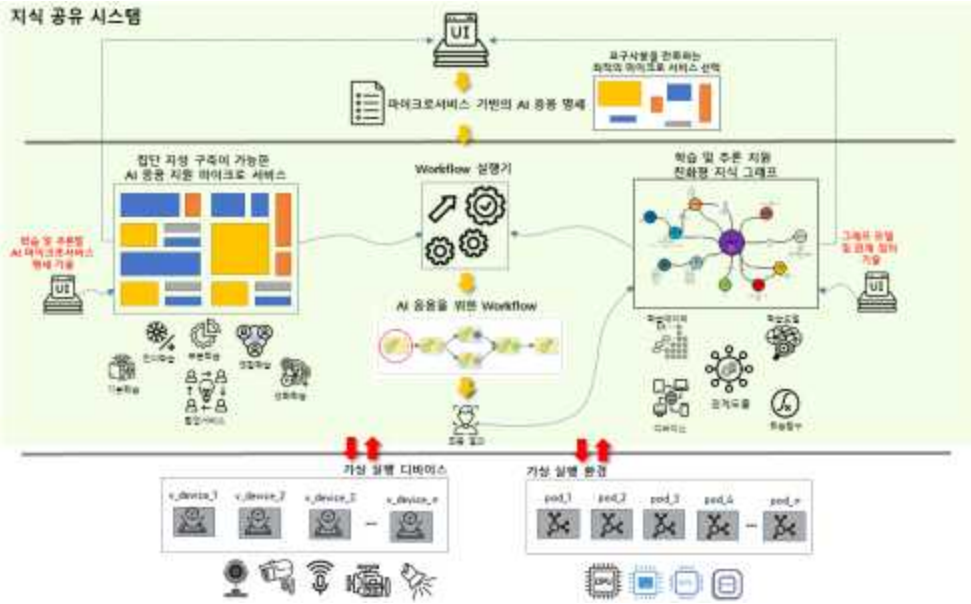
    (2) 서버 자원 가상화 연구 방향
      . 서버 자원 가상화를 위해서는 서버에서 사용하는 HW 가상화 기술, 대규모 마이크로 서비스를 수행하기 위한 가상화 자원 오케스트레이션 기술, 가상화 자원 모니터링

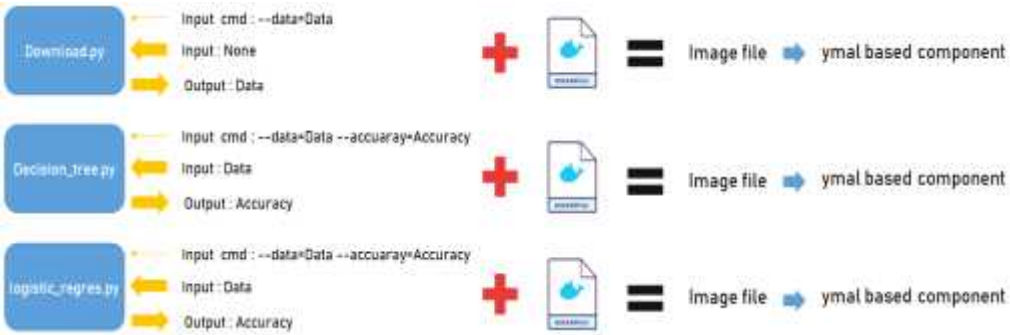
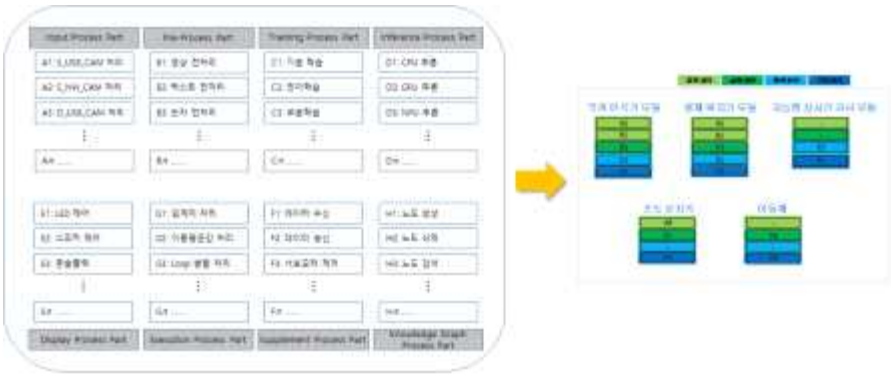
수행기관	개발 목표	개발내용 및 범위
		기술 개발이 필요함 . 1차년도는 인공지능 연산 가속 지원 HW 자원 가상화 기술에 대한 정의와 가상화된 자원을 공유하기 위한 모델을 설계하였음 · 서버 HW 가상화 자원 기술 정의 (1) 인공지능 연산 가속을 위한 서버 HW 자원 분류 . 범용 컴퓨팅 연산기 : CPU와 같이 인공지능 연산 뿐 아니라 모든 데이터를 연산/처리하는 연산기 . 인공지능(AI) 전용 컴퓨팅 연산기 : GPU/FPGA/NPU와 같이 인공지능 연산(학습/추론)을 최적 수행하기 위한 전용 연산기 (2) 서버 HW 가상화 기술 분류 . Virtual Machine : 하이퍼 바이저를 사용하여 독립된 공간에 대한 HW 자원을 할당하는 기술 . Container : Host OS 커널을 공유하여 필요한 만큼 동적으로 HW 자원을 사용하는 기술 (3) 서버 자원 가상화 기술 정의 . 하나의 HW 자원을여러 응용이 나누어 쓸 수 있는 자원 파티셔닝 가상화 기술 . 하나의 HW 자원을 여러 응용이 공유하는 가상화 자원 공유 기술 . 실시간 자원 상황을 모니터링 하는 기술 . 가용 HW 자원에 대한 수요-공급 논리와 우선순위에 따른 최적 스케줄링 기술 . 추가적인 디바이스 확장이 가능한 가상화 자원 관리 기술  그림 120. 서버 자원 가상화 자원 시나리오 · 서버 가상화 자원 관리 모델 설계 (1) 서버 가상화 자원 관리 모델 구조 정의 . 1차년도에서는 마이크로 서비스가 수행될 서버 HW 자원을 가상화 하고 관리하는 모델을 설계하였음 . 서버 가상화 자원 관리 시나리오는 다음과 같음 * 마이크로 서비스 분석을 통한 자원 요구사항을 수집 * 수집된 요구사항을 분석하여 서버 가상화 자원 정의 * 정의된 서버 가상화 자원이 할당 가능한지 확인 * 할당이 가능하면 서버 가상화 자원 할당, 그렇지 않으면 대기하거나 할당되지 않음을 알림 * 할당된 서버 가상화 자원 정보를 디바이스 트윈 모델 관리 시스템에 전달

수행기관	개발 목표	개발내용 및 범위
		* 새로운 연산/가속기 자원이 추가될 경우 가상화를 수행하여 자원풀에 추가 (2) Request Analysis . 서버 가상화 자원을 할당하기 위한 요구사항 수집 모듈 * 엣지 서비스 지원을 위한 마이크로서비스 기술' 과 연계하여 서버 자원에 대한 요구사항 수집을 수행함 * 마이크로 서비스(빅데이터, AI 등)의 서버자원 요구사항 명세(profile)와 수집을 수행함 * 요구사항 명세에서 정의된 서버 자원(CPU/GPU 및 메모리 등) 종류를 결정함 . 서버 가상화 자원 요구사항 분석 수행 * 마이크로 서비스 요구사항에 맞는 자원을 할당 * 요구사항에 맞는 자원이 없을 경우 대체자원(GPU/FPGA->CPU)을 할당하도록 요구사항 수정 (3) Time-sensitive Virtual Resource Scheduling . 서버 가상화 자원 스케줄링 모듈 * 마이크로 서비스(빅데이터, AI 등)의 저지연 수행 목적으로 스케줄링을 수행함 * 서버 자원(CPU/GPU 및 메모리 등) 종류에 따른 시간 결정적(time-deterministic) 서비스를 제공함 . 마이크로 서비스에 할당된 서버자원 시간 동기화 수행 * 연동하는 마이크로 서비스들의 시간(Start time, Current time, Deadline 등) 자원을 공유하도록 시간 동기화 수행 * 연동하는 마이크로 서비스들이 동시에 동작 가능한 최대 서버 자원 조합을 찾아서 자원을 할당함  그림 121. GPU 가속 지원 스케줄링 구조 . 서버 자원 관리 수행 * Time-synchronized Virtual Resource Pool 모듈과 연계하여 현재 가용 가능한 서버 가상화 자원을 할당함 * Resource Monitoring 모듈과 연계하여 서버 가상화 자원의 실시간 가용률을 확인함 (4) Resource Management . 서버 자원 가상화 모듈 * 서버 자원(CPU/GPU 및 메모리 등) 종류에 따른 가상화와 관리를 수행함

수행기관	개발 목표	개발내용 및 범위
		* 가상화된 자원은 Time-synchronized Virtual Resource Pool에 등록됨 * FPGA, NPU 등 새로운 연산 HW 가상화 및 자원 할당 서비스를 지원함 * 자원 활용률을 높이기 위한 물리적 자원의 논리적 파티션을 수행함 * 가상화된 논리적 파티션 자원을 관리함  그림 122. GPU 메모리 자원 기술 설계 (5) Resource Monitoring . 서버 자원 모니터링 모듈 * Request Analysis 모듈에서 정의된 마이크로 서비스 별 자원 모니터링을 수행 * 서버 클러스터 전체/서버 노드별/자원 종류별 등 다양한 사용량 모니터링을 수행 . 서버 자원 정보 제공 * Time-sensitive Virtual Resource Scheduling 모듈에 실시간 자원 가용률 정보 제공 * Resource Provisioning 모듈을 통해 할당된 가상화 자원 모니터링을 수행하고 자원 할당 시점에서 남은 자원 정보를 제공함 (6) Resource Provisioning . 서버 자원 할당 모듈 * 마이크로 서비스의 서버자원 요구사항에 맞게 서버 자원을 할당함 * Resource Management 모듈과 연동하여 새로운 자원 추가 시 새로운 자원에 대한 할당/관리를 수행함 . EdgeCPS 자원 동기화 * 디바이스 트윈 관리 시스템과 연동하여 할당된 자원을 등록하고 실시간 동기화를 수행함 · 1종의 HW 가속기(GPU) 공유 모델 개발 (1) 물리 GPU 공유를 통한 가상 GPU 자원 할당 모듈 정의 . 1차년도에서는 가상화 컨테이너 환경인 kubernetes를 기반으로 마이크로 서비스가 수행 될 서버자원 중 GPU를 가상화하고 pod 형태 응용이 할당 모듈을 구현하였음 . 개별 응용들은 가상GPU 관리모듈을 통해 컨테이너 기반의 개별 서비스들이 GPU를 활용할 수 있도록 지원함

수행기관	개발 목표	개발내용 및 범위
		 그림 123. GPU 가상화 자원 관리 모듈 설계 (2) GPU 가상화를 통한 vGPU 할당 검증 . 초기 서비스 구성시 개별 컨테이너에 대한 필요 GPU 자원을 분할 함 . 분할된 가상 GPU에 대한 서비스 제공을 지원하는 인터페이스를 구성함 . MIG(Multi-Instance GPU) 기술을 사용하여 하나의 GPU를 여러 응용에서 공유하는 HW 지원 가상화를 수행하였음 . 클러스터 환경에서 GPU를 사용하는 복수의 노드를 통한 GPU 가상화를 위한 연구가 필요함 그림 124. GPU 가상화를 통한 서비스 동시 지원 - 향후 개발 내용 · 서버 HW 자원 공유 기술 개발 (1) 하나의 GPU 자원에 대한 복합 공유(HW 지원 + SW 지원) 가상화 기술 (2) Time-synchronized Virtual Resource Pool 관리 기술 · 서버 가상화 자원 스케줄링 기술 개발 (1) 동시 수행되는 다수의 서비스의 자원 간섭을 줄이기 위한 우선순위 기반 스케줄링 기술 (2) 스케줄링을 위한 실시간 모니터링 기술 (3) 서버 공유자원 할당 관련 연계 기술 개발 (4) Request Analysis 기술 (5) Resource Provisioning 기술

수행기관	개발 목표	개발내용 및 범위
		• 집단 지성 구축을 위한 지식 공유 미들웨어 설계 및 프로토타입 - 기술 개요 · 연구 정의 : AI 지원 마이크로 서비스와 지식공유 그래프의 연계를 통한 EdgeCPS 환경에 최적의 수행 가진 AI 응용을 제공하기 위한 플랫폼 개발 · 연구 목표 : EdgeCPS 기반의 인공지능 서비스 제공을 위해서 AI 지원 마이크로서비스 실행기술과 지식공유 그래프와 연계 가능한 지식 공유 시스템 프로토타입 개발 (1) 파이프라인 기반의 AI 마이크로서비스 지원 플랫폼 개발 (2) Component 기반의 AI 응용 시나리오 및 AI 응용 구현 (3) AI 응용 지원을 마이크로서비스 인터페이스와 지식 그래프 인터페이스 통합  그림 125. 지식 공유 시스템 연구개발 범위 - 세부 개발 내용 · 개발환경 (1) 서버 시스템: Master Node: 1대, Worker Node: 2대 . CPU: Intel(R) Core(TM) i7-9800X Octa-core CPU @ 3.80GHz . Memory: 126GB . GPGPU: NVIDIA Corporation TU116 [GeForce GTX 1660] . 운영체제: Ubuntu 20.04 LTS . 설치된 SW: Kubernetes 1.18.19, Kubeflow 1.19, Neo4j, OpenCV 4.5.1, python 3.4, pytorch1.4 (2) 단말 시스템: Raspberry PI4, RGB webcam . CPU: ARM 7 Cortex-A72 Quad-core CPU (1.5Ghz) . Memory: 4GB . GPGPU: NONE (But, VideoCore VI QPU) . 운영체제: Ubutun 20.04 . 설치된 SW: python 3.4, OpenCV 4.5.1, pytorch1.4 · 파이프라인 기반의 AI 마이크로서비스 지원 플랫폼 개발 (1) 파이프라인이 가능한 Component 기반의 AI 마이크로서비스 집합 정의

수행기관	개발 목표	개발내용 및 범위
		. 마이크로서비스 정의 시에 두 가지를 고려 * single responsibility를 가지는 독립 모듈화: 각 image에 대해서 입력 명령라인, 입력 값, 출력 값의 개수를 최소화하여 모듈의 독립성을 극대화  그림 126. 파이프라인 기반 AI 마이크로 서비스 정의 * AI 응용을 고려한 모듈 최적화: ① AI 응용에 대한 전체적인 코드 흐름을 분석한 결과를 바탕으로 8 개의 대분류 코드를 생성: 입력처리파트, 전처리파트, 학습파트, 추론파트, 출력파트, 실행파트, 기타파트, 지식그래프파트 ② 각 파트에 있는 세부 모듈은 다양한 AI 응용 (직원인식기모듈, 물체탐지기모듈, 지능형실시간감시모듈, 등) 생성 시에 차례대로 stacking되어 실자원 시스템 혹은 가상자원 시스템에서 실행  그림 127. 모듈 기반 AI 응용 구조 설계 (2) Component 기반의 AI 응용에 대한 파이프라인화 및 실행 기술 . 1차년도의 파이프라인은 오픈소스 파이프라인 ML 프레임워크인 Kubeflow 을 활용하였음 . Kubeflow의 KFP SDK 기반의 Component 기술 활용: ML 워크플로우를 지원하는 공개용 파이프파인 SDK를 활용하여 특정 프로그램이나 특정 entrypoint를 컨테이너화 할 수 있도록 ymal 기반의 Component 생성

수행기관	개발 목표	개발내용 및 범위

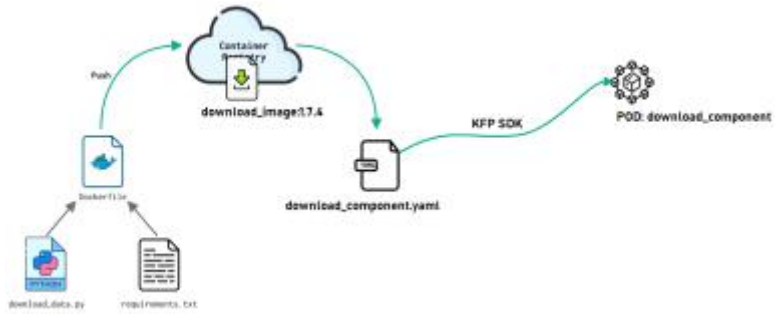


그림 128. AI 파이프라인 예제

. Component 기반의 파이프라인 생성 및 실행 검증: ML 워크플로우를 생성하는 kubeflow에서 데이터수집 Component, 의사결정트리 Component, 로지스틱회귀 Component, 결과물출력 Component 로 구성된 워크플로우를 PIPELINE(파이프라인)으로 생성하고 생성된 PIPELINE(파이프라인)을 RUN(실행)하여 정상적인 결과값을 도출하는지 확인

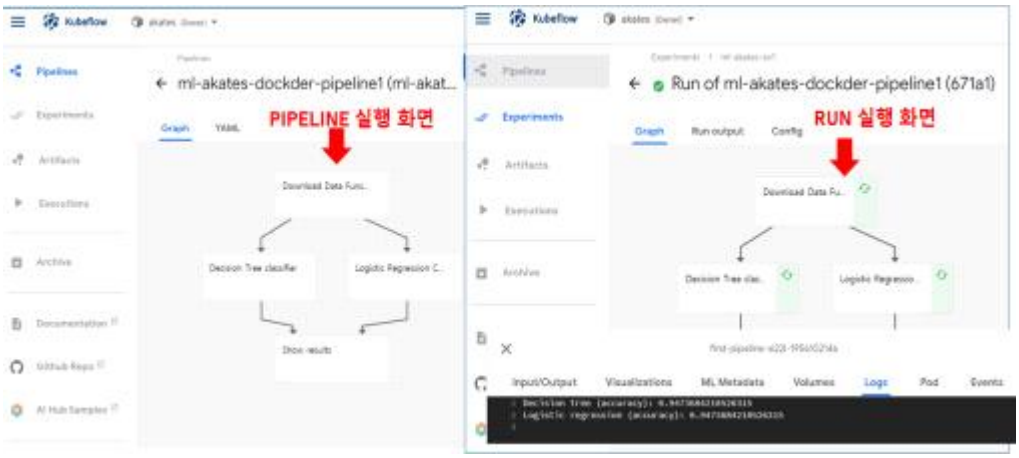


그림 129. ML 파이프라인 프레임워크 동작 화면

. EdgeCPS 환경에 AI 응용의 마이크로서비스화 적합 여부 실험적 검증: OpenCV 기반의 얼굴 인식기를 이용하여 large-scale과 small-scale 시스템에서 stand-alone 형태의 프로그램과 network 형태의 프로그램으로 구분하여 FPS, CPU 활용률, Memory 활용률을 측정

* stand-alone 형태의 프로그램: 응용 프로그램이 large-scale과 small-scale 시스템에 모놀리틱 구조로 수행되는 형태
* network 형태의 프로그램: large-scale centric 구조 (LSD-Centric)와 small-scale centric 구조 (SSD-Centric)로 나누어져 있고, LSD-Centric 구조는 large-scale 시스템이 모든 처리를 다하며 small-scale 시스템은 단지 IoT 디바이스처럼 영상 수집 기능만 한다. 그리고 SSD-Centric 구조는 응용 프로그램의 최종 결과물을 제공받는 서비스를 small-scale 시스템에서 수행하는 구조
* stand-alone과 network 형태의 프로그램 비교: stand-alone 기반의 small-scale 시스템에서 얼굴인식기를 수행한 결과, FPS는 불과 12.5 프레임인데 network 기반의 small-scale 시스템에서 얼굴인식기를 수행한 결과, FPS는 stand-alone 기반의 large-scale 시스템에서 나온 결과와 유사하게 22 프레임이므로, AI 응용을 위한 마이크로서비스화 적합 여부를 확인

수행기관	개발 목표	개발내용 및 범위

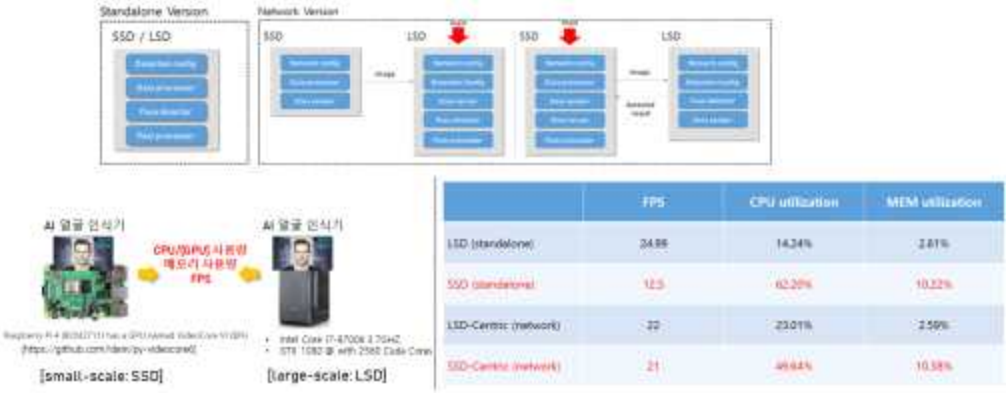


그림 130. 디바이스 별 응용 수행 성능 실험 결과

· Component 기반의 AI 응용 시나리오 및 AI 응용 구현

(1) 제안한 지식 공유 시스템 기반으로 한 서비스 시나리오: 열악한 환경을 가진 임의의 공장에서 AI 응용을 통해서 직원들의 얼굴인식과 장비착용 인식 서비스를 하기 위해서 지식 공유 시스템을 활용하는 절차를 시나리오로 구상

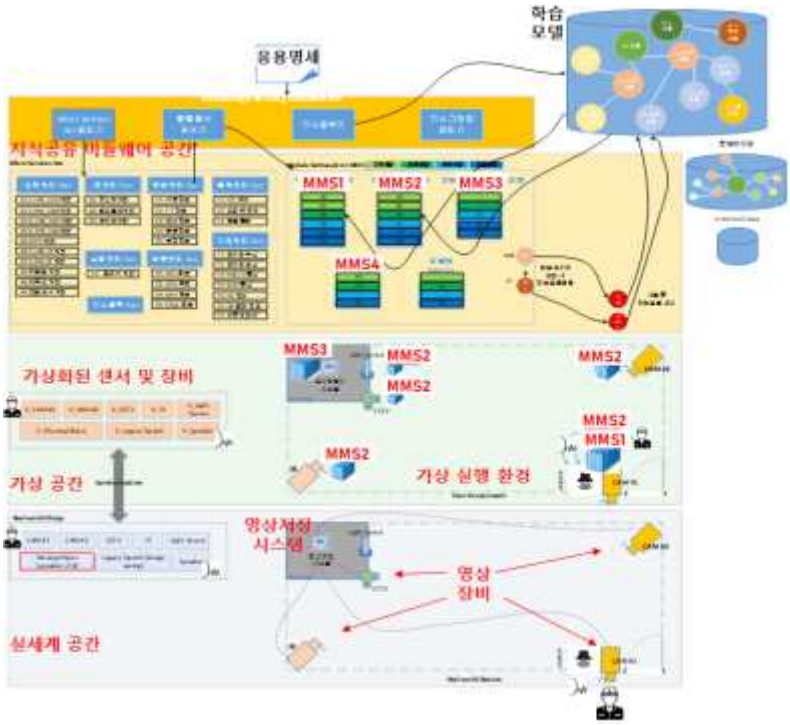


그림 131. AI 응용 구동 시나리오

(2) Component 기반의 AI 응용 설계 및 구현

. Component 기반의 OpenCV based 얼굴 인식기: Raspberry PI4에서는 영상을 수집하여 서버로 전송하고, 서버에서 추론하여 그 결과인 x,y,h,w 값을 이용하여 탐지 결과에 대해서 boxing하고, 그 실행은 pipeline화하여 실행시킴

. Component 기반의 ResNet50 based 얼굴 인식기: 학습모델을 기반으로 얼굴 인식을 해야 하므로 지식 공유 그래프에 있는 학습 모델을 참조

· AI 응용 지원을 마이크로서비스 인터페이스와 지식 그래프 인터페이스 통합

(1) AI 응용 마이크로서비스내에서 지식 공유 그래프 활용

수행기관	개발 목표	개발내용 및 범위

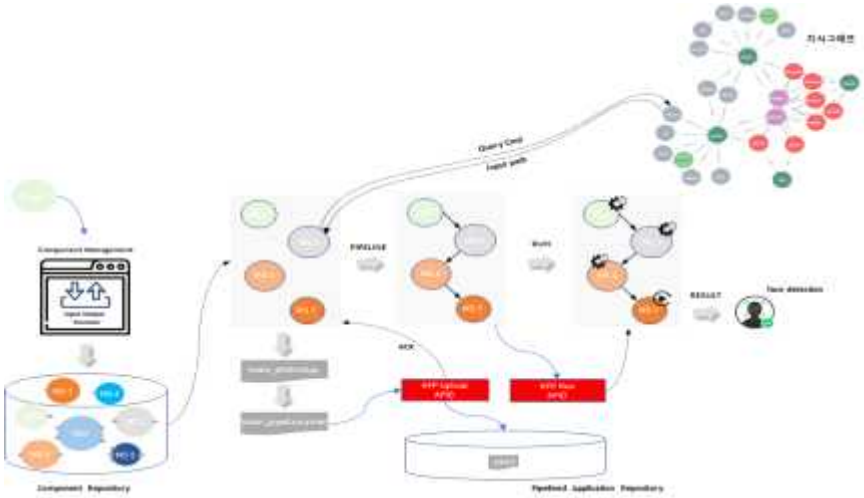
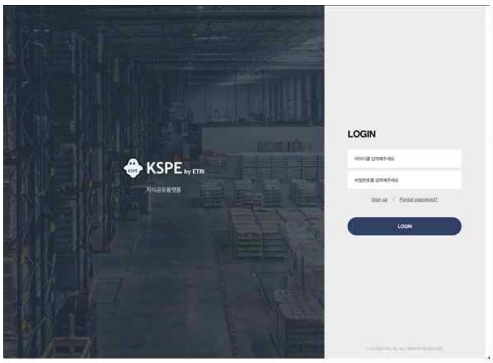


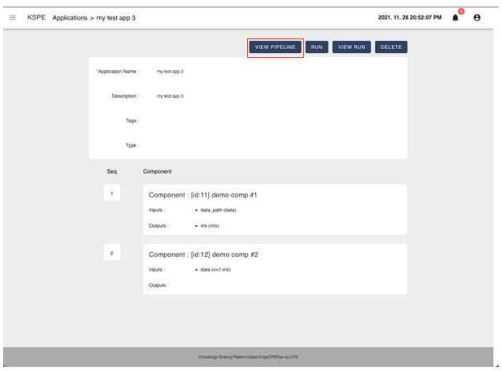
그림 132. 지식 그래프 활용 구조

(2) 지식 공유 미들웨어 통합 인터페이스 구축: Component 관리 인터페이스, Application 관리 인터페이스, 지식 그래프 관리 인터페이스로 구성된 웹페이지 구축

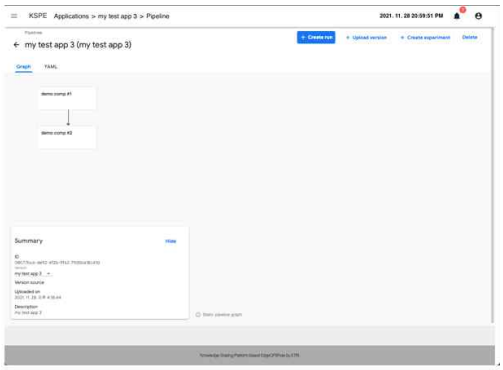


Home Page

Menu



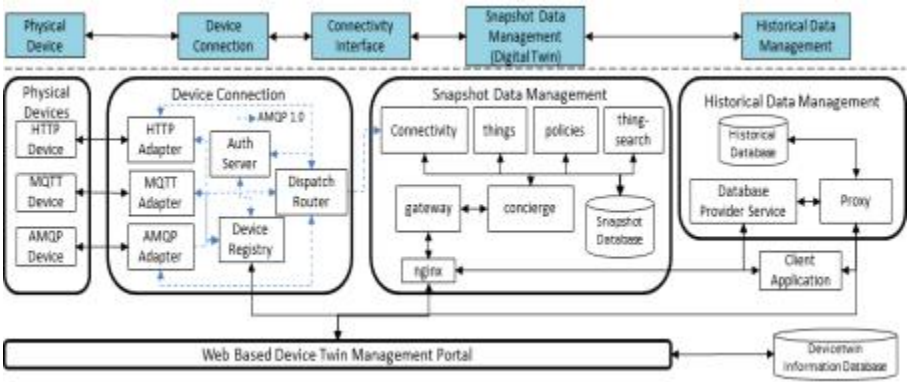
App. Detail - Click View Pipeline

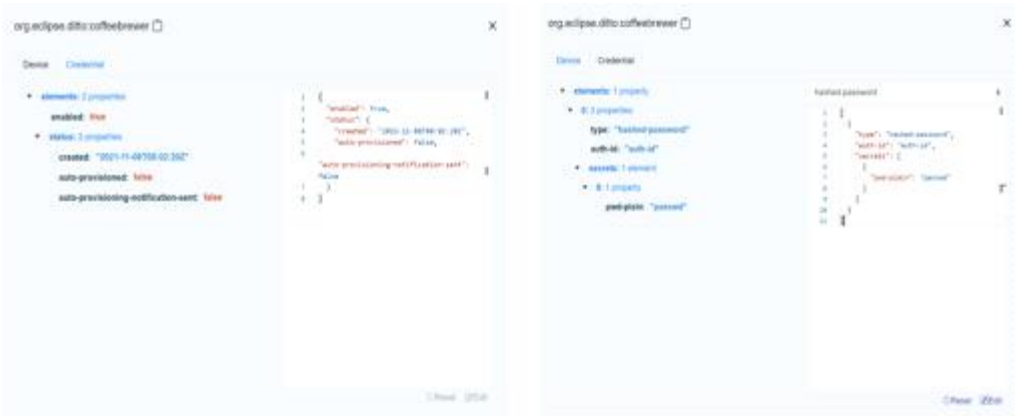


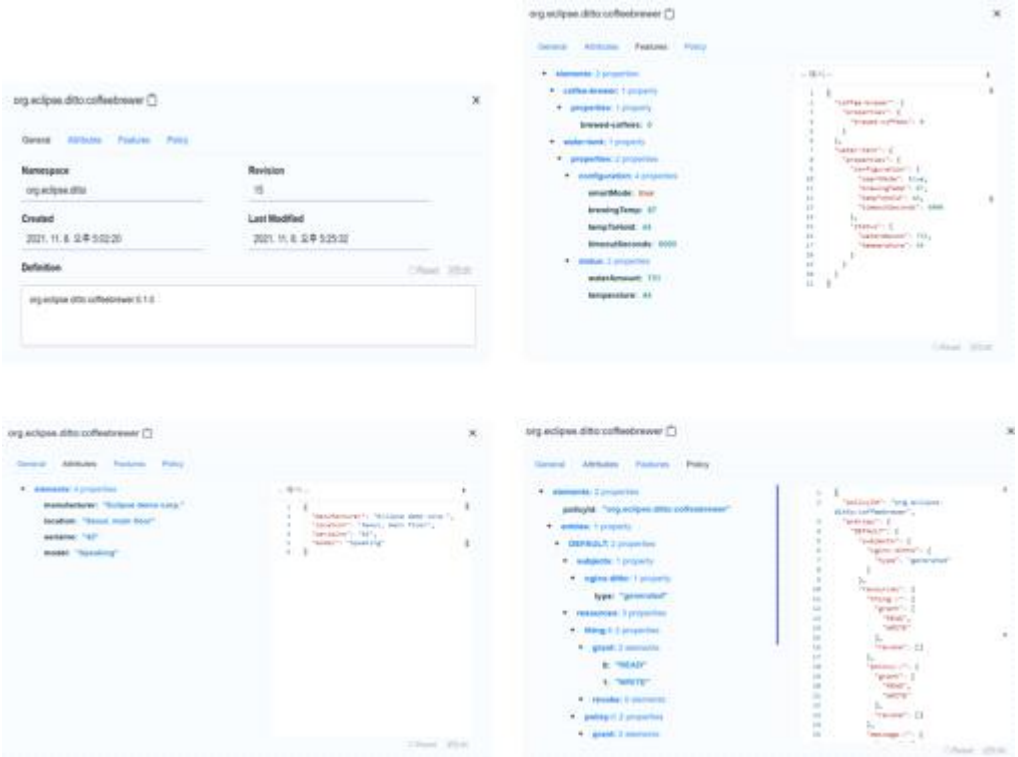
Pipeline View

그림 133. 지식 공유 미들웨어 통합 인터페이스

- 향후 개발 내용
  · 파이프라인 기반의 AI 마이크로서비스 지원 플랫폼 고도화
    (1) AI 응용을 위한 사용자 요구사항 정의
    (2) 시스템별/상황별을 고려한 EdgeCPS에 최적화된 AI 마이크로서비스화
    (3) 마이크로서비스화 된 Component에 대한 실용적인 성능 측정 및 디버깅 방안 수립
  · Component 기반의 AI 응용 시나리오 및 AI 응용 구현
    (1) 가상자원 및 가상 입력값 등을 고려한 EdgeCPS AI 응용 구현

수행기관	개발 목표	개발내용 및 범위
		(2) 다양한 학습 모델간 지식 공유 그래프 참고 방법 연구 (3) 가상하된 개별 노드 학습 / 추론 기술 연구 · AI 응용 지원을 마이크로서비스 인터페이스와 지식 그래프 인터페이스 고도화 (1) 멀티 사용자를 위한 웹 인터페이스 설계 및 구현 (2) 사용자 요구사항 처리 모듈 개발 (3) 마이크로서비스와 지식그래프간의 활용 자유도 극대화
	② 엣지 사이버물리시스템을 위한 대규모 디바이스 가상화 기술 설계	**• 디바이스트윈 기반의 시스템 운용 모델 분석 및 실자원 명세 기술** - 기술 개요 · 연구 정의 : 디바이스트윈이란 실제 물리 디바이스를 엣지사이버물리시스템 플랫폼에서 표현하기 위하여 가상의 트윈 형태로 추상화 한 것으로, 실제 물리 디바이스의 데이터를 수집하고 이를 플랫폼 및 백엔드 응용 서비스에서 사용할 수 있도록 인터페이스를 연구 · 연구 목표 (1) 디바이스트윈 관리 시스템 설계 및 프로토타입 제작 (2) 디바이스트윈을 표현하기 위한 데이터베이스 스키마 정의 (3) 100개 이상의 가상 자원 관리 성능 검증 (정량 목표) - 세부 개발 내용 · 디바이스 트윈 관리 시스템 설계 (1) 디바이스 트윈 구성 요소 정의 . 물리 디바이스 (Physical Device) . 디바이스 연결 (Device Connection) . 연결 인터페이스 (Connectivity Interface) . 스냅샷 데이터 (Snapshot Data) . 누적 데이터 (Historical Data) (2) 디바이스 트윈 구성 관리 시스템 구현 환경 구축 . OS: Ubuntu 20.04 . Kubernetes Version: v1.21.1 . 디바이스 연결 도구 : Eclipse Hono v1.8.2 . 디지털트윈 관리 도구 : Eclipse Ditto v2.0.1 . 디바이스트윈 관리 DB : mongoDB v10.28.4  그림 134. 디바이스트윈 관리 시스템 구조 · 디바이스트윈 표현 데이터베이스 스키마 정의 . "_id" : "namespace:deviceId" * primary key 값 . "namespace" : ""

수행기관	개발 목표	개발내용 및 범위
		* 디바이스트윈이 속한 namespace . "deviceId" : "" * 디바이스트윈의 ID . "physicaldevice" : { } * physical device의 정보 (key, value 쌍의 리스트로 관리) . "interfaces" : [ ] * connectivity interface의 정보 (connection id의 리스트로 관리) . "historicalValues" : [ ] * historical data의 정보 . namespace와 deviceId 정보를 활용해 Eclipse Hono, Eclipse Ditto의 API 서버로 데이터를 등록, 조회, 수정, 삭제할 수 있음 · 디바이스트윈 운용 플랫폼을 위한 UI 개발 (1) 주요 기능 . 디바이스트윈을 구성하는 다섯 가지 컴포넌트를 웹 UI상에서 쉽게 관리할 수 있도록 함 (등록, 수정, 삭제 등) (2) 웹 UI 스크린샷 . 디바이스 연결 관리 : Eclipse Hono 연결 정보 관리 (Device, Credentials 정보)  그림 135. 디바이스 연결 관리 UI 스크린샷 . 스냅샷데이터 관리 : Eclipse Ditto의 Thing 객체 정보 관리 (Attribute, Feature, Policy)

수행기관	개발 목표	개발내용 및 범위
		 그림 136. 스냅샷데이터 관리 UI 스크린샷 . 연결 인터페이스 관리 : Connectivity Interface 정보 관리 (연결 주체, 매핑 함수) 그림 137. 연결 인터페이스 관리 UI 스크린샷 . 클러스터 리소스 모니터링 : 클러스터 노드 자원 모니터링

수행기관	개발 목표	개발내용 및 범위

그림 138. 클러스터 리소스 모니터링 UI 스크린샷

· 가상 자원 100개 이상의 관리 성능 측정을 통한 정량적 목표 검증

(1) 실험 방법 정의

. 100개의 워크로드 생성기를 통한 패킷 전송

. 전송된 패킷에 대응하는 디바이스트윈의 가상 자원과의 공유 모델 구축

. RPS(Request Per Second) 측정을 통해 초당 100건 이상의 로드 처리 검증

(2) 실험 환경 구성

. 4개의 노드로 구성된 클러스터

표 18. 실험 환경 구성

Nodes	Resource Information			
	CPU	***RAM***	***OS***	***K8s Version***
Node1	*Socket 1:* Intel Xeon Silver 4208 2.1GHz [Total: 16 cores]	62GB	Ubuntu 20.04	Kubernetes v1.21.1
Node2	*Socket 1:* Intel Xeon Silver 4208 2.1GHz [Total: 16 cores]	62GB		
Node3	*Socket 1:* Intel Xeon Silver 4216 2.1GHz *Socket 2:* Intel Xeon Silver 4216 2.1GHz [Total: 64 cores]	125GB		
Node4	*Socket 1:* Intel Xeon Silver 4216 2.1GHz *Socket 2:* Intel Xeon Silver 4216 2.1GHz [Total: 64 cores]	125GB		

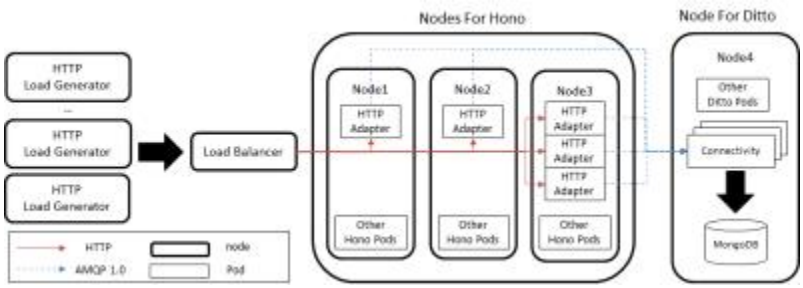
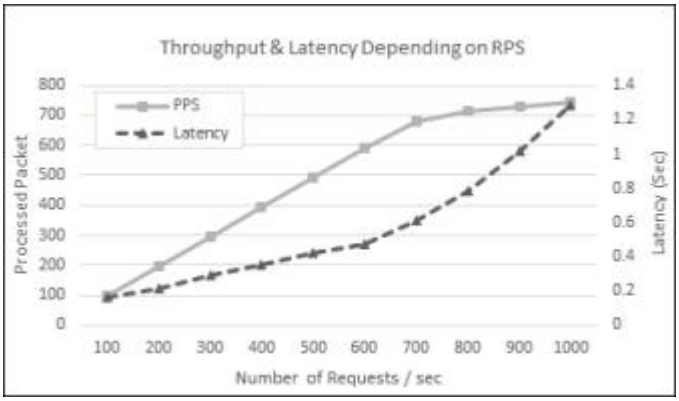
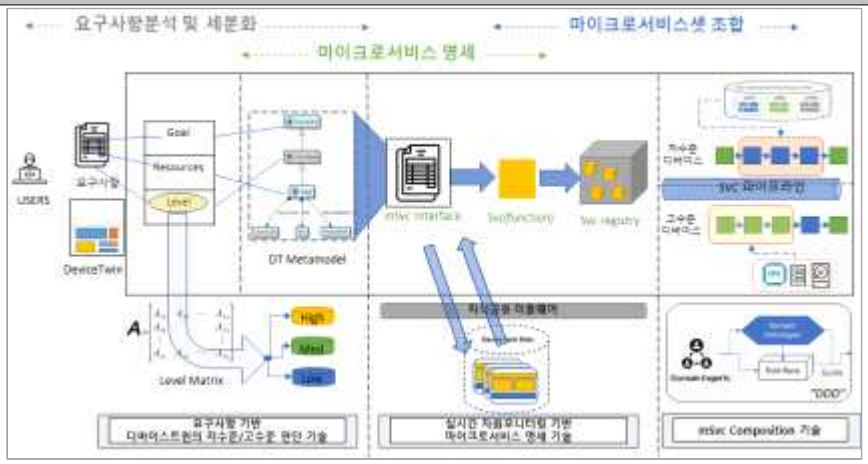


그림 139. 정량목표 검증을 위한 시스템 구조

수행기관	개발 목표	개발내용 및 범위
		(3) 실험 결과 . 워크로드 생성기에서 발생하는 처리요청에 대한 선형적 결과 도출 : 약 700건의 패킷 요청에 수렴됨 . 응답시간에 대한 증가 량도 700건이상부터 급격하게 나타남 . 1차년도 실험 결과 초당 100건의 자원 공유 모델을 충족하나 700개 이상의 자원모델 지원을 위한 연구가 필요함  그림 140. 요청에 따른 데이터 처리 결과 - 향후 개발 내용 · 디바이스트윈 기반의 단일 CPS 운용 명세 및 가시화 프레임워크 기술 * 디바이스트윈의 CPS 운용을 위한 인터페이스 기술 개발 : 디바이스와 응용의 Command & Control 메시지 처리를 위한 인터페이스 기술 * Historical Data 관리를 위한 Database 관리 모듈 구현 : 스냅샷 데이터만 관리하는 Ditto 데이터베이스 확장 * 디바이스트윈 모니터링을 위한 UI 기술 개발 : 디바이스트윈 관리 UI에서 확장한 디바이스트윈 현황에 대한 모니터링 UI 개발 **• 디바이스트윈과 서비스 간의 인터페이스 기술** - 기술 개요 · 연구 정의 : 컨테이너 기반 응용의 목적 또는 요청에 따라 필요한 서비스를 연계하는 기술로서 마이크로서비스 명세를 통한 디바이스트윈과 인터페이스의 연계 기술에 대한 설계를 위한 연구 수행 · 연구 목표 (1) 요구사항 기반 디바이스트윈의 저수준/고수준 판단 기술 (2) 실시간 자원모니터링 기반 마이크로서비스 명세 기술 (3) 마이크로서비스 합성 기술

수행기관	개발 목표	개발내용 및 범위
		 그림 141. 연구내용 대표도 - 세부 개발 내용 (개발 환경, 세부 개발/연구 내용) · 요구사항 기반 디바이스트윈의 저수준/고수준 판단 기술 (1) 요구사항 분석 . 개발자가 제공하는 요구사항을 기반으로 디바이스트윈 서비스 구동에 필요한 HW/SW Spec.을 세분화 . SW Inspection을 기반으로 디바이스트윈 Workload 자동 계산 . 도메인에서 정의한 요구사항을 기반으로 HW/SW Spec을 프로파일링 . 쿠버네티스에서 관리하는 Resource 정보에 맞춰 HW/SW Spec을 yaml에 적용 (2) 마이크로서비스를 위한 Domain Driven Design(DDD) . 사용자(Domain Experts)와 개발자(IT)간의 공통 된 Communication을 위해 필요한 모델링 프로세스 . 비즈니스 요구사항을 bounded context로 분리하고 각 도메인별 서비스 API, UI, Data를 도출 . Domain Driven Design을 참조해서 모놀리틱 시스템을 마이크로서비스 단위로 나뉠 수 있는 근거 모델로 활용 그림 142. Domain Driven Design Approach · 실시간 자원 모니터링 기반 마이크로서비스 명세 기술 (1) 마이크로서비스 명세 . 마이크로서비스 등록을 위한 메타데이터 정의 . EdgeCPS 디바이스트윈 관리자(Ditto)에서 정의한 메타데이터 참조 . 마이크로서비스 명세를 통한 디바이스트윈 관리자(Ditto)와의 연계

수행기관	개발 목표	개발내용 및 범위

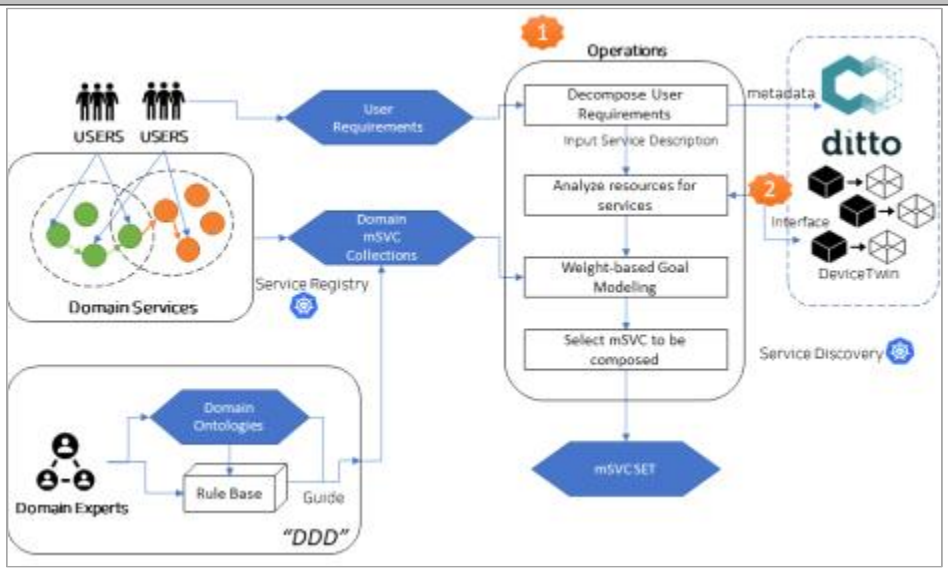


그림 143. 마이크로서비스와 Ditto와의 연계

(2) 마이크로서비스 유저 인터페이스

. 메타데이터를 기반으로 한 유저 인터페이스 개발

. 인터페이스를 통한 애플리케이션 스켈레톤 생성 및 서비스 연계

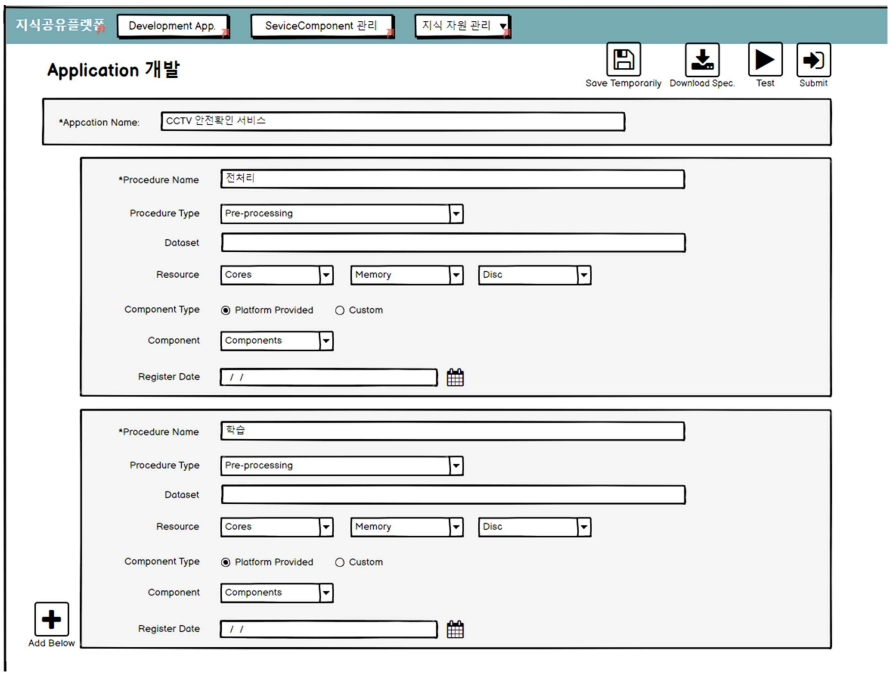


그림 144. 응용 명세 인터페이스

· 마이크로서비스 합성 기술

(1) 가상자원 조회 및 디바이스트윈과 연계 방법

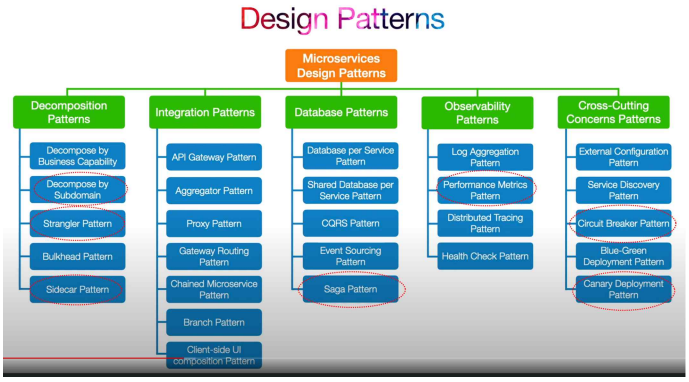
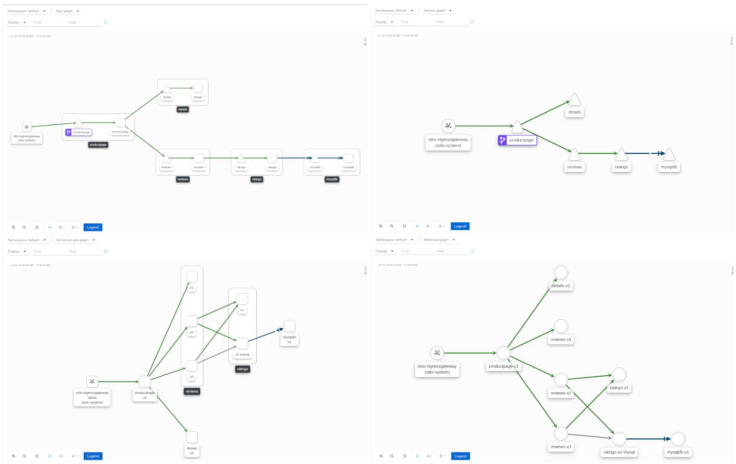
. 디바이스트윈 자원과 엣지 서버에서 관리하는 자원 정보 조회 및 수집

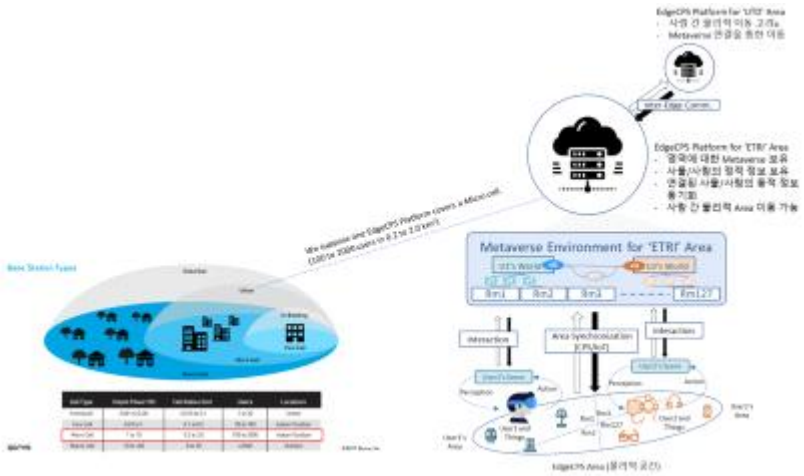
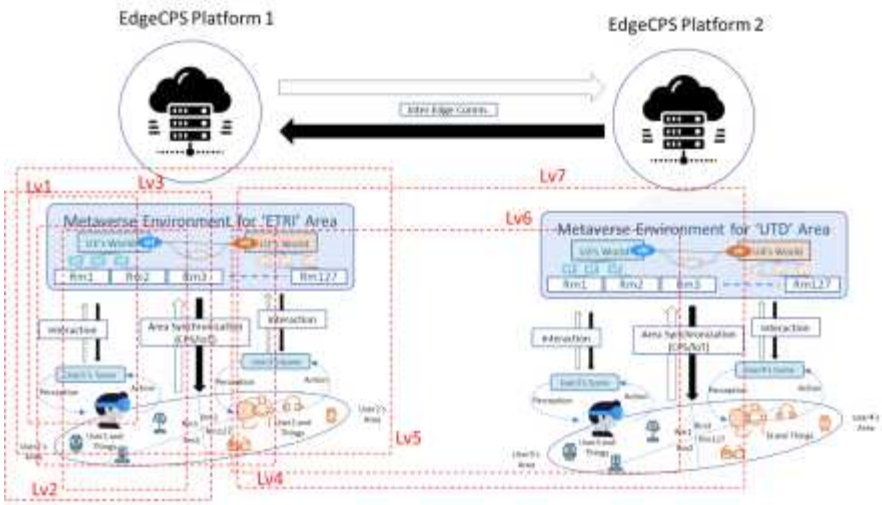
. 서비스 구동에 필요한 가상자원의 연계를 UI를 통해 구현

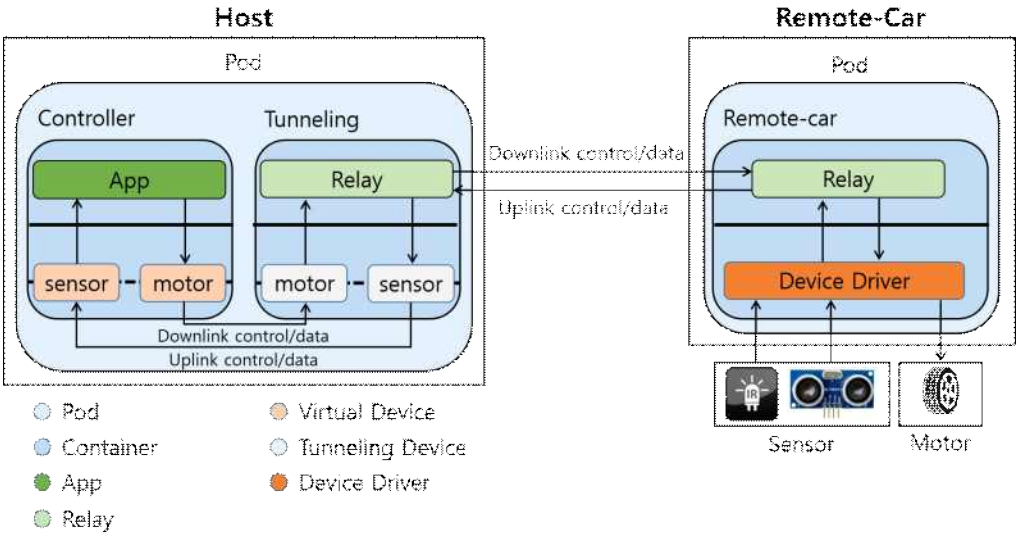
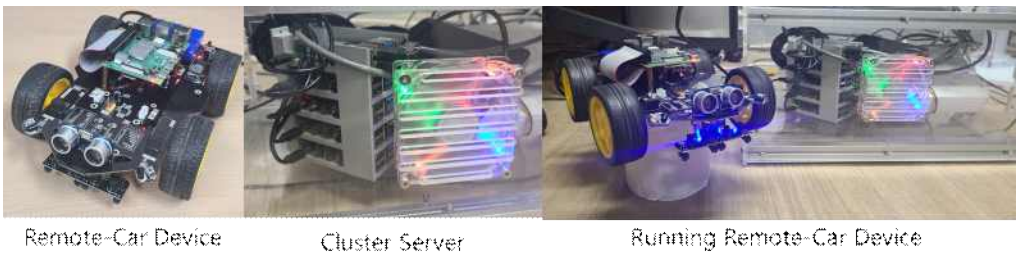
(2) 마이크로서비스 패턴 분류

. 44가지 마이크로서비스 패턴 중 도메인/Context에 맞는 패턴을 연구

. 마이크로서비스 패턴을 활용해서 마이크로서비스 집합을 구성하는데 활용

수행기관	개발 목표	개발내용 및 범위
		 그림 145. 마이크로서비스 패턴 분류 (3) 마이크로서비스 집합 표현 방법 연구 . 마이크로서비스 워크플로우 시스템 연구 . Istio, Kiali와 같은 서비스매쉬를 표현할 수 있는 오픈소스툴 활용  그림 146. 마이크로서비스 워크플로우 시각화 - 향후 개발 내용 · 이종가상화 자원 실시간 상황인지 기술 · 자원 사용량의 초고속 추출을 통해 정확한 리소스 사용량 정보 제공 · 서비스 프로파일 레지스트리를 통한 즉각적이고 효율적인 프로비저닝 지원 **• 비정형 멀티모달 데이터 정의 및 수집 인터페이스 기술** - 기술 개요 · 연구 정의 : EdgeCPS를 통해 임의의 영역(통신 장치의 전파 영역) 내 Massive Things에 대한 가상화가 가능하며, 가상화의 객체인 디바이스 트윈 간의 상호작용은 음성/영상/촉각 등 비정형 멀티모달 데이터의 교환을 통해 이루어짐. 본 연구에서는 단일 EdgeCPS 플랫폼의 영역 또는 타 EdgeCPS 플랫폼 영역내의 Things간의 상호작용 시 현실감있는 사용자 경험을 위한 데이터 정의 및 수집 인터페이스를 연구함 · 연구 목표 (1) EdgeCPS 상에서의 멀티모달 인터랙션 정의 (2) 로컬/원격 EdgeCPS 간의 멀티모달 인터랙션을 위한 플랫폼 요구사항 도출 - 세부 개발 내용 (개발 환경, 세부 개발/연구 내용) · EdgeCPS 상에서의 멀티모달 인터랙션 정의

수행기관	개발 목표	개발내용 및 범위
		(1) EdgeCPS 영역의 구성 요소 정의 . Area(무선 장치의 전파 유효 범위 / User의 실제 물리적 공간) . World(EdgeCPS플랫폼 내 가상세계 / 각 User'sworld가 존재) . Thing(Area 내의 물리적 사물) / vThing(가상세계 내에 모델링된Thing) . User(플랫폼 사용자) / vUser(가상세계 내에 모델링된사용자) . Scene(World를 통한 User의 1인칭 시점) . NPC(가상세계 운영을 위해 존재하는 가상의 인간) (2) EdgeCPS를 통한 메타버스 환경 개념 . EdgeCPS 플랫폼의 영역은 셀 반경 2km 이내, 100~2000명의 사용자 수준을 고려 . 반경 내 사용자 및 디바이스에 대한 가상 세계인 메타버스 환경을 EdgeCPS에 구현 . 메타버스 환경에는 연결된 사물/사람의 정적 정보, 동적 정보 동기화 (Digital Twin)  그림 147. EdgeCPS 기반의 메타버스 환경 - 향후 개발 내용 · 지리적으로 떨어진 EdgeCPS 영역 상의 각 메타버스 환경 간의 상호작용 기술 정의 · 메타버스 간 상호작용 단계 정의 및 프로토타입 개발  그림 148. EdgeCPS 간 상호작용 수준
대구대 (위탁1)	저자원 디바이스 자원증강 가상화	**• 실 디바이스 가상화 정의 및 응용 배포 기술**

수행기관	개발 목표	개발내용 및 범위
	인프라 구조 연구	- 기술 개요 · 연구 정의 : 저자원 디바이스 자원증강을 위한 디바이스 가상화 기술 연구와 실-가상 디바이스간 응용 배포 기술 연구 · 연구 목표 (1) 실-가상 디바이스 플랫폼 정의 (2) 가상 디바이스 드라이버 개발 및 가상 장치 연동 기술 (3) 실-가상 디바이스간 응용 배포 기술 - 세부 개발 내용 · 실-가상 디바이스 플랫폼 정의 . 컨테이너 기반 엣지 클라우드 환경 구축 . Pod 형태의 가상 플랫폼 정의  그림 149. 실-가상 디바이스 플랫폼 간의 관계 · 가상 디바이스 드라이버 개발 및 가상 장치 연동 기술 . 가상 디바이스 드라이버(모터, 적외선 센서) 개발 . 실 디바이스 장치 - 가상 디바이스 장치 간의 동기화 모듈 개발  그림 150. 실 디바이스 장치-가상 디바이스 장치간 연동 · 실-가상 디바이스간 응용 배포 기술 . Pod 기반 Car 응용 프로그램 개발 . 실-가상 플랫폼 Pod에 대한 응용 배포 . 응용 배포 관리를 위한 UI 화면 개발

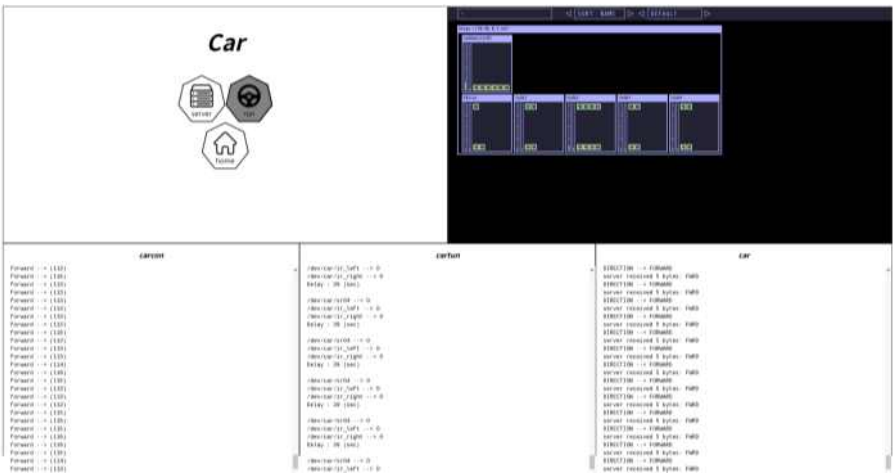
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>


그림 151. 실-가상 디바이스 간 Pod 응용 배포 UI

- 향후 개발 내용
· 실 디바이스 환경의 가상 디바이스 드라이버 고도화 (예: GPU, NPU)
· 모듈 기반의 플랫폼 구조 설계 (Vertical SW Stack 정의)
</td></tr>
<tr><td>경북대
(위탁2)</td><td>실디바이스-가상화 플랫폼 연동 데이터 관리 모델 연구</td><td>

• 오프로딩 최적화를 위한 성능 지표 모델 연구

- 기술 개요
· 연구 정의 : 실-가상 환경간의 태스크 오프로딩 최적화를 위한 성능 영향 인자 도출 및 데이터 오프로딩 모델 정의

· 연구 목표
(1) 클라우드 컴퓨팅에 대한 오프로딩 관련 선행 기술 조사
(2) 데이터 오프로딩 정의를 위한 오프로딩 성능 영향 인자 도출

- 세부 개발 내용
· 클라우드 컴퓨팅에 대한 오프로딩 관련 선행 기술 조사
* 오프로딩 관련 논문 56개 조사
* 논문에서 주장하는 요구사항 분류 (latency, energy, bandwidth)

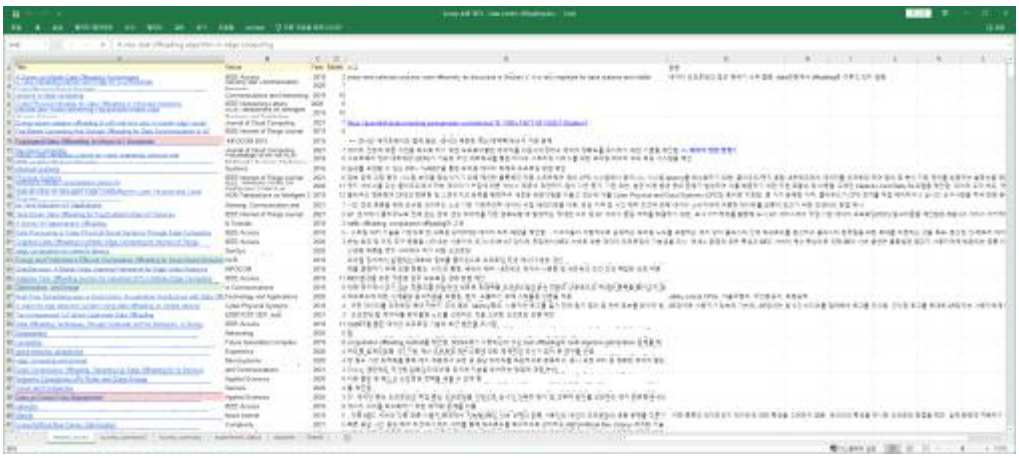
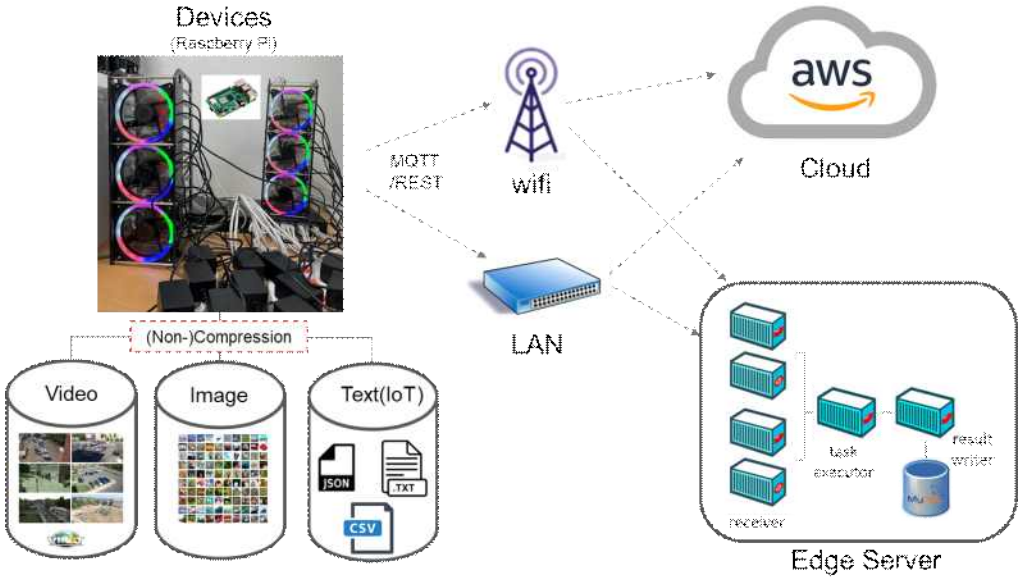
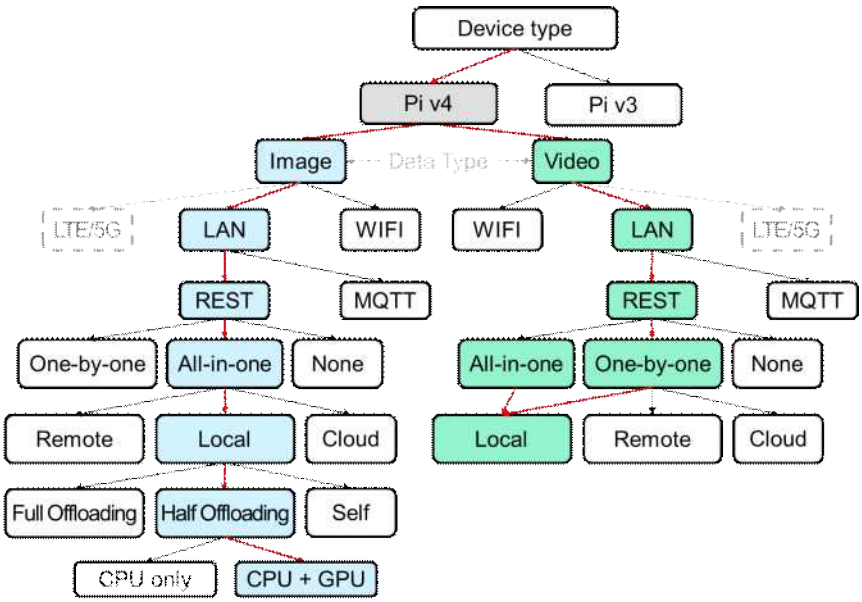
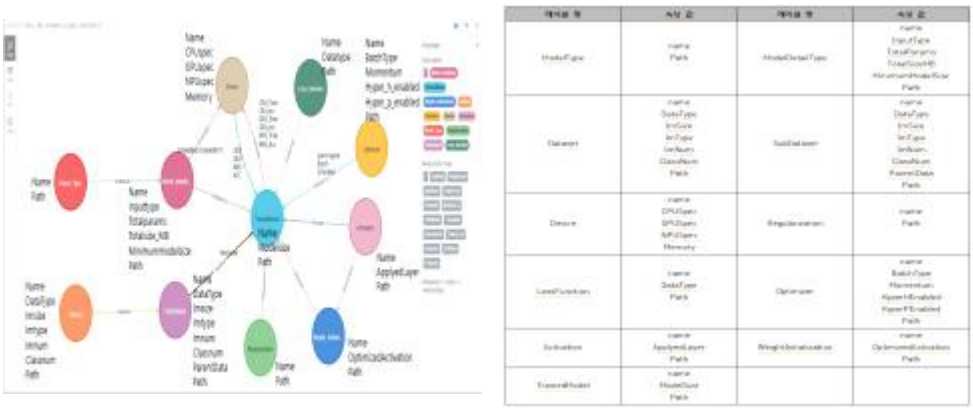


그림 152. 관련 연구 조사
</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		· 오프로딩 성능 인자 도출을 위한 실험 수행  그림 153. 상황별 오프로딩 성능 평가 실험 · 데이터 타입/네트워크 유형/디바이스 형태에 따른 오프로딩 규칙 도출  그림 154. 응용 수행 조건에 따른 오프로딩 규칙 - 향후 개발 내용 · 디바이스리스 컴퓨팅 환경의 데이터오프로딩 지원 기술 연구 · 디바이스리스를 위한 DaaS(Database as a service) 기술 연구
인하대 (위탁3)	EdgeCPS 기반의 인공지능 서비스를 위한 지식 공유 모델 연구	**• EdgeCPS 기반의 인공지능 서비스를 위한 지식 공유 모델 연구** - 기술 개요 · 연구 정의: 지식 공유 미들웨어의 EdgeCPS 기반의 인공지능 서비스를 위한 지식 공유

수행기관	개발 목표	개발내용 및 범위
		모델을 위한 지식 그래프 자동 구축 알고리즘 개발 · 연구 목표: EdgeCPS 기반의 인공지능 서비스를 위한 지식 공유 모델을 위해서 CNN 기반의 지식 그래프 자동 구축 알고리즘에 대한 연구 개발 (1) EdgeCPS 기반의 지식 공유 모델 설계 (2) EdgeCPS 기반의 지식 공유 그래프 구축 (3) 노드간의 관계 자동 생성 연구 - 세부 개발 내용 (1) 개발환경 정의 * 서버 시스템: Master Node: 1대, Worker Node: 2대 * CPU: Intel(R) Core(TM) i7-9800X Octa-core CPU @ 3.80GHz * Memory: 126GB * GPGPU: NVIDIA Corporation TU116 [GeForce GTX 1660] * 운영체제: Ubuntu 20.04 LTS * 설치된 SW: Kubernetes 1.18.19, Kubeflow 1.19, Neo4j, OpenCV 4.5.1, python 3.4, pytorch1.4, Neo4j (2) EdgeCPS 기반의 지식 공유 모델 설계 . 노드를 11개의 레이블로 나누고, 각 레이블은 특성에 맞는 속성을 갖고 있음. 레이블 명과 속성 명은 Upper camel case 표기법을 기반으로 작성하였고, name과 같이 지식그래프 플랫폼에서 자주 사용되는 속성만 소문자로 표기  그림 155. EdgeCPS 기반의 지식 공유 모델 설계 (3) EdgeCPS 기반의 지식 공유 그래프 구축 . 72개의 노드와 187개의 관계 형성 * 사용자에 의해 UI상에서 ModelType,ModelDetailType, Device, Dataset, SubDataset,Regularization, WeightInitialization, Activation, Optimizer, LossFunction 노드가 추가 * 사용자가 학습을 원한다면, 마이크로서비스로 인공지능 서비스를 요청하고, 마이크로서비스는 인공지능 명세를 기반으로 지식그래프를 통해 검색하여 필요한 컴포넌트들을 구성 * 학습이 수행되었다면, 학습 기본 정보 및 학습된 모델정보를 기반으로 지식그래프에 TrainedModel노드를 추가하고 관련된 관계를 연결 (4) 노드간의 관계 자동 생성 연구 . (인공지능 키워드) 지식공유 모델 디자인을 기반으로 레이블 및 속성값 비교 기반으로 특정 조건에 맞을 때 관계를 자동생성

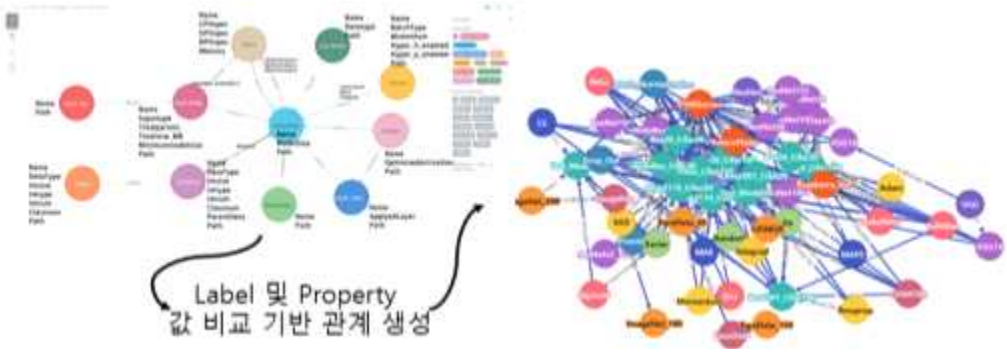
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>


그림 156. Prototype → Knowledge Design 기반 인공지능 키워드 관계 자동 생성

. (비 인공지능 키워드) BPedia라는 구글에서 정보로부터 구조화된 내용을 추출하기 위한 디비로부터 개체 관계를 분석
* 사용자로부터 새롭게 입력받은 데이터가 기존 데이터와 포함관계인지 DBPedia를 통해 분석하여 Subdata 관계를 연결
* SUBdata관계를 알았을 경우 어떠한 이점이 있나 Feasibility Test

<table>
<tr><th>Model</th><th>Accuracy (모델 B)</th><th>Params (MB)</th><th>Training Time (hour)</th><th>Accuracy (모델 C)</th><th>성능 차</th></tr>
<tr><td>ResNet20</td><td>60.15%</td><td>27</td><td>3</td><td>60.12%</td><td>0.03%</td></tr>
<tr><td>ResNet50</td><td>67.06%</td><td>85</td><td>6</td><td>66.26%</td><td>0.8%</td></tr>
<tr><td>ResNetV2-110</td><td>74.41%</td><td>170</td><td>11</td><td>73.14%</td><td>1.27%</td></tr>
<tr><td>ResNet-1001</td><td>77.3%</td><td>1940</td><td>71</td><td>76.08%</td><td>1.22%</td></tr>
</table>

그림 157. 인공지능 모델별 성능 분석

- 향후 개발 내용
· EdgeCPS 기반의 지식 공유 모델 고도화
· 다단계 형성이 가능한 지식 공유 그래프 구조 설계
· 협업필터링 기반 지식 공유모델 관계 가중치를 활용한 학습/추론 노드 추천 시스템
</td></tr>
<tr><td>경상대
(위탁4)</td><td>EdgeCPS 기반의 무인 이동체 운용 기술 연구</td><td>

• 디바이스트윈 연동형 무인이동체 운용 기술

- 기술 개요

· 연구 정의 : EdgeCPS의 영역 내의 무인 이동체 단말의 상태 정보를 획득하고, 이를 디바이스 트윈에 저장하는 프레임워크를 검증하고, 저장된 디바이스 트윈 데이터를 실제 응용 사례에 적용하는 사례를 연구함

· 연구 목표
(1) Pixhawk 비행 제어 시스템으로부터 드론의 운용 데이터를 추출하고 이를 Eclipse Ditto에 저장
(2) 디바이스 트윈 가시화 모듈에서 모사된 드론의 운용 데이터를 Ditto에 저장

- 세부 개발 내용
· 개발 및 운용 환경
(1) 무인 이동체와 지상 제어 시스템(GCC)
. 지상의 GCC(Unity)와 드론(아두이노) 간의 통신 및 정보 처리 영역을 Eclipse Ditto로 대체
. 배터리 상태, 자세(Roll, Pitch, Yaw), 고도, 속도, 충격 등 다양한 센서 장비 연계
</td></tr>
</table>

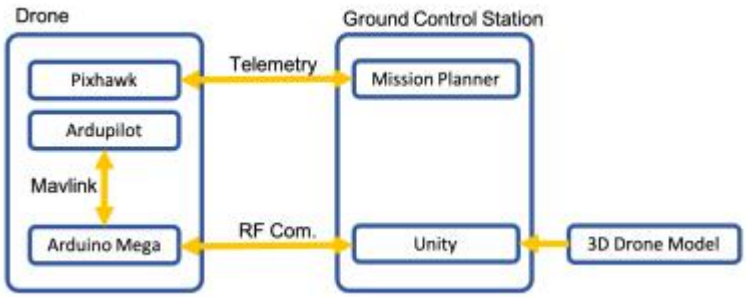
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>


그림 158. 지상과 드론 간 비행 제어 아키텍처

(2) 드론과 디바이스 트윈 간의 자세 동기화
. 드론의 자세 센서 데이터를 디바이스 트윈에 동기화하고, 이를 가시화
. 데이터 수신 지연은 평균 0.8초 정도이며, 지연에 따른 데이터 오차는 미미함

<table>
<tr><th>Operation</th><th>Mission Planner</th><th>Arduino (Received)</th><th>Difference</th></tr>
<tr><td>Roll (deg)</td><td>-1.07</td><td>-1.07</td><td>-</td></tr>
<tr><td>Yaw (deg)</td><td>181.38</td><td>181.37</td><td>0.01</td></tr>
<tr><td>Pitch (deg)</td><td>0.63</td><td>0.63</td><td>-</td></tr>
<tr><td>Altitude (m)</td><td>-4.35</td><td>-4.28</td><td>0.07</td></tr>
<tr><td>Battery (v)</td><td>10.52</td><td>10.53</td><td>0.01</td></tr>
<tr><td>Speed (m/s)</td><td>0.07</td><td>0.07</td><td>-</td></tr>
</table>
그림 159. 드론과 디바이스 트윈 간 자세 동기화 성능

- 향후 개발 내용
· Ditto의 성능 이슈 도출 및 해결 방안 연구
· EdgeCPS를 통한 화재 감지 알고리즘 동작 사례 개발
· 드론 상태 동기화 응용의 확장 개발 및 실드론-EdgeCPS와의 통합

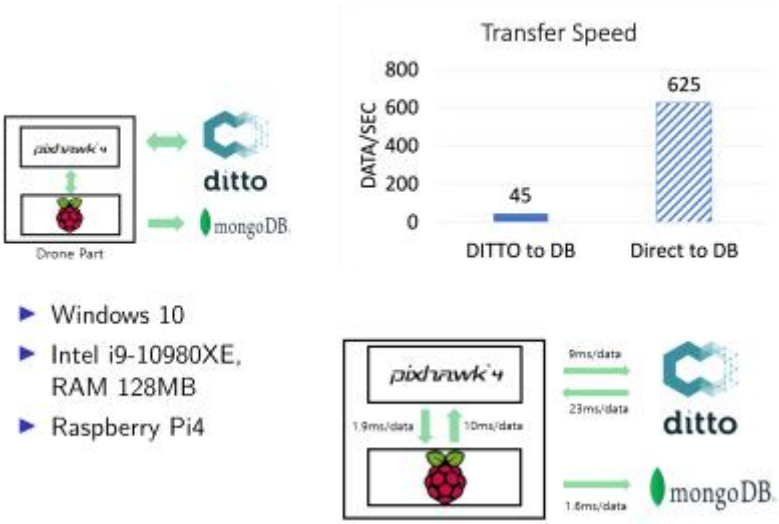


그림 160. Ditto의 성능 이슈 도출
</td></tr>
<tr><td>경희대
(위탁5)</td><td>EdgeCPS 기반의 비정형 멀티모달 데이터 처리 기술 연구</td><td>
• 다중 엣지CPS 서버 간의 하이브리드 인터랙션 기술

- 기술 개요 (정의, 목표)

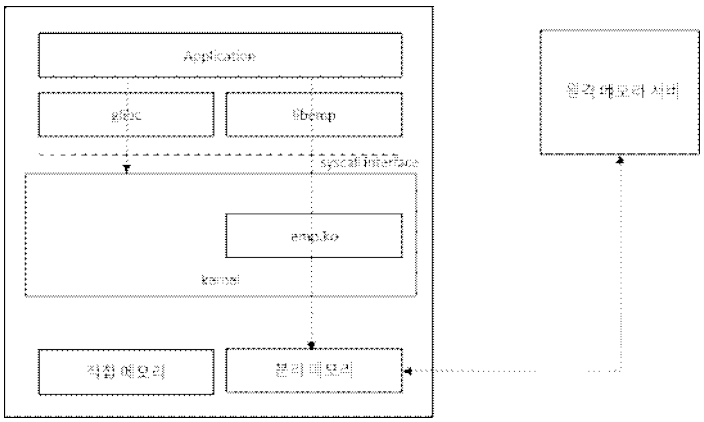
· 연구 정의 : EdgeCPS 간의 환경의 상태와 정보의 예측가능성/불가능성에 따라 적응형
</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		알고리즘 적용을 통해 저지연 텔레오퍼레이션 사례 연구 · 연구 목표 (1) 예측 가능한 원격지의 객체에 대한 비정형 멀티모달 데이터 구축 및 모델링 (2) 예측 불가능한 원격지의 객체 정보에 대한 실시간 모델링 및 예측 - 세부 개발 내용 (개발 환경, 세부 개발/연구 내용) · 개발 및 운용 환경 (1) 원격 환경 객체의 정확한 디바이스 트윈 모델링을 위한 데이터 기반 햅틱 모델링 . 촉감 모델링을 위한 촉감 데이터 수집 장치 활용 . 표면 거칠기에 대한 모델링과 렌더링 기술 활용 그림 161. 비정형 멀티모달 데이터 수집 (2) 원격 촉감 전달 테스트베드 시스템 구축 . 마스터 장치 (Omega3): 사용자의 제어기 . 슬레이브 장치 (Touch Premium): 원격 로봇 . 서버 및 EdgeCPS 시스템 . 데이터 전송: 단일 EdgeCPS를 통한 데이터 통신 환경 구축 그림 162. 원격 촉감 전달 테스트베드 - 향후 개발 내용 · 원격 객체에 대해 로컬 EdgeCPS에서의 촉감 실시간 추정 알고리즘 개발 · EdgeCPS 간 통신 상태 및 촉감 신호의 종류에 따른 스트리밍-모델링 스위칭 알고리즘 개발

수행기관	개발 목표	개발내용 및 범위
		· 원격 촉감 전달 테스트베드를 위한 멀티 EdgeCPS 환경(ETRI, 경희대) 구축

(3) (2단계) 1차년도(2022년도) 개발 내용 및 범위

[패브릭 메모리]

수행기관	개발 목표	개발내용 및 범위
ETRI (주관)	① 페타바이트급 주소 공간 관리 기술 연구	**• 네이티브 (Native) 응용 지원 대용량 메모리 분리 플랫폼 기술 개발** - 기술 개념 · 기 개발한 IaaS (Infrastructure as a Service)를 지원하는 가상머신용 패브릭 메모리 시스템인 Elastic Memory Platform (EMP)을 네이티브 (Native) 응용 프로그램 혹은 컨테이너 상에서 직접 적용. PaaS (Platform as a Service)와 SaaS (Service as a Service)를 지원하는 클라우드 데이터센터 메모리 분리 플랫폼으로의 지원 범위를 확대 · 네이티브 응용 프로그램이 다른 노드의 메모리 (원격 메모리)에 접근하기 위해서 추가의 응용 프로그램 개발 없이 대용량 메모리를 사용할 수 있도록 하는 장점을 가지며, 보유기술인 EMP에서 지원하는 응응의 메모리 접근 패턴에 따른 가변블록 관리, 빅팀 (Victim) 페이지 선택, 메모리 블록 프리패칭, 성능 일관성 제공 기능 등을 지원  그림 163. 네이티브 응용 지원 대용량 메모리 배치 기술 구조 - 기술 개발 내용 · 유저 레벨에서 동작하는 응용 프로그램이 메모리를 사용시 라이브러리(glibc 등)에서 제공하는 메모리 관리 함수(malloc, calloc, free 등)나 시스템 콜 함수(mmap, munmap 등)를 통하게 됨 · 응용 프로그램에서 메모리 사용이 발생할 경우 메모리 관련 함수와 시스템 콜 함수의 API 가로채기 (Intercept) 기법을 사용하여 EMP에서 관리하는 분리 메모리를 사용할 수 있도록 함. 응용 프로그램 소스 수정이나 재컴파일 없이 분리 메모리를 사용할 수 있다는 장점이 있음 · 예를 들어, 응용 프로그램이 링크되어 있는 C라이브러리의 malloc 함수를 호출할 경우, glibc 라이브러리의 malloc() 함수가 호출되는 대신 프리로드된 emplib의 malloc() 함수가 호출되게 함. libemp의 malloc() 함수는 직접 메모리가 아닌 분리 메모리를 대상으로 메모리 할당을 요청하게 되고 할당받은 메모리 주소를 응용 프로그램에 결과값으로 반환. · libemp 라이브러리(libemp.so)를 프리로드하는 방법은 아래 3가지 방법과 같이 LD_PRELOAD 환경변수를 설정하여 사용할 수 있음 ⑴ 특정 바이너리 실행 시에만 사용: 바이너리 실행시 LD_PRELOAD 환경 변수를 설정하여 LD_PRELOAD 동작 - LD_PRELOAD=libemp.so ./binary(실행파일) ⑵ 어떠한 바이너리든지 실행 시 마다 사용: 쉘에 등록하여 LD_PRELOAD 동작 - export LD_PRELOAD=libemp.so ⑶ 계정에 상관없이 사용: /etc/ld.so.preload에 설정하여 LD_PRELOAD 동작 · 본 과제에서 개발한 libemp 라이브러리는 응용 프로그램의 변경없이 메모리 관련 함수와 시스템 콜 함수를 지원하기 위하여 원격 메모리 서버와의 설정 및 연결을 수행하는 모듈, 메모리 함수 모듈, 메모리 시스템 콜 모듈, 라이브러리 내부 시스템 콜 모듈로 구성됨

수행기관	개발 목표	개발내용 및 범위
		· EMP 커널 모듈은 libemp에서 통해 전달받은 분리 메모리에 대한 접근 제어를 커널 레벨에서 수행. 원격 메모리 접근에 따른 페이지 폴트에 개입하여 EMP가 관리하는 영역일 경우 분리 메모리에 대한 관리를 수행하며, 원격 메모리 서버로부터 원격 메모리를 제공 받아 관리 · EMP 커널 모듈이 네이티브 응용 프로그램을 지원하기 위한 기능을 추가: 여러 가상주소 공간을 하나의 분리 메모리로 관리할 수 있는 기능 구현, 서브블록 단위로 정렬되지 않은 주소 공간을 지원하기 위한 partial mapping 지원 기능 구현

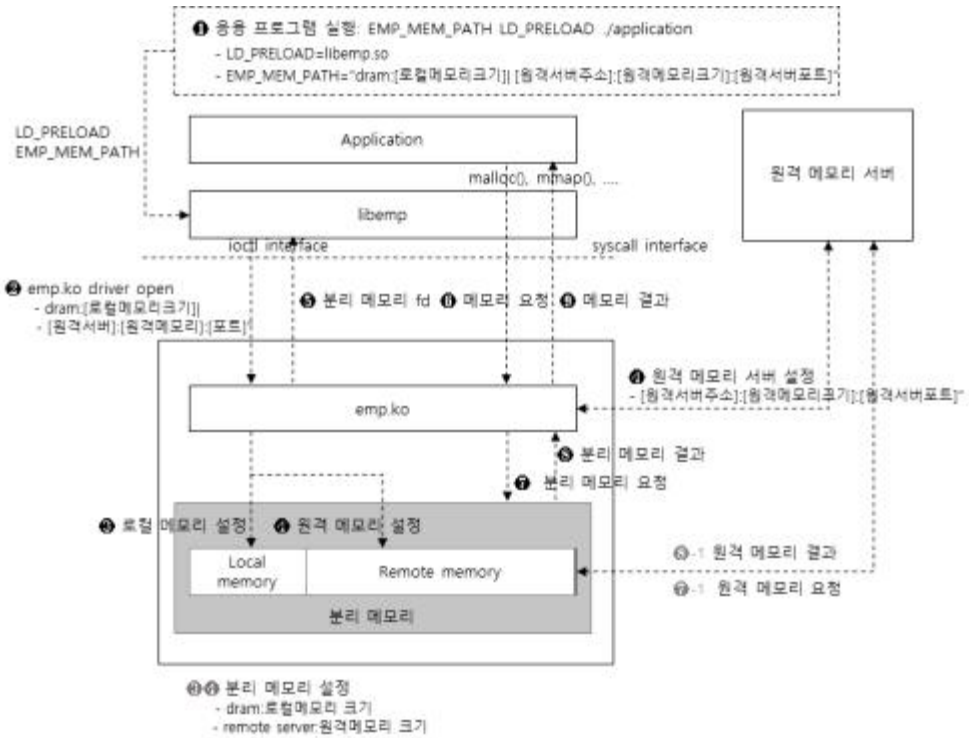


그림 164. 유저 레벨 응용 지원 메모리 라이브러리(libemp) 흐름도

표 19. 네이티브 응용 지원 라이브러리 함수

libemp	
C라이브러리의 메모리 관리 함수	aligned_alloc(), calloc(), free(), malloc(), malloc_usable_size(), memalign(), posix_memalign(), realloc(), valloc()
메모리 시스템 콜	mmap(), mremap(), munmap(), madvise()

· 아래 그림에서 나타난 바와 같이, 대용량 메모리 요구 응용 중 대표적인 메모리 캐시 서버(memcached, redis)를 활용하여 동작 시험 수행 완료. memcached와 redis 응용 프로그램을 실행하여 네이티브 대용량 메모리 지원이 정상 작동하는 것을 확인함

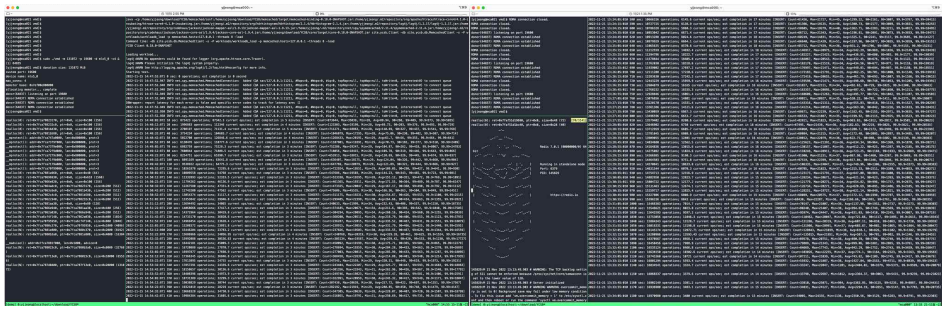
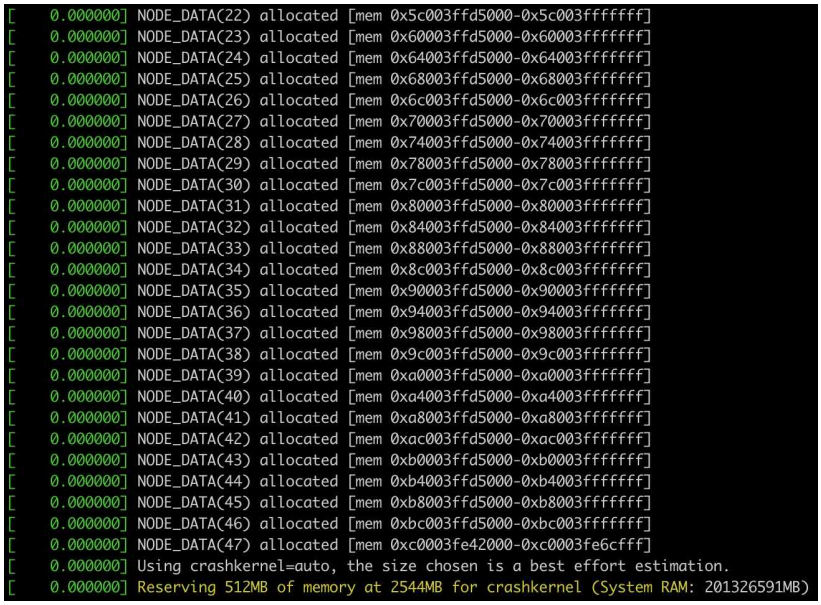
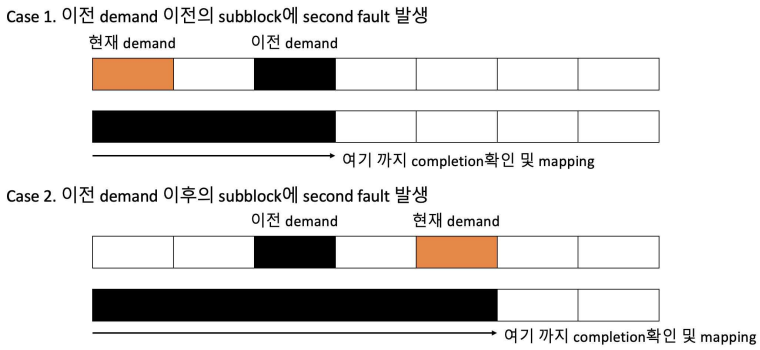
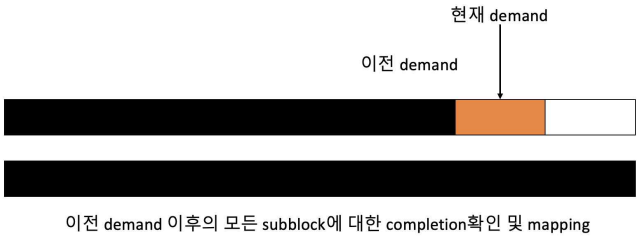
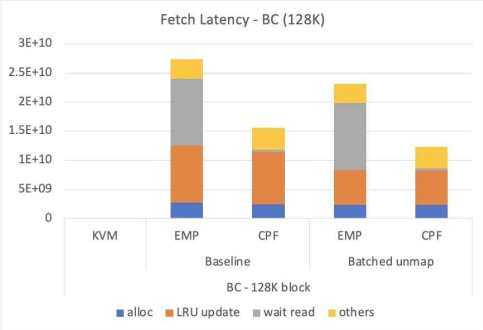
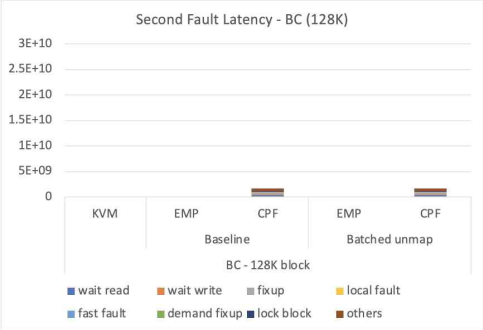
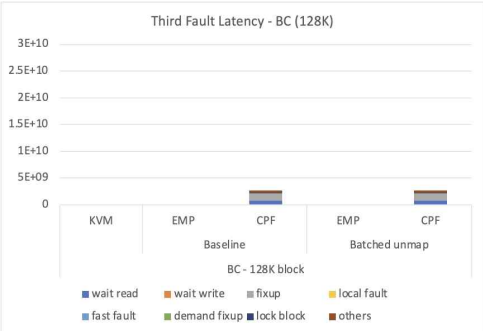
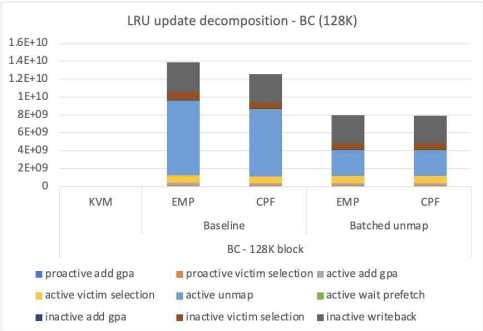


그림 165. EMP 네이티브 응용 지원 기술 실행 화면: memcached(좌), redis(우)

수행기관	개발 목표	개발내용 및 범위
		• [정량 목표] 192TB 규모 패브릭 메모리 시스템 구축을 통한 페타바이트급 메모리 지원 검증 - 192TB 규모 패브릭 메모리 시스템을 가상머신 형태로 구축하기 위해 다음과 같이 하드웨어 및 소프트웨어의 한계를 극복 · 192TB 확장 메모리를 구축하기 위해, 총 32개의 7.68TB U.2 SSD를 이종 성능의 링크를 활용하여 통합함. 가상머신 서버에 지역 (Local) PCIe 링크를 활용하여 24개 SSD 연결, InfiniBand 네트워크로 연결된 다른 서버에 장착한 8개 NVMeOF (NVMe-Over-Fabric) 기반 네트워크 블록 장치 연결 · 4-level paging 구조는 유저 레벨 가상메모리 주소공간이 최대 128TB 크기의 지원 한계를 갖는 반면, 5-level paging은 유저 레벨 가상메모리 주소공간이 64PB까지 지원되므로, 5-level paging을 지원하는 CPU와 리눅스 커널을 사용. 또, QEMU에서 가상머신의 메모리 공간을 mmap할 때 2^47 이상의 값을 address hint로 주도록 하여, 5-level paging을 사용하도록 설정 · 192TB 크기의 가상머신 메모리를 관리하기 위해서 Extended Page Table (EPT)만 약 1TB 정도 이상이 요구되므로, 대용량 커널 데이터를 지원하기 위해 2TB 크기의 호스트 DRAM을 할당 · QEMU에 구현된 Sea Bios는 구조상 8TB~16TB 정도의 메모리만 지원 가능. 192TB의 메모리를 지원할 수 있도록 BIOS 내 메모리 자원 관련 부분의 확장 수행 - 192TB 규모의 패브릭 메모리 시스템 실행 화면  [0.000000] NODE_DATA(22) allocated [mem 0x5c003ffd5000-0x5c003fffffff] [0.000000] NODE_DATA(23) allocated [mem 0x60003ffd5000-0x60003fffffff] [0.000000] NODE_DATA(24) allocated [mem 0x64003ffd5000-0x64003fffffff] [0.000000] NODE_DATA(25) allocated [mem 0x68003ffd5000-0x68003fffffff] [0.000000] NODE_DATA(26) allocated [mem 0x6c003ffd5000-0x6c003fffffff] [0.000000] NODE_DATA(27) allocated [mem 0x70003ffd5000-0x70003fffffff] [0.000000] NODE_DATA(28) allocated [mem 0x74003ffd5000-0x74003fffffff] [0.000000] NODE_DATA(29) allocated [mem 0x78003ffd5000-0x78003fffffff] [0.000000] NODE_DATA(30) allocated [mem 0x7c003ffd5000-0x7c003fffffff] [0.000000] NODE_DATA(31) allocated [mem 0x80003ffd5000-0x80003fffffff] [0.000000] NODE_DATA(32) allocated [mem 0x84003ffd5000-0x84003fffffff] [0.000000] NODE_DATA(33) allocated [mem 0x88003ffd5000-0x88003fffffff] [0.000000] NODE_DATA(34) allocated [mem 0x8c003ffd5000-0x8c003fffffff] [0.000000] NODE_DATA(35) allocated [mem 0x90003ffd5000-0x90003fffffff] [0.000000] NODE_DATA(36) allocated [mem 0x94003ffd5000-0x94003fffffff] [0.000000] NODE_DATA(37) allocated [mem 0x98003ffd5000-0x98003fffffff] [0.000000] NODE_DATA(38) allocated [mem 0x9c003ffd5000-0x9c003fffffff] [0.000000] NODE_DATA(39) allocated [mem 0xa0003ffd5000-0xa0003fffffff] [0.000000] NODE_DATA(40) allocated [mem 0xa4003ffd5000-0xa4003fffffff] [0.000000] NODE_DATA(41) allocated [mem 0xa8003ffd5000-0xa8003fffffff] [0.000000] NODE_DATA(42) allocated [mem 0xac003ffd5000-0xac003fffffff] [0.000000] NODE_DATA(43) allocated [mem 0xb0003ffd5000-0xb0003fffffff] [0.000000] NODE_DATA(44) allocated [mem 0xb4003ffd5000-0xb4003fffffff] [0.000000] NODE_DATA(45) allocated [mem 0xb8003ffd5000-0xb8003fffffff] [0.000000] NODE_DATA(46) allocated [mem 0xbc003ffd5000-0xbc003fffffff] [0.000000] NODE_DATA(47) allocated [mem 0xc0003fe42000-0xc0003fe6cfff] [0.000000] Using crashkernel=auto, the size chosen is a best effort estimation. [0.000000] Reserving 512MB of memory at 2544MB for crashkernel (System RAM: 201326591MB) 그림 166. 192TB 패브릭 메모리 시스템 실행 화면 · 위 그림의 가상 머신 내 동작하는 Guest 운영체제 커널의 부트 메시지는 총 48개의 NUMA 노드가 설정되어 있고 시스템 메모리가 201326591MB로, 192TB인 것을 확인할 수 있음 **• 요구 페이지 우선 처리 기술 고도화를 통한 원격 메모리 접근 지연시간 단축** - 원격 메모리 데이터 전송과 응용의 동시 수행을 위한 3단계 폴트 처리 기법 개발 · 요구 페이지 우선 처리 기술 (Critical Page First, CPF)의 경우 요구되는 페이지를 먼저 처리하고 해당 블록의 다른 페이지들은 블록에 대한 다음 폴트 수행 시에 모두 처리되는 방식을 취하고 있음. 하지만, 이 두 번째 폴트가 바로 수행이 된다면, 폴트 수행 시 블록에 해당하는 모든 페이지들을 기다려야 하는 오버헤드가 발생하게 됨 · 이를 해결하기 위해서 지금까지 가지고 온 페이지만큼만 기다려서 처리한 후, 세 번째 폴트 처리 시에 나머지 페이지들을 기다려 처리하는 방식을 사용하여 두 번째 폴트 처리 시 불필요하게 기다리는 시간을 최소화하는 방법을 구현 · 해당 구현은 RDMA를 통해서 데이터를 전송하는 경우 (RC: Reliable Connection) 데이터

수행기관	개발 목표	개발내용 및 범위
		의 순서 (ordering)가 지켜진다는 속성을 이용함 · 그림 167은 처음 CPF를 통해서 요구 페이지만 처리된 블록에 대해서 두 번째 폴트가 발생하는 두 가지 경우를 나타냄. 첫 번째 케이스는 이전 폴트의 요구 페이지가 현재 폴트의 요구 페이지보다 뒤쪽에 있을 경우에 대한 예임. 이때는 이전 폴트의 요구 페이지만큼 가지고 오면 현재 폴트의 요구 페이지도 가지고 오는 것이 완료되었기 때문에 이전 요구 페이지만큼 completion을 확인하고 페이지를 매핑하여 폴트를 처리함 · 두 번째 케이스는 이전 폴트의 요구 페이지보다 현재 폴트의 요구 페이지가 더 뒤에 있는 경우를 나타냄. 이때는 현재 폴트의 요구페이지만큼 가지고 온다면 이전 요구페이지와 함께 처리가 가능함  그림 167. 두 번째 폴트 시에 처리되는 페이지  그림 168. 세 번째 폴트 (마지막 폴트) 시에 처리되는 페이지 · 두 번째 폴트 이후로 해당 블록에 대해서 폴트가 나타나게 되면 그림 168과 같이 해당 블록의 모든 페이지들을 기다리고 매핑시킴으로써 블록의 처리를 끝냄. 만약 첫 번째 나 두 번째 폴트 이후에 다음 폴트가 발생하지 않고 로컬 메모리에서 매핑이 끊어지게 된다면 이때 해당 블록에 대한 페치를 모두 기다린 후에 매핑을 끊도록 함 · 위 고도화 기법을 통해서 페이지 폴트의 횟수는 증가할 수 있지만, 불필요하게 페치를 기다리는 시간을 최소화하는 장점을 가짐 - 페이지 마커 기반 RDMA 읽기 폴링을 활용한 적극적인 원격 메모리 접근시간 단축 수행 · CFP환경에서 페이지 별로 마커를 할당하고 이 마커를 통해서 해당 페이지가 페치되었다는 것을 확인할 수 있음. 이러한 메커니즘을 확장하여 모든 서브-블록에 대한 RDMA completion을 이 마커를 통해서 확인하도록 하여 RDMA 시그널을 제거 · 가지고 오는 블록의 페이지에 대해서 마커까지 가지고 왔는지 계속해서 확인이 필요하기 때문에 폴링은 하게 되지만, send queue에 대한 drain 요청 이외에는 RDMA 읽기를 위한 시그널은 사라짐 - 원격 메모리 접근 시간 단축 기법의 성능 평가 · 기술한 고도화 기법을 포함하여 CPF가 성능에 얼마나 영향을 끼치는 지에 대한 성능 평가를 수행. 해당 성능 평가는 EMP와 CPF의 경우 로컬 메모리의 크기를 전체 메모리 사용량의 30%만 가지도록 설정하였고, 128KB 블록과 32KB 서브-블록 환경에서 진행함. 이러한 설정으로 STREAM, BC, PageRank와 같은 워크로드를 사용하여 성능을 평가함

수행기관	개발 목표	개발내용 및 범위
		· 아래의 세 개의 그래프는 BC 워크로드에 대한 페치 지연시간과 두 번째, 세 번째 폴트에 대한 지연시간을 나타내는 그래프임. 그림 169의 그래프를 보면 EMP에 비해서 CPF의 wait read시간 즉, RDMA 페치명령 후에 페치를 기다리는 시간이 급격하게 줄어드는 것을 확인할 수 있음. 이는 기존 EMP에서 블록 전체를 읽는 데 기다리는 시간이 critical path에 속하게 되는데, CPF에서는 요구 페이지에 대해서만 기다리기 때문에 이 시간이 급격하게 줄어듦 · 하지만 CPF에서는 EMP에 비해서 폴트의 횟수가 늘어남. 두 번째 폴트와 세 번째 폴트를 통해서 해당 블록에 대한 페치를 모두 기다리며, 블록에 대한 매핑을 완수 할 수 있음  그림 169. 첫 번째 폴트 페치 지연시간 분석  그림 170. 두 번째 폴트 지연시간 분석  그림 171. 세 번째 폴트 지연시간 분석  그림 172. LRU update 지연시간 분석 · 그림 170, 그림 171와 같이 두 번째, 세 번째 폴트에서는 많은 지연시간이 걸리지는 않지만 특히 세 번째 폴트 시에 블록의 나머지 페이지들의 페치를 완전히 기다려야 하기 때문에 wait read의 지연시간이 많이 나타남 · 그림 169을 보면 CPF를 통해서 페치를 기다리는 시간은 거의 다 줄었지만 이로 인해서 LRU update로 인한 지연시간이 많은 부분을 차지함을 확인할 수 있음. 따라서, 본 연구에서는 LRU update에서 어떠한 부분이 지연시간에 영향을 많이 끼치는지 확인을 위해서 LRU update의 지연시간을 분석함 · 그림 172은 BC 워크로드를 수행하면서 발생하는 LRU update의 지연시간을 분석한 그래프임. 해당 그래프에서 Baseline쪽으로 보면 active unmap의 부분이 매우 큰 것을 볼 수 있음. 이는 EMP 내부 LRU list를 관리할 때, active list에 있는 데이터가 inactive로 넘어가는 경우 unmap을 수행하여 이때의 지연시간이 큰 영향을 줌 · 해당 unmap에 대한 최적화는 아래 페이지 언맵을 한꺼번에 수행하는 최적화 방법을 통해서 해소함. 이는 다음 고도화 기법에서 설명하도록 함

수행기관	개발 목표	개발내용 및 범위

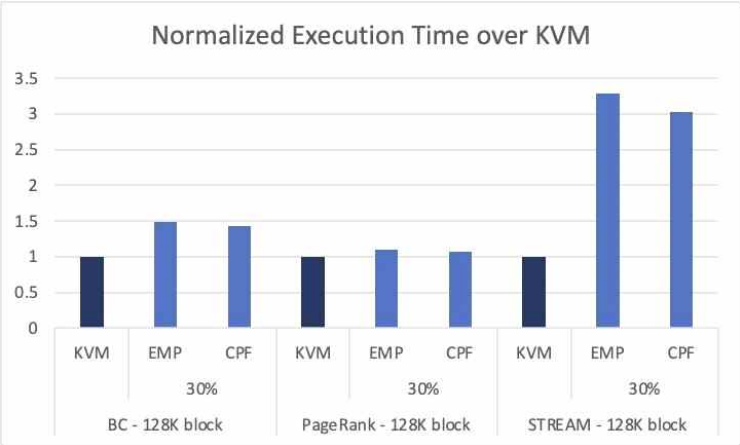


그림 173. CPF의 성능 평가

· 그림 173는 앞서 설명한 CPF 기법을 바탕으로 워크로드의 성능을 KVM, EMP와 비교한 결과임. KVM의 실행시간에 정규화한 결과로 바가 낮을수록 성능이 좋음. BC, PageRank, STREAM 워크로드에서 각각 3.8%, 2.4%, 8.1%의 성능 향상을 나타냄을 확인. 이는 메모리 폴링과 페이지 언맵 고도화 기법을 통해서 EMP에서의 성능의 향상을 반영한 결과

· CPF를 통해서 불필요한 페치 기다리는 시간을 다음 폴트 쪽으로 상쇄시켜서 성능의 이점을 얻을 수 있음을 확인함. 하지만 CPF로 인해서 폴트의 개수가 증가하기 때문에 많은 성능의 향상을 만들지는 못함

- 주소 공간의 블록 기반 언맵 (batched unmap)을 활용한 원격 메모리 접근 최적화 수행

· EMP는 KVM가상화 환경에서 동작하는데, 기존 KVM환경에서는 unmap이 많이 발생하지는 않기 때문에 unmap을 수행할 때 4KB 페이지 단위로 하나하나 reverse map을 찾아다녀야 하는 번거로움이 있음. 하지만, EMP환경과 접목이 되었을 때는 보통 서브-블럭과 같이 연속적인 공간을 할당하기 때문에 많은 페이지들을 한번에 unmap하기 쉬운 구조로 되어 있음

· 따라서, 본 연구에서는 EMP구조에 맞도록 unmap을 4KB페이지 단위가 아닌 블록이나 서브-블록 혹은 그 이상으로 한꺼번에 수행할 수 있도록 하여 reverse map을 횡단(traverse)하는 오버헤드를 줄이도록 구현함

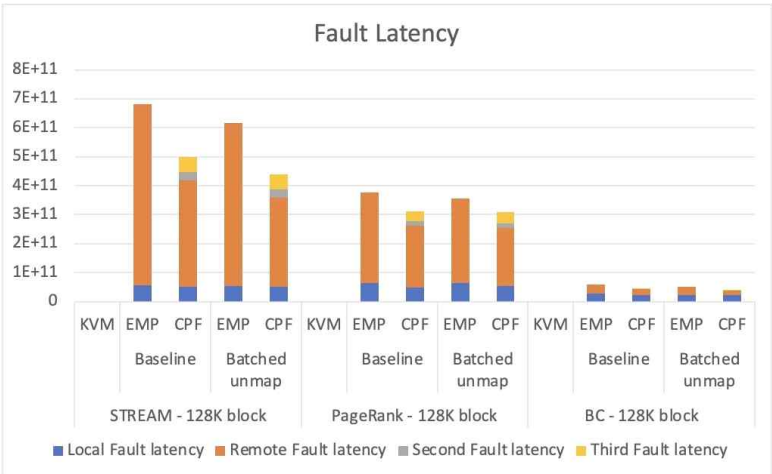


그림 174. 한 번에 언맵 수행 기법의 성능 평가

· 해당 고도화 방법을 통해서 그림 174의 Batched unmap부분과 같이 EMP와 CPF에서 언맵에 소요되는 시간을 절반 이상으로 줄여서 전체적인 성능을 높이는 결과는 도출함

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>

· 이 고도화 기법을 통해서 LRU update로 인해서 발생하는 전체 지연시간을 줄일 수 있었으며, 페치를 기다리는 시간과 오버랩 되어있던 부분의 지연시간을 줄여 그림 174과 같이 전체적인 페치 지연시간을 줄이는 데 큰 영향을 줌

• EMP기술의 Couchbase 적용 실험 환경 및 실험 결과

- EMP 분리 메모리 시스템을 여러 가지 응용에 도입 및 적용하는 시험을 수행함. 본 실험에서는 Couchbase 데이터베이스를 EMP환경에서 수행시키고 실험 결과를 도출함
- 해당 실험은 지난해 기술이전을 통해 이전한 업체에서 테스팅 결과를 요구함에 따라 수행하였으며, 업체와의 계속적인 협업과 지원을 제공함
- 본 과제의 기술 (산출물)에 대한 평가를 수행하였으며, 해당 결과를 바탕으로 실제 대용량 메모리를 필요로 하는 응용의 수행에 도움을 준다는 것을 확인할 수 있음
- 아래 표는 Couchbase를 이용하여 EMP의 성능을 파악하기 위한 호스트 시스템과 가상머신의 설정을 나타낸 표이며, EMP의 환경설정 옵션을 나타냄

표 20. Couchbase를 이용한 EMP 실험 호스트 환경설정

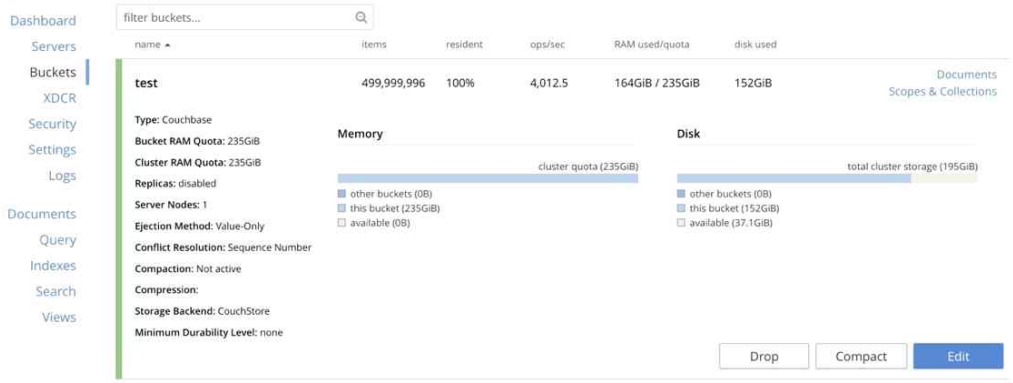
Host	
Processor	Intel(R) Core□ I9-9820X 3.30GHz (10 Core 20 Threads)
Memory	128 GB
NVMe	[OS용] Samsung SSD 970 PRO 512GB [메모리 확장용] Intel SSDPED1K750GA (Optane SSD DC P4800X)
OS/Kernel	CentOS 8 / 4.18.0-240.10.1.el8.x86_64.emp

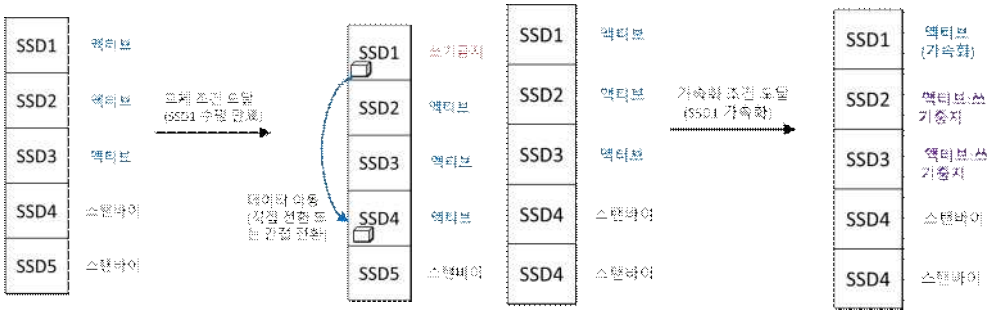
표 21. Couchbase를 이용한 EMP 실험 가상머신 환경설정

Virtual Machine	
Processor	Intel(R) Core□ I9-9820X 3.30GHz (vCPU 20개)
Memory	400 GB (DRAM cache : 24 GB, Extended to NVMe: 400 GB)
Storage	qcow2 이미지의 가상디스크 방식으로 구성. 200GB로 설정
EMP option	initial_block_size: 4 KB initial_subblock_size : 4 KB initial_els_disabled : 0 initial_ignore_zero_page : 0 initial_mark_empty_page : 0 initial_critical_subblock_first : 0 initial_critical_page_first : 0 initial_enable_transition_csf : 0 initial_async_invlept : 0 initial_eager_writeback : 0 initial_emulate_insn : 0
OS/Kernel	CentOS 8 / 4.18.0-240.10.1.el8_3.x86_64

· Couchbase 서버에 데이터를 삽입하기 위해서 4개의 클라이언트 프로세스가 병렬적으로 아이템을 삽입하는 방식을 사용하여 성능을 평가함. 총 500 million개의 아이템을 4개의 프로세스가 125 million개씩 삽입을 수행

</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		 그림 175. Couchbase 실험의 서버 프로세스 결과 · EMP 사용시에도 Couchbase 서버에서도 500 million개의 아이템에 대한 삽입이 정상적으로 수행되었음을 확인할 수 있음. 총 시간은 5시간 33분이 걸렸고, 시간당 약 25,000개의 작업이 수행되었음을 확인 가능함 · 총 164GB의 메모리가 사용되었으며, 실제 호스트에서 사용되는 메모리는 24GB만이 사용되었고, 총 응용의 메모리 사용량 대비 15%의 메모리만으로도 응용이 수행될 수 있음을 확인하였음
	② 혼성·다계층 메모리 중심 컴퓨팅 시스템 데이터 배치 기술 연구	**• 저 내구성 (low endurance) 메모리 활용을 위한 2-계층 메모리 관리 기술 연구** - 기술 개발의 필요성 · EMP는 서버 내의 SSD (Solid State Drive) 등 고성능 저장매체 혹은 원격 서버의 메모리 등을 메모리 확장에 사용하는 분리 메모리 기법을 제공. 특히 저내구성 (low endurance)을 특징으로 갖는 SSD를 활용하는 경우, 용량 당 가격이 저렴하여 분리 메모리 시스템 구성에 유리하나, 쓰기 횟수 제한이 있기 때문에 많은 쓰기를 유발하는 분리 메모리 시스템에 불리함 · 따라서, 본 연구에서는 SSD 등 쓰기 횟수에 제한이 있는 매체를 활용하는 2-계층 메모리 시스템을 구성할 때, 쓰기 횟수 제한에 따른 오류를 막아주고, 쓰기 횟수 제한에 도달했을 때 해당 SSD를 교체하는 동안 추가적인 오류가 일어나는 것을 방지해야 함 - 기술 개발 내용 · 본 연구에 따른 2-계층 메모리 관리 기술은 데이터 분류부, SSD설정부, 수명 만료 검출부, 전환 제어부, 쓰기 가속화부로 구성 될 수 있음. 데이터 분류부는 성능 민감 데이터를 DRAM에 할당하고 일단 데이터를 SSD에 할당하는 기존의 EMP 구조에 따름 · SSD 설정부는 복수의 SSD 중 하나 이상의 SSD를 액티브 SSD로 설정하고, 다른 하나 이상의 SSD를 스탠바이 SSD로 설정할 수 있음. 복수의 SSD 중 어느 SSD를 액티브 SSD로 설정할 것인지는 SSD의 일련번호 순서, 제조일 순서, 최초 사용 개시일 순서, 임의의 설정 순서 등에 의해 결정될 수 있음 · 수명 만료 검출부는 액티브 SSD의 수명이 만료하였는지 검출할 수 있음. 수명 만료 여부는 액티브 SSD의 쓰기 동작이 수행된 쓰기 횟수를 검출하여 TBW (Total Bytes Written)와 비교하거나, 액티브 SSD에 대한 주기적인 쓰기 및 읽기에 의해 액티브 SSD가 정상적으로 동작하는지 검출 하는 방법 등에 의해 결정 가능함 · 수명이 완료되는 경우 액티브 SSD의 쓰기를 금지하고, 스탠바이 SSD 중 하나를 액티브 SSD로 설정하여 원격 메모리로서 사용할 수 있음 · 전환 제어부는 수명 만료된 액티브 SSD에 저장되어 있는 데이터를 스탠바이 또는 액티브 SSD로 이동 또는 복사할 수 있음. 이때, 데이터의 이동 또는 복사는 직접 전환 방식 또는 간접 전환 방식에 의해서 수행될 수 있음 · 직접 전환 방식은 전환 제어부 직접 액티브 SSD의 데이터를 스탠바이 SSD로 이동함. 시스템의 작업 부하가 낮을 때 수행 · 간접 전환 방식은 수명 만료된 액티브 SSD에 대한 새로운 쓰기 동작이 시도될 때 액티브

수행기관	개발 목표	개발내용 및 범위
		SSD 대신 스탠바이 SSD로 해당 데이터를 쓰는 방식  그림 176. 수명 만료된 액티브 SSD 처리의 예 그림 177. 액티브 SSD의 쓰기가속화 기법 · 다른 액티브 SSD로의 유효 데이터의 저장이 완료되면 전환 제어부는 쓰기 금지된 SSD의 교체가 필요하다는 알림 메시지를 생성할 수 있고, 관리자는 SSD의 수명 만료를 인식하고 새로운 SSD로 교체 가능함. 이를 통해 스탠바이 SSD의 최소 개수 유지 가능 · 쓰기 가속화부는 액티브 SSD가 가속화 조건을 만족하는지 판정하고, 가속화 조건을 만족하는 경우 조건을 만족하는 하나의 액티브 SSD를 가속화 액티브 SSD로 설정하여 가속화 액티브 SSD에 쓰기 동작을 집중시키고, 다른 액티브 SSD로의 쓰기 동작을 중지. 이를 통해 복수의 액티브 SSD의 수명 만료일이 비슷한 기간에 발생하는 것을 막아서, SSD 교체를 수월하게 할 수 있도록 함 · 가속화 조건은 복수의 액티브 SSD 중 어느 하나의 액티브 SSD의 쓰기 횟수가 미리 설정된 임계값을 초과하였을 때와, 복수의 액티브 SSD 중 수명 만료 예정일이 가장 빨리 도래하는 두 개의 SSD의 수명 만료 예정일의 차이가 미리 설정된 임계값 이하인 경우임 · 위 그림과 같이 SSD1이 가속화 조건에 도달하는 경우 다른 복수의 액티브 SSD의 쓰기를 중지시키고 SSD1으로만 쓰기를 몰아주어서 빠르게 하나의 SSD에 수명을 소진하여 빠르게 스탠바이 SSD로 대체시킬 수 있도록 함 **• 혼성·다계층 메모리 데이터 배치를 가능하게 하는 분리 메모리 가상머신 이주 기능 개발** - 기술 개발의 필요성 · 분리 메모리 시스템 상에서 동작하는 가상머신들은 물리머신 내의 다른 가상머신과 로컬 메모리 사용에 있어 간섭 현상을 일으킬 수 있음. 이를 해결하기 위해 가상머신 이주 기능이 필수적 · 분리 메모리 시스템은 많은 양의 메모리가 물리머신 바깥에 있는 상황을 가정하므로, 기존의 가상머신 이주보다 훨씬 적은 양의 데이터만을 옮겨서 가상머신 이주 실현 가능. 따라서 클러스터 단위의 메모리 관리가 더 용이할 수 있음 · 본 연구에서는 EMP 플랫폼 기반 분리 메모리를 사용하는 가상머신에 대한 활성 이주(live migration) 기능을 개발함 - 기술 개발 내용 · 가상머신 활성 이주 과정: 1) 전체 메모리를 이주 대상 머신으로 전송, 2) 전송 도중 수정된 메모리에 대한 재복사 3) 과정 2에서 재복사 되는 양이 일정 수준 이하가 될 때까지 과정 2 반복, 3) 가상머신 중지, 4) 전송이 필요한 메모리 및 기타 컨텍스트 복사 · EMP 기반 분리메모리에서는 로컬 메모리의 내용에 대해서만 전송함으로써 대용량 메모리 가상머신에 대한 이주 시간을 줄임 · EMP가 관리하는 페이지의 LRU 순서가 바뀌나, 이는 이주 후에 가상머신이 다시 실행되면서 LRU 순서가 다시 정립됨

수행기관	개발 목표	개발내용 및 범위

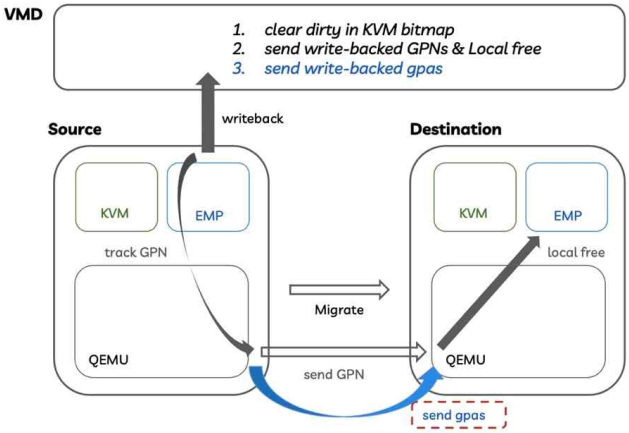


그림 178. 분리 메모리 가상머신 이주시 write-back 처리

· 가상머신의 이주 과정 도중 원격 메모리로 write-back이 된 경우, 이후 과정에서 재전송할 필요가 없음. 따라서 write-back이 일어난 페이지에 대해서는 수정된 메모리 대상에서 빼고, 이주 대상 머신에 관련 정보를 전송하는 등의 기능을 구현

· 분리 메모리 관리를 위한 GPA descriptor에 대해서도 pre-copy 기술을 적용하고, 수정된 GPA descriptor에 대해서만 재전송을 반복하는 과정을 거쳐서 가상머신 이주 도중의 가상머신 중지 시간을 최소화

· EMP의 성능 최적화 기술 중 하나인 Elastic Block Size에 대해서는 가상머신 이주 기능과 충돌하는 부분이 많아서 가상머신 이주 시작 전에 Elastic Block Size 기능을 끄고, 가상머신 이주가 완료된 후 다시 켜는 것으로 함

• 혼성 메모리 데이터 배치 연구를 위한 커널 수준 메모리 성능 에뮬레이터 연구

- 기술 개발의 필요성

· CXL 등 메모리 인터커텍트 기술이 개발되면서 다양한 성능의 메모리 미디어가 등장할 것으로 예측됨. 이에 따라 에뮬레이터를 통해 다양한 메모리 미디어에서의 응용 성능을 예측하고 응용 개선 연구를 수행할 필요성이 대두됨

· 기존의 응용 수준 에뮬레이터인 Quartz 등이 개발됐지만, 응용의 쓰레드 수가 코어 개수보다 많은 경우에는 에뮬레이션이 불가능하며, 응용의 signal 사용에 제한이 있는 등의 문제가 있었음

· 본 연구에서는 커널 수준의 메모리 성능 에뮬레이터를 개발하여 응용 수준 에뮬레이터의 제약을 없앴고, 실제 비휘발성 메모리인 DCPMM으로 정확성을 검증함

- 기술 개발 내용

· 메모리 성능 에뮬레이션을 위해, 먼저 응용의 실행 시간에서 메모리 접근에 걸리는 시간을 분리하는 기술이 필요. Performance Counter의 값을 사용하는 기존 응용 수준 에뮬레이션과 비슷한 방식을 차용함. 단, 커널 수준으로 개발함으로써 각 쓰레드의 Performance Counter 값이 가상화되도록 함

$$WeightedLat = L2latency \times \#L2hit + L3latency \times \#L3hit + LocalDRAMlatency \times \#LocalDRAMaccess + RemoteDRAMlatency \times \#RemoteDRAMaccess$$

$$LocalDRAMstall = L1Dstall_cycle \times LocalDRAMlatency \times \#LocalDRAMaccess / WeightedLat$$

$$RemoteDRAMstall = L1Dstall_cycle \times RemoteDRAMlatency \times \#RemoteDRAMaccess / WeightedLat$$

· 각 메모리에 접근된 시간을 바탕으로 응용에 삽입해야 할 추가 딜레이는 다음과 같이 계산

$$AddDelay = DRAMstall \times NewLatency / DRAMlatency - DRAMstall$$

· 아래 그림은 커널 수준 메모리 성능 에뮬레이터의 딜레이 삽입 과정을 나타냄. 리눅스의 preempt notifier와 exit profiling 기능을 활용하여 각 스레드가 스케줄링에 들어오는 순간과 나가는 순간, 스레드가 끝나는 순간을 감지할 수 있음. 해당 이벤트가 감지될 때 추가 지연 시간을 계산하고 스레드를 실제로 지연시킴

수행기관	개발 목표	개발내용 및 범위

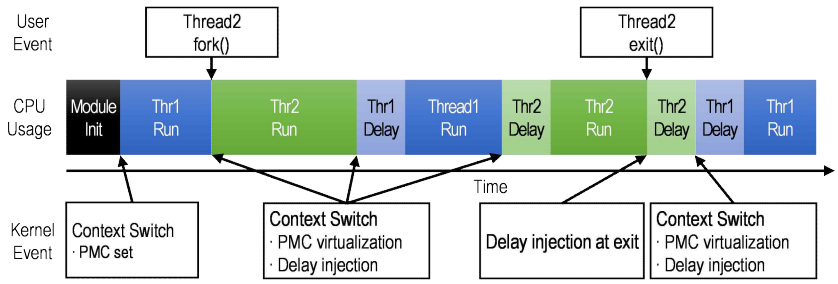


그림 179. 커널 수준 차세대 메모리 에뮬레이터 동작

- 인텔 Memory Latency Checker 기반 에뮬레이터 정확성 검증
  · 인텔 5215M CPU 2개와 384GB의 DRAM, 1.5TB의 DCPMM이 장착된 환경에서 커널 수준 메모리 성능 에뮬레이터의 정확성을 검증함

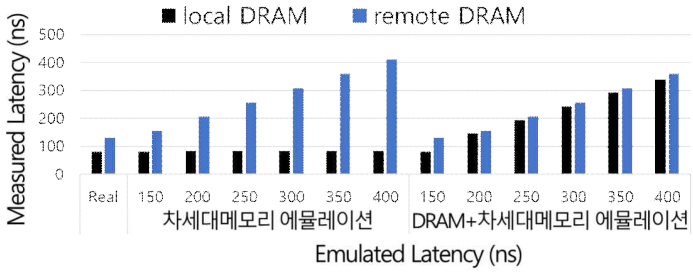


그림 180. MLC를 이용한 에뮬레이터 정확성 검증

  · 위 그림을 인텔 Memory Latency Checker를 이용하여, 개발한 에뮬레이터가 지정한 메모리 성능이 그대로 측정되는 가를 검증. 에뮬레이션하는 메모리 접근 지연시간을 150ns에서 400ns까지 변화시켰을 때, 설정값 대비 평균 오류는 3%, 최대 오류는 4%였음

- SPECCPU2017과 DCPMM을 이용한 에뮬레이터 정확성 검증
  · 비휘발성 메모리인 DCPMM에서 SPECCPU2017을 실행하여 성능을 측정한 뒤, 커널 성능 에뮬레이터가 DRAM만으로 DCPMM의 성능을 정확하게 에뮬레이션할 수 있는 지를 검증

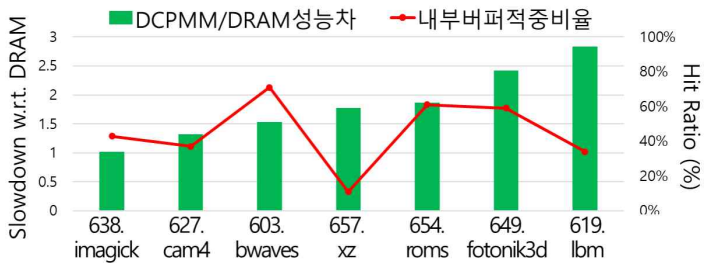


그림 181. 정확도 검증에 사용한 벤치마크 특성

  · 위 그림은 SPECCPU2017의 특성을 나타냄. DRAM 실행 대비 DCPMM 실행의 성능이 다양한 벤치마크를 선정함
  · DCPMM은 내부에 버퍼가 있어서 해당 버퍼의 적중률이 높으면 DCPMM을 접근할 때의 평균 지연 시간이 짧아지고, 적중률이 낮으면 평균 지연 시간이 길어짐. 개발한 에뮬레이터는 내부 버퍼까지 모사하지는 않음. 그래서 각 응용의 DCPMM 내부 버퍼 적중률을 pmwatch 프로그램을 통해 측정하고 각 응용의 평균 DCPMM 접근 시간을 계산하여 이를 바탕으로 에뮬레이터의 정확도 검증

수행기관	개발 목표	개발내용 및 범위

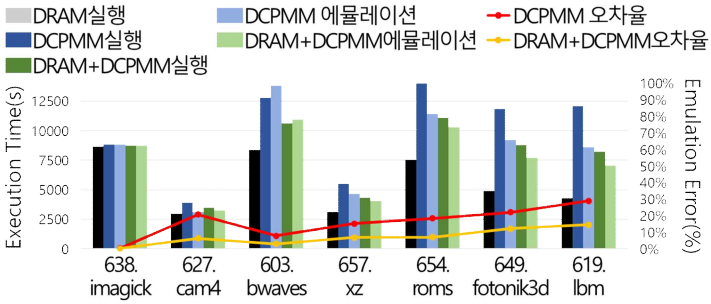


그림 182. SPECCPU2017을 이용한 에뮬레이터 정확성 검증

· 실험 결과 평균 11.6%의 오차율을 보임. 매크로 응용에 대해서도 10% 내외의 오류를 보이는 것이므로 차후 다양한 메모리에 대한 응용 및 시스템 소프트웨어 연구에 활용이 가능할 것으로 보임

• 분리 메모리 시스템에서의 보안 문제를 위한 무결성 검증 기술 연구

- 기술 개발의 필요성

· 분리 메모리 시스템에서 로컬 노드는 메모리 데이터가 믿을 수 있는 반면, 분리 메모리의 다수의 노드에 대한 서비스를 수행시 원격 노드는 오류 문제 등으로 인해 보안 문제가 존재함. 이는 네트워크를 통하여 메모리 데이터가 전송됨에 따라 이에 물리적인 장애로 인한 데이터 손실, 유실, 변조 등에 기인함

· 문제가 발생하면 응용 오류 또는 데이터에 대한 신뢰성의 문제가 발생하므로 이 문제의 해결을 위하여 원격 메모리로부터 전송되는 데이터에 대한 무결성 검증이 요구됨

- 메모리 분리 시스템에서 데이터의 무결성 검증 기술

· 본 연구에서는 무결성 검증을 지원하는 분리 메모리 관리자 구조를 제안함. 분리 메모리 관리자는 무결성 검증부, 지연시간 검출부, 프리페치 수행부 및 스케줄링부로 구성됨. 이 분리 메모리 관리자는 원격 메모리에 접근하는 NIC (Network Interface Card)와 통합 혹은 연결된 가속기 형태로 설계

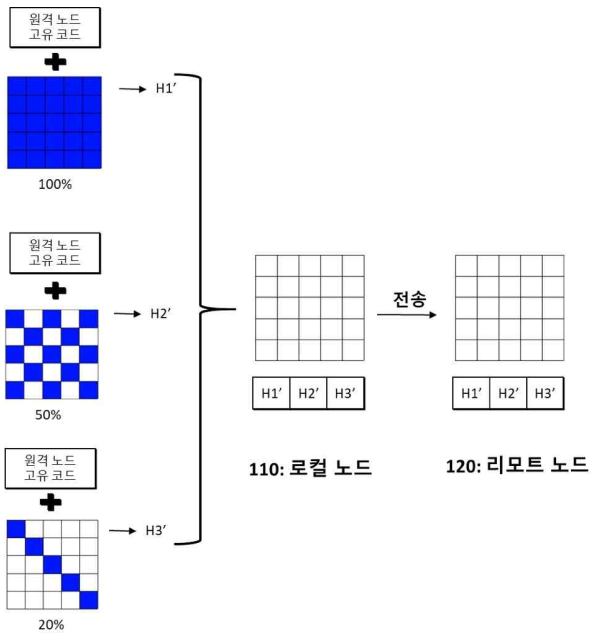
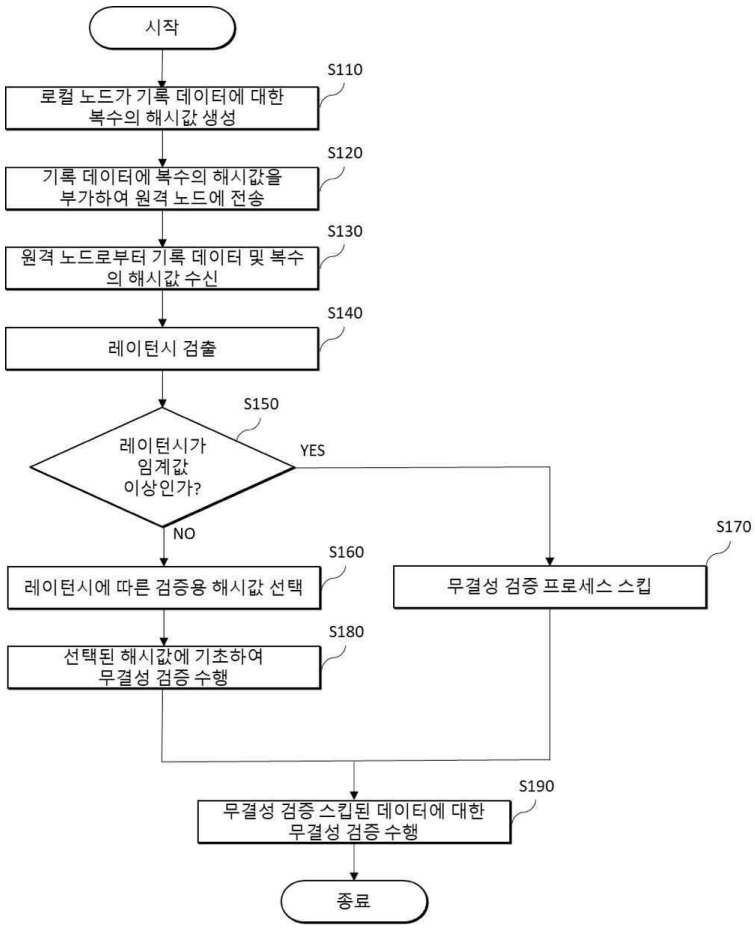
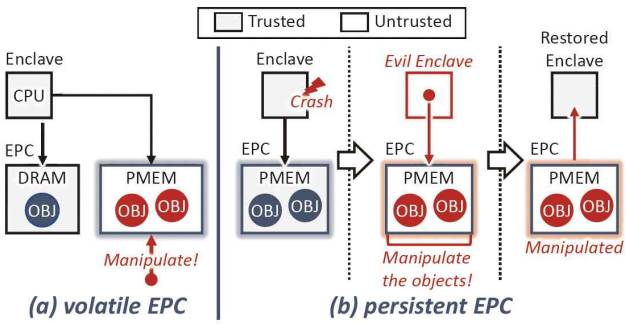


그림 183. 데이터의 해시값 생성 및 전송 프로세스

· 무결성 검증부는 원격 메모리에 저장할 기록 데이터를 복수개의 세그먼트로 분할하고, 분

수행기관	개발 목표	개발내용 및 범위
		할된 복수개의 세그먼트 중 적어도 하나의 세그먼트를 포함하는 세그먼트 세트에 대한 복수의 해시값을 생성 및 전송하고, 복수의 해시값 중 적어하도 하나의 해시값에 기초하여 데이터의 무결성을 검증할 수 있음 · 해시값은 데이터를 입력으로 하는 MD5, SHA (SHA-256, SHA-512) 등의 해시 함수에 의해서 생성될 수 있음. 기록 데이터와 원격 노드의 고유 코드 (IP 주소, 머신 ID 등)를 반영해 해시값 생성. 원격 노드의 고유 코드를 바탕으로 원격 노드의 출처에 대한 검증도 가능 · 무결성 검증부는 데이터를 복수개의 세그먼트로 분할하고, 데이터의 10%, 20%, 50%, 70%, 100%의 비율을 차지하는 세그먼트를 포함하는 세트들로 설정. 또한, 해당 세그먼트 세트 각각에 대한 해시값을 생성하고 원격 메모리로 전송하여 저장 · 지연시간 검출부는 원격 메모리에 대한 접근 지연시간을 나타내는 지연시간을 측정. 지연시간에 따라서 데이터 비율에 따라 복수개로 만들어 둔 해시값 중 하나가 선택될 수 있음 · 프리페치 수행부는 원격 메모리에 저장된 데이터 중 접근이 예상되는 데이터를 미리 저장해두는 프리페치를 수행. 데이터 무결성 검증을 수행하는 경우 처리 속도가 느려질 수 있으므로, 이를 사전에 방지하기 위해 프리페치 수행 · 스케줄링부는 시스템에 의해 요구되는 요구 데이터 (demand data)와 프리페치 데이터를 구분하고, 요구 데이터에 대한 무결성 검증에 우선권을 부여  그림 184. 무결성 검증 프로세스의 흐름도 · 위 그림은 본 연구에서 제안하는 무결성 검증 프로세스를 나타냄. 데이터를 원격 메모리에 기록하는 경우, 로컬노드는 데이터에 대한 복수의 해시값을 생성하고, 데이터 및 복수의 해시값을 원격 노드에 전송.또한, 원격 노드에 저장된 데이터에 대한 읽기 요청이 있는 경우 원격 노드는 읽기 요청과 관련된 데이터 및 복수의 해시값을 로컬 노드에 전송 · 로컬 노드는 원격 노드에 대한 지연시간을 측정하고, 지연시간이 임계값 이상인 경우 무결성 검증 프로세스를 생략하고, 임계값 이하인 경우 지연시간에 따라 검증용 해시값을 선택

수행기관	개발 목표	개발내용 및 범위
		하여 무결성 검증 프로세스 수행. 여기서 임계값은 네트워크 상태 또는 원격 노드의 장치 성능에 의해서 결정 될 수 있음 **• 비휘발성 메모리로의 메모리 확장시 보안 문제를 해결하기 위한 데이터 보호 프레임워크** - 비휘발성 메모리에서의 데이터 보호 프레임워크의 필요성 · 비휘발성 메모리를 사용하는 환경에서는 시스템 파워가 없는 경우에서도 데이터가 저장되어있기 때문에, 휘발성 메모리만을 사용하는 환경에 비해서 데이터나 코드에 대한 보호가 필요한데, 특히, 하나의 데이터를 이전의 valid한 데이터로 바꾸는 공격인 rollback 공격에 대해서 보호가 요구됨 · 최근 프로세서들은 하드웨어를 통한 데이터 보호 기능을 지원하고 있으나 (e.g., Intel SGX) 이러한 지원은 대부분 DRAM만을 지원하므로, 비휘발성 메모리에 대한 지원 필요 - 비휘발성 메모리와 혼용될 수 있는 신뢰할 수 있는 실행환경 · 본 연구에서는 하드웨어 기반 신뢰할 수 있는 실행 환경을 이용해 아래 두 가지 환경에서 데이터 보호를 제공해주는 프레임워크 구현 · (volatile EPC) 기존 Intel SGX환경에서는 enclave라는 고립되고 보호된 영역을 제공하고, 이를 보호하기 위해서 DRAM영역에 하드웨어적으로 보호된 메모리 영역 (EPC)을 설정함. 따라서, Intel SGX환경을 그대로 활용하여 비휘발성 메모리의 데이터를 enclave에 가져와서 데이터 처리를 하고, 다시 비휘발성 메모리에 안전하게 저장하는 방식을 사용하는 환경 · (persistent EPC) Intel SGX에서 제공하는 enclave영역이 비휘발성 메모리까지 확장되는 환경 - 비휘발성 메모리상에서 발생할 수 있는 데이터 공격 · 기본적인 데이터에 대한 공격으로 세 가지가 있음. 첫 번째로 데이터를 가로채는 공격 (snooping), 두 번째로 데이터를 변조하는 방법 (manipulation), 마지막으로 데이터에 대한 접근을 막는 방법 (not available)이 존재함. 본 연구에서는 availability에 대해서는 공격 모델에서 제외 · 데이터를 가로채는 공격에 대해서는 데이터를 암호화하여 저장하는 방법을 사용하며, 데이터를 변조하는 공격에 대한 방어를 위해서는 무결성 검증을 통해서 변조를 확인함. 무결성 검증에서 특히 추가적으로 검증해야할 사항은 무결성 검증에 통과한 이전의 데이터로 최근 데이터를 바꾸게 되었을 때에 대한 확인임 (rollback protection). · 이전 데이터로의 변조를 확인하기 위해서는 각 데이터에 대한 카운터 값을 사용하여 최신의 데이터인지 아닌지에 대한 검증을 수행할 수 있음. 이 카운터 값은 시스템의 파워나 나간 후에도 최신의 카운터를 확인 가능하여야 함 · 비휘발성 메모리 환경에서 앞서 설명한 데이터 공격들이 발생할 수 있음. 그림 185와 같이 volatile EPC와 persistent EPC환경에서도 데이터에 대한 공격이 가능하며 이러한 공격을 막아줄 수 있는 환경에 제공되어야 함  그림 185. 비휘발성 메모리에서 발생할 수 있는 공격

수행기관	개발 목표	개발내용 및 범위

- LibTPO의 디자인
  - · 비휘발성 메모리를 사용하는 프로그래밍 모델들은 데이터를 오브젝트 단위로 생성하고 관리하는 형태를 취하는데, 본 연구에서도 데이터를 오브젝트 단위로 신뢰할 수 있는 영속적으로 관리함. 이 오브젝트를 TPO (Trusted Persistent Object)라고 정의하고, 해당 프레임워크를 LibTPO라고 정의함
  - · LibTPO는 그림 186와 같이 주로 사용하는 Intel PMDK의 libpmemobj의 API와 유사하게 만들어 사용자들이 쉽게 프레임워크를 사용할 수 있도록 함. libpmemobj에서 제공하는 API에 "s_"의 prefix를 추가하면 사용 가능

```
/* initialize trusted transaction supports */
S_TX_BEGIN(pop) {
  /* copy a secret from PMEM to enclave
   * integrity verification & decrypt secret */
  struct data *p = s_pmemobj_direct(secret);
  /* generate a log for the object
   * store the encrypted log with MAC */
  s_pmemobj_tx_add_range(secret,0,sizeof(*p));
  /* after commmit the log, process memcpy() */
  memcpy(p->buf,"secret",7);
  /* copy and store the encrypted secret object
   * update a monotonic counter clear the log */
} S_TX_END
```

그림 186. LibTPO API

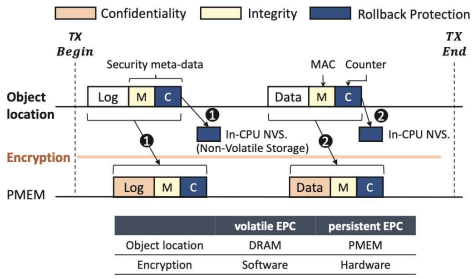


그림 187. LibTPO 데이터 보호 방법

  - · volatileEPC환경에서는 비휘발성 메모리 영역이 보호되는 영역은 아니기 때문에 암호화되고 무결성 검증된 오브젝트들을 DRAM에 존재하는 enclave영역으로 카피를 한 다음, 오브젝트를 복호화하고 무결성 검증한 후 데이터를 처리 (트랜젝션 처리)하고, 데이터 처리 후 다시 암호화 및 무결성 검증을 위한 메타데이터를 생성하여 비휘발성 메모리로 보내는 작업을 수행함
  - · PersistentEPC환경에서는 비휘발성 메모리 영역이 하드웨어 지원을 통해 보호되는 영역이기 때문에 암호화는 하드웨어 지원을 통해서 수행되고, 무결성 검증만을 수행한 후 그대로 비휘발성 메모리에 저장하면 됨
  - · 위와 같은 처리들을 프로그래머가 직접하기에는 에러가 많이 발생할 수 있고, 실수가 유발되기 때문에 라이브러리 차원에서 제공하여 쉽게 프로그래머가 비휘발성 메모리의 데이터를 신뢰할 수 있고 충돌 일관성 있게 사용할 수 있도록 해줌
  - · LibTPO는 충돌 일관성을 제공하기 위해서 트랜젝션을 제공하며, write-ahead log방법을 사용해서 트랜젝션 중간에 컴퓨팅 파워가 나가는 경우 Log를 통해서 복구가 가능하도록 하게 되는데, 이 Log에 대해서도 보호를 하기 위해서 암호화와 무결성 검증방법을 제공
  - · Rollback공격에 대해서 방어하기 위해서 각 오브젝트에 대해서 카운터 값을 저장하도록 하였으며, 데이터를 사용할 때 해당 카운터를 이용해서 rollback공격을 검증 가능하도록 함. 또한, 모든 오브젝트의 카운터 값을 영속적으로 보관할 수 없기 때문에 그림 188과 같이 해시트리의 형태로 관리하고 트리의 루트 값만을 CPU내부의 영속적인 레지스터에 저장하는 방식을 사용함

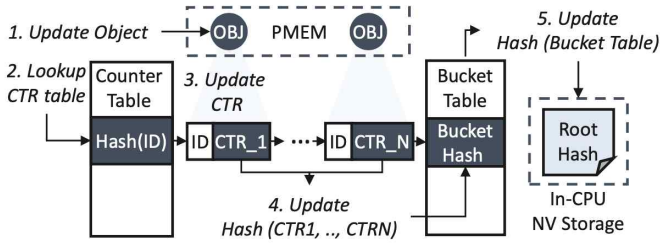
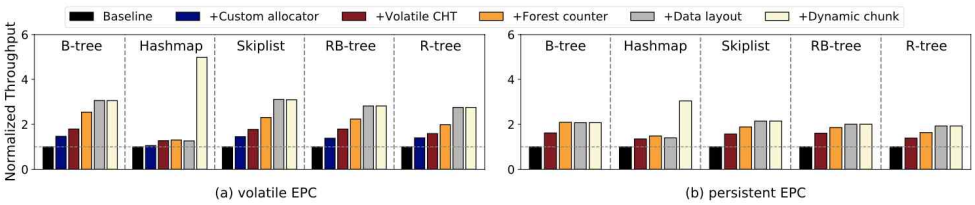
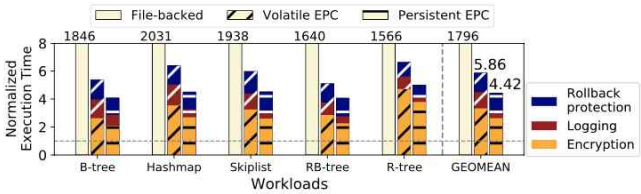


그림 188. Rollback 공격 방어를 위한 카운터 관리 자료구조

- LibTPO의 최적화 기법
  - · 개별 비휘발성 메모리 관리기: enclave 컨텍스트 내에서 비휘발성 메모리를 관리할 수 있

수행기관	개발 목표	개발내용 및 범위
		도록 하여 Ocall의 횟수를 획기적으로 줄임 · 카운터 저장 형태 변경: 카운터를 DRAM에 저장하여 성능 향상. 리커버리시에는 오브젝트와 함께 저장되는 카운터 값을 이용하여 복구 · 데이터 구조 변경: 데이터에 대한 메타데이터를 오브젝트 헤더에 저장함으로써 빠르게 메타데이터를 확인 · 동적 청크-기반 오브젝트 관리: 오브젝트를 작은 크기의 청크 단위로 암호화하는 방식을 사용하고, 동적으로 최적희 청크 크기를 찾아가도록 하여 성능 최적화 · 포레스트 방식의 카운터 관리: 단일 루트가 아닌 여러 개의 루트를 만들어 내는 포레스트 방식의 카운터를 관리하여 계산을 줄임. CPU 내부에 1KB 정도의 비휘발성 저장소면 충분한 것을 확인함 - 리커버리 프로세스 · 정전이나 시스템 크래쉬로 인해서 데이터의 복구가 필요한 경우, LibTPO는 데이터의 복구를 위해서 로그를 저장하고, rollback 공격에나 데이터 변조에 대한 방어를 위해서 카운터 사용 · 리커버리를 위해서는 메타데이터를 이용해 카운터 해시트리를 복구하는 것과, 복구된 해시트리의 루트 카운터 값과 CPU 내부 저장소의 루트 카운터 값 비교 검증이 필요  그림 189. 최적화 기법에 따른 LibTPO의 성능 변화  그림 190. LibTPO 성능 평가 - LibTPO의 성능평가 · LibTPO의 성능 평가를 위해서 5개의 워크로드 (B-tree, Hashmap, Skiplist, RB-tree, R-tree) 자료구조에 10,000개의 오브젝트를 삽입하여 성능을 측정함 · 그림 189은 LibTPO에서 기능구현에 따른 성능을 나타냄. 기본의 구현을 Baseline으로 하고, enclave내에서 비휘발성 메모리의 관리 기능을 구현한 내용을 Custom allocator, DRAM에 카운터 해시트리를 구현한 내용을 Volaitle CHT, 포레스트 방식으로 카운터 루트를 만든 내용을 Forest counter, 데이터 구조를 바꾼 구현을 Data layout, 동적 청크-기반 오브젝트 관리 기능을 Dynamic chunk로 표현함 · 각각의 최적화 기능들은 LibTPO의 성능을 올리는데 큰 영향을 끼치는 것을 확인할 수 있음. 특히, Dynamic chunk의 경우 Hashmap에서 성능의 향상이 매우 큰 것을 확인할 수 있는데, 이는 매우 큰 크기의 오브젝트를 관리할 때 동적 청크-기반 관리가 커다란 성능 향상을 끼치기 때문임 · Baseline에 비해서 VolatileEPC환경에서 3.34배, persistentEPC환경에서 2.24배의 성능향상을 도출함 · 그림 190와 같이 PersistentEPC의 경우 volatileEPC에 비해서 전체적인 성능은 크기 때문에 최적화 기법의 영향이 크지 않음을 확인할 수 있음

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>· 전체적으로 LibTPO는 volaitleEPC환경과 persistentEPC환경에서 비보호 프레임워크에 비해서 5.86, 4.42배 성능 하락을 유발하지만 비휘발성 메모리에서의 데이터 보안과 충돌 일관성을 지켜줄 수 있는 프레임워크임</td></tr>
<tr><td></td><td>③ 대규모 메모리 기반 데이터 분석 가속 알고리즘 개발</td><td>

• 대규모 메모리 기반 유전체 분석 가속 기법 고도화

- 본 연구의 이전 연차에 진행했던 대규모 메모리 기반 유전체 분석 기법에 대한 고도화 및 추가 실험을 진행함
- 기존 연구와의 비교
  · Seeding 단계 가속 기법: BWT와 FM-index 알고리즘을 사용하는 Seeding 단계를 가속하는 방법들은 다음과 같은 것들이 연구됨. 첫째, K-mer FM-index는 FM-index의 탐색 단위를 K글자로 늘림. ERT는 다양한 길이의 글자 탐색을 지원하는 FM-index를 제안. 둘째, FM-index의 구조를 바꾸어 메모리 접근의 순서를 바꾸어 접근 속도를 최적화하거나, 프리페칭을 적용하기도 함. 셋째, learned-index를 적용하여 가속하는 기법도 연구됨. 본 연구에서 개발한 기법은 FM-index의 캐시를 만들어서 Seeding 단계를 가속함
  · Extension 단계 가속 기법: Dynamic Programming을 사용한 string matching 알고리즘을 사용하는 Extension 단계를 가속하는 방법들은 다음과 같은 것들이 연구됨. 첫째, SIMD 명령어를 사용한 가속 기법이 연구됨. 둘째, FPGA나 ASIC을 사용해 전용 가속기를 개발. 셋째, pre-alignment filtering을 통해 Extension 단계를 수행하는 seed 개수를 줄이는 방법. 본 연구에서는 Exact Match Filter를 통해 seeding과 extension 단계를 모두 생략할 수 있는 기법을 제시. Exact Match Filter를 부분적으로만 사용함으로써 메모리 용량과 성능의 교환 관계도 만들어냄.
  · 기타 가속 기법: GenCache와 GenStore는 여러 단계로 구성된 유전체 분석 기법을 제시함. 앞 단계에서는 간단한 연산으로 가능한 매핑을 찾고, 앞 단계에서 매핑 찾기에 실패할 경우 다음 단계의 더 복잡한 연산으로 서열 정렬을 수행. 하지만 특별한 하드웨어가 필요하다는 단점이 있음. 본 연구에서는 상용 시스템에서 서열 정렬 가속 효과를 얻음.

- 유전체 분석 가속 기법 고도화
  · 메모리 용량 별 성능 효과를 고려하여, FM-index Accelerator의 크기를 총 1.5GB 정도로 최적화함. 그에 따라 Selective Index Loading 알고리즘을 다음 그림과 같이 수정

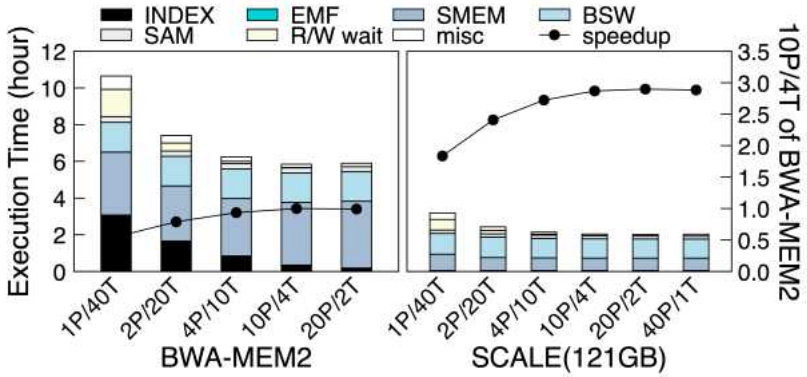
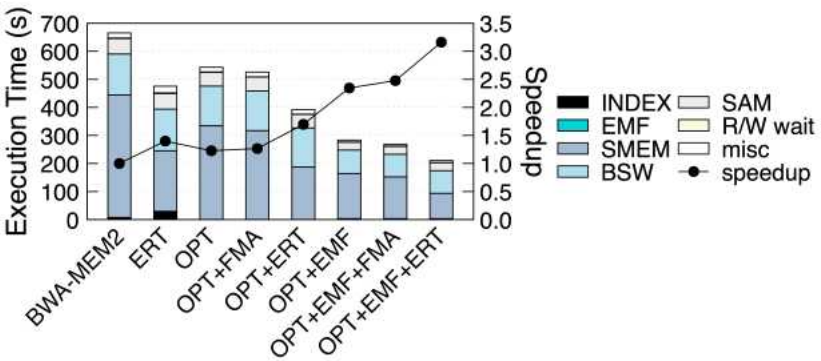
Index	Size	Perf ↑
Reference genome (Ref)	7GB	-
FM-index (FM)	10GB	-
FM-index Accelerator (FMA)	1.5GB	3.2%
Exact Match Filter (EMF)	54GB	125%
Enumerated Radix Tree (ERT) [ISCA'21]	60GB	33%

Memory	Ref	FM	FMA	EMF	ERT
17GB	O	O	X	X	X
19GB	O	O	O	X	X
20-72GB	O	O	O	Part	X
73GB	O	O	O	O	O
121GB	O	X	X	O	O

그림 191. 대규모 메모리 기반 유전체 분석 가속 기법의 Selective Index Loading 기법

- 멀티프로세스를 활용한 성능 가속과의 비교
  · BWA-MEM2는 병렬성이 완벽하지는 않으므로, 메모리 용량이 충분할 때 멀티프로세스로 실행하면 성능 향상 효과를 볼 수 있음. 이는 유전체 분석에서 많이 쓰이는 성능 향상 기법임. 본 연구에서 개발한 BWA-MEM-SCALE에 대해서도 같은 기법을 사용하여 성능을 측정함. HG001 데이터셋의 871개 파일을 모두 분석하는데 걸리는 시간을 측정

</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		 그림 192. 멀티프로세스를 사용했을 때의 유전체 분석 성능 비교. 10P/4T는 10개의 프로세스를 사용하고 각 프로세스가 4개의 스레드를 사용함을 의미 · 위 그림을 보면 BWA-MEM2는 10개의 프로세스가 각각 4개의 스레드를 사용했을 때 최고의 성능을 보여주며, 단일 프로세스 사용시에 비해 1.83배의 성능 향상을 보임. 단일 프로세스의 BWA-MEM-SCALE은 BWA-MEM2의 최고 성능에 비해서도 1.59배의 성능 향상을 보였으며, 멀티프로세스 사용시 최대 2.89배의 성능 향상을 보임. 또한 멀티프로세스가 실행될 때 in-memory index store의 효과로 전체 메모리 사용량이 오히려 줄어듬 · 성능 향상의 가장 큰 부분은 in-memory index store부터 얻을 수 있었음. BWA-MEM2의 index loading은 단일 스레드로 수행될 수밖에 없는데 BWA-MEM-SCALE은 이를 생략할 수 있으므로 전체적인 CPU 활용율도 증가하였음 - 워크스테이션 기반 성능 실험 · BWA-MEM-SCALE이 다양한 환경에서 성능 향상 효과를 얻을 수 있는지 분석하기 위해 16코어 규모의 워크스테이션에서 BWA-MEM2와 비교한 성능 실험을 진행함  그림 193. 16코어 워크스테이션에서의 BWA-MEM-SCALE 성능 실험 · 위 그림을 보면 최대 성능 향상은 3.16배로, 40코어 서버 환경보다 살짝 높은 수준. 반면 최적화 기법(OPT에 해당)들의 성능 향상 효과는 미미함. 그 이유는 워크스테이션 환경이 서버 환경에 비해 코어 수가 적고 AVX512 지원이 없는 등 컴퓨팅 파워는 떨어지는데 비해, PCIe 4.0을 탑재하여 I/O가 빠르기 때문임. 이로 인해 Index loading이나 read/write stage 등 I/O 관련 최적화에 대한 성능 향상 효과는 떨어지는 반면, mapping stage에 대한 성능 향상 효과는 더 극대화되는 것임 - 클라우드 기반 성능 및 비용 비교 · FGPA를 활용할 수도 있는 클라우드 시스템에서 성능과 비용에 대한 비교를 수행함

수행기관	개발 목표	개발내용 및 범위

표 22. FPGA-ERT와 BWA-MEM-SCALE에 대한 클라우드 성능 및 비용 비교

	AWS Instance	#cpu	DRAM	$/year	Speedup
FPGA-ERT	f1.4xlarge (FPGA)	16	244GB	18,242	2.1
BWA-MEM-SCALE	i3.8xlarge	32	244GB	13,928	2.6

· 위 표에서 FPGA-ERT는 최신의 가속기를 활용한 BWA-MEM2 가속 연구에 해당함. 이를 구동하기 위해서는 FPGA가 포함된 클라우드 인스턴스를 사용해야 함. 16개의 CPU가 탑재된 인스턴스를 사용했을 때 1년에 18,242달러가 필요하고, 논문에 언급된 BWA-MEM2 대비 성능 향상은 2.1배

· 본 연구에서 제안하는 BWA-MEM-SCALE은 FPGA가 필요 없으므로 더 저렴한 인스턴스에서 수행 가능. 단, 메모리 용량은 충분히 많아야 함. 표를 보면 FPGA-ERT가 사용한 것보다 코어 수가 2배 많은 인스턴스를 사용했는데도 연간 비용은 더 저렴함. 그리고 해당 인스턴스에서 얻을 수 있는 BWA-MEM2 대비 성능 향상은 2.6배로 더 높음

• 계층적 메모리 기반 양자 시뮬레이터 가속 기법 연구

- 이전 연도에 진행했던 양자 시뮬레이터에 대해 실제 알고리즘에 대한 시뮬레이션을 실행하면서 추가 요구 사항을 발견하여 추가 기술 개발을 진행함
  · 실험에 사용한 알고리즘은 다음과 같음: Bernstein Vazirani, QFT, grover
- 양자 상태 서버에서 메모리 풀 구현
  · 본 연구에서 개발한 양자 시뮬레이터는 계층적 메모리를 기반으로 하면서 분산 환경 실행을 지원하기 위해 양자 상태 서버와 양자 상태 클라이언트를 구현함. 각 서버에서 실행 중인 시뮬레이션에 필요한 양자 상태가 다른 서버에 있을 경우 이를 주고 받기 위한 것
  · 기존 구현에서는 양자 상태 서버 스레드들이 각자의 메모리 버퍼를 확보하는 구조였으나, 많은 양자 게이트로 구성된 양자 회로를 시뮬레이션하기 위해서는 각자의 메모리 버퍼를 확보하다가 메모리가 부족해지는 상황이 발생함
  · 이 같은 상황을 방지하기 위해 각 서버에서 양자 상태 서버를 위한 공유 메모리 풀을 만들고, 양자 상태 서버가 데이터 수신/송식을 위해 메모리 버퍼가 필요할 때 공유 메모리 풀에서 메모리를 할당 받도록 구현함
  · 각 양자 게이트를 시뮬레이션할 때 대부분의 경우 2개 서버가 짝을 이루어 서로의 양자 상태 데이터를 송수신하므로 크지 않은 공유 메모리 풀만으로 경쟁 없이 메모리 버퍼를 사용하도록 할 수 있음
- 디스크 결함 내성 구조 설
  · 본 연구에서 개발한 양자 시뮬레이터는 많은 수의 디스크를 사용하기 때문에, 많은 수의 양자 게이트로 이루어진 양자 회로를 시뮬레이션 할 경우 시뮬레이션이 종료되기 전에 디스크 결함이 발생할 확률이 높음. 따라서, 디스크 결함 상황에서도 시뮬레이션을 지속할 수 있게 하기 위한 기법을 설계함
  · 본 연구에서 개발한 양자 시뮬레이터는 디스크를 두 개의 세트로 나누어 각 게이트를 시뮬레이션할 때 한 세트의 디스크에서 양자 상태를 읽고 다른 세트의 디스크에 게이트를 통과한 양자 상태를 씀. 다음 게이트를 시뮬레이션할 때는 세트를 바꿈
  · 디스크 결함 내성 제공을 위해 각 세트의 디스크에서 2개씩을 ECC를 위한 디스크로 할당

수행기관	개발 목표	개발내용 및 범위

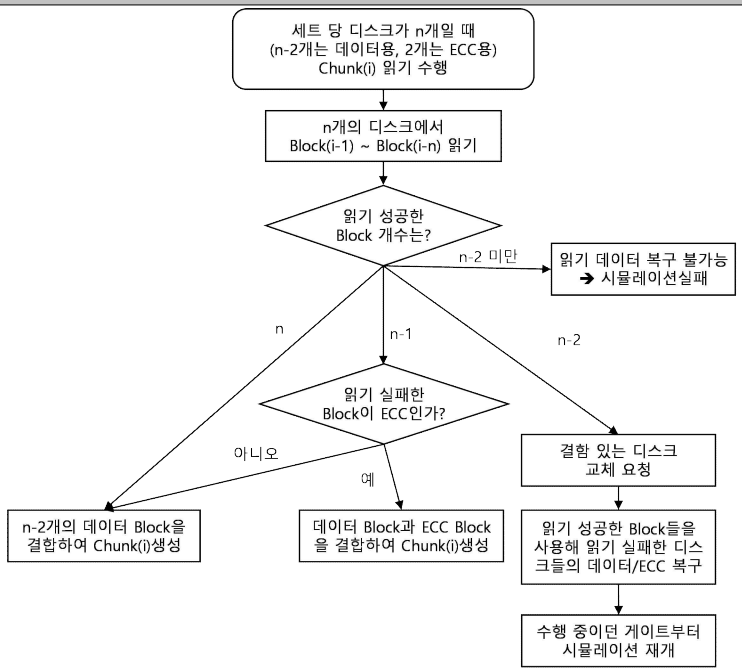


그림 194. 양자 상태 디스크 관리 장치: 읽기 동작

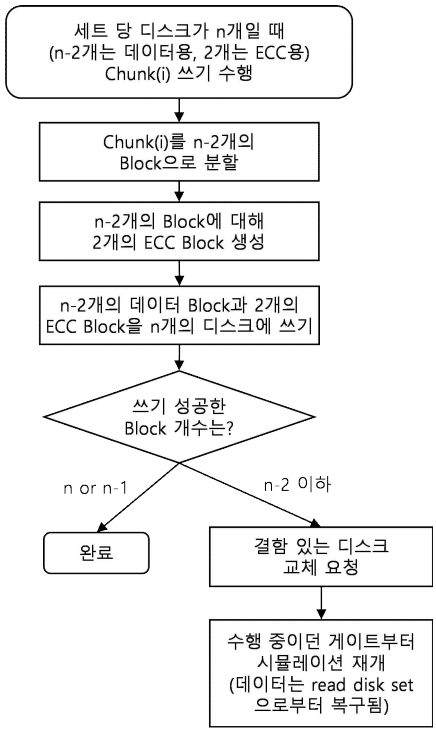


그림 195. 양자 상태 디스크 관리 장치: 쓰기 동작

· 디스크 결함 내성을 제공하기 위한 기법은 위 그림과 같음. 읽기 동작 시에는 읽기에 실패한 디스크가 1개 이하일 때는 읽기 성공한 블록들만으로 요청했던 양자 상태 데이터를 만들어 제공하고 완료. 읽기에 실패한 디스크가 2개라면, 디스크 교체를 요청한 뒤, 디스크 교체 후에 가용한 데이터를 사용해 전체 데이터를 복구하고 중지됐던 게이트의 처음부터 시뮬레이션 재개. 만약 3개 이상의 디스크가 동시에 실패한다면 시뮬레이션이 실패함

· 쓰기 동작 시에는 쓰기에 실패한 디스크가 1개 이하일 때는 그대로 지속함. 그리고 쓰기에 실패한 디스크가 2개 이상일 경우에는 디스크 교체를 요청한 뒤, 디스크 교체 후에 중지됐던 게이트의 처음부터 시뮬레이션 재개. 직전 게이트의 결과 상태가 읽기 디스크 세트에

수행기관	개발 목표	개발내용 및 범위

존재할 것이므로 데이터 복구가 필요하지 않으며, 많은 수의 디스크가 동시에 결함이 발생하더라도 상관없음

- Bernstein Vazirani 알고리즘에 대한 성능 평가
  · 양자 알고리즘 중 하나인 Bernstein Vazirani 알고리즘에 대해 QuEST와 비교한 성능을 평가함. 아래 그림에서 x축의 숫자는 시뮬레이션에 사용된 노드의 숫자

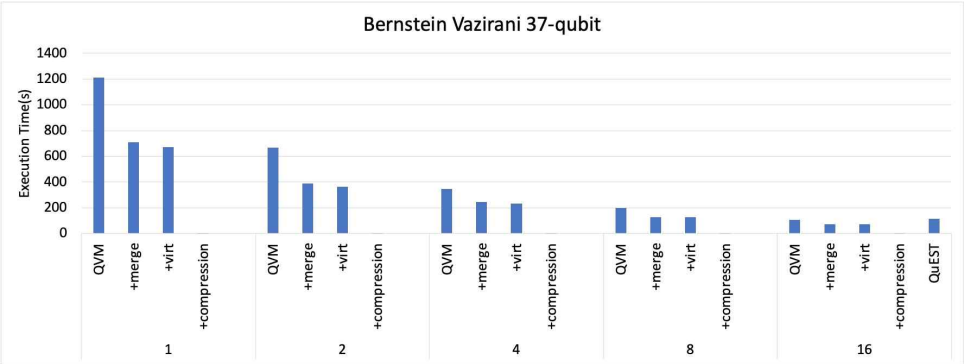


그림 196. 양자 시뮬레이터 성능 평가(Bernstein Vazirani 알고리즘, 37큐빗)

  · QuEST는 메모리만을 사용해 시뮬레이션을 수행하므로 37큐빗 규모의 시뮬레이션을 하려면 16대가 필요. 본 연구에서 개발한 양자 시뮬레이터는 1대만으로도 시뮬레이션 가능
  · 16대를 사용했을 때, 본 연구에서 개발한 양자 시뮬레이터가 최적화 기법 없이도 더 좋은 성능을 보였으며, 게이트 합성(merge), 큐빗 가상화(virt), 양자 상태 압축(compression) 등의 최적화 기법이 추가됐을 경우에는 월등히 나은 성능을 보임

• **[정량 목표] 대규모 그래프 분석 가속 연구**
- 개념: 1 trillion edge 규모의 그래프 분석 응용에 대한 연구
- 시스템 구성: 1 trillion edge는 데이터 규모가 크기 때문에 고성능의 장치 및 많은 양의 메모리를 갖춘 시스템 구성이 필요
  · CPU: Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz X 2개 (총 112코어)
  · DRAM: 1024GB
  · 그래프 저장용 SSD: Intel SSD D7-P5510 (7.68TB) X 4개
  · 그래프 분석용 SSD: Intel SSD D7-P5510 (7.68TB) X 20개
- 소프트웨어 구성
  · 운영체제: CentOS 8 Stream, Linux 4.18.0-358el8
  · 그래프 생성 플랫폼: TrillionG (SIGMOD'17)
  · 그래프 분석 플랫폼: Graphene (FAST'17)
- 개발 내용
  · TrillionG를 사용해 Vertex 4,294,963,201개, Edge 976,767,207,659 개(약 1 trillion)의 그래프 생성
  · Graphene은 사용하는 디스크 개수와 병렬성이 비례하므로 시스템 내 최대한 많은 디스크를 그래프 분석용으로 배정
  · 성능 실험: BFS 알고리즘 수행 결과, 1356.78초 소모

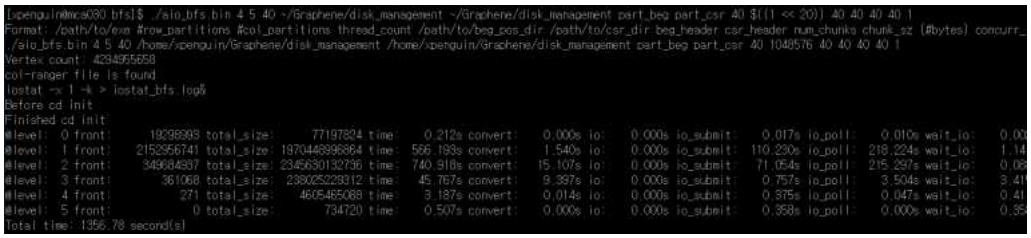
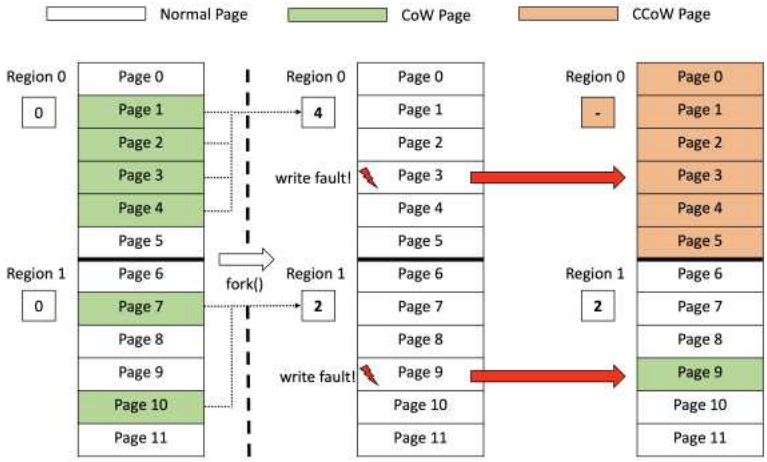
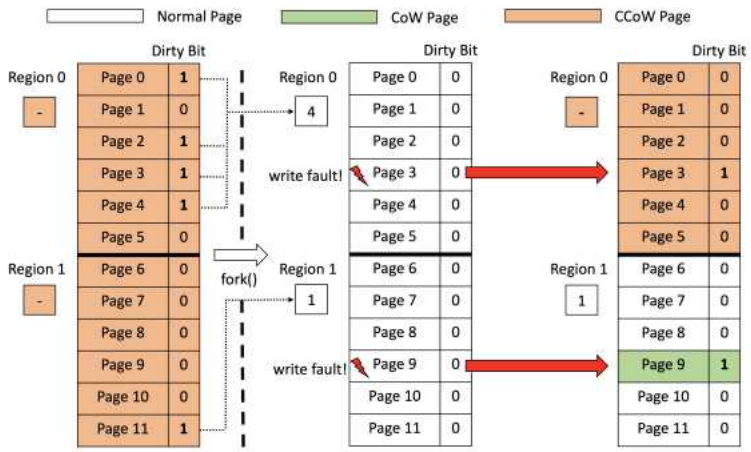
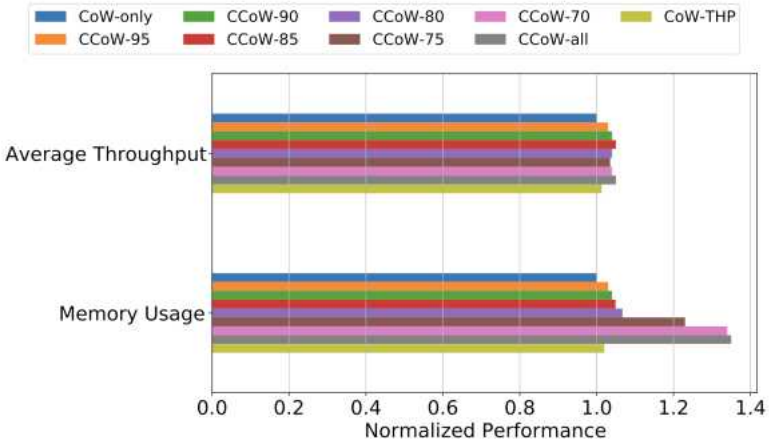


그림 197. 1 trillion edge에 대한 BFS 알고리즘 수행 결과

수행기관	개발 목표	개발내용 및 범위
아주 대학교 (공동1)	• 고속 데이터 접근을 위한 소프트웨어 통합설계 기술	• **Spatial locality 적응형 Copy-on-Write 최적화 기술** - 연구 개요 · 대규모의 메모리를 사용하는 응용에서 데이터의 변화 추이를 기록하거나 예상치 못한 장애 발생시 데이터의 정합성이 보장되는 상태로 복원(recovery)하기 위해 일정 시간마다 메모리의 스냅샷(snapshot)을 남기는 기술이 보편적으로 사용되고 있음. 특히 이러한 기술은 대규모의 데이터를 장시간 처리해야 하는 memory-intensive compute-intensive한 데이터센터 응용에서 자주 관찰되는 기능임 · 이러한 메모리 스냅샷을 만들기 위해 응용이 사용하고 있는 메모리 전체를 복사(copy)하여 복제본을 만드는 방식을 고려해 볼 수 있겠으나 수 십에서 수백 GB 규모의 메모리를 사용하는 memory-intensive한 응용에서는 메모리 복사가 매우 오래 걸리게 됨. 또한 참조 지역성(locality of reference)이 높은 워크로드에서는 스냅샷 사이에 발생하는 변화는 전체 데이터셋의 크기에 비하면 작은 편인데 이를 고려하지 않고 전체를 복사하는 방식은 공간 및 처리 시간 측면 모두에서 매우 비효율적임 · 이러한 단점을 극복하기 위해 fork() 시스템 콜과 운영체제 수준에서 제공하는 copy-on-write를 활용하여 snapshot process를 만드는 방식으로 메모리 스냅샷을 만드는 방법이 Redis 등을 비롯한 memory-intensive 응용에서 활용되고 있음 · POSIX standard에 따라 fork() 시스템콜을 통해 process를 만드는 경우 parent process의 address space와 동일(identical)한 child address space를 만들어야 함. 하지만 앞서 스냅샷의 경우와 마찬가지로 프로세스를 새로 만드는 시점에 parent process의 모든 메모리 데이터를 복사하여 복제하는 것은 매우 비효율적이어서 대부분의 OS는 copy-on-write라는 기법을 활용함 · 운영체제는 fork등을 위해 어떤 데이터의 복제본이 필요해지는 시점에 해당 데이터를 별도의 공간에 복사하지 않고 processes의 page table을 수정하여 서로 다른 프로세스 간에 같은 page를 가리키게 하여 데이터를 공유하여 사용하게 하는 기술임. 이렇게 공유되는 페이지들은 읽기 요청의 경우 한 개의 같은 페이지를 통해 처리될수 있으며, 추후 쓰기 요청이 발생하는 경우에나 다른 페이지에 원래 내용을 복사하여 처리함으로써 데이터 복제 발생을 최소화 하는 기술임 · 이러한 기능은 기본적으로 OS의 page fault hander를 통해 처리, 구현됨. 같은 페이지를 공유하는 두 processes의 page table entries는 쓰기 권한이 없는 상태로 설정됨. 이후 process가 쓰기 접근을 할 경우 MMU는 해당 요청을 처리할 수 없게 되고, 이는 시스템 초기화에 정의된 운영체제 내의 page fault handler를 통해 처리하게 됨 · Linux는 fork 뿐만 아니라 같은 내용을 저장하는 메모리 공간을 간결하게 정리하는 KSM(kernel same page merging)에도 같은 기법을 활용하고 있음. 따라서 copy-on-write를 효과적으로 처리하는 방법은 시스템 수준에서 메모리 관리 성능을 결정짓는 중요한 기능이라고 볼 수 있음 · 현재 Linux에서의 copy-on-write는 페이지 단위로 처리가 됨. 즉 fork 시점에 parent 및 child process의 address space가 모두 동일한 데이터를 가리키게 설정된 이후, write가 발생할 때 마다 page fault가 발생, page 단위로 데이터의 복사가 이루어짐. 이러한 접근 방법은 복사되는 데이터의 양을 최소화 할 수 있는 측면에서 메모리 사용을 효과적으로 할 수 있는 장점이 있으나 매 page fault마다 유저-커널 간의 CPU mode 전환이 이루어져야 하고 이 과정에서 많은 overhead가 발생함. 특히 이는 write-intensive한 workload에서 큰 overhead를 유발하게 됨 · 본 연구에서는 이러한 copy-on-write의 효율을 개선하기 위해 메모리 영역별로 copy-on-write가 발생할 정도를 프로파일하고 이를 바탕으로 copy-on-write를 위해 page fault가 발생한 페이지 근처에서 근시간 내에 다시 copy-on-write를 위한 page fault가 발생할 가능성을 유추하고 이를 바탕으로 필요시 data copy를 미리 수행(precopy)하는 coverage copy-on-write를 (CCoW)기술을 제안, 개발하였음 - Spatial locality 적응형 Copy-on-Write 최적화 기술 연구 개발 내용

수행기관	개발 목표	개발내용 및 범위
		· 기본적으로 CCoW는 address space를 일정 크기의 region으로 나누고, 각 region별로 copy-on-write가 발생할 수 있는 빈도(copy-on-write intensity)를 추적, 관리함. 만약 어떤 page에 대해 write 접근으로 인해 copy-on-write를 처리해야 하는 경우 해당 page가 속한 region의 과거 copy-on-write 발생 빈도를 통해 다시 page fault가 발생할 가능성을 평가하게 됨  그림 198. Normal region에서 fork와 page fault 처리 · 기본적으로 copy-on-write를 처리하는 과정에서 OS의 page fault handler가 불리우므로, 이를 통해 각 region에서 발생한 page fault 개수를 카운트할 수 있음. 해당 count는 fork를 통해 child process가 만들지는 과정에서 parent의 address space와 함께 복사되어 전달됨. 실제 시스템에는 process address space를 구성하는 virtual memory area (VMA)마다 해당 VMA내의 각 region에 대한 정보를 저장하도록 구현 · 위의 그림에서 왼편의 region 0과 1은 각각 copy-on-write가 4회 및 2회 발생하였음. 이때 fork를 수행한다면 parent 및 child는 해당 region에서 발생한 copy-on-write count인 4와 2가 전달됨. 이 때 만약 page 3에서 page fault가 발생했다면 region 0은 copy-on-write가 자주 발생한다고 판단하여 page 3 뿐만 아니라 region 0의 모든 페이지에 대해 precopy(미리 복사해 놓음)를 수행함 · 반면 page 9에 대해 page fault가 발생한 경우 해당 region의 page fault 횟수가 적으므로 이 경우에는 precopy를 수행하지 않고 기존과 동일하게 page 9에 대해서만 copy-on-write를 처리함 · 이러한 적응형 접근 방법을 통해 기존에 여러 번의 page fault를 통해 copy-on-write가 이루어지는 region은 page보다 큰 region 단위로 copy-on-write가 이루어지게되어 page fault 횟수를 획기적으로 감소시킬수 있게됨. · 하지만 이러한 방식을 사용한 경우 한 번 precopy가 일어난 region에는 다음에는 precopy를 할 수 없게 됨. 예를 들어 위의 예제에서 region 0이 precopy를 통해 미리 복사되었다고 가정하면 해당 region에 속한 페이지로의 모든 접근은 더 이상 page fault가 발생하지 않게 됨. 따라서 해당 region의 copy-on-write intensity는 매우 낮게 판단되게 되게 됨. 따라서 바로 이후의 fork 직후 page 3에 page fault가 발생했다고 하더라도 precopy는 이루어지지 않게 됨. 만약 region 0이 지속적으로 copy-on-write가 발생하는 영역이더라도 해당 fork를 처리하는 동안에는 CCoW 방식을 통한 이득을 볼 수 없게 되는 문제가 있음.

수행기관	개발 목표	개발내용 및 범위
		 그림 199. Precopy region에서 fork와 page fault handling · 본 연구팀은 page fault의 발생 여부와 상관없이 precopy를 통해 이득을 볼 수 있는 region을 알아낼 수 있는 매커니즘을 발굴하였음 · PTE의 dirty bit은 해당 PTE가 담당하고 있는 메모리 영역에 write 접근이 일어났을 경우 MMU가 자동으로 set하는 field임. 따라서 어떤 page의 dirty bit이 set되어 있다는 의미는 해당 page에 쓰기 접근이 일어난 것을 의미함. · 이에, 특정 region에서 copy-on-write intensity를 측정함에 있어 page fault 발생 카운트 대신 해당 region에 대응되는 page table에서 dirty bit이 set된 PTE의 개수를 카운트 하는 방식을 사용하도록 개선하였음. 위의 그림에서 region 0과 1 모두 바로 이전 fork 이후 precopy를 통해 미리 복사된 region이라고 가정하면, 이 중 어떤 페이지로의 접근도 page fault는 유발되지 않음. 이 시점에 fork가 발생한다면 해당 region의 copy-on-write intensity는 dirty bit이 set된 PTE의 개수인 4와 1이 되고, 이를 바탕으로 region 0은 copy-on-write가 자주 발생하여 precopy의 이득을 볼 수 있는 region으로, region 1은 그렇지 않은 region으로 판단할 수 있음. - Spatial locality 적응형 Copy-on-Write 최적화 기술 성능 평가 · 제안하는 방법을 Linux 5.7.7에 구현하였음. 구현에는 약 400라인 정도의 kernel 수정이 필요하였음 · 실험은 Intel Xeon Gold 5215 CPU, 128 GB의 메모리를 가진 노드에서 YCSB와 Redis를 통하여 성능을 평가하였음. 제안하는 방식은 kernel에 구현되어 YCSB나 Redis의 변경은 필요하지 않았음. 또한 실험에는 2MB의 region을 사용하여 PMD수준에서의 page table manipulation을 수행하며 PTE의 dirty count를 처리하도록 하였음 · 특정 region이 precopy를 해서 이득인지 여부를 판단하는 threshold값을 바꾸어가며 Redis의 성능과 메모리 사용량을 측정하였음. 아래의 그림은 threshold 값의 변화에 따른 성능 변화를 나타냄. CCoW-80의 경우 한 region에 속한 page 중 80% 이상이 dirty인 경우 precopy를 수행하는 것을 의미함. 따라서 threshold가 높을수록 기존의 copy-on-write와 같게 되고, threshold 값이 작을수록 미리 precopy를 수행하는 형태임 · Threshold가 작아짐에 따라 precopy를 더 많이 수행하게 되므로 Redis의 성능은 최대 5%까지 향상됨. 특히 기존의 THP를 사용했을 때 보다 우수한 성능을 보이는 것을 확인하였음. 하지만 이로 인해 fork 직후 메모리 복사가 많이 일어나게 되어 Redis 응용의 memory footprint가 증가하고, 특히 threshold가 매우 작은 경우에는 기존 copy-on-write 대비 30% 이상의 추가 메모리를 사용함. 따라서 적절한 값의 threshold를 사용해야 함을 알 수 있음 · 실험 결과를 바탕으로 응용이 사용하는 메모리가 영역별로 매우 상이한 접근 특성을 가지

수행기관	개발 목표	개발내용 및 범위
		고 있음을 확인할 수 있었음. 특히 write-intensive한 접근은 process address space에서 특정 영역에 집중되는 것을 확인하였음. 이러한 관찰 결과를 바탕으로 분리형 메모리 시스템에서 write-intensive한 영역에 속한 데이터는 evict하지 않고 최대한 local에 배치함으로써 빠른 처리를 유도하고, read-only 데이터는 원격 메모리에 배치하여 데이터 접근 속도를 향상시키는 방안에 대해 검토 중임  그림 200. Precopy threshold값에 따른 Redis 성능 및 메모리 사용량
성균관 대학교 (공동2)	• 비휘발성 메모리 기반 인메모리 저장소 연구	**• Disaggregated Clean Cache (DCC) 연구** - 연구의 필요성 · 컴퓨팅 자원의 사용량에 대한 예측은 종종 정확하지 않기 때문에 효율적으로 데이터 센터의 컴퓨팅 자원을 할당하는 것은 어려운 문제임. 메모리 사용량은 응용 프로그램의 종류뿐만 아니라 시간에 따라서도 메모리 사용량이 다르기 때문에 최적의 메모리 크기를 결정하는 것은 더욱 어려운 문제이며, 이는 메모리 활용률 저하 문제로 이어짐 · 그런데 단일 프로세스의 메모리 풋프린트는 메모리 크기를 초과하는 경우가 많지 않으며, 여러 프로세스가 동시에 수행이 될 때에만 발생함. 즉, 익명 페이지를 위한 메모리 공간이 부족한 문제는 스케줄링을 통해 어느 정도 해결이 가능 · 그러나, 메모리가 부족한 시스템에서 익명 페이지가 아닌 파일 시스템으로부터 읽은 데이터셋의 크기가 큰 경우, 클린 파일 페이지(Clean Page)들은 메모리에서 삭제되고 그 메모리 공간은 익명 페이지를 위해 사용됨 · 클린 페이지는 즉각적인 디스크 IO로 이어지지는 않지만, 장기적으로 캐시 미스가 많아지고 성능 감소를 초래함 · 데이터 분석 응용이나 머신러닝 프레임워크는 상당한 메모리 공간을 요구하는데 이는 익명 페이지가 아니라 파일 페이지 때문인 경우가 많음

수행기관	개발 목표	개발내용 및 범위

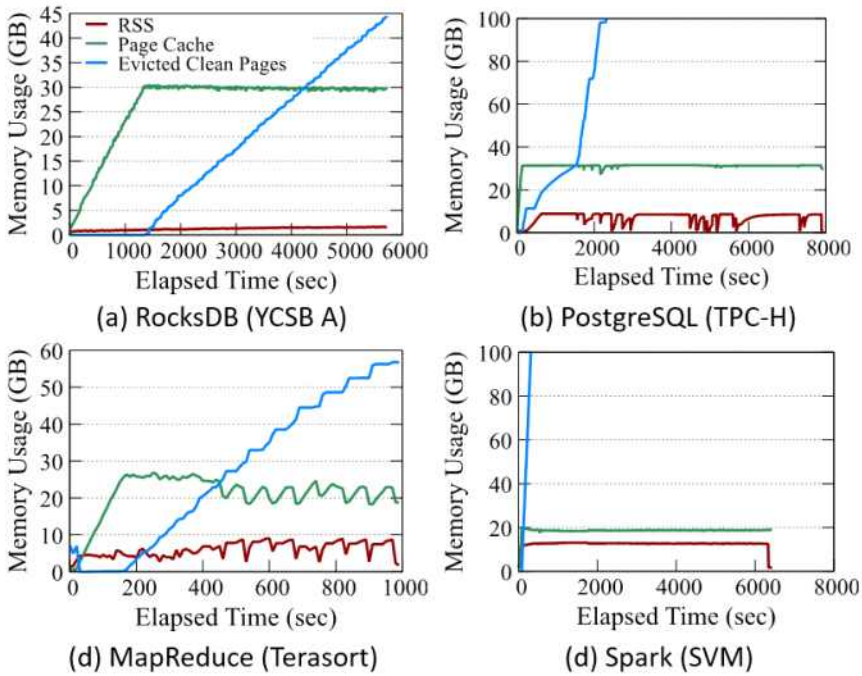


그림 201. 시간에 따른 응용의 메모리 풋프린트

· 위 그림은 RocksDB, PostgreSQL, Hadoop MapReduce 및 Spark와 같은 단일 애플리케이션 프로세스의 메모리 풋프린트를 측정한 결과임. RSS(Residual Set Size)는 익명 페이지를 포함한 워크로드의 페이지, 페이지 캐시는 파일 시스템으로부터 읽은 파일 페이지의 크기, Evicted Clean Pages는 페이지 캐시에서 제거된 클린 페이지의 누적 크기를 나타냄
· 전반적으로 익명 페이지는 메모리 사용량의 매우 작은 부분을 차지하는 반면, 클린 페이지가 메모리 사용량의 대부분을 차지함을 확인할 수 있음. 즉, 물리적 메모리의 크기가 매우 작거나 과도한 수의 애플리케이션 프로세스로 리소스가 과도하게 커밋되지 않는 한 RSS는 물리적 메모리의 크기를 거의 초과하지 않음

- 대용량 메모리를 활용한 Clean Page를 위한 분리형 메모리 시스템
· 본 연구에서는 클린 페이지를 Intel Optane DCPMM 등 대용량 메모리를 활용한 분리형 메모리 (Disaggregated Memory) 서버로 보내 활용함으로써 메모리 활용률 저하를 해결하고자 함
· DCC (Disaggregated CleanCache)는 로컬 메모리에서 유지할 수 없는 클린 파일 페이지를 원격 대용량 메모리로 보내 재활용할 수 있게 만드는 2차 캐시임
· DCC는 메모리 전용 서버를 활용하는 Dedicated (Client-Sever) 모드와 클러스터 내의 유휴 메모리를 활용하는 Shared (Client-Client) 모드를 지원함. Dedicated 모드의 DCC를 개발 완료했고, Shared 모드를 개발 중

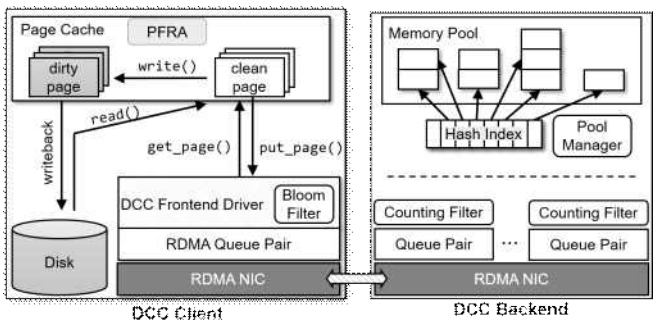


그림 202. DCC 아키텍처

수행기관	개발 목표	개발내용 및 범위
		· DCC는 위 그림과 같이 클린 페이지를 원격 메모리 서버로 전송하는 클라이언트인 프론트엔드 드라이버 모듈과 클린 페이지를 받아 관리하는 백엔드로 구성되며, 이 둘은 RDMA 통신으로 데이터를 교환 · 기존 메모리 Disaggregation 연구들은 빠른 One-sided RDMA를 사용하지만, DCC는 원격 메모리를 유연하게 관리하기 위해서 Two-sided RDMA를 사용 · 클린 페이지를 백엔드로 전송하는 Put 연산은 inode 번호와 페이지 오프셋을 키로 사용하며, 클린 페이지를 RDMA_SEND를 통해 원격 메모리에 저장함 · 페이지 캐시에서 캐시 미스로 백엔드에서 페이지를 읽는 Get 연산은 RDMA_WRITE_IMM 연산으로 키와 전송받을 페이지의 DMA 주소를 백엔드의 등록된 메모리 영역에 쓰고 RDMA_WRITE_IMM 연산으로 페이지를 전송 받음 · 페이지를 삭제하는 Invalidate 연산은 페이지를 프론트엔드로 전송받지 않는다는 점을 제외하고는 Get 연산과 동일하게 동작함 - 캐시 정책 (Exclusive & Non-Inclusive Caching) · DCC는 페이지 캐시의 2차 희생 캐시로서 페이지는 페이지 캐시 혹은 DCC 둘 중 한 곳에만 있어야 하는 exclusive 캐시 정책을 기본적으로 사용함. 그러나, 프론트엔드가 클린 페이지를 백엔드로부터 읽은 후, 자주 사용하지 않는다면 얼마 지나지 않고 다시 페이지를 백엔드로 전송하여 네트워크 대역폭이 불필요하게 낭비되는 결과를 초래할 수 있음 · 이에 대한 대안으로 non-inclusive 캐시 정책을 추가하였음. 이 경우 전체 용량은 페이지 캐시와 DCC의 캐시에 중복되지 않은 크기의 합이 됨. Get 연산은 DCC에서 읽은 페이지를 삭제하지 않기 때문에 읽어들인 페이지가 수정된 경우 페이지 플래그를 통해 수정을 탐지해야 하며, 업데이트된 페이지는 DCC로 전송해서 원래의 페이지를 업데이트해야 함 · 페이지 캐시 레벨의 캐시 일관성을 유지하는 오버헤드를 줄이기 위해 DCC가 NFS와 같은 공유 파일 시스템을 사용하지 않는 경우만 non-inclusive 정책을 사용하는 것을 권장함 - 접근 패턴에 따른 캐싱 · Sequential Read는 소프트웨어적으로는 커널의 read-ahead 매커니즘, 하드웨어적으로는 RAID와 같은 도움으로 최적화할 수 있는 여지가 있으나, Random Read의 경우 저장장치의 성능이 향상되어도 여전히 Sequential Read에 비해 높은 지연시간과 낮은 처리량을 보임 · 리눅스의 페이지 회수 정책은 inactive list에서 참조된 페이지를 active list로 이동(승진)하고 그렇지 않은 경우는 제거하며, active list에서 참조된 페이지는 그대로 남아있고 그렇지 않으면 inactive 리스트로 이동(강등)함. 따라서, inode 별로 제거되는 페이지 오프셋이 인접하다면 이러한 페이지들은 향후에 readahead의 효과를 보거나 공간 지역성이 높기 때문에 한 번에 읽힐 수 있어서 IO 비용이 높지 않다고 판단할 수 있음 · 이런 정보를 바탕으로 접근 패턴에 따라 IO 비용이 비싼 페이지들을 선별적으로 캐싱함으로써 네트워크 대역폭이 낭비되지 않으면서 효율적으로 사용할 수 있는 클린 캐시 구현 - 프론트엔드 메타데이터 · 프론트엔드에서 캐시 미스가 발생할 때마다 DCC 백엔드를 접근하는 것은 효율적이지 못하므로 프론트엔드는 블룸필터를 포함한 메타데이터를 관리함으로써, 백엔드와 불필요한 통신을 피하게 함 · 프론트엔드 메타데이터는 크기가 작아야 하며 거짓 양성 비율이 높으면 안 됨. 이를 만복하기 위해 프론트엔드에는 블룸필터, 백엔드에는 카운팅 블룸필터가 협동하여 낮은 거짓 양성 비율을 보이는 분산 협동 필터를 제안함

수행기관	개발 목표	개발내용 및 범위

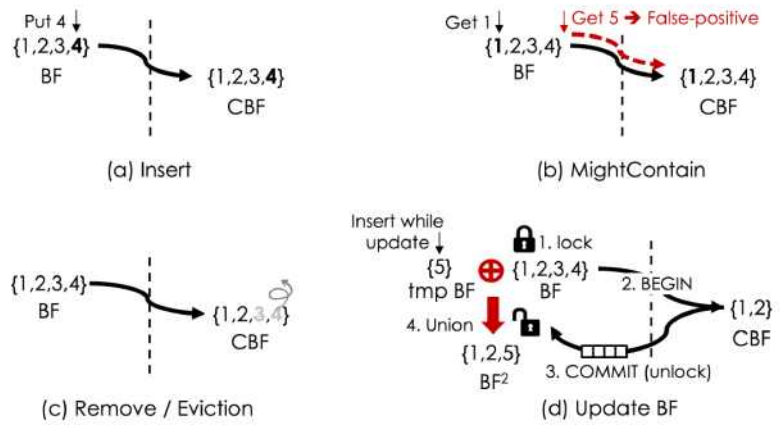


그림 203. 분산 협동 필터의 동작과정

· Put 연산 시에 양쪽의 필터에 키가 추가되며, Invalidate 연산 시에는 백엔드의 카운팅 블룸필터의 값만 제거함. 그리고 주기적으로 카운팅 블룸필터를 블룸 필터로 변환하여 최신본을 프론트엔드로 반영함

· 이때 동시에 블룸필터에 추가되는 키의 누락을 막기 위해 반영본은 임시 블룸필터로 반영되고 그 이후에 기존 블룸필터와 합집합 연산을 통해 새로운 버전의 블룸필터로 버전이 변경됨

- 해시 기반의 고성능 캐시

· 낮은 지연시간을 제공하기 위해 DCC는 해시 충돌 교체 기반의 Approximate LRU 캐시를 사용함. Exclusive 캐시 정책에서 Get 연산은 엔트리를 제거하기 때문에 캐시에서 재참조가 없기 때문에 완전한 LRU 순서를 유지할 필요가 없으며, 타임스탬프를 이용하는 근사 LRU를 사용하기 때문에 inclusive 캐시 정책도 지원할 수 있음. 또, 백엔드의 원격 메모리 풀의 크기를 알고 있기 때문에 정적 해시테이블을 할당하고 리해싱을 피함

· 충돌은 선형 탐사법으로 해시 충돌을 해결하나, 고정된 탐사길이 이내에서 삽입을 하지 못하면 탐사 길이 내에 있는 가장 오래된 버킷과 교체함

- Disaggregated 메모리의 구조

· Dedicated 모드에서는 메모리 전용 서버가 다른 작업을 하지 않기 때문에 폴링 모드를 사용함

· Shared 모드에서는 동료 노드의 유휴 메모리를 사용하기 때문에 원격 메모리 접근 시에 동료 노드의 CPU를 최소한으로 사용해서 동료 노드를 방해하지 않아야 함. 따라서 이벤트 방식과 바쁜 대기 폴링 방식을 혼합해서 사용하는 하이브리드 폴링 방식을 사용함. 하이브리드 폴링은 CPU를 적절하게 적게 사용하면서 폴링 모드에 준하는 성능을 보여줌

- 필터 관리 오버헤드 측정

표 23. 클라이언트 메타데이터에 따른 메모리 풋프린트

	CoBF	BF	Hash Table	RB Tree
Per GB	628 KB	314 KB	4 MB	10 MB

· 표 23은 원격 메모리 1 GB 당 필요로 하는 메타데이터의 메모리 양을 보여줌. DCC는 Two-sided RDMA를 사용하므로 확률적 자료구조를 사용할 수 있음. 0.01 의 거짓 양성 비율을 설정한 경우 Bloom Filter는 314 KB의 메모리를 요구함. CoBF는 버전 업데이트를 위한 임시 버퍼를 추가로 요구하므로 양이 좀더 커짐

· TPC-H 벤치마크를 실행햇을 때 CoBF의 거짓 양성 비율은 40초일 때 18%이며 높지만 동

수행기관	개발 목표	개발내용 및 범위

기화 간격을 5초로 줄이면 3%까지 낮아짐

- Dedicated 모드 (클라이언트-메모리 전용 서버)에서의 PostgreSQL 성능 측정

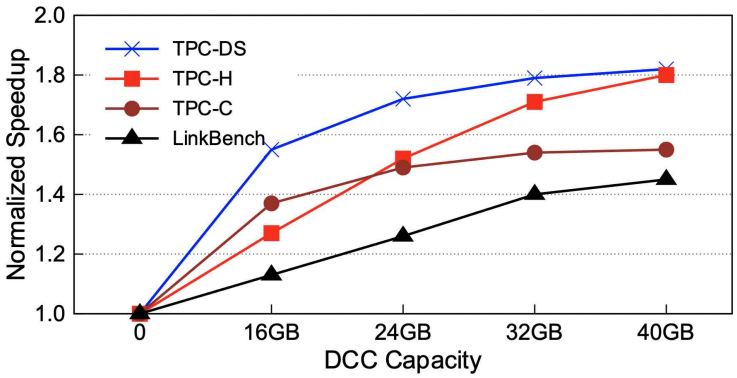


그림 204. PostgreSQL: DCC를 통한 성능 향상

· 그림 204는 다양한 DCC 캐시 크기로 PostgreSQL의 표준화된 속도 향상을 보여줌. TPC-H 및 TPCDS는 총 쿼리 런타임을 측정하고 TPC-C 는 처리량(초당 트랜잭션)을 측정하며 LinkBench 는 초당 요청 수를 측정함

· TPC-H 및 TPC-DS는 대규모 순차 읽기를 수행하는 읽기 집약적 복합 쿼리로 구성되어, DCC 캐시 크기가 증가할수록 성능 향상이 증가함

· TPC-C는 소량의 임의 I/O를 수행하고 쿼리의 2/3만 읽기 연산임에도 여전히 DCC로 인한 성능 향상을 얻음

- Shared 모드 (클라이언트-클라이언트) 성능 측정

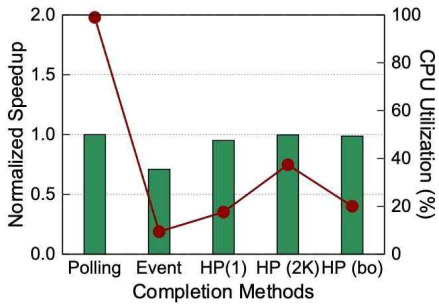


그림 205. 완료 처리 방식에 따른 TPC-H의 쿼리 수행시간

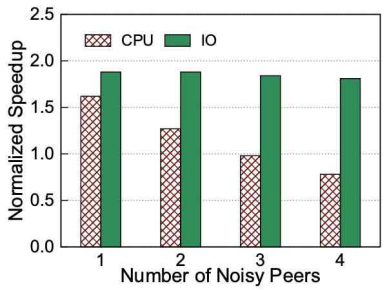


그림 206. Noisy 피어와 함께 수행된 결과

· Shared 모드는 클러스터 내의 노드의 충분히 활용되지 않는 메모리를 원격 메모리로 활용함으로써 메모리 활용률을 높임. 동료 노드의 메모리를 사용하기 때문에 원격 메모리 액세스 시에 CPU를 계속해서 점유하는 폴링 방식을 사용하기 어려움

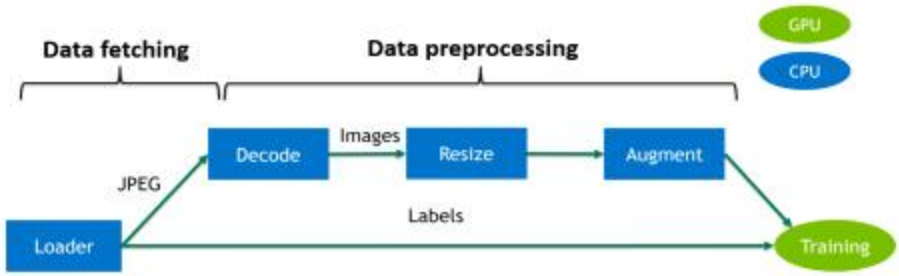
· 왼쪽 그림 205는 완료 처리 방식에 따른 TPC-H의 정규화된 속도향상과 쿼리 수행시간 동안 사용한 동료 노드의 CPU 사용량을 보여주며, 하이브리드 폴링(HP)과 폴링, 이벤트 완료 방식을 비교

· 하이브리드 폴링의 괄호 안에 숫자는 폴링 모드가 수행되는 시간(밀리초)를 나타내고 백오프(bo)는 1 밀리초부터 2048 밀리초 사이에서 폴링 모드의 지속 시간이 유동적으로 변함

· HP(bo)는 폴링에 비해서 2%만 성능이 열등한 반면 20%라는 낮은 CPU 활용률을 보임. 이벤트 방식 보다 11% 더 많은 CPU를 사용하면서 27%의 성능 향상을 보임

· 오른쪽 그림 206는 동료 노드에서 CPU(행렬 곱셈), IO (파일서버) 집약적인 작업이 수행될 때 DCC를 사용하지 않았을 때에 정규화된 TPC-H의 성능을 보여줌

· DCC는 원격 메모리 접근 시에 동료 노드의 CPU를 사용함에도 불구하고 IO 집약적인 작

수행기관	개발 목표	개발내용 및 범위
		업에는 영향을 받지 않음. 하지만 CPU 집약적인 작업이 실행 중일 때는 DCC 데몬이 CPU를 스케줄링 받는 것이 지연될 수 있음 · 그 결과 4 개의 노드에서 모두 CPU 집약적인 작업이 수행되고 있으면 TPC-H의 성능이 DCC를 사용하지 않았을 때보다도 22 % 감소하는 결과를 보임 · 동료 노드의 CPU를 사용하기 어려운 경우는 이를 감지하여 원격 메모리 접근을 방지하는 메커니즘은 본 연구에서 개선해야 할 부분으로 남김
UNIST (공동3)	• 공유 메모리 기반 분산 인공지능 플랫폼 연구	• 공유 메모리 기반 데이터 전처리를 통한 인공지능 학습 성능 향상 연구 - 기술 개발의 필요성 · 본 연구에서는 데이터 전처리가 중요한 이미지 관련 인공지능 모델을 대상으로 동적 데이터 증강 단계를 병렬화하여 전체 DNN 학습 속도를 높임 · 인공지능 학습에 사용되는 원본 이미지는 480x480 크기의 JPEG 이미지로써 이미지 해독 과정(Decoding)을 거쳐야 이미지 처리가 가능함. 이러한 이미지들은 원본 그대로 학습되지 않고 이미지 학습 과정에 해로운 극단적인 아웃라이어(outlier)를 제거하기 위해서 여러 과정을 거침. 기본적인 방법으로는 이미지의 크기를 변경시키고, 색조를 변경하는 등의 이미지 변환 이후에 학습을 위한 크기로 잘라내는(crop) 과정을 거치게 됨 · 이 과정에서 이미지 학습 모델의 학습 능력을 높이기 위해 여러 데이터 증강 기법(data augmentation)이 적용됨. 2019년도 CVPR에 데이터 증강 기법의 매개변수를 자동으로 찾아내는 방법을 발표한 논문인 AutoAugment에 따르면, 전통적으로 이용되는 데이터 기법 조합 2가지를 추가하여 ImageNet을 학습할 때 ResNet-200에서 1.5% 정도 정확도를 올렸고, 2020년도 ICLR에 발표된 AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty에서는 기존 데이터 증강 기법의 조합으로 CIFAR-10-P를 학습했을 때 ResNeXt 모델에서 학습된 모델의 견고성(학습된 데이터 이외의 데이터가 들어왔을 경우 모델의 판별 능력)이 2.6배 정도 향상되는 것을 확인함  그림 207. 기본적인 데이터 로딩 파이프라인 (2019년 NVIDIA GTC 발표 자료) · GPU를 이용한 모델 학습과 학습을 위한 데이터 전처리 과정은 순차적으로 이루어져 있으므로 CPU에서의 데이터 전처리가 늦어지면서 GPU의 활용율이 떨어져 전체 학습 성능 저하로 이어지는 현상을 발견 · 이런 문제를 해결하기 위해 본 연구에서는 CPU-GPU를 통합적으로 사용하는 공유 메모리 기반 DNN 학습 성능 최적화 기법을 개발함 - Manticore: 세분화된(fine-grained) 협력적 병렬화(coorerative-parallel)를 통한 DNN 데이터 로딩 시스템 · DNN 학습 중 CPU의 데이터 준비 대기 시간(data preparation latency)을 숨기기 위해 기존의 DNN 프레임워크(framework)는 CPU에서 동시에 여러 개의 미니 배치(mini-batch)를 사전 처리하는 전략인 Inter-Batch 병렬 처리(Inter-P)를 사용함 · 그러나 위의 실험 결과에서 알 수 있듯이 충분하지 않은 GPU 당 CPU 수나 데이터 처리 파이프라인의 한계로 인하여 데이터 준비 지연이 발생하고, 해당 미니 배치가 학습 시간에 맞춰 준비되지 않는 경우가 많음. 이것은 현재 미니 배치가 필요한 모델 학습의 임박한 반

수행기관	개발 목표	개발내용 및 범위
		복(iteration)을 늦추고 결과적으로 학습 처리량(throughput)을 저하시킴. 또한 지연된 모델 학습으로 인해 GPU가 유휴 상태(idle)일 수 있으며 하드웨어 효율성이 낮아짐 · 다중 GPU 환경에서는 문제가 더욱 악화될 수 있는데 CPU의 병목 현상(bottleneck)으로 인해 GPU 스트래글러(straggler)가 생길 수 있음. 모든 학습 반복(iteration)이 끝날 때 모든 GPU는 다음 반복으로 이동하기 위해 훈련에서 얻은 그라디언트(gradient)와 동기화(synchronize)해야 하고 증강의 동적, 임의적 반영 특성은 각 배치 당 CPU에 다른 양의 로드(load)를 가하기에 결과적으로 GPU의 반복 시간을 변경시킬 수 있음. 그리하여 이상적으로는 GPU가 비슷한 시간에 훈련 반복을 완료해야 하지만 CPU의 계단식 부하 변동(cascaded load variation)으로 인해 GPU 스트래글러가 생성되고 전체 시스템 속도가 느려짐 · 본 연구에서 우리는 CPU와 GPU에 대해 협력적으로 Intra Batch 병렬 처리를 활용하여 데이터 증강 속도를 높이고 균형을 맞추는 DNN 프레임워크인 Manticore를 설계함. Manticore는 세 가지 주요 설계 통찰력을 기반으로 구성됨 · 첫째, Manticore에서는 학습 미니 배치(mini-batch)를 더 작은 타이니 배치(tiny-batch)들로 수직으로 분할하여 CPU 또는 GPU에 타이니 배치들을 할당하여 병렬로 처리하는 Intra-Batch 병렬 처리(Intra-P)를 수행함. 더 작은 배치들을 병렬로 처리하는 Intra-batch 병렬 처리(Intra-P)를 활용하여 기존 데이터 로딩 워크플로(workflow)의 한계를 극복하고 CPU와 GPU 모두에서 데이터 증강 작업을 협력적으로 조정(schedule)할 수 있도록 구현. 임박한 미니 배치에 우선순위를 지정하고 컴퓨팅 자원(resource)을 할당함으로써 선형 속도 향상을 얻을 수 있고 아주 작은 병렬화 오버헤드(overehead)만 발생시킴 · 둘째, Manticore에서는 유휴 GPU 사이클(idle GPU cycle)을 최대한 활용하여 빠른 속도로 타이니 배치(tiny-batch)를 처리하여서 종합적으로 미니 배치(mini-batch) 계산 시간을 줄였다. GPU 커널 구성(kernel construction) 및 메모리 할당에 상당한 오버헤드를 유발할 수 있는 GPU 작업(task) 생성을 모든 타이니 배치마다 하지 않도록 사전(ahead-of-time) 작업 할당을 사용하여서 GPU kernel 구성 및 생성으로 인한 전체 워크플로우의 오버헤드를 감소시킴. 이것은 런타임 스택(runtime stack)을 우회하기 위해 가능한 모든 데이터 증강 파이프라인(pipeline)조합을 미리 생성하고 호스트와 GPU 간의 GPU 메모리 사용량 및 메모리 복사 작업을 최소화함 · 셋째, Manticore는 미니 배치가 먼(remote) GPU에 속하더라도 현재 반복(iteration)의 미니 배치들을 준비 완료할 수 있도록 워커(worker)를 동적으로 조정(schedule)할 수 있도록 시스템을 구성함. 학습은 모든 GPU에서 현재 미니 배치를 완료하지 않고는 다음 반복(iteration)으로 이동할 수 없기에 다음 반복을 위해 미니 배치를 준비하는 것은 도움이 되지 않음. 따라서 Manticore는 GPU에 할당된 로컬(loacl) CPU 워커(worker)들을 재조정(rebalance)하여 글로벌 도메인(global domain)에서 느린 데이터 준비를 촉진시킴 · Manticore는 로컬 매니저(local manager), 글로벌 매니저(global manager) 및 워커(worker)로 구성. 시스템 전반에 걸쳐 효과적인 Intra-Batch 병렬 처리 실행을 위해 이러한 구성 요소를 조정하는 통신 프로토콜(protocol)을 활용함. 들어오는 각 미니 배치(mini-batch)에 대해 로컬 매니저(local manager)는 더 작은 타이니 배치(tiny-batch)로 나누어 Intra-Batch 병렬 처리를 할 수 있도록 하고 작업 스케줄링(scheduling)을 수행 · 글로벌 매니저(global manager)는 로컬 매니저(local manager)의 상태를 추적하고 진행 상황을 조정. 글로벌 매니저의 주요 임무는 동적 리소스 재배치를 통해 데이터 병렬 학습에서 스트래글러(straggler)를 완화. 반복(iteration) 중에 스트래글러가 식별되면 글로벌 매니저는 스트래글러 영향(straggler effect)을 완화하기 위해 점진적으로 작업자를 재배치. 진행 속도가 느린 로컬 매니저에는 반복이 끝날 때까지 보류 중인 작업을 추가로 병렬화하기 위해 더 많은 워커가 부여됨. Manticore 시스템에서는 이러한 작업자 증강의 경우, 오예측의 위험을 없애는 방향으로 워커 스케일링(worker scaling)을 수행하고 워커를 재배치함

수행기관	개발 목표	개발내용 및 범위

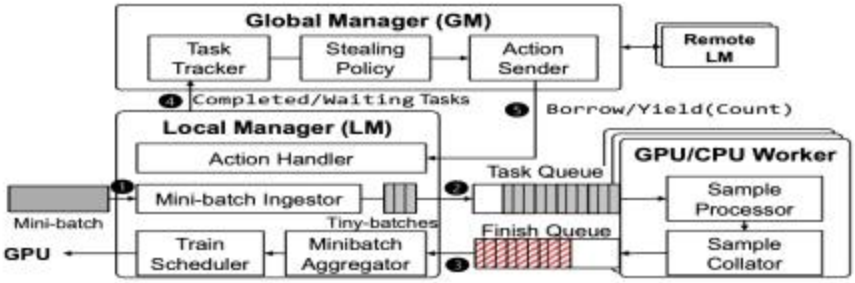


그림 208. Manticore 시스템 구조

· 그림 208은 Manticore 시스템의 구조를 도식화 한 것임. 가장 먼저 DNN 프레임워크에서 정의한 순서에 따라 미니 배치들을 하나씩 받아오고(1번) 세분화된 Intra-P 작업을 생성하여 워커들에게 전달하고(2번) 처리된 작업 된 미니 배치로 집계하여 현재 반복에 대한 모델 학습을 시작함(3번). 로컬 매니저는 각각의 다른 로컬 매니저와 유사한 속도로 반복 위해 CPU 워커들을 확장하거나 축소하는 작업 처리기(action handler)가 있음. 이를 용이하게 하기 위해 각각의 로컬 매니저는 주기적으로 글로벌 매니저에 Intra-Batch 병렬 작업(task) 처리에 대한 진행 상황을 보고하고(4번) 적당한 워커 확장에 대한 지침을 받음(5번)

· 작업 처리에 대해서는 GPU를 충분히 활용할 수 있도록 적응형 작업 스케줄링(adaptive task scheduling)을 사용. 이러한 적응형 작업 스케줄링에 대한 이해를 돕기 위하서 다음 그림 209을 통하여 스케줄링이 어떤 식으로 일어나는지 시나리오를 구현. 적응형 작업 스케줄링은 다음과 같이 일정량의 타이니 배치가 남아 있을 경우, 해당 타이니 배치들을 빠르게 GPU로 처리. 그림 209에서 볼 수 있듯이 만약 기준 타이니 배치의 수가 5개라면, 처리해야 하는 남은 배치(그림 209에서의 59로 표기된 배치)가 5개 미만으로 남았을 경우, 남은 타이니 배치들은 GPU 작업자로 할당한다. 이를 통하여 빠르게 처리할 수 있는 하드웨어 자원을 최대한으로 활용함

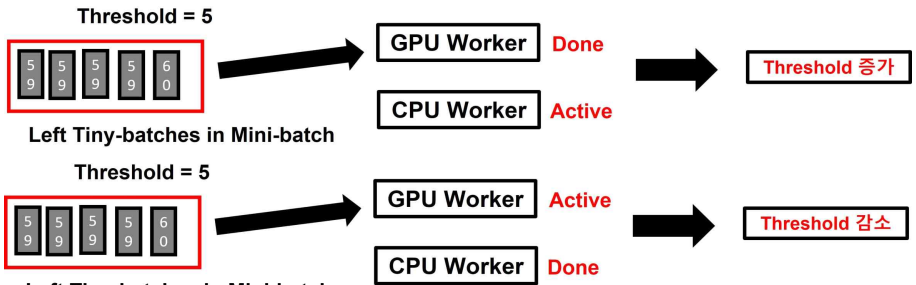


그림 209. 적응형 작업 스케줄링(adaptive task scheduling)의 구조도

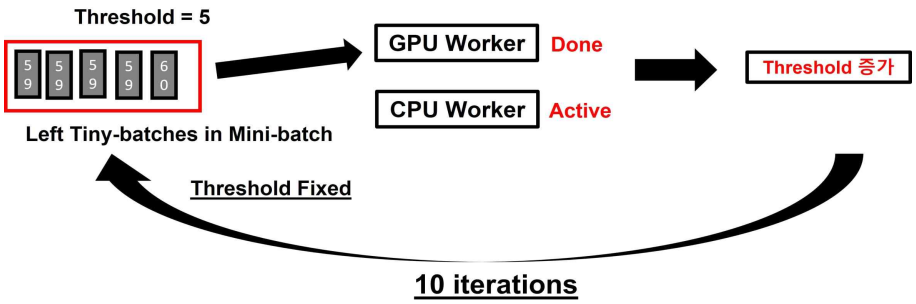


그림 210. 적응형 작업 스케줄링(adaptive task scheduling)의 Manticore 이식도

· 이러한 GPU 오프로드 잔여 작업 기준(remaining task threshold)을 전체 인공지능 모델 학습 시간 동안 매 학습 반복 (iteration)마다 수정하는 것은 오히려 잔여 작업 기준의 잦

은 변화로 인한 학습 성능의 손해를 일으킬 수 있음. Manticore에서는 전체 학습 중 초기 반복에서 잡은 잔여 작업 기준을 그대로 유지함으로써 이러한 문제를 해결함.

- Manticore 시스템의 성능 평가 결과

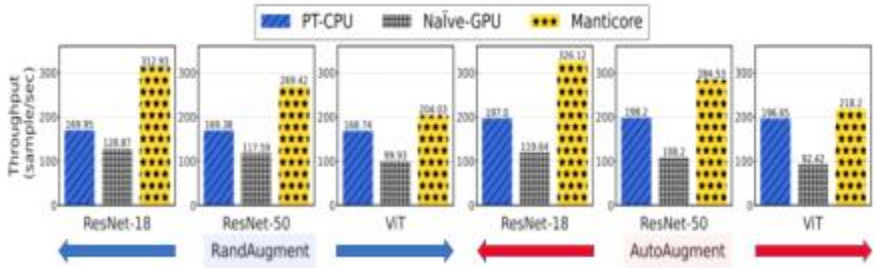


그림 211. 단일 GPU에서 ResNet-18, ResNet-50 및 ViT 학습의 처리량. 왼쪽 3개 및 오른쪽 3개 그래프는 각각 RandAugment 및 AutoAugment에서 처리량

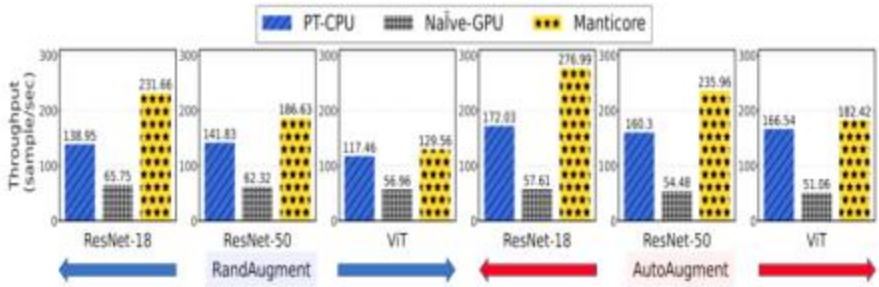


그림 212. 데이터 병렬 처리에서 6개의 GPU에서 ResNet-18, ResNet-50 및 ViT 학습의 처리량

· 그림 211는 서로 다른 모델과 데이터 증강 알고리즘에 대한 세 가지 병렬 처리 방법의 처리량을 보여줌. 비교의 용이성을 위해 처리량(throughput)은 단일 GPU의 처리량(samples/sec)으로 표기. Manticore와 비교하기 위한 두 개의 비교군은 다음과 같음

· PT-CPU는 CPU에서 iner-batch 병렬화를 수행하는 PyTorch의 기본 방법으로 본 연구에서 베이스라인(baseline)으로 삼은 방법. Naive-GPU의 경우, GPU에서 intra-batch 병렬화를 유휴 로컬 GPU 사이클을 이용하여 수행하지만, 사전 작업 할당(ahead-of-time task allocation)이나 GPU 내에서 처리된 작업의 모델 학습 직접 공급(DIM, direct input materialization) 등의 몇 가지 Manticore의 주요 최적화 기술을 사용하지 않은 상태를 보여줌

· 그림 211에서 Manticore는 모든 경우에 대해서 PT-CPU와 Naive-GPU를 능가하는 것을 볼 수 있음. 특히 Manticore는 ResNet-18 모델에 대해서 PT-CPU와 Naive-GPU보다 각각 65.6~84.1% 및 140.9~172.5% 더 높은 처리량을 달성하면서 가장 큰 성능의 향상을 보임. 이와 같은 높은 성능 향상 폭은 ResNet-18의 상대적으로 가벼운 연산 학습량으로 인한 충분한 GPU 사이클(cycle)을 얻을 수 있었기 때문이지만, 연산 학습량이 큰 ViT 모델 학습에서도 Manticore는 RandAugment와 AutoAugment에 대해서 PT-CPU보다 각각 20.9%와 10.9% 더 높은 처리량(throughput)을 달성. 이는 GPU 연산 학습량이 높은 모델에 대해서도 Manticore는 충분한 성능 향상을 보이는 것임

· 해당 데이터에서 Naive-GPU의 경우 Manticore뿐만 아니라, PT-CPU보다 성능이 떨어지는데, 이는 단지 GPU의 유휴 사이클을 활용하는 것이 아니라 GPU와 CPU를 잘 고려한 시스템의 필요성을 보여주며, 사전 작업 할당과 같은 Manticore의 주요 기술의 효율성을 보여줌

· 그림 212은 동일한 실험을 멀티 GPU(multi GPU) 환경에서 진행한 것. 멀티 GPU를 활용

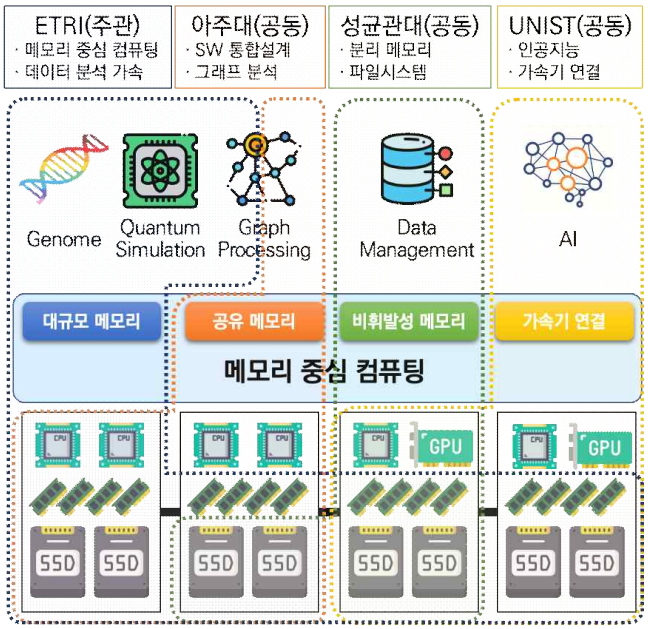
수행기관	개발 목표	개발내용 및 범위
		한 인공지능 모델의 학습 중 데이터 병렬화(data parallelism)의 경우, 모든 학습의 반복(iteration)마다 업데이트된 모델을 동기화(synchronization)하기 위해서 학습된 그래디언트(gradient) 교환이 필요한 분산 학습 · 그림 212에서는 6개의 동종 GPU(homogeneous GPU) 환경에서 시스템의 처리량을 비교. PT-CPU 및 Naive-GPU와 비교하여 Manticore는 RandAugment 와 AutoAugment 모든 경우에서 높은 처리량을 달성함. PT-CPU와 비교하여 Manticore는 두 데이터 증강 기법에 대해서 평균적으로 ResNet-18 모델의 경우 63.8%, ResNet-50 모델의 경우 39.3%, ViT 모델에서는 9.9% 더 높은 처리량을 보여줌

ETRI (주관)
· 대규모 메모리 중심 컴퓨팅 핵심 기술 연구
· 메모리 중심 컴퓨팅 기반 소프트웨어 통합 설계 기술 연구
김강호

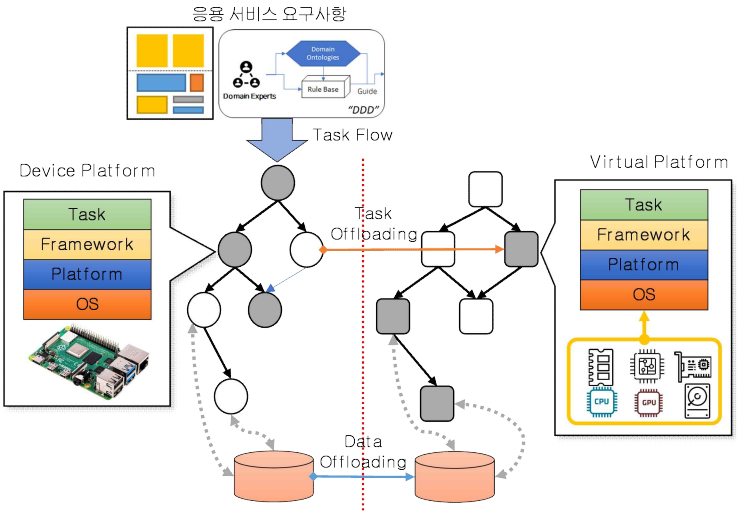
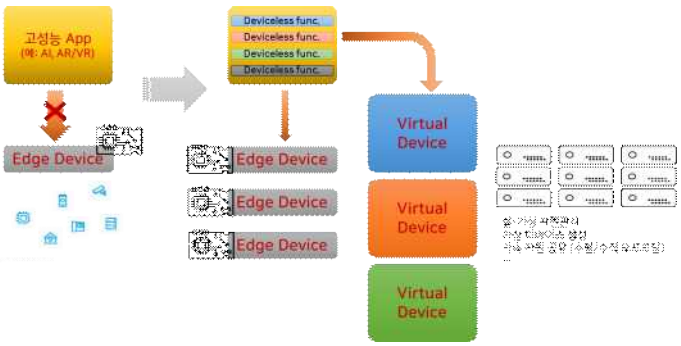
아주대학교 (공동1)
· 메모리 중심 컴퓨팅 시스템을 위한 기반 소프트웨어 기술 연구

성균관대학교 (공동2)
· 비휘발성 메모리 기반 스케일러블 인메모리 저장소 연구

UNIST (공동3)
· 메모리 중심 컴퓨팅 기반 인공지능 플랫폼 연구



[EdgeCPS]

수행기관	개발 목표	개발내용 및 범위
ETRI (주관)	③ 저저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 핵심 기술 개발	**• 디바이스의 자원 증강을 지원하는 응용 실행환경 가상화 기술 연구** - 기술 개요 · 연구 정의 : 저자원 디바이스 단말의 연산 성능 한계극복을 위해 가상화된 공간에서 가상 디바이스를 생성하고 자원 공유의 탄력성을 제공하여 학습/추론 등의 고성능 AI 연산을 가능하게 하는 디바이스리스 컴퓨팅을 제공하는 가상화 프레임워크 기술 연구  그림 213. 디바이스리스 컴퓨팅 동작 개념도 - 연구 범위 및 당해연도 목표 (1) 디바이스리스 컴퓨팅을 위한 가상 디바이스 생성과 응용 실행 환경을 제공하는 디바이스리스 컴퓨팅 프레임워크 설계 및 디바이스 관리 기술 개발 (2) 자원 성능을 반영하는 중앙집중식 디바이스 관리 기법 및 실-가상 데이터 동기화 기술 연구 (3) 실행 환경(실/가상) 변화에 따른 적용형 디바이스리스 SW 구조 및 관리 기술 연구 - 세부 개발 내용 (1) 디바이스리스 컴퓨팅 실현을 위한 디바이스 가상화 프레임워크 기술 연구 . 가상 디바이스 기반의 컴퓨팅 연산 증강을 지원하는 프레임워크 구조 및 기능 정의 : 온디맨드 방식의 가상화 디바이스 생성과 오프로딩 기법을 활용한 연산 증강 수행  그림 214. 디바이스리스 컴퓨팅의 실-가상 디바이스 환경 . 디바이스리스 컴퓨팅 실현을 위한 연구 기술 선정과 요구 사항 도출 : 디바이스리스 컴퓨팅 실현을 위한 인프라 시스템 기술, 디바이스 자원 활용 지원 기술 및 응용 수행시 적응형 연산 증강을 지원할 수 있는 SW 구조에 대한 기능 정의

수행기관	개발 목표	개발내용 및 범위

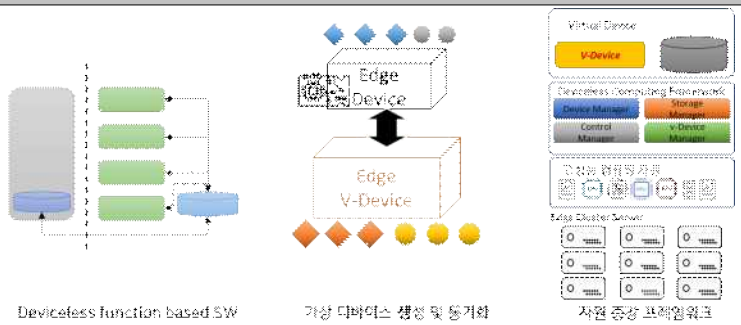


그림 215. 디바이스리스 컴퓨팅 세부 연구 영역

. 프레임워크 기반의 디바이스리스 컴퓨팅의 증강 시나리오 설정 및 개별 매니저간 연관 기술 연구 : 디바이스 가상화 과정, 응용 실행환경 생성 및 응용 실행 제어에 대한 시나리오 정의와 개변 매니저들 간의 연관 관계 수립

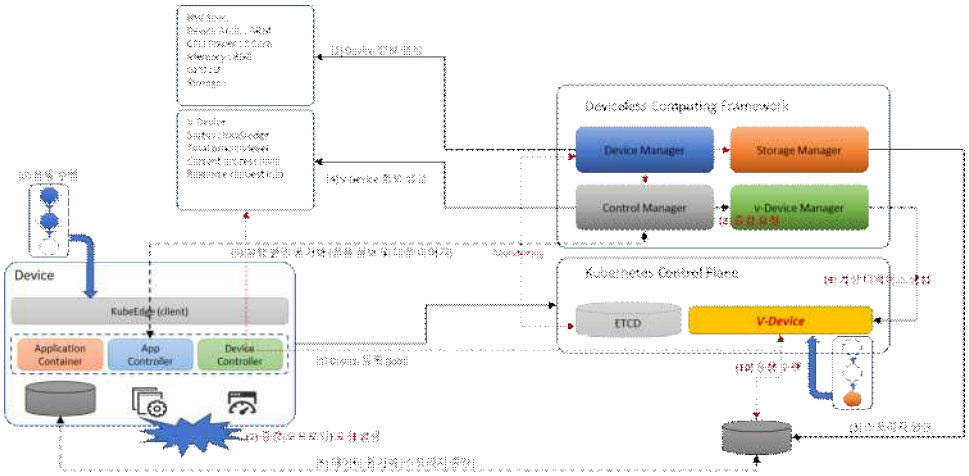


그림 216. 디바이스리스 프레임워크 정의 및 동작 과정

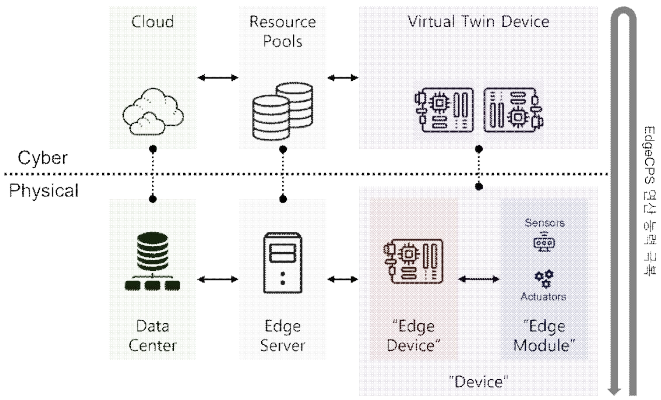
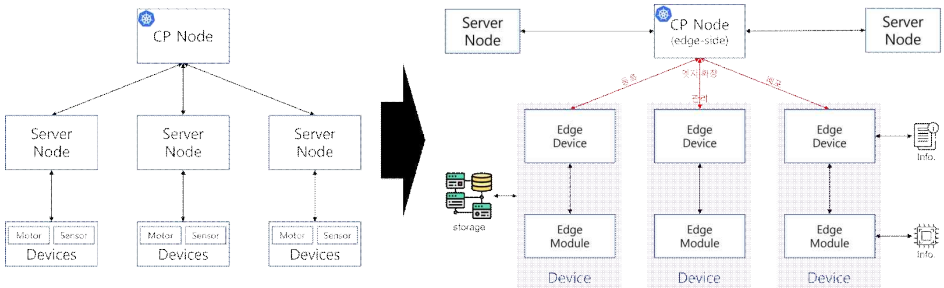
. 개별 매니저는 프레임워크는 기능에 따라 마이크로서비스 형태로 개발되며, 각 서비스들은 개별적인 역할 수행과 함께, 서비스간의 제어 관리를 통해 연동되어 서버 자원의 최적 활용을 지원함

표 24. 디바이스리스 컴퓨팅 프레임워크 필수 매니저 및 역할

매니저 이름	기능	실 디바이스 연동 정보
Device Manager	증강 요청을 위한 실 디바이스 정보 관리	실 디바이스 정적/동적 정보
Storage Manager	실 디바이스 스토리지 증강을 위한 가상 스토리지 생성 관리	스토리지 마운트 정보 및 동기화 요청 데이터
Controll Manager	연산 증강 지원을 위한 디바이스 응용 연산 증강 실행 관리	실행 응용의 워크플로우 및 실시간 상태 정보
V-Device Manager	가상 디바이스를 통한 실행환경 라이프사이클(생성/동작/삭제) 관리	디바이스 및 응용 정보를 활용한 가상 장치 생성 요청 정보

(2) 실-가상 디바이스 자원 관리 및 동기화 기술 연구

가. 실 디바이스의 성능 및 기능 특성에 따른 디바이스 계층 구조 분석과 용어 정의

수행기관	개발 목표	개발내용 및 범위
		. 엣지 환경에 연결되는 단말 장치는 성능적, 기능적으로 서로 다르므로, 체계적이고 효율적인 관리를 위해 장치별 속성에 따른 계층적 분리가 필요함 . 스케줄러가 포함된 리눅스 기반의 운영체제(OS)가 동작 가능한 소형 임베디드 보드는 다양한 응용 프로그램이 동시에 동작 가능한 상대적으로 좋은 성능의 장치로 분류되며, 디바이스리스 컴퓨팅 프레임워크에서는 이를 "엣지디바이스"로 정의 . 센서나 엑츄에이터 등은 매우 한정적인 기능만을 가지며 주로 "엣지디바이스"에 종속적인 장치로써, 디바이스리스 컴퓨팅 프레임워크에서는 이를 "엣지모듈"로 정의 . 엣지서버와의 연동 및 효율적 관리와 용어의 함의를 위해 "엣지디바이스"와 "엣지모듈"을 "디바이스"로 정의하여 추상화를 진행  그림 217. 디바이스리스 컴퓨팅 프레임워크의 계층적 장치 정의 나. 클라우드-엣지 컴퓨팅 모델 확장을 통한 엣지 디바이스 종속 모듈 관리 기술 연구 . IoT 장치 등의 저자원 엣지 다비이스와 디바이스 종속 모듈 관리 위해 당해년도에는 컨테이너 기반의 클라우드 환경에서 다루고 있는 디바이스 자원 관리 기술(IoT 장치별 등록/연결 관리)을 엣지 컴퓨팅 환경에 맞게 적용하는 연구를 진행하였음 . 증강을 지원하는 클러스터에 클라우드 환경에서 제공하는 개별 디바이스 온라인/오프라인 상태 정보와 클라우드와 동일한 인터페이스의 관리 및 서비스 배포 환경을 구축하였음  그림 218. 엣지 컴퓨팅 기반 디바이스 관리 구조 다. 엣지 디바이스 연계와 종송 모듈 관리를 위한 모듈 데이터 동기화 기술 개발 . 엣지 모듈은 기능적 특성에 따라 엣지 디바이스에 종속적이며 단발성, 휘발성의 데이터 기반 동작을 수행함 . 생성된 데이터의 전달 및 관리를 엣지 디바이스에 위임하고 성능 증강 시 필요한 정보로 재활용할 수 있도록 가상화된 트윈 데이터의 동기화 기술 연구를 통해 개발 자원들에 대한 통합 관리 및 동기화 구조를 개발하였음

수행기관	개발 목표	개발내용 및 범위
		. 동기화된 정보는 엣지 서버에 위치한 관리 모듈을 통해 중앙집중식 통합 모니터링이 가능함

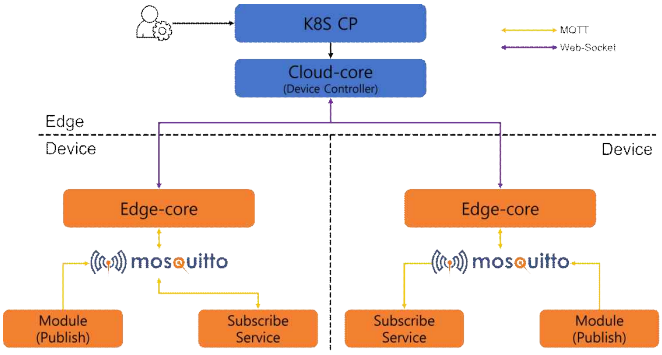


그림 219. 가상 디바이스 데이터 동기화 구조

(3) 디바이스리스 컴퓨팅 지원을 위한 적용형 SW 구조 및 관리 연구

가. 디바이스리스 컴퓨팅 프레임워크向 AI 응용 개발

. 엣지 디바이스와 가상화 디바이스가 동일한 AI 모델을 공유하고, AI 모델의 추론을 수행할 디바이스를 실시간으로 결정할 수 있는 제어 관리자 구조 설계

. AI 모델은 복수 개의 부분 모델로 분할하여 각각의 디바이스에서 실행 및 속개 가능

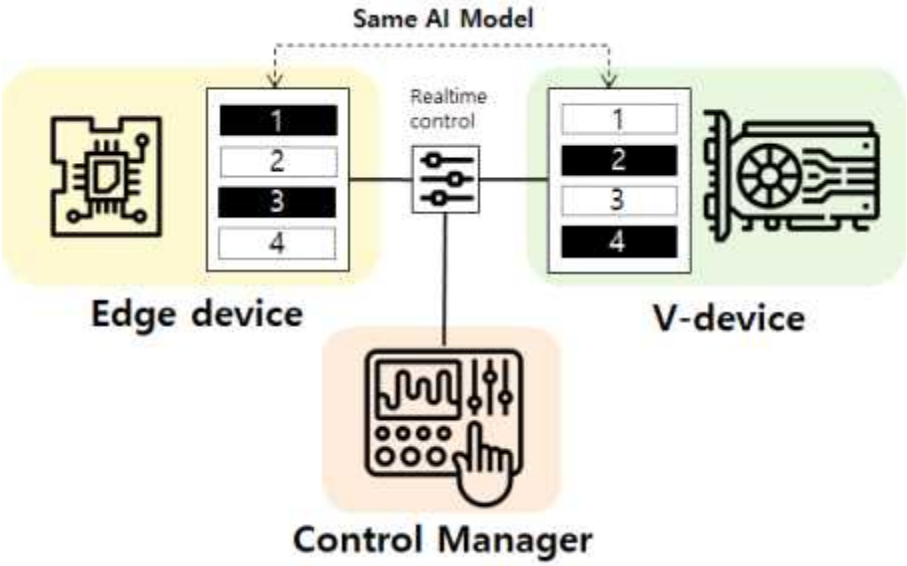


그림 220. 엣지 디바이스와 가상화 디바이스 및 제어 관리자를 이용한 AI 모델 공동 추론 방법론 개념도

. AI 모델 추론을 위한 코드(프로그래밍 언어, AI 모델 추론용 라이브러리 등)는 엣지 디바이스와 가상화 디바이스가 최대한 동일한 것을 사용하여 엣지 디바이스 기반으로 응용을 개발한 사용자가 쉽게 자신의 응용을 적용할 수 있도록 설계

. 제어 관리자는 현재 응용의 진행 상황을 모니터링하면서 디바이스의 현재 상태 등의 추가 정보를 이용하여 AI 모델을 어느 디바이스에서 수행할지를 실시간으로 판단 및 결정

. 디바이스리스 컴퓨팅 프레임워크에서 디바이스의 소비전력 감소, 단위시간 내 최대 응용 활성화 등 별도의 외부 정책을 설정하여 제어 관리자를 통해 응용 수행을 제어할 수 있도록 설계

나. 엣지 디바이스와 가상화 디바이스 간 데이터 교류를 위한 메타데이터 구조 제시

수행기관	개발 목표	개발내용 및 범위

. 엣지 디바이스와 가상화 디바이스가 공동으로 AI 모델의 추론을 수행할 때 제어 관리자가 효율적으로 응용의 진행 상황을 파악할 수 있도록 메타데이터를 구성

. 메타데이터에는 디바이스 ID 등의 정보 외에 AI 모델 추론에 소요된 시간, 현재 AI 응용 수행 진척도 등이 포함됨

표 25. control manager의 주요 메타데이터

이름	형식	의미
block_number	int	현재 응용 수행 진척도
network_name	string	추론을 수행 중인 네트워크 정보
model_file_name	string	실제 딥러닝 모델 파일명
model_file_size	float	실제 딥러닝 모델 크기
input_layer_name	string	입력 레이어 이름
input_data_shape	array<int>	입력 데이터 차원 정보
output_layer_name	string	출력 레이어 이름
output_data_shape	array<int>	출력 데이터 차원 정보
use_cuda	bool	가속기 (nvidia gpu) 사용 여부
architecture	string	CPU 구조 (x64, arm64 등)
elapsed_inference	double	추론 소요 시간
elapsed_communication	double	통신 소요 시간
offloaded	bool	오프로딩 여부
cpu_usage	int	디바이스의 cpu 사용량
memory_usage	int	디바이스의 메모리 사용량

. 엣지 디바이스와 가상화 디바이스는 정의된 메타데이터 규격을 준수하여 데이터를 주고받으며, 제어 관리자는 이 메타데이터를 확인하여 현재 AI 응용 서비스가 사용자의 의도 또는 희망하는 사양에 맞게 작동하고 있는지를 파악할 수 있고, 이를 기반으로 가상화 디바이스의 증강 자원 사용량 증대 또는 축소를 요구할 수 있음

- 연구 개발 결과

(1) 디바이스리스 컴퓨팅 실현을 위한 디바이스 가상화 프레임워크 기술

. 디바이스와 서버간의 프레임워크 구성 및 매니저 기능 설계: 엣지 디바이스는 엣지 서버와 연동되며, API gateway를 통해 디바이스 등록, 스토리지 생성, 응용 실행환경 정의 및 가상 디바이스 생성등의 기능을 수행하며, 각 기능들은 프레임워크의 담담 매니저를 통해 동작됨

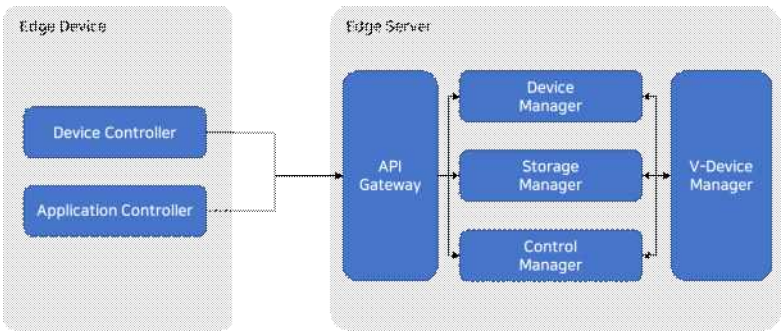


그림 221. control manager의 주요 메타데이터

수행기관	개발 목표	개발내용 및 범위

. 디바이스 매니저 개발 : 개별 매니저들은 마이크로서비스 형태로 생성되며 Restful API를 통해 정보 전달 및 동작을 수행하며, 디바이스 매니터에서는 가상 디바이스 생성을 위한 디바이스들의 정적 정보를 관리함

Restful API (URI)	TYPE	설명
/devices/	POST	새로운 엣지 디바이스 입력
/devices/	GET	등록된 전체 엣지 디바이스 리스트
/devices/id	GET	엣지 디바이스 정적 정보 읽기
/devices/name	GET	이름을 통한 엣지 디바이스 리스트 읽기
/devices/id	PUT	엣지 디바이스 정보 수정
/devices/id/	REMOVE	엣지 디바이스 삭제

그림 222. 디바이스 매니저의 디바이스 정적 정보 관리

. 디바이스의 정적 정보를 기반한 가상 디바이스 생성 기능 개발 : 엣지 서버에서는 현재 개별 디바이스 정보를 서버 클러스터 내부의 노드 단위로 관리가 가능하며, 컨테이너 인프라 설치가 어려움 디바이스는 별도의 연동 관리 기능을 적용할 수 있도록 개발 예정임

그림 223. 디바이스 실시간 온라인/오프라인 상태 확인

(2) 실-가상 디바이스 자원 관리 및 동기화 기술 연구

. 디바이스의 종송 모듈 관리 지원을 위해 KubeEdge 기반의 엣지 서버 관리 환경 설정하고 엣지 디바이스에 등록된 자원에 대한 중앙 통제 관리 기술을 개발 하였음

. 디바이스에 종속된 엣지 모듈은 엣지 코어로 MQTT를 통한 데이터를 전달하고, 클라우드 코어 간 웹소켓 통신을 통해 데이터를 중앙집중식으로 통합 관리 함

. 다양한 형태의 엣지 모듈에 대응하기 위해 string 형태의 자유로운 특성 데이터 반영 가능성 확보

. ReadOnly, ReadWrite 모드를 통한 양방향 데이터 연동 기능 제공

수행기관	개발 목표	개발내용 및 범위

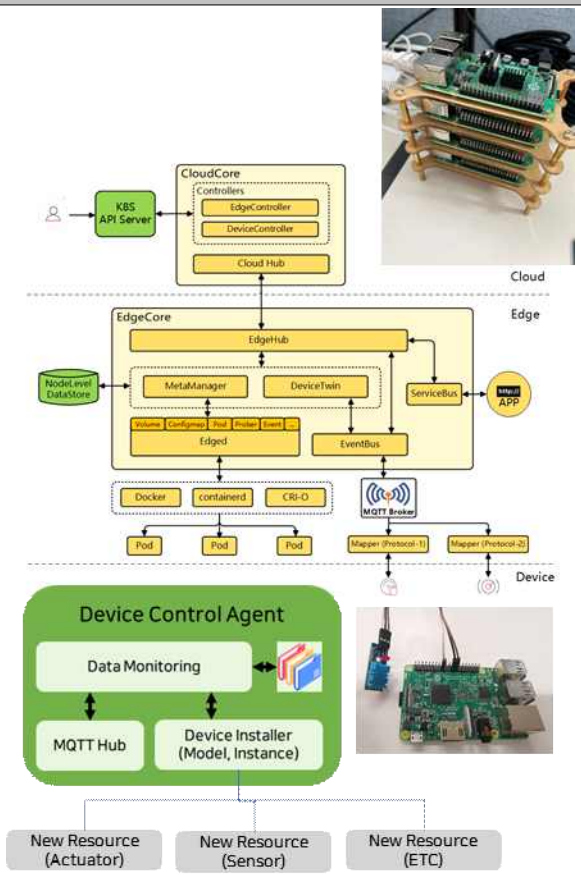


그림 224. 개별 자원에 대한 데이터 동기화 및 관리

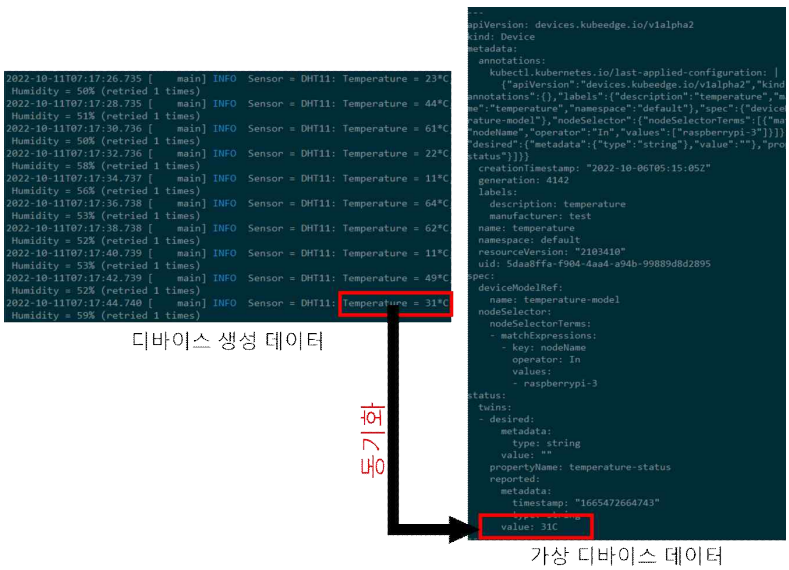


그림 225. 엣지 모듈 실시간 데이터 동기화 확인

(3) 디바이스리스 컴퓨팅 지원을 위한 적용형 SW 구조 및 관리 연구

. 디바이스리스 컴퓨팅 프레임워크向 AI 응용 프로토타입 개발 및 성능 평가 수행

. 2개의 서로 다른 Architecture (x64 & aarch64)에서 동일한 딥러닝 추론 코드 (ONNXRuntime 기반)를 이용하여 공동으로 AI 모델(YOLOv5n)을 추론하고, 실시간으로 추론 디바이스를 조정할 수 있는 컨테이너 기반 어플리케이션 개발

. AI 모델을 분리 후, 분리된 각 부분 모델의 오프로딩 여부에 따른 응용 수행 시간

수행기관	개발 목표	개발내용 및 범위

및 aarch64 디바이스의 CPU 사용량 측정

. 오프로딩 정도에 따라 응용 수행 시간 및 aarch64 디바이스의 CPU 사용량이 감소하는 것을 확인하였음. 응용 수행 시간에 있어 부분 모델의 FLOPs에 따라 그 Gain이 커지며, 통신 환경에 영향을 받는 것으로 확인됨. aarch64 CPU 사용량도 오프로딩 정도에 따라 감소하지만 FLOPs에 의한 영향은 적은 것으로 판단됨

표 26. 실험에 사용한 YOLOv5n 모델 정보

	Block 0	Block 1	Block 2	Block 3
레이어 수 (개)	5	25	26	622
파라미터 수 (개)	1,744	27,840	28,928	1,809,188
계산량 (GFLOPs)	0.09	0.18	0.09	0.76
입력 데이터 차원	3×320×320	16×160×160	64×40×40	64×40×40
출력 데이터 차원	16×160×160	64×40×40	64×40×40	6300×85

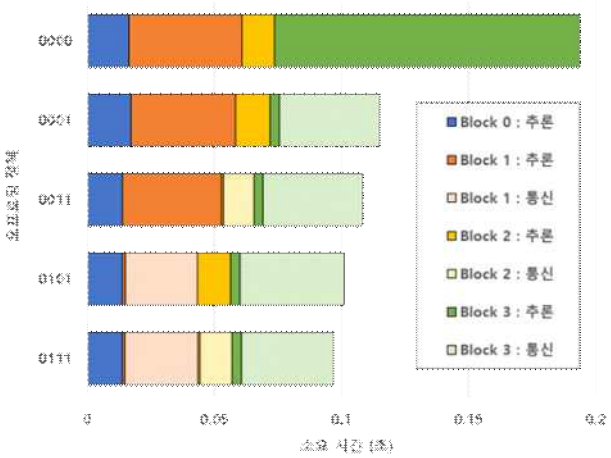


그림 226. 오프로딩 정책에 따른 응용 수행 시간 측정 결과 (0:일반, 1:오프로딩)

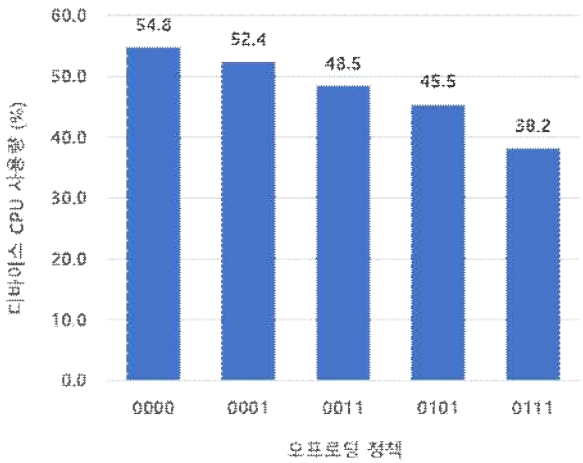


그림 227. 오프로딩 정책에 따른 aarch64 디바이스의 CPU 사용량 측정 결과

- 향후 연구 계획

수행기관	개발 목표	개발내용 및 범위
		(1) 디바이스리스 컴퓨팅 실현을 위한 디바이스 가상화 프레임워크 기술 연구 . 실 디바이스의 상태 정보에 따른 가상 디바이스 생성 및 최적 서버 배치를 위한 스케줄링 정책 기술 개발 (2) 실-가상 디바이스 자원 관리 및 동기화 기술 연구 . 이기종 환경을 위한 다중 멀티 디바이스 연동과 대단위 가상 자원 관리를 지원하는 실시간 동기화 기술 연구 (3) 디바이스리스 컴퓨팅 지원을 위한 적용형 SW 구조 및 관리 연구 . 실시간으로 수집되는 정보를 추가적으로 활용하고, 다수의 응용을 동시에 제어할 수 있는 컨테이너 기반 제어 관리자 어플리케이션 개발 및 디바이스리스 컴퓨팅 프레임워크 적용 **• 다중 노드 기반의 GPU 자원 공유 기술 연구** - 기술 개요 · 연구 정의 : 다수의 디바이스에서 한정된 서버 가속 HW 들을 동시에 활용하기 위해서는 가속 자원에 대한 공유 기술과 병목 발생을 해소하는 스케줄링 기술 연구가 필요하며, 본 연구에서는 대표적인 HW 가속 장치인 GPU 자원 활용의 제한적인 문제를 해결하여 다중 노드 기반의 GPU 자원을 공유하기 위한 프레임워크를 연구함 - 당해연도 연구 목표 및 범위 (1) 다중 노드 기반의 GPU 자원 공유 기술 연구 방향 수립 (2) 다중 노드 기반의 GPU 가속기 공유모델 설계 (3) 다중 노드 기반의 GPU 가속기의 효율적인 공유 기술 연구 (4) 멀티 GPU 환경의 가상 메모리 관리를 통한 마이그레이션 기법 개발 - 세부 연구 개발 내용 (1) 다중 노드 기반의 GPU 자원 공유 기술 연구 방향 수립 . EdgeCPS 플랫폼에서는 다양한 HW 자원들의 가상화를 통해 저자원 디바이스의 성능 한계를 극복하도록 지원하고 있으며, GPU는 대표적인 HW 가속 장치로 저자원 디바이스에 대한 대표적인 증강 자원임 . 엣지 컴퓨팅을 수행하는 엣지 컴퓨팅 서버에서 대규모 마이크로 서비스를 수행하기 위해서 엣지 컴퓨팅 서버의 GPU 자원의 가상화가 필요함 . 한정된 자원을 가진 엣지 컴퓨팅 서버에서 높은 수준의 GPU 가속기 자원의 활용률과 분산 처리 능력을 제공하기 위하여 다중 노드로 구성된 클러스터에서 GPU 가속기 자원을 실시간으로 파악하고 관리하는 연구가 필요함 . 기존 클라우드 환경에서는 GPU가 탑재된 노드에 직접 가상 머신 혹은 컨테이너를 할당하는 방식을 통해 GPU 자원의 활용이 제한적인 문제가 있음 . 다중 노드로 구성된 클러스터 환경에서 특정 노드의 GPU를 활용하기 위한 자원 가상화 및 공유모델을 지원하는 프레임워크 개발이 필요함 (2) 다중 노드 기반의 GPU 가속기 자원 공유모델 설계 . 2차년도에서는 마이크로 서비스가 수행 될 서버의 GPU 가속기 자원을 여러 응용이 공유할 수 있는 자원 공유 모델을 설계하였음 . 다중 노드 기반의 GPU 가속기 공유모델 동작 시나리오는 다음과 같음 - 마이크로 서비스의 자원 요구사항 수집 - 현재 클러스터에서 가용한 GPU 가속기 자원 수집 - 요구사항 분석을 통한 GPU 자원 파티셔닝 정보 생성 - GPU 자원 공유를 위한 마이크로 서비스에게 필요한 자원 요구사항 정보 생성 - GPU 자원 공유 모델 생성(GPU 가상화)

수행기관	개발 목표	개발내용 및 범위

- 마이크로 서비스를 가상화된 GPU 자원에 할당하여 수행

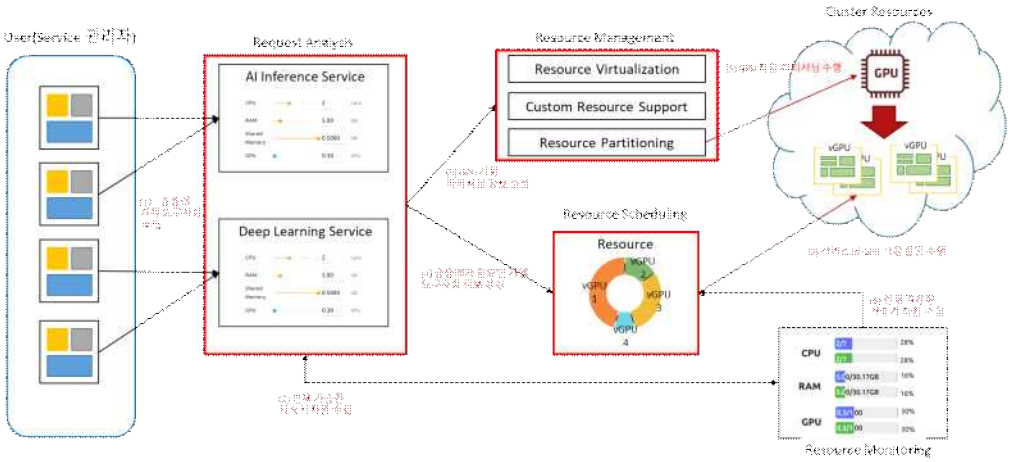


그림 228. 다중 노드에 대한 자원 공유 모델 동작

. Request Request Analysis
- GPU 가속기 자원의 공유모델을 만들기 위한 요구사항 수집/분석 모듈
- 마이크로 서비스의 자원 요구사항 정보 생성
- 요구사항 분석을 통한 GPU 가속기 자원 공유모델 생성

. Resource Management
- EdgeCPS 서버 GPU 가속기 자원 관리 모듈
- GPU 가속기 자원 가상화 및 관리
- GPU 가속기 자원 파티셔닝 수행
- 자원 활용률을 높이기 위한 GPU 가속기 자원의 논리적인 파티션 수행 및 관리

. Resource Scheduling
- EdgeCPS서버 GPU 가속기 자원 할당 스케줄링
- 마이크로 서비스에 대한 가상화된 GPU 가속기 자원 할당
- 클러스터 내의 GPU 가속기 자원 스케줄링

(3) GPU 가속기의 효율적인 공유 기술 연구

. GPU 자원 사용률을 높이기 위한 자원 스케줄링 기술 개발

. 최대한 많은 응용을 동시에 수행시킬 수 있도록 GPU 자원 할당이 목표

. 클러스터로 구성된 다중 노드의 GPU 자원에 대한 복합 공유 (HW지원 +_SW 지원) 가상화 기술

. MIG를 지원하는 환경에서는 동적 구성 런타임을 통해 GPU 파티셔닝 기능을 동적으로 수행하도록 지원

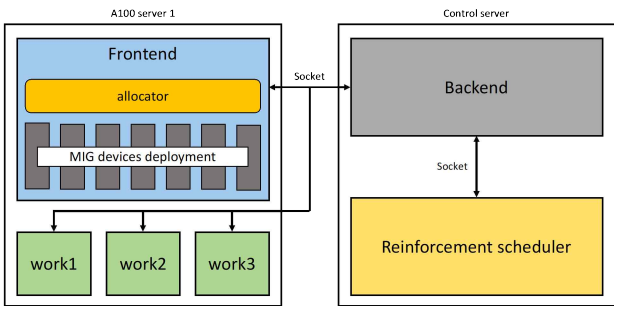


그림 229. MIG 동적 구성 지원 런타임

(4) 멀티 GPU 환경의 가상 메모리 관리를 통한 마이그레이션 기법 개발

수행기관	개발 목표	개발내용 및 범위
		. low-Level GPU 가상 메모리 관리를 이용하여 동종/이종 및 MIG 간 등 다양한 환경의 멀티 GPU 환경에서 마이그레이션 개발을 진행함 . 동작의 원리는 타셍 GPU의 가상 주소를 원본 GPU와 동일하게 설정함으로써 마이그레이션이 동작하도록 개발하였으며, 새로운 CUDA 가상 메모리 라이브러리를 사용하여 타겟 GPU 메모리를 할당/복사 과정을 통해 마이그레이션을 수행함 - 연구 개발 결과 (1) GPU 자원 분할/관리 . 가상화 컨테이너 환경인 kubernetes를 기반으로 마이크로 서비스가 수행될 서버자원 중 GPU를 가상화하고 pod 형태로 제작된 응용이 할당됨 . 초기 서비스 구성 시 필요한 GPU 자원을 분할 . 분할된 가상 GPU를 여러 서비스가 동시에 사용할 수 있도록 인터페이스를 제공함 . MIG(MULTI-INSTANCE GPU) 기술을 사용하여 하나의 GPU를 여러 응용이 공유할 수 있도록 HW 지원 가상화를 수행 . Kubernetes extended resource 기술을 사용하여 하나의 GPU를 여러 응용이 공유할 수 있도록 SW 지원 가상화를 수행 . 응용의 GPU 자원에 대한 최소 요구사항(VRAM 용량)에 따라 단일 GPU 분할 가능 그림 230. 하나의 GPU를 여러 개의 vGPU로 가상화 (2) GPU 요구사항에 맞는 자원 요청 수행 및 동작 확인 . 소형응용(Vector 연산)은 논리적으로 분할한 GPU로 배치 . 중형 및 대형 응용(추론/학습)은 HW 독립적(MIG)으로 분할한 GPU로 배치 . 배치된 응용의 동시 수행 확인

수행기관	개발 목표	개발내용 및 범위

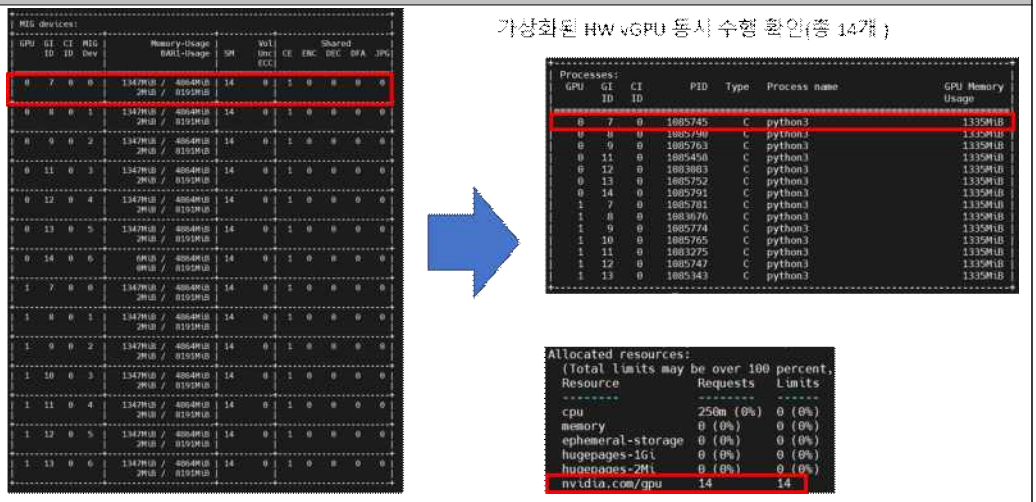


그림 231. 가상화된 여러 개의 HW vGPU 동시 수행

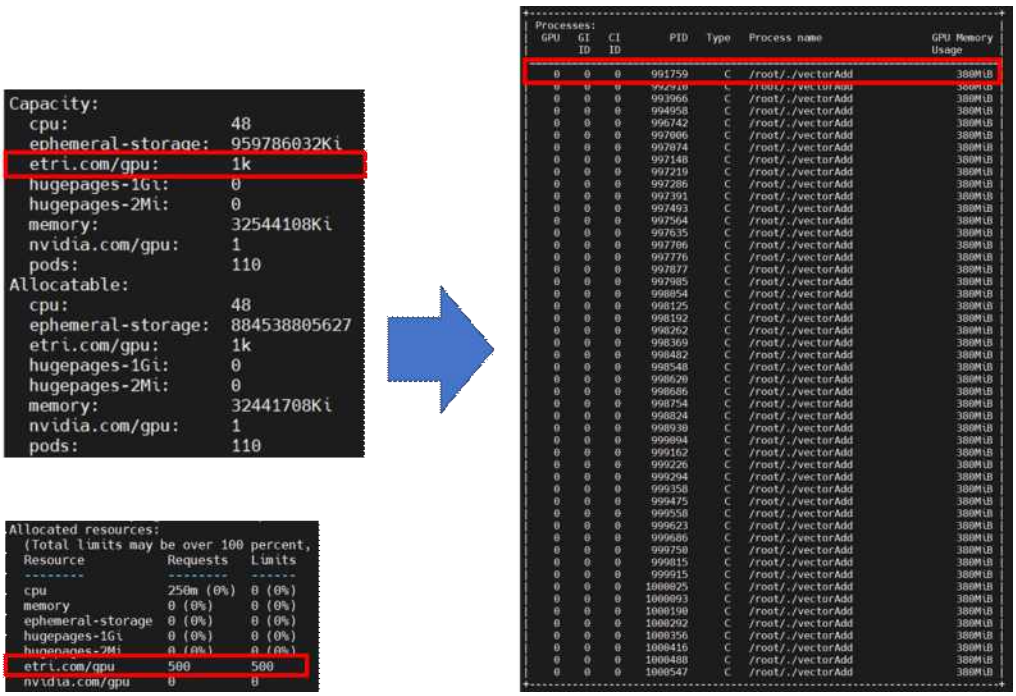


그림 232. 가상화된 여러 개의 SW vGPU 동시 수행

(4) 멀티 GPU 환경의 가상 메모리 관리를 통한 마이그레이션 기법 개발

. 1,2번 GPU를 가지는 호스트에서 파이토치 워크로드를 1번 GPU에서 실행을 시작한 후 2번 GPU에서 실행이 종료되는 시연 수행

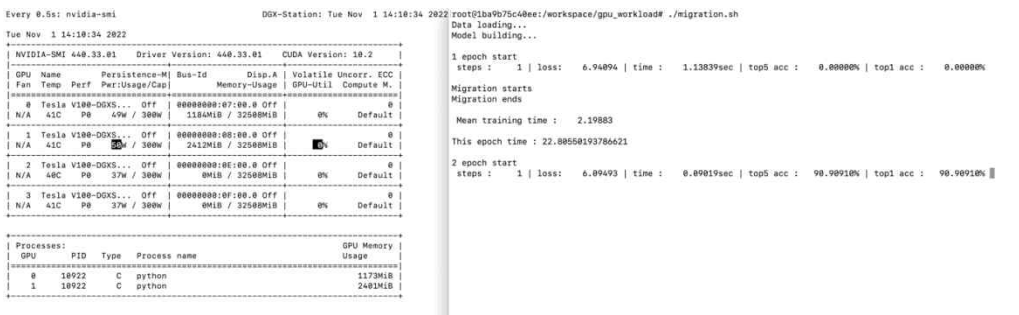


그림 233. GPU 마이그레이션을 통해 2번 GPU에서 실행되는 워크로드

수행기관	개발 목표	개발내용 및 범위
		- 향후 연구 계획 (1) 실시간으로 동적 HW/SW GPU 자원 가상화 및 자원 관리 가능 (2) FPGA, NPU 등 다양한 가속기들을 공유하기 위한 공통 자원 관리 구조 설계 및 개발 (3) 클러스터 환경의 GPU 서버(노드) 간 워크로드 마이그레이션 기법 개발 **[정량 목표] 가속기 공유 모델 1종 검증** - 실험 목적 : 다중 노드 기반의 GPU 자원 공유 기술 검증 (1) GPU 가속기 공유모델 제공 . 가속기 공유모델을 통한 한정된 GPU 가속기 자원의 공유 가능 (2) GPU 가속기의 효율적인 공간 활용 . 단일 가속기 동시 동작(최대 1000개 응용) . 응용의 자원 요구사항에 따른 특정 가속기 자원 할당 - 시스템 구축 (1) 소프트웨어 . Ubuntu 20.04 (커널 5.4.0) . Kubernetes 1.25 . CUDA 11.6 환경 (2) 하드웨어 . Intel(R) Xeon 5317 플랫폼 . NVIDIA A100 * MIG 구조(최대 7개 동시 동작 가능) * 40GB 메모리(7개 분할시 총 35(5*7)GB 메모리 사용 가능) (3)목표 검증 응용 . GPU 컴퓨팅을 수행하는 CUDA 응용 * 소형 응용 : 50000개의 vectorAdd 연산을 수행하는 워크로드(GPU memory 최소 380MB 이상 사용), vector 연산을 위한 CUDA runtime 포함 * 중형 응용 : 이미지 추론 워크로드(GPU memory 1GB 이상 사용), 추론 연산을 위한 CUDA+추론모델(ONNX, Pytorch 등) 포함 - 실험 결과 (1) GPU 가속기 공유모델 수행 결과

수행기관	개발 목표	개발내용 및 범위

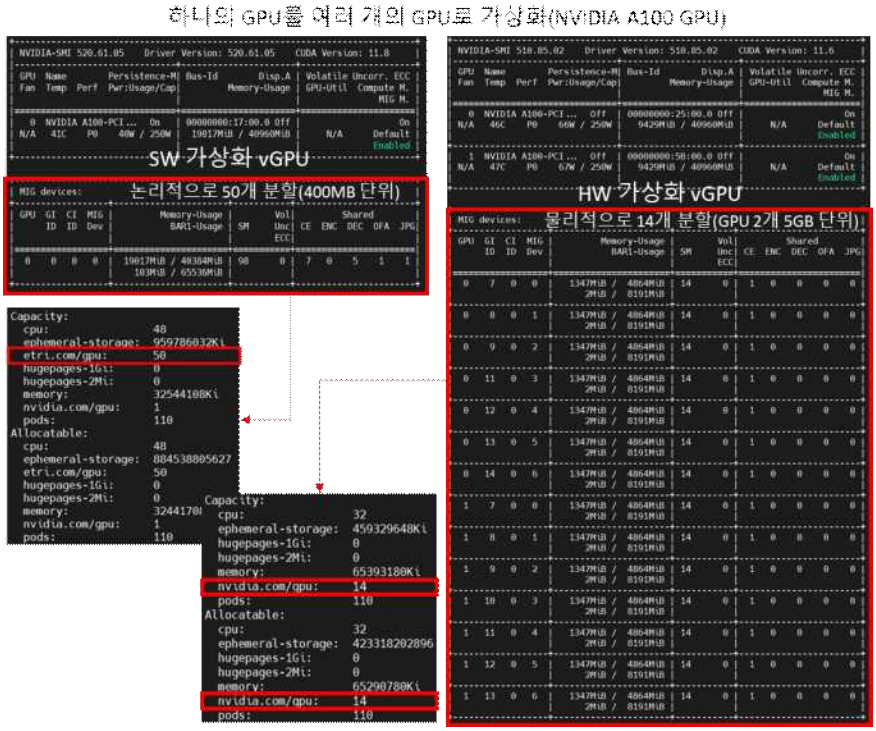


그림 234. GPU 가속기 공유모델 수행 결과

(2) GPU 가속기 공유모델을 적용한 효율적인 공간 활용

. 소형 응용과 중형 응용의 GPU 가속기 요청

* 자원 요구사항에 따른 가상화된 가속자원 배치
* 소형 응용은 논리적로 분할된 가상화된 vGPU에 배치
* 중형 응용은 HW 독립적으로 분할된 가상화된 vGPU에 배치

. 1000개의 소형 응용을 단일 GPU에 수행 요청

* 1000개의 Pod 생성 및 수행 요청
* 50개의 GPU 동작이 동시에 수행됨을 확인
* Pod이 종료되면 다음 Pod 수행 총 1000개의 Pod 수행
* 단일 GPU 가속기에서 1000개의 소형 응용이 모두 정상 동작함을 확인

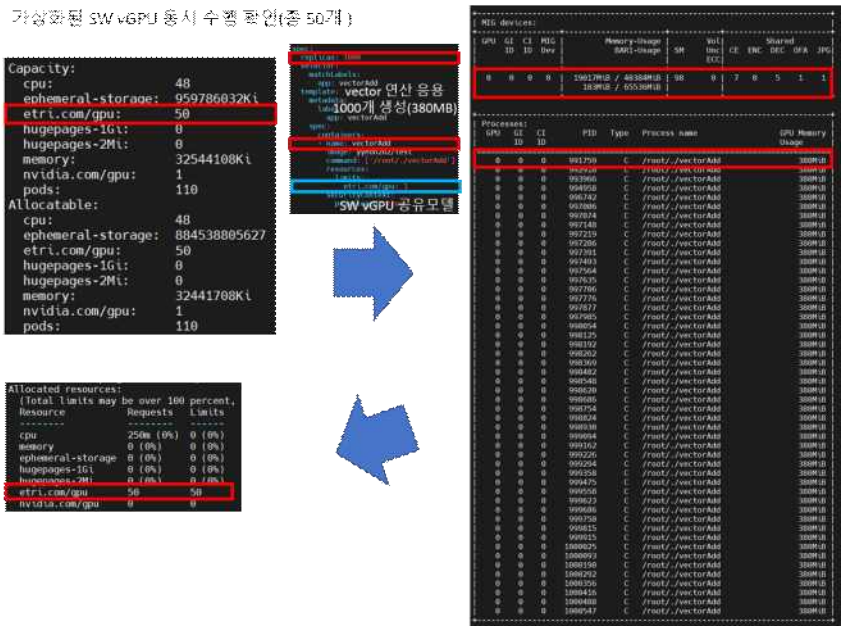
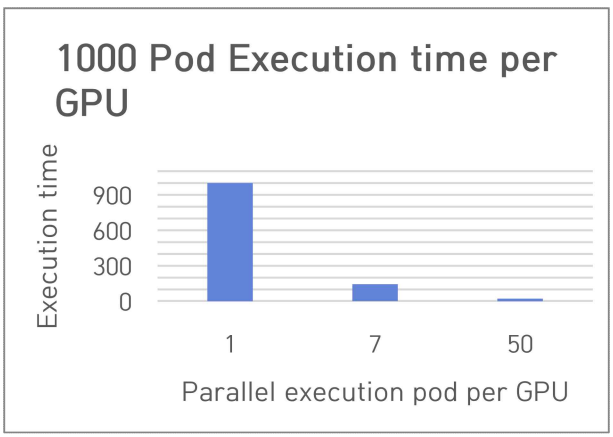
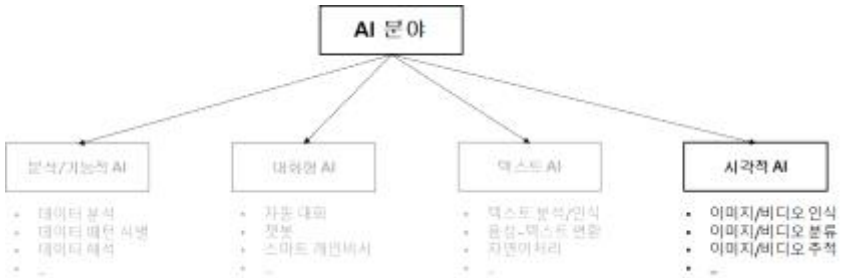
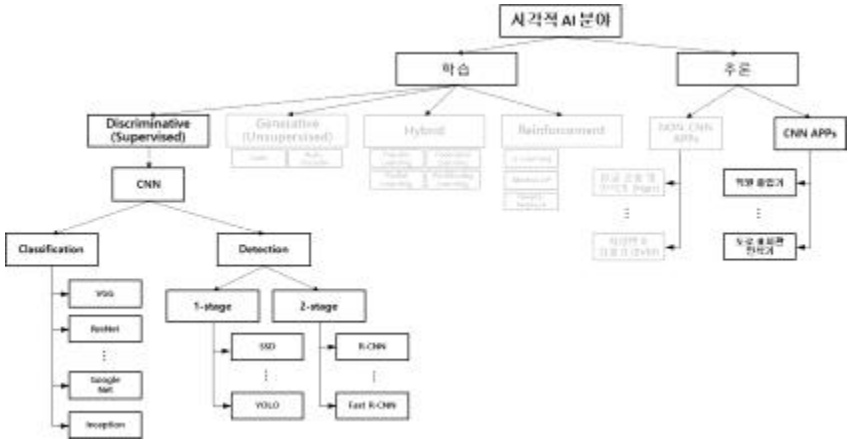
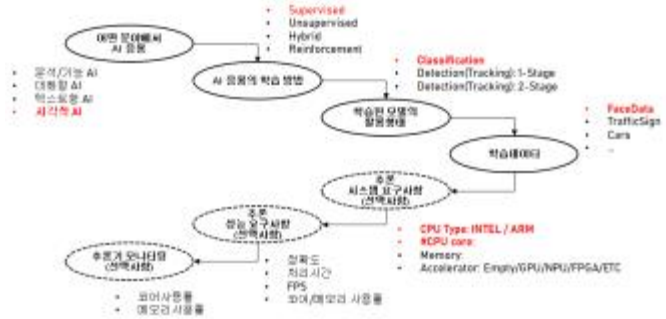
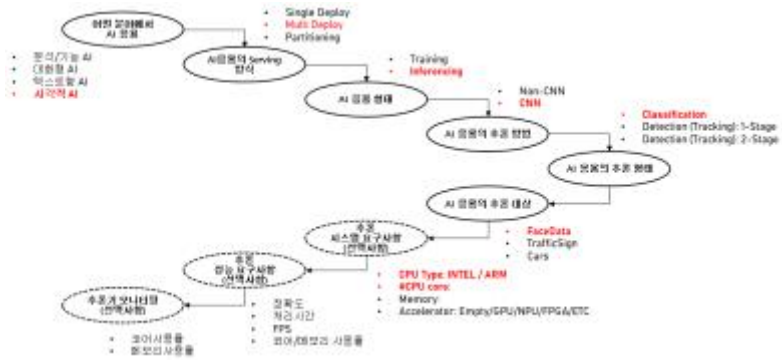


그림 235. 단일 GPU 가속기에서 1000개의 소형 응용 정상 동작 확인

수행기관	개발 목표	개발내용 및 범위
		(3) 14개의 중형 응용을 2개의 MIG GPU를 가진 단일 노드에 수행 요청 . 총 14개의 Pod 생성 . 50개의 GPU 동작이 동시에 수행됨을 확인 . 2개의 MIG GPU 가속기에서 14개의 중형 응용이 모두 정상 동작함을 확인 그림 236. 14개의 중형 응용을 2개의 MIG GPU를 가진 단일 노드에 수행 요청 - 결론 (1) 실험을 통해 GPU 가속기를 논리적으로 분할하여 다수의 응용을 단일 GPU 가속기에서 동시에 동작할 수 있음을 확인, 이를 통해 쿠버네티스 클러스터에서 동작하는 기존 가속기 활용 방식보다 최대 50배 빠른 처리 가능함을 검증하였음 (HW GPU 가상화 7배, SW GPU 가상화 50배) (2) 본 연구를 통해 개발된 GPU 가속기 공유 모델은 GPU 자원을 요구하는 소형 응용은 논리적으로 분할한 GPU로 할당하고, 중형 응용은 HW 독립적으로 분할한 GPU로 자원 할당을 진행함으로써 GPU 자원 요구 최대 1000개에 대해 원활한 자원할당을 수행함을 보였으며, 이를 통해 정량적 목표인 가속기 공유 모델 1종에 대한 검증을 완료함  그림 237. GPU당 1000 Pod 실행 시간 비교

수행기관	개발 목표	개발내용 및 범위
		• 디바이스별 시스템 자원의 연산 능력 분석 기술 연구 - 기술 개요 · 연구 정의 : 디바이스별 시스템 자원에 민감한 AI 응용의 성능 요구사항을 보장하기 위해 응용 SW 명세와 시스템 자원 기반의 디바이스 명세를 정의하고, 성능 요구사항을 충족하기 위한 최적 디바이스 선정을 통한 응용 배치 및 교체 가능한 응용 서비스 제공 알고리즘 기술 연구 - 당해연도 연구 목표 및 범위 (1) AI 응용 SW 명세 방법론 정의 (2) 마이크로 서비스 기반의 SW 명세 플랫폼 설계 (3) 시스템별 연산 능력 측정 기술 (4) 자원 정보 시스템과의 연동 - 세부 연구 개발 내용· (1) AI 응용 SW 명세 방법론 정의 . AI 분야는 아래의 그림처럼 크게 분석/기능적 AI, 대화형 AI, 텍스트AI, 그리고 시각적 AI 분야로 정의함 . 당해연도에서는 시각적 AI 분야에 해당하는 SW 명세 기능을 제공함  그림 238. 응용 SW 명세를 위한 AI 분야 정의 . 시각적 AI 는 아래의 그림처럼 학습과 추론으로 나누어짐 . 학습적인 측면에서는 지도학습 (Discriminative), 비지도학습(Generative), 반지도학습 (Hybrid), 강화학습 (Reinforcement)로 나누어지고 당해연도에는 지도학습에 해당하는 SW 명세 기능을 제공함 . 추론적인 측면에서는 CNN 기반의 응용을 지원하기 위한 SW 명세 기능을 제공함  그림 239. 시각적 AI분야 . AI 응용 명세 정의를 위한 Flow 기반의 절차 정의를 위해 학습기 생성을 위한 7단계 학습 명세 절차를 정의하고, 각 단계마다 사용자 및 개발자가 선택할 수 있는

수행기관	개발 목표	개발내용 및 범위
		속성을 정의함  그림 240. 7단계 학습 명세 절차 . 추론기 생성을 위한 9단계 추론 명세 절차를 정의하고, 각 단계마다 사용자 및 개발자가 선택할 수 있는 속성을 정의  그림 241. 9단계 추론 명세 절차 . 추론기 지원을 위한 4 종류의 지원형태를 기반으로 6단계 추론 지원 명세 절차를 정의하고, 각 단계마다 사용자 및 개발자가 선택할 수 있는 속성을 정의 그림 242. 6단계 추론 지원 명세 절차 . AI 응용 명세는 아래의 그림처럼 크게 5개의 부분인 Input, Preprocessing, AI, Output, Supplement 로 구성됨 . AI 응용 명세에서 대규모 디바이스를 활용할 수 있도록 지원하는 것은 serving에서 multi deploy에 해당하는 것이고, 시스템 자원의 한계 등으로 AI 가중치 모델을 수행하지 어려운 상황이면 partitioning을 지원함

수행기관	개발 목표	개발내용 및 범위

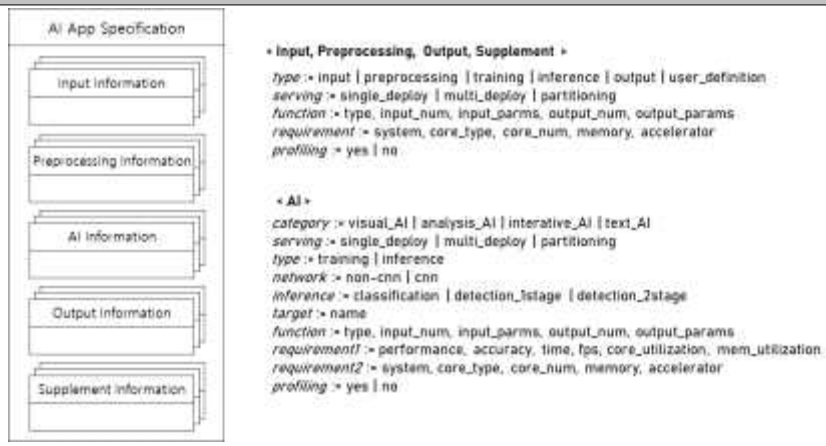


그림 243. AI 응용 명세 구성

. multi deploy 방식으로 카메라를 이용하여 얼굴을 인식하고 출력으로 LED로 결과를 출력하는 AI 응용 서비스에 대한 명세임

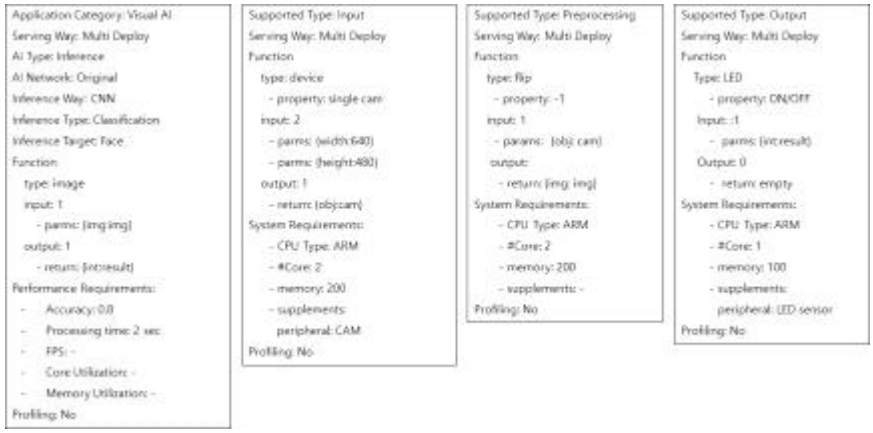


그림 244. 얼굴 인식 AI 응용 서비스 명세

(2) 마이크로 서비스 기반의 SW 명세 플랫폼 설계

· AI 응용 명세를 기반으로 파이프라인 중간표현을 생성하고, 생성된 중간표현을 이용하여 최적의 코드 및 하드웨어를 제공받기 위해서 질의어를 생성함

· 생성된 질의어는 자원정보명세 및 지식 공유 시스템에 추천을 받음

· 자원정보명세는 코드 명세 집합과 하드웨어 명세 집합으로 구성됨

· AI 응용 명세와 중간표현을 거쳐서 생성된 AI 응용을 시스템에 배포하기 위해서 단일 배포 명세와 다중 배포 명세를 지원함

· 배포 시에 필요에 따라서 partitioning을 지원할 수 있음.(이 기능은 Visual AI에서 학습된 모델에서만 적용 가능)

· 배포 명세를 통해서 시스템에서 동작하면서 profiling 여부에 따라서 실행중에 측정된 값들인 정확도, 추론시간, CPU활용률, GPU활용률, 메모리활용률 등을 측정되고, 측정된 값은 코드명세 혹은 하드웨어명세쪽에 업데이트되거나 지식공유시스템에 업데이트 될 수 있음

수행기관	개발 목표	개발내용 및 범위

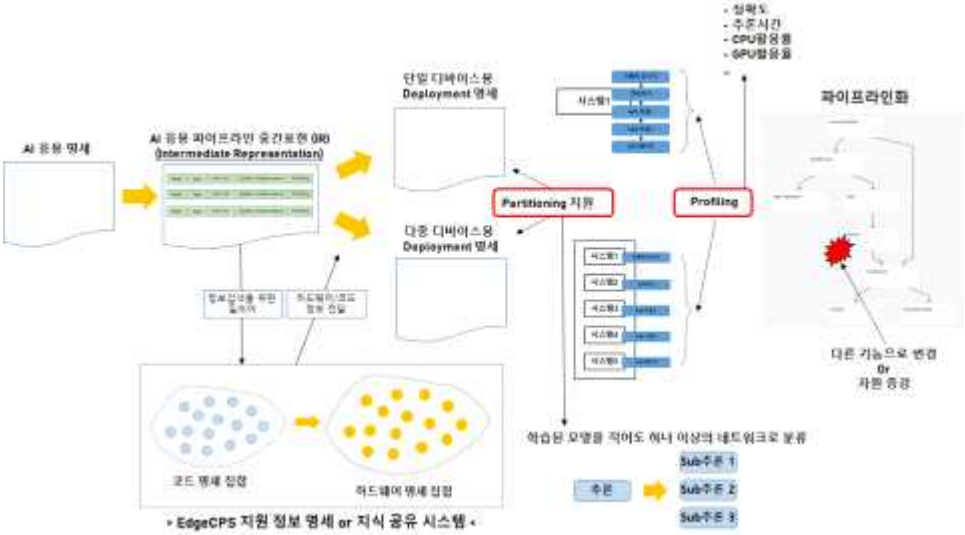


그림 245. 파이프라인 기반의 AI 응용 수행 시나리오

meta information := deployment_type, func_num, input_type, specification_type, output_type, developer

head := input | preprocessing | training | inference | output | user_definition

type := head_feature

func_I/O := input_num, parameters, output_num, parameters

requirements := system requirement | performance requirements

profiling := YES | NO

[Example]

- Multi-deployment, 4, cam, CNN-Classification, LED, YJKIM
- input, {type: device=single_cam}, {func_input_parms:2, {640, 480}}, {func_output_params:1, {cam:obj}}, {system: arm, core=2, mem=200, peripheral=cam}, {profiling=no}
- preprocessing, {type: flip} {func_input_parms:1, {cam:obj}}, {func_output_parms:1, {img:img}}, {system: arm, core=2, mem=200, peripheral=cam}, {profiling=no}
- inference, {type=CNN | Classification, target=faceData}, {type: input=image}, {func_input_parms:1, {img:img}}, {func_output_parms:1, {int:return}}, {performance: acc=0.8, speed(sec)=2.0, FPS=empty, core_util=empty, mem_util=empty, tx=empty, rx=empty}, {profiling=no}
- output, {type: device=LED}, {func_input_parms:1, {int:return}}, {func_output_parms:0, empty}, {system: arm, core=2, mem=200, peripheral=cam}, {profiling=no}

그림 246. AI 응용 파이프라인 중간표현 포맷과 중간표현의 예

· 코드 명세에서 Type Definition은 코드에 대한 특징을 설정, Function Inforamtion은 함수의 이름, 입력, 출력 값들을 정의, Provisioin Information은 코드의 위치 정보등을 포함, Dynamic Information은 실제 특정 디비아이스에서 수행한 성능 정보들을 포함함

수행기관	개발 목표	개발내용 및 범위

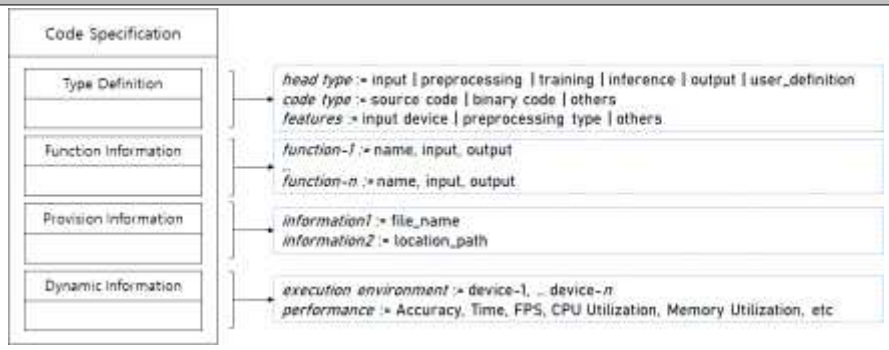


그림 247. 코드 명세 예

· 디바이스 명세에서 Basic Information은 디바이스 스펙에 해당하는 정보, Add-On Information은 디바이스의 기본적인 스펙에 추가적으로 설치된 정보, SW Information은 디바이스에 설치된 프로그램 정보, Dynamic Information은 디바이스에서 다양한 응용들이 수행하면서 사용되고 있는 시스템 전체 사용량 등에 관련된 정보들을 포함함

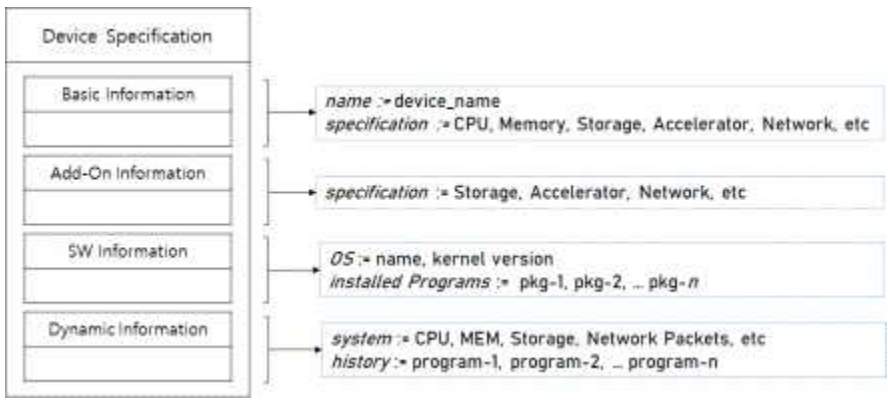


그림 248. 디바이스 명세 예

· 배포 명세에서 Global Information은 배포 방식을 결정하는 정보, Basic Information은 배포이름과 응용명세의 타입에 관련된 정보, System Information은 배포할 시스템에 대한 정보, Application Information은 실제 배포되어 실행해야 될 프로그램에 대한 정보들을 포함함.

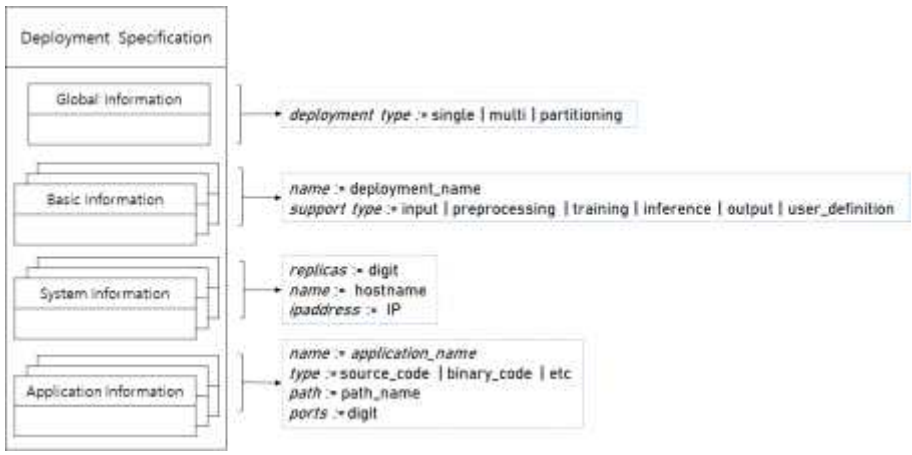
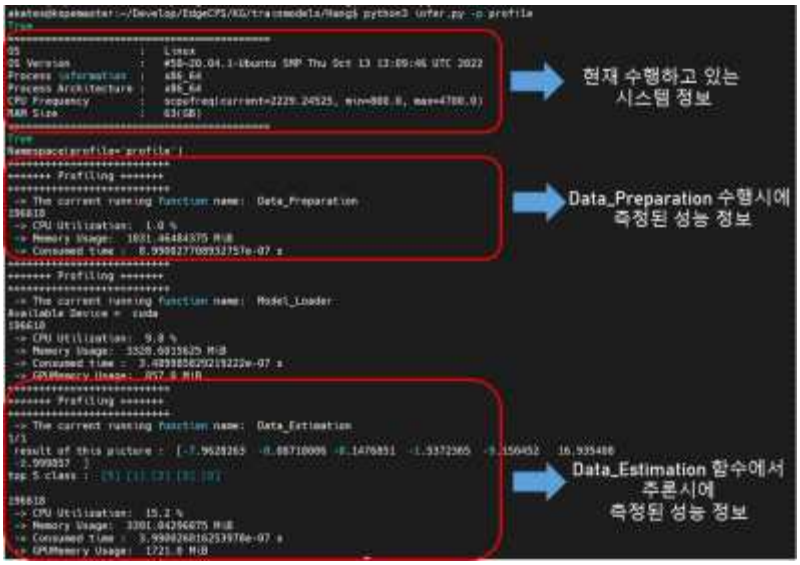
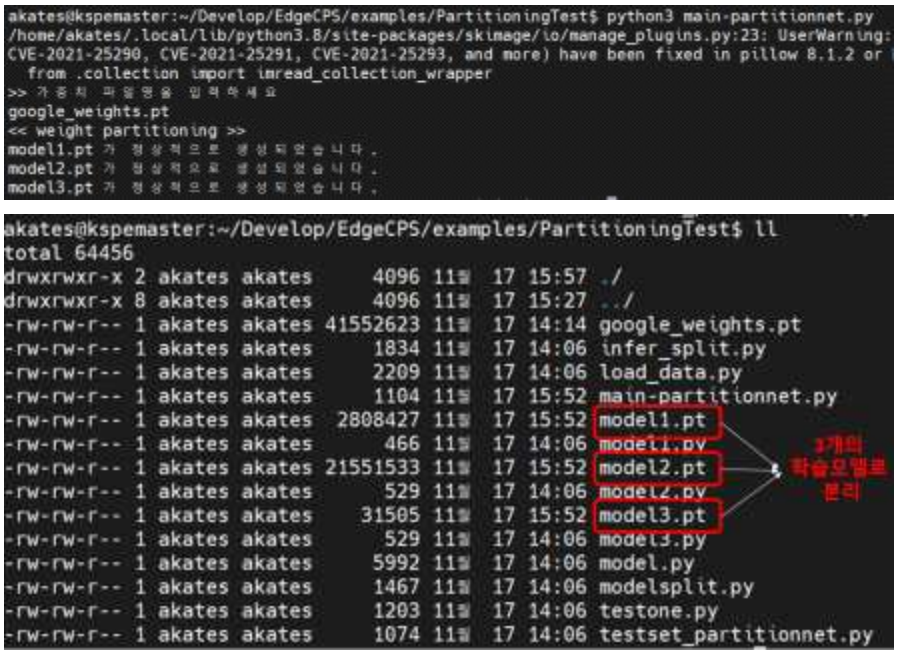
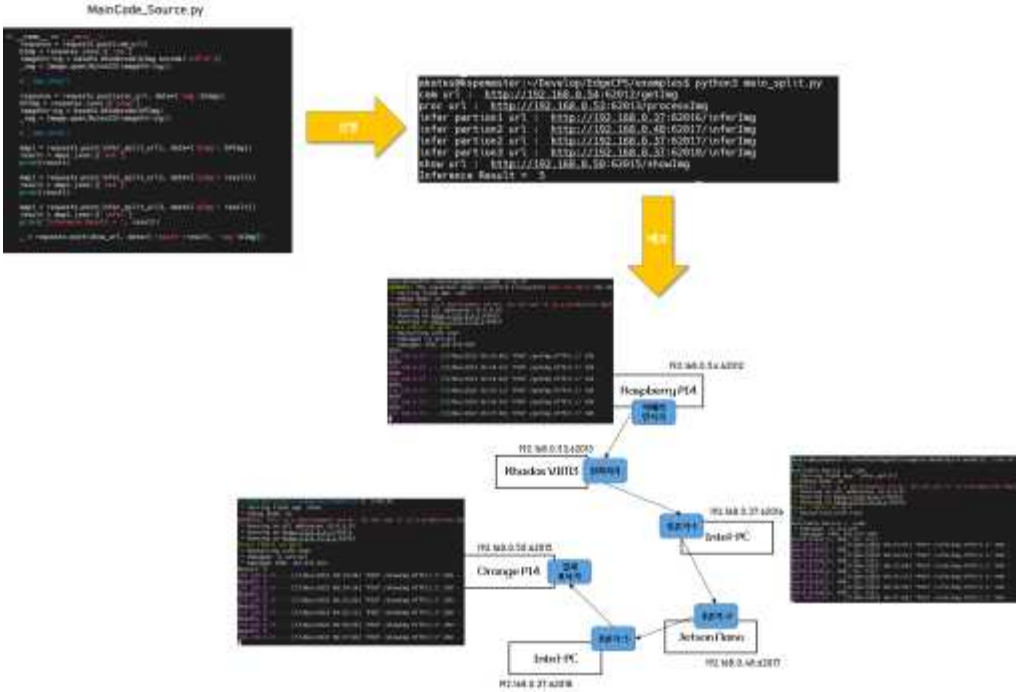
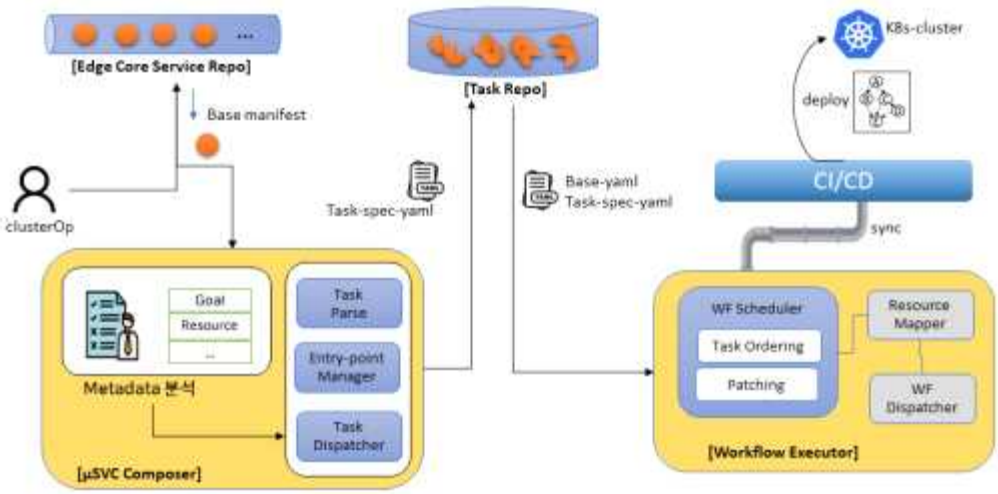


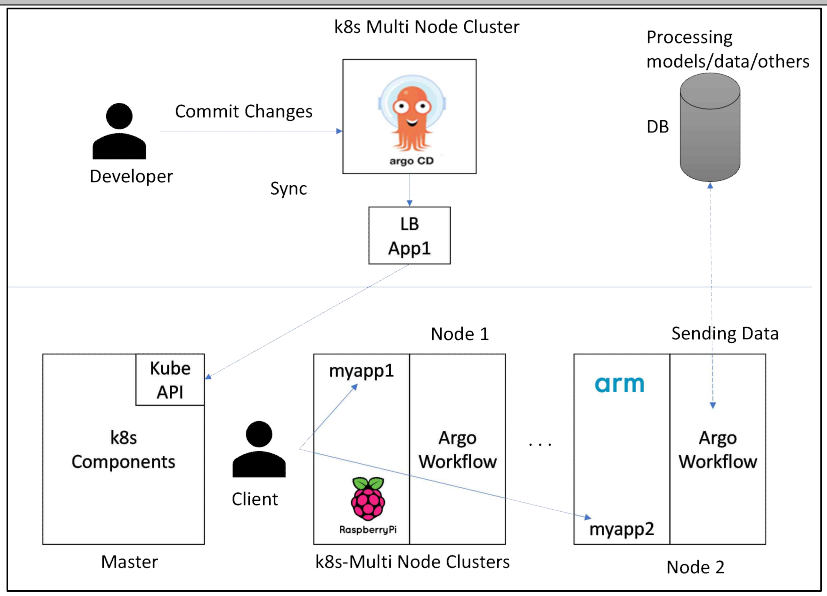
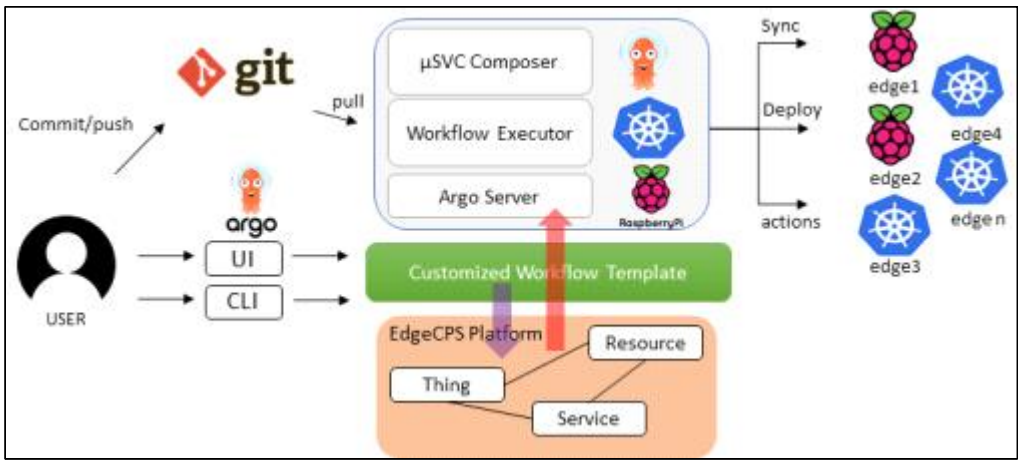
그림 249. 배포 명세 예

- 연구 개발 결과

수행기관	개발 목표	개발내용 및 범위
		(1) 각 디바이스에 수행한 AI 응용 서비스들에 대한 실시간 성능 측정 · 수행한 디바이스의 시스템정보 · 수행한 AI 응용에 대한 함수별 혹은 컴포넌트별 성능 측정 · 성능 정보는 CPU 점유률, MEM 사용량, GPU 메모리 사용량, 소요시간 등이 측정  그림 250. AI 응용 서비스들에 대한 실시간 성능 측정 (2) 학습된 모델에 대해서 사용자가 원하는 만큼 Partitioning 할 수 있는 기능 · partitioning 할 가중치 파일을 입력하면 사용자가 원하는 만큼으로 분리  그림 251. 학습 모델 partitioning (3) SW 명세 기반으로 생성된 마이크로서비스 기반의 AI 추론기의 실행 · 5종의 이기종 디바이스에 배포되어 마이크로서비스가 동작 · 학습된 모델이 3개의 추론기 나누어져 동작 · 추론 결과는 원하는 목적지에서 출력

수행기관	개발 목표	개발내용 및 범위
		 그림 252. 마이크로서비스 기반 AI 추론기 - 향후 연구 계획 · 다양한 AI 응용 지원을 위한 명세 고도화 · 성능정보 profiler와 가중치 모델 partitioner의 고도화 · 마이크로 서비스 기반의 SW 명세 플랫폼 개발 · EdgeCPS 자원정보 및 지식공유 시스템 활용을 통한 명세 최적화
	④ 엣지사이버물리시스템을 위한 대규모 디바이스 제어 기술 개발	**• 디바이스와 가상실행환경의 디바이스트윈간 동기화 기술 연구** - 기술 개요 · 연구 정의: EdgeCPS 플랫폼에서는 디바이스의 임무를 실행하기 위해 필요한 규격(자원, 기능, 성능)의 총합에서 실제 디바이스의 규격과의 차이만큼을 증강하여 실행하기 위한 디바이스 트윈이 존재함. 본 연구에서는 실디바이스와 디바이스 트윈간 동기화 및 이에 기반한 모듈 증강을 위한 프레임워크를 연구함 - 당해연도 연구 목표 및 범위 (1) 엣지 사이버물리시스템 운영상황인지형 워크플로우 구성 기술 (2) 엣지 디바이스 간 모듈 공유를 위한 미들웨어 기술 - 세부 연구 개발 내용 (1) 대규모 디바이스의 시스템 자원을 효율적으로 활용하기 위한 워크플로우 구성 기술 설계 . 마이크로서비스 단위의 Task 분할 기법 조사 . 마이크로서비스 단위의 Task 분할 기능(μSVC Composer) 설계 . 구성된 워크플로우 관리를 위한 Task Repository 개발 . 쿠버네티스 환경 연계를 위한 Workflow Executor 설계 (2) 대규모 워크로드 관리와 제한적인 데이터 가시성 극복을 위해 마이크로서비스 기반 서비스 동기화 프로토타입 기술 개발 . 라즈베리파이를 이용한 엣지 클러스터 구축

수행기관	개발 목표	개발내용 및 범위
		. 디바이스트윈 동기화 환경에서 동작하는 워크플로우 구성 요소 정의 (디바이스, 자원, 서비스) . 오픈소스 Argo CD/Workflow 조사 및 개선 방향 수립 . 쿠버네티스 환경 연계 및 동작 테스트 (3) 디바이스트윈을 통한 사물 간 동적 모듈 증강 아키텍처 개발 . 디바이스트윈을 통한 모듈 간 증강을 위한 미들웨어 설계 . 가상 디바이스 모듈 합성을 위한 Synthetic Twin Manager 설계 . 모듈 증강 프로세스에 따른 백엔드 서비스를 내재한 인스턴스 실행을 위한 Instance Manager 설계 - 연구 개발 결과 (1) 대규모 디바이스의 시스템 자원을 효율적으로 활용하기 위한 워크플로우 구성 기술 . µSVC Composer와 Workflow Executor을 통한 워크플로우 구성 기술 설계  그림 253. 워크플로우 구성 Architecture . "Edge Core Service Repository"에 적재 된 코어서비스를 마이크로서비스 단위의 Task로 분할(µSVC Composer) . 분할 된 Task는 사용자가 요구하는 목적에 맞게 워크플로우를 구성할 수 있도록 "Task Repository"에 저장 . 개별 Task는 하나의 application으로 구성될 수 있으며 app. of apps. 형태의 워크플로우로 재정의 가능 . Task 수행을 위해 필요한 리소스를 연계하고 쿠버네티스 환경에 배포 및 동기화 작업을 수행(Workflow Executor) (2) 대규모 워크로드 관리와 제한적인 데이터 가시성 극복을 위해 마이크로서비스 기반 서비스 동기화 프로토타입 기술 개발

수행기관	개발 목표	개발내용 및 범위
		 그림 254. 라즈베리파이를 이용한 워크플로우 프로토타입 시스템 구축 . 컨테이너 환경에서 마이크로서비스 단위의 응용을 개발 및 배포하기 위해 라즈베리파이 3대를 이용한 엣지 클러스터 구축 . Customized Workflow-template을 통한 워크플로우 구성요소(디바이스, 자원, 서비스) 정의  그림 255. Customized Workflow Template을 통한 디바이스트윈과 자원 연계 . 오픈소스 Argo CD/Workflow를 통해 배포, 동기화 작업 병렬 수행 . 쿠버네티스 상에서 Yaml 명세를 수정해서 사용자가 기본적인 워크플로우를 정의할 수 있음 . Customized Workflow Template을 통해 사용자가 수행하고자 하는 태스크와 리소스를 맵핑 후, 클러스터 내 노드에 워크플로우 재배포

수행기관	개발 목표	개발내용 및 범위

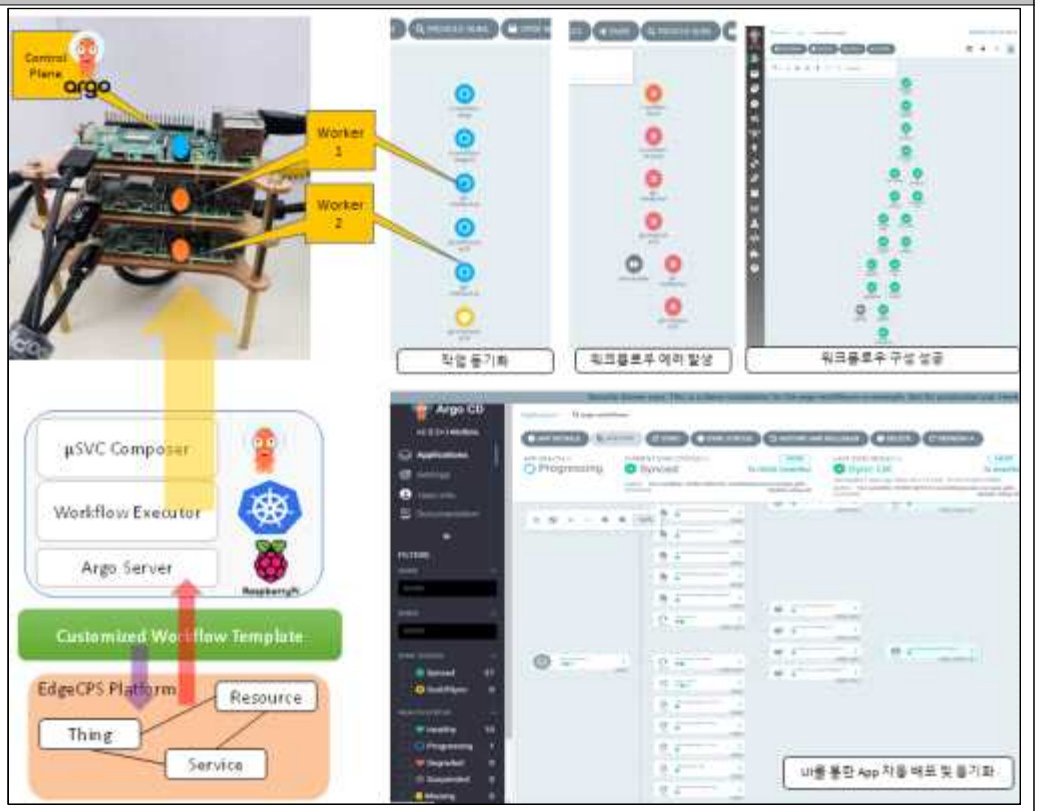


그림 256. 오픈소스 기반의 워크플로우 구성 UI

(3) 디바이스트윈을 통한 사물 간 동적 모듈 증강 아키텍처 개발

. 실세계의 디바이스를 디바이스트윈과 디바이스의 하위 모듈인 모듈트윈으로 계층적으로 관리할 수 있도록 구조 설계

. 디바이스의 물리적 제약을 극복하기 위해 다른 디바이스들의 모듈을 가져와 논리적으로 새롭게 동작하는 디바이스로 합성하는 개념 및 동작 구조 설계

. 합성 디바이스가 사용하는 백엔드 서비스에 대한 정의와 동작 구조 설계

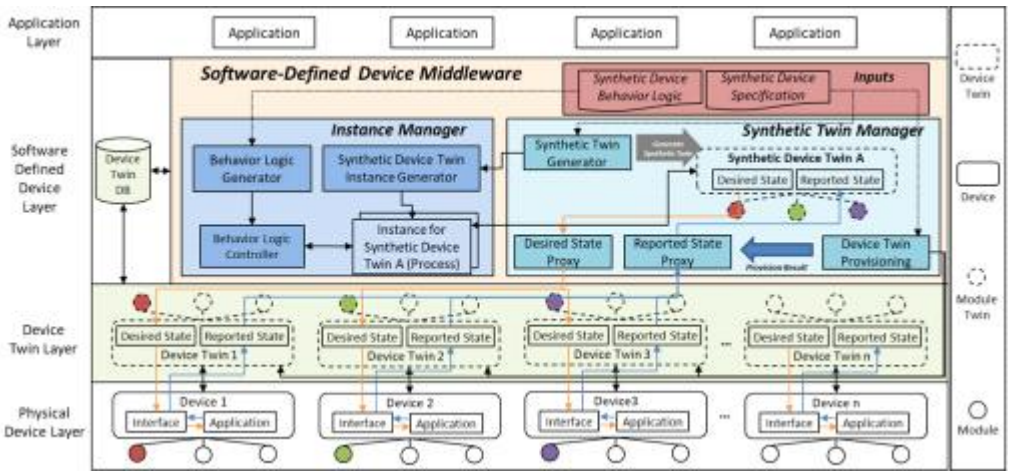


그림 257. 디바이스트윈을 통한 사물간 동적 모듈 증강 아키텍처

- 향후 연구 계획
  · 디바이스와 디바이스트윈 간의 빠른 동기화를 위한 핵심 관리 요소 분석
  · Customized Workflow Template 고도화 및 워크플로 실행 엔진에 적용

수행기관	개발 목표	개발내용 및 범위

· 디바이스트윈을 통한 동적 모듈 증강 서비스 구현

• 근거리 엣지사이버물리시스템간 데이터 고속 처리 기술 연구

- 기술 개요

· 연구 정의 : 지리적으로 한정된 엣지 환경 내에 이종의 규격을 갖는 다수의 자율 사물들이 존재할 때, 규격에 관계 없이 개별 사물에 할당된 원활한 임무 수행을 위하여 EdgeCPS는 자원 및 기능 증강을 제공하며, 이를 실현할 수 있는 아키텍처 및 이를 기반으로 데이터를 고속 처리할 수 있는 기법 연구

- 당해연도 연구 목표 및 범위

(1) 근거리 EdgeCPS 내의 이종 디바이스-플랫폼 간 자원 증강 클러스터 구성 기술 연구

(2) 저/중/고성능 디바이스-플랫폼 간 영상 추론 데이터 고속 처리 기술 연구

- 세부 연구 개발 내용

(1) 근거리 EdgeCPS 내의 이종 디바이스-플랫폼 간 자원 증강 클러스터 구성 기술 연구

. 다양한 성능의 디바이스들로 구성된 엣지 클러스터에서 저성능 디바이스가 EdgeCPS의 자원/기능 증강을 통해 한계를 극복하는 시나리오 정의

. 엣지 디바이스는 AI 응용 수행 시나리오에서 저/중/고성능을 대표할 수 있는 임베디드 보드로 구성됨. 구성된 엣지 디바이스의 성능 스펙은 표와 같음

표 27 실험을 위한 클러스터 노드 및 자원

Role	CPU	Mem	GPU
CP (server)	Xeon W-1250	64G	N/A
worker1 (server)	Xeon E-2246G	32G	GTX 1660 Super
worker2 (Rpi4)	Cortex A72	4G	N/A
worker3 (Odr)	Cortex A73, A53	4G	N/A
worker4 (Jetson)	Cortex A57	4G	NVIDIA Maxwell 128-c

. 엣지 디바이스들은 AI 모델을 기반으로 실시간 영상 추론 서비스를 수행하고 있으며 원활한 서비스 제공을 위해 EdgeCPS를 통해 자원/기능 증강을 실시함

. 자원/기능 증강은 AI 네트워크의 일부를 EdgeCPS에서 계산하게 함으로써 구현됨. 저/중/고 성능별 자원/기능 증강 정도의 차이를 둠으로써 엣지 클러스터의 전체 자원을 효율적으로 활용하도록 함

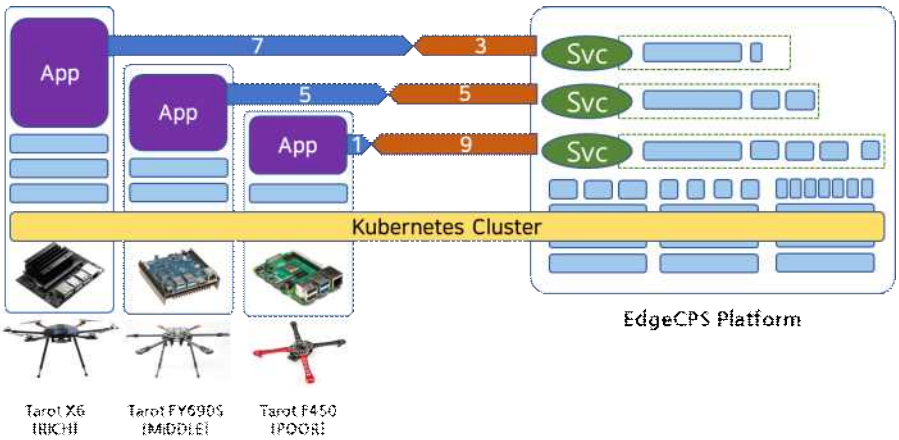


그림 258. EdgeCPS 플랫폼을 활용한 저/중/고성능 디바이스의 자원/기능 증강

(2) 저/중/고성능 디바이스-플랫폼 간 영상 추론 데이터 고속 처리 기술 연구

수행기관	개발 목표	개발내용 및 범위

. EdgeCPS에서 다양한 아키텍처(amd64, arm64 등)간 자원/기능 증강을 위하여 서버-클라이언트 방식을 채택함. 디바이스에서 계산 완료된 중간 결과물을 EdgeCPS 고성능 서버에 전달하고 최종 결과값을 리턴받음

. 구현을 위하여 저/중/고성능 디바이스와 자원/기능 증강을 위한 고성능 서버를 별개의 노드로하여 쿠버네티스 클러스터로 구성하였음.

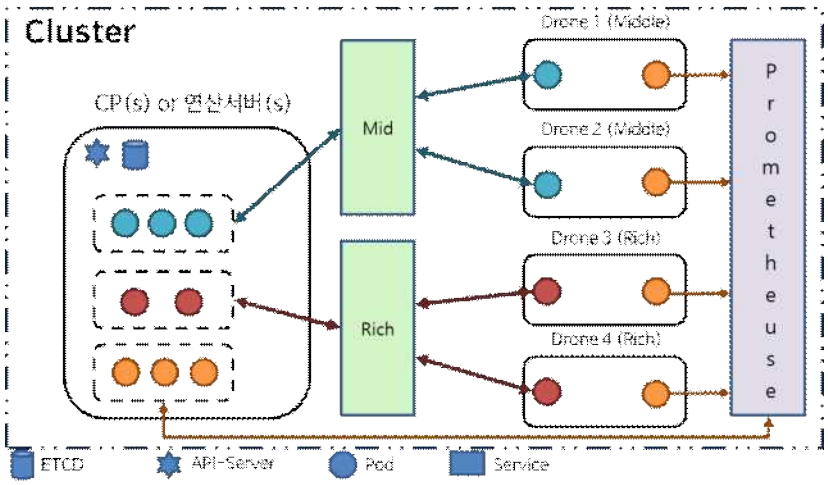


그림 259. EdgeCPS 기반 사이버 물리 시스템을 위한 자원/기능 증강 아키텍쳐

- 연구 개발 결과

(1) 근거리 EdgeCPS 내의 이종 디바이스-플랫폼 간 자원 증강 클러스터를 통한 성능 향상

. EdgeCPS 기반 자원 증강을 통해 엣지 디바이스의 AI 추론 성능이 최소 34.9% 이상 향상되었음을 보임. 그림 260는 엣지 디바이스의 단일 및 증강을 통한 성능 그래프를 보여줌

. 증강을 통해 성능 향상이 이뤄졌으나 네트워크를 통해 중간/최종 결과물이 전송되므로 단일 계산보다 프레임의 편차가 늘어난 것을 확인할 수 있음. 향후 일관성 있는 자원/기능 증강 서비스 제공을 위한 연구를 수행할 필요가 있음

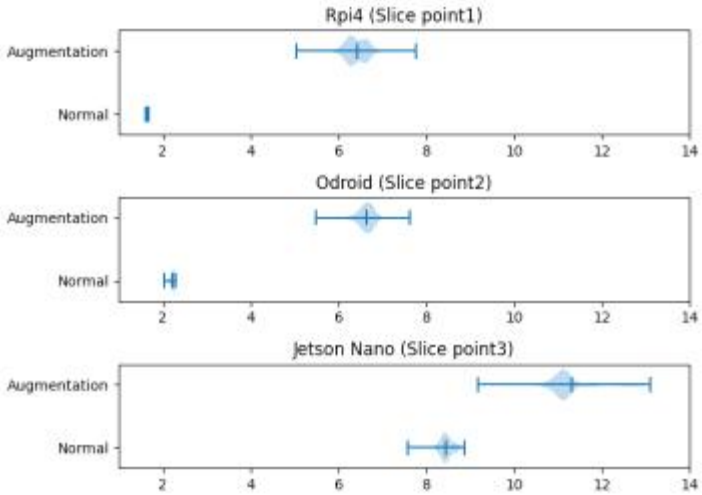


그림 260. 저/중/고성능 디바이스별 Yolov5s 모델의 단일/증강 연산 성능(FPS) 비교

. 그림 261은 고성능 디바이스인 Jetson nano을 기준으로 각 분할지점을 적용한 자원/기능 증강 성능 그래프를 나타냄. 디바이스에서의 단일 AI 네트워크 계산보다 point1 지점에서의 자원/기능 증강 성능이 더 낮게 나왔으며, point2와 point3 사이에서 최적의 성능을 도출할 수 있었음. 이를 통해 디바이스의 성능과 중간 결과물의 크기, 네트워크 상태 등 여러 요소를 고려한 AI 네트워크 분할이

수행기관	개발 목표	개발내용 및 범위
		필요하다는 결론을 도출

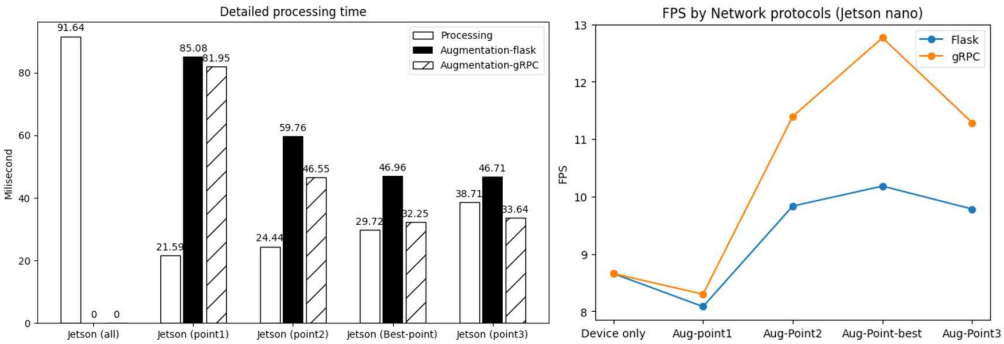


그림 261. Jetson nano에서의 분할 지점별 성능 평가

(2) 저/중/고성능 디바이스-플랫폼 간 영상 추론 데이터 고속 처리 효과 검증

. 엣지 디바이스가 수행하는 실시간 영상 분석 AI는 객체 탐지 모델인 Yolov5s를 활용하였으며, 이 모델을 디바이스/증강 서버용으로 분할하였음

. 엣지 디바이스는 카메라로부터 가져온 이미지를 나누어진 AI 모델을 통하여 생성된 중간 벡터를 자원/기능 증강 서버로 전달함

. 고성능 서버는 전달받은 중간 벡터를 기반으로 나머지 AI 연산을 실시하며 연산 결과를 디바이스에게 리턴함

. Yolov5 모델은 내부적으로 1) Backbone, 2) Neck, 3) Head의 3가지 논리적은 블록 연산으로 객체 탐지를 실시함. 본 연구에서는 AI 모델을 저/중/고성능 디바이스별 증강 정도에 차이를 두어 그림 262과 같이 분할함

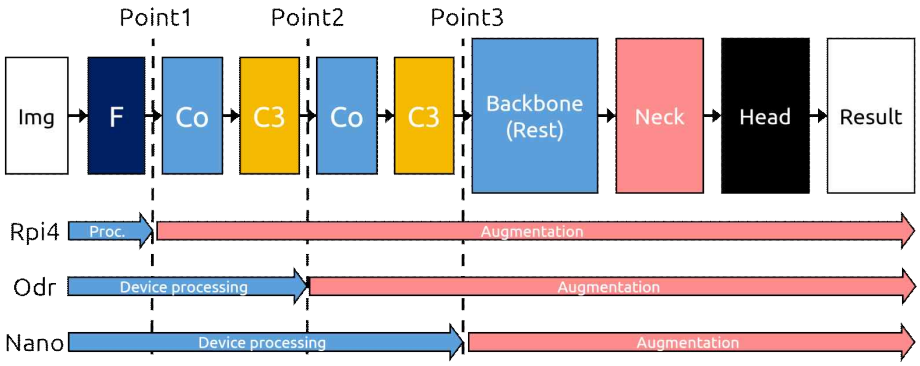
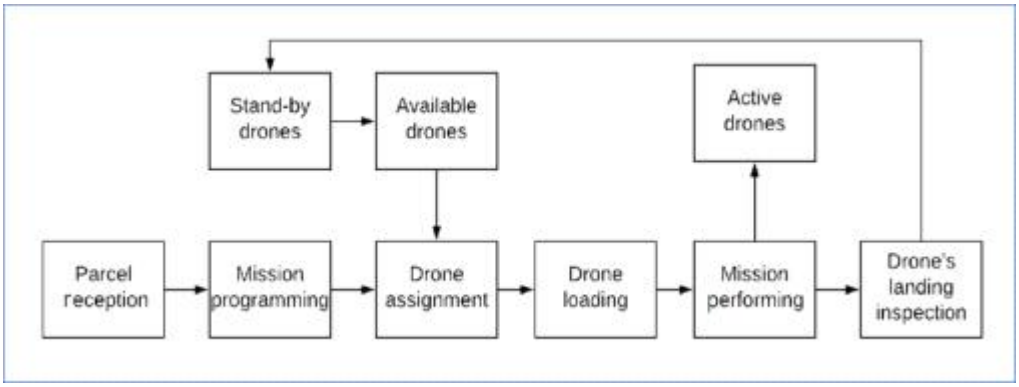
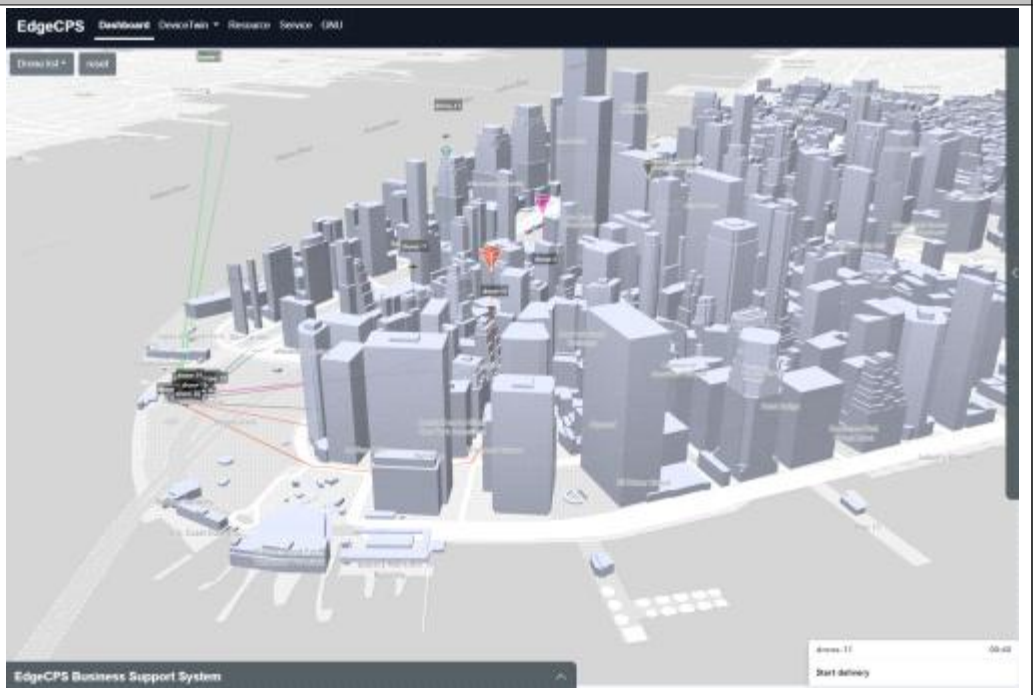
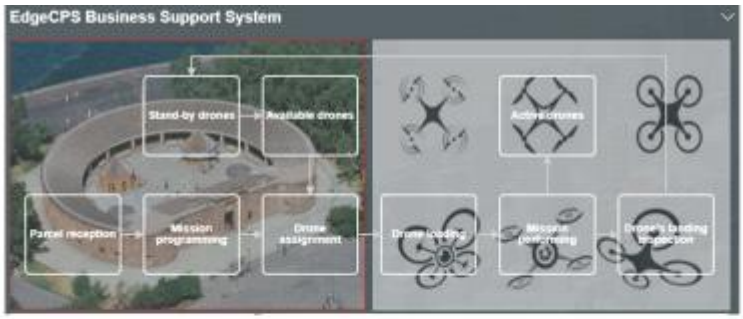


그림 262. 저/중/고성능 엣지 디바이스 별 네트워크 분할/증강

그림 263. Jetson nano 에서의 실행 결과(왼쪽: 디바이스, 오른쪽: 증강을 통한 객체 탐지

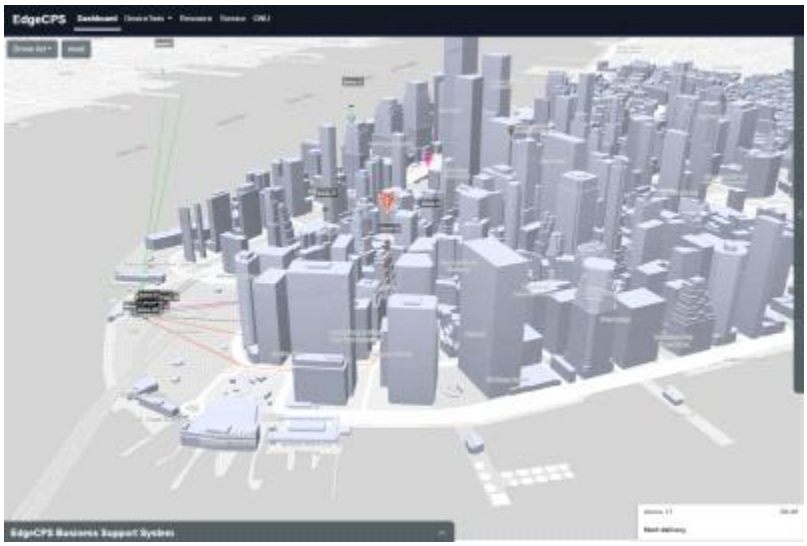
- 향후 연구 계획
  · 자원 증강 클러스터 환경에서의 기기간 모듈 증강 프로세스 연구
  · 저/중/고성능 디바이스-플랫폼 간의 데이터 고속 처리를 위한 시스템 SW 기술 연구

수행기관	개발 목표	개발내용 및 범위
		• 가상화된 대규모 시스템자원 관리 아키텍처 연구 - 기술 개요 · 연구 정의 : 지리적으로 한정된 엣지 환경 내에 대규모 디바이스에 대한 자원 증강을 지원하기 위해서는 EdgeCPS 플랫폼 내의 시스템 자원은 가상화되어 관리되고, 필요에 따라 디바이스에 분배될 수 있어야 함. 가상화된 대규모 시스템 자원에 대한 관리 및 제어를 위한 아키텍처가 요구되며, 이를 검증할 수 있는 개념 증명(PoC) 시스템이 필요함 - 당해연도 연구 목표 및 범위 (1) 대규모 디바이스에 대한 가상 디바이스 운용 시나리오 연구 (2) 가상 디바이스와 연계한 대규모 시스템 자원 관리 기술 연구 - 세부 연구 개발 내용 (1) 대규모 디바이스에 대한 가상 디바이스 운용 시나리오 연구 . 지리적으로 한정된 엣지 환경 내에 대규모 디바이스가 운용되는 시나리오로서 가상화된 도시를 배경으로 한 드론 배송 시나리오 정의 . 출처: Troudi, A.; Addouche, S.-A.; Dellagi, S.; Mhamedi, A.E. Sizing of the Drone Delivery Fleet Considering Energy Autonomy. Sustainability 2018, 10, 3344. https://doi.org/10.3390/su10093344  그림 264. 시나리오 속 자율 사물로서 드론의 임무 프로세스 . 저/중/고성능의 드론이 동일한 임무를 수행(물품을 주문 고객에게 배송) 시 규격에 따른 차이(최고 고도, 페이로드 등)를 고려한 주문 시나리오 개발 . 20대의 드론이 연속되는 주문을 처리하는 것을 가시화하는 가상 도시 인터페이스 개발 (2) 가상 디바이스와 연계한 대규모 시스템 자원 관리 기술 연구 . 가상 도시 인터페이스 속 개별적인 드론이 목적지 이동 후 고객 확인 작업에서 호출할 수 있는 얼굴 인식 서비스의 dummy pod를 EdgeCPS에 개발 (저/중/고성능 디바이스 별로 서로 다른 서비스 개발) . 가상 도시 인터페이스와 연계되는 실제 드론을 개발하고, 상호 동기화될 수 있도록 인터페이스 통합 (위탁연구 연계) - 연구 개발 결과 (1) 대규모 디바이스에 대한 가상 디바이스 운용 시나리오 연구 . 가상 도시 속에서 10초마다 요청되는 주문(목적지, 고도, 페이로드 특성 차이)에 따라 적합한 드론이 배치되어 배송을 처리하는 시나리오 구현

수행기관	개발 목표	개발내용 및 범위
		 그림 265. 가상 디바이스 운용 시나리오 . 요청된 주문에 대한 상세 정보와 배치된 드론의 정보, 그리고 해당 드론의 상세 정보를 조회할 수 있는 인터페이스 개발  그림 266. 가상 디바이스의 실시간 정보 조회 인터페이스 . 전체 배송 서비스를 수행중인 EdgeCPS 플랫폼 내의 노드(디바이스 및 서버), 서비스(저/중/고성능 얼굴인식), 자원 정보 조회를 위한 인터페이스 개발

수행기관	개발 목표	개발내용 및 범위
		 그림 267. 가상 디바이스 운용 시나리오 수행 중 EdgeCPS 플랫폼의 정보 조회 인터페이스 (2) 가상 디바이스와 연계한 대규모 시스템 자원 관리 기술 연구 . 시나리오를 구성하는 가상 디바이스(트윈), 주문, 디바이스와 자원간 매핑 관계를 관리할 수 있는 MongoDB 클러스터 구성 . 가상 디바이스 운용 시나리오 상에서 디바이스의 수를 1000대로 증가하고, 이에 따른 얼굴 인식 서비스의 실행 및 상태 모니터링 . EdgeCPS 플랫폼에서 최대 1000개의 디바이스 트윈(본 시나리오에서는 얼굴 인식 서비스와 매핑) 인스턴스를 생성해 개별 디바이스에 대한 서비스 제공 확인 그림 268. EdgeCPS 플랫폼 상에서 최대 1000개의 디바이스 트윈 실행

수행기관	개발 목표	개발내용 및 범위
		- 향후 연구 계획 · 가상 디바이스 운용 시나리오 인터페이스와 실 드론과의 동기화 기술 연구 · 가상-실드론 동기화 기반 전체 배송 프로세스 실행 시나리오 개발 · 저/중/고성능 디바이스의 연산 능력 차이에 따른 자율성 증강 기술 연구 **[정량 목표] 가상 자원 관리 성능 (1,000개)** - 실험 목적 (1) 정량 목표 . 정량목표(2022년): 가상 자원 관리 성능 (1,000개) . 최종목표(2025년): 가상 자원 관리 성능 (1,000,000개) (2) 산출 근거 . EdgeCPS 플랫폼과 유사한 엣지 환경에서 동작하는 소규모 클라우드 인프라에서는 수십만대(GE: 20만, 삼성SDS: 10만)의 실디바이스 연동을 요구하며, 5G 이동통신에서도 1㎢당 100만개의 사물 인터넷을 수용하는 능력이 필요하다고 제시 (3) 검증 방법 . EdgeCPS 플랫폼에 1,000개의 가상 디바이스가 접속하는 예제 서비스를 구현 . 접속한 가상 디바이스에 대응되는 1,000개의 디바이스 트윈을 EdgeCPS 플랫폼에 구현 * EdgeCPS 플랫폼 상의 1,000개의 디바이스 트윈은 CPU 자원만 사용하는 것으로 제한 - 시스템 구축 (1) 소프트웨어 . Ubuntu 20.0.04 LTS . Kubernetes v1.21.1 . Docker v20.10.12 . MongoDB v5.0.10 (2) 하드웨어 . 플랫폼 노드 * CPU: Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz * Memory: 64GB * Storage: 2TB * GPU: RTX 2060 Super . 디바이스 노드 (가상 디바이스의 모체가 되는 노드의 실제규격) * Worker#1 (Odroid N2+) CPU: Quad-core Cortex-A73 1.8GHz / Dual-core Cortex-A53 1.9GHz Memory: 4GB * Worker#2 (RaspberryPi 3) CPU: Dual-core Cortex-A53 1.9GHz Memory: 1GB * Worker #3 (RaspberryPi 4) CPU: Quad-core Cortex-A73 1.8GHz Memory: 2GB (3) 목표 검증 응용 . EdgeCPS 기반의 드론 배송 서비스 시제품 * 시제품명: 실디바이스와 디바이스 트윈간 동기화에 기반한 디바이스리스 컴퓨팅 포털 * 자바스크립트 (Apache ECharts, JointJS, Mapbox GL, three.js, Vue.js) . 가상 디바이스의 서비스 요청 패턴 * 1,000개의 드론에 대응되는 EdgeCPS 플랫폼 서비스(Kubernetes Pod)가 실행됨 * 개별 드론은 할당된 주문에 따라 목적지로 이동 후 고객 확인을 위한 EdgeCPS 플랫폼 서비스(얼굴인식)을 요청

수행기관	개발 목표	개발내용 및 범위
		- 실험 결과 (1) 가상 디바이스 생성을 위한 시나리오 . 가상 시험 환경 구성 * 1,000대의 드론이 상품을 고객에게 배송하는 가상의 도시를 구현하고, 10초마다 요청되는 주문(목적지, 고도, 페이로드 특성 차이)에 따라 가장 적합한 드론이 배치되어 배송 임무를 수행하는 환경 구현  그림 269. 가상 디바이스 운용 시나리오 (좌측 드론 기지에서 개별적으로 목적지로 이동) . 가상 디바이스의 생성 * 시나리오 시작 시 1,000대의 드론이 생성되어 개별 임무를 수행하며, 각자 EdgeCPS 플랫폼에 접속하여 서비스를 요청함 * 시나리오 속 EdgeCPS Business Support System 메뉴에서는 각 드론의 정보와 임무를 확인할 수 있으며, 총 1,000대의 드론이 생성된 것 확인 (Drone ID: drone-999) 그림 270. 1,000대의 가상 드론의 생성 확인

수행기관	개발 목표	개발내용 및 범위
		. 디바이스 트윈의 생성 * 가상 디바이스에 대응되는 EdgeCPS 플랫폼에서는 디바이스 트윈(개별 가상 디바이스에 대한 서비스 인스턴스)이 생성됨 * Kubernetes의 PoD 개수 조회를 통해 동시 접속 가상 디바이스 1,000개에 대응되는 Kubernetes PoD(디바이스 트윈)의 개수가 그림과 같이 1,000개까지 늘어난 것을 확인 그림 271. 1,000대의 디바이스 트윈(Pod) 생성 확인
UT달라스 (공동4)	• EdgeCPS 기반 메타버스 환경 구축 및 성능 실험	- 기술 개요 · 연구 정의 : 한국-미국 수준의 장거리 환경에 구축된 EdgeCPS 플랫폼 간 사람과 사물이 디바이스 트윈을 통해 상호작용과 제어를 수행할 수 있도록 양 단의 현실 세계와 이를 복제한 가상 세계가 강하게 동기화된 디지털 트윈 기반 메타버스(MetaTwin)를 정의하고 실제로 한-미간 메타버스 환경을 구축하고, 이를 통한 다양한 상호작용 기법을 연구 - 연구 개발 내용 (1) 디지털 트윈 기반 메타버스(MetaTwin)의 요구사항 정의 (2) 디지털 트윈 기반 메타버스(MetaTwin) 프로토타입 개발 (3) ETRI-UTD 간 MetaTwin 통합 (4) MetaTwin을 통한 종단간 멀티모달 상호작용 수행 - 연구 개발 결과 (성능 평가 혹은 당해연도 연구 결과물) (1) 디지털 트윈 기반 메타버스(MetaTwin)의 요구사항 정의 . 디지털 트윈 기반 메타버스(MetaTwin)를 위한 5가지 구성요소를 그림과 같이 정의함

수행기관	개발 목표	개발내용 및 범위

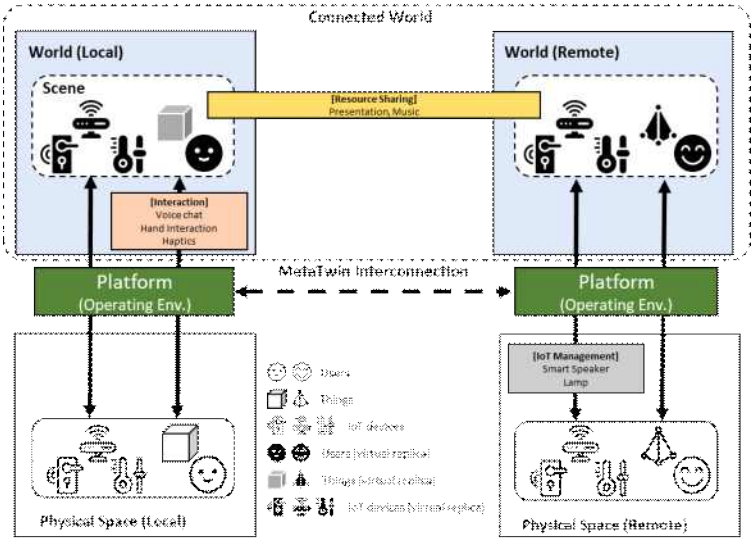


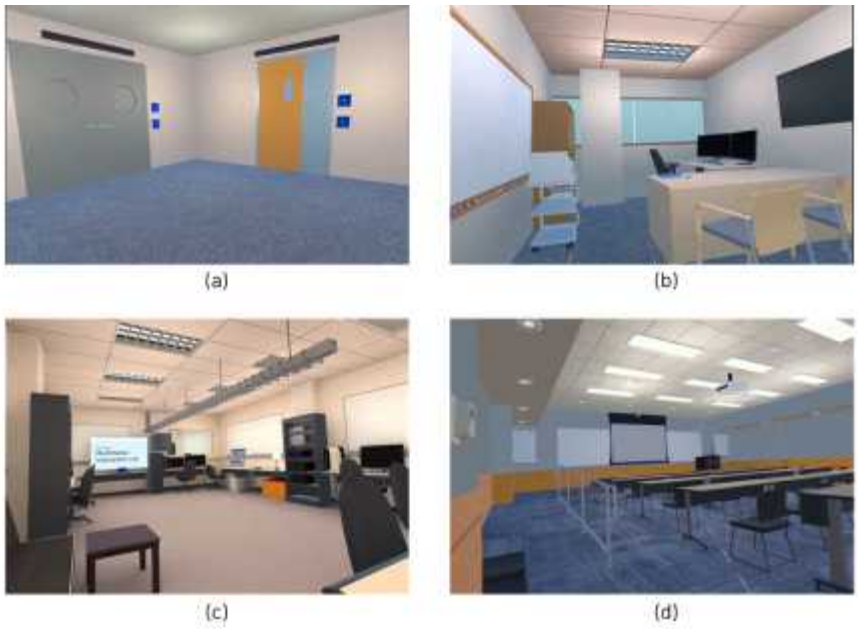
그림 272. 디지털 트윈 기반 메타버스(MetaTwin)의 주요 기술 요소

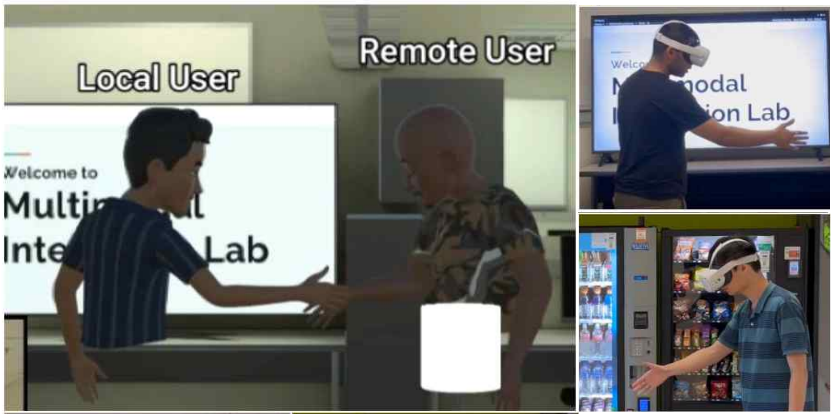
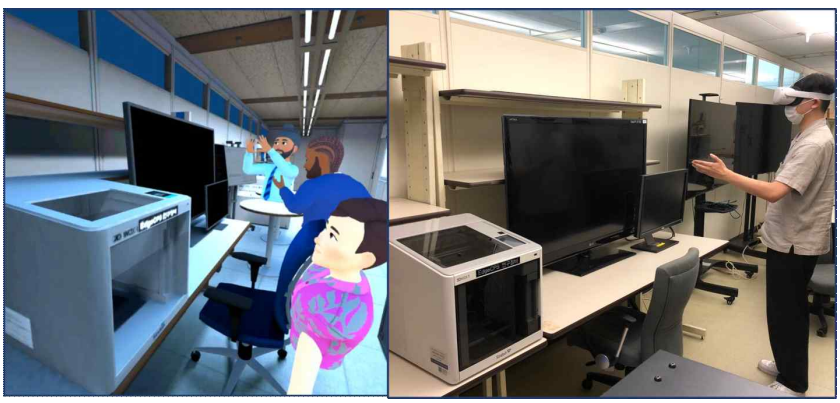
- Platform: MetaTwin을 구동하는 역할 (즉, EdgeCPS 플랫폼), 이를 위해 1) IoT 디바이스의 관리 및 리소스 공유, 2) 가상 세계에서 아바타, 가상 객체에 대한 분석 기술, 3) EdgeCPS간 통신 기술이 필요함
- World: 현실과 동기화된 가상 세계를 의미함. 가상 씬, 아바타, 객체로 구성되며 가상 객체들은 현실 IoT 기기와 연동되어 있어 실감 상호작용이 가능. World 구현을 위해 원격 접속 기술 및 실 객체와 가상 객체의 동기화 기술이 필요함
- Interaction: 아바타-아바타 간 멀티모달 상호작용 기술 (음성, 제스처 등)
- IoT Device Management: MetaTwin에 존재하는 가상 디바이스와 현실의 IoT 기기 동기화 및 제어 기술
- Resource Sharing: 프레젠테이션, 미디어 등 가상 환경 내에서 사용자 간의 컨텐츠 공유 기술

(2) 디지털 트윈 기반 메타버스(MetaTwin) 프로토타입 개발

. ETRI에서는 7연구동 415호 회의실과 L60 실험실을 대상으로 가상 환경을 구축함

그림 273. ETRI 7동 415호 회의실 (가상 공간)

수행기관	개발 목표	개발내용 및 범위
		 그림 274. ETRI 7연구동 L60 회의실 (가상 공간) (3) ETRI-UTD 간 MetaTwin 통합 . UTD Multimodal Interaction Lab의 연구실, 실험실, 강의실 등에 대한 가상 환경을 구축하였고, ETRI와 UTD의 가상환경 통합 및 한-미간 통합 접속, 상호작용 테스트를 실시하였음 . ETRI-UTD 간 MetaTwin은 중앙 Hub를 중심으로 ETRI의 회의실과 실험실, UTD의 교수 연구실, 실험실, 강의실을 자유롭게 넘나들 수 있도록 구축되었음  (a) (b) (c) (d) 그림 275. UTD의 MetaTwin 가상 공간 (a: 허브, b: 교수 연구실, c: 실험실, d: 강의실) . MetaTwin 내의 사용자는 EdgeCPS와 연동된 IoT 기기인 램프, 스피커 등을 켜고 끌 수 있으며, 가상환경 내에 실행되는 미디어(빛, 소리 등)는 사용자 간 공유됨

수행기관	개발 목표	개발내용 및 범위
		 그림 276. 가상 공간을 통해 EdgeCPS와 연동된 IoT 기기에 대한 조작 (4) MetaTwin을 통한 종단간 멀티모달 상호작용 수행 . ETRI와 UTD를 MetaTwin으로 통합하고, 구성원 간에 ETRI와 UTD의 회의실, 실험실을 넘나들며 상호작용에 대한 실험과 공동연구 회의 등을 수행함  그림 277. MetaTwin을 통한 사람 간 멀티모달 인터랙션  그림 278. MetaTwin을 활용한 ETRI-UTD 간 공동연구 회의

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>(6) 국제 우수학술지 2편 작성
. MetaTwin: Synchronizing Physical and Virtual Spaces for Seamless World, ACM VRST 2022
. Data Abstraction for Visual and Haptic Representations in Immersive Analytics, ACM VRST 2022

- 향후 연구 계획
· ETRI-UTD MetaTwin 상에서의 실감형 촉감 인터랙션 기술 연구
· EdgeCPS 간 실감형 촉감 전달을 위한 디바이스 미러링 기술 연구
· 현실 사용자(AR)와 가상 사용자(VR) 간의 MetaTwin 내에서의 상호 인지 기술 연구
· MetaTwin 내에서 데이터의 시촉각적 조작(manipulation) 기술 연구</td></tr>
<tr><td>경북대
(위탁1)</td><td>• 스토리지 증강을 위한 컨테이너 기반 가상화 기술 연구</td><td>- 기술 개요
· 연구 정의 : EdgeCPS 플랫폼에 연동된 저자원 디바이스들의 스토리지 증강을 지원하는 컨테이너 기반 스토리지 가상화 연구를 수행하고, 이를 통해 대규모 가상 스토리지에 대한 동적 관리를 제공

· 연구 목표 :
. 컨테이너 환경의 최신 표준 IO 인터페이스 동향 조사 및 특징 분석
. 동적 자원 할당 기술 및 스토리지 가상화 프로토타입 개발
. 실-가상 디바이스 간의 데이터 공유 및 동기화 모델 설계

- 연구 개발 내용
(1) 컨테이너 환경의 최신 표준 IO 인터페이스 동향 조사 및 특징 분석
. 컨테이너 기반 가상화를 지원하는 Kubernetes의 스토리지 기술에 대한 비교 수행

<table>
<tr><th>Features</th><th>OpenEBS</th><th>Rook (w/ Ceph)</th><th>Longhorn</th></tr>
<tr><td>Provisioning</td><td>Thin-Provisioning</td><td>Thin-Provisioning</td><td>Thin-Provisioning</td></tr>
<tr><td>Replication</td><td>Synchronous Replication</td><td>Multi-site Replication</td><td>• Synchronous Replication
• Auto Balance</td></tr>
<tr><td>Snapshot</td><td>Incremental Snapshot</td><td>Incremental Snapshot (Snapshot Layering)</td><td>• Recurring Snapshot
• Off-site Snapshot</td></tr>
<tr><td>Backup and Restore</td><td>Velero Plugin</td><td>• Incremental Backup
• Disaster Recovery (with Multi-site Asynchronous Replication)</td><td>• Incremental Backup
• Object Storage Backup (NFS, AWS S3)
• Disaster Recovery</td></tr>
<tr><td>Monitoring</td><td>Prometheus Plugin</td><td>Prometheus Plugin</td><td>REST API (Prometheus, Graphite, Telegraf text format)</td></tr>
<tr><td>Dashboard</td><td>Grafana Dashboard</td><td>Ceph Dashboard</td><td>• Longhorn UI
• Grafana Dashboard</td></tr>
</table>
그림 279. Kubernetes 스토리지 기술 비교

. Kubernetes 시스템 구축을 통한 스토리지 가상화 실험 수행

(2) 동적 자원 할당 기술 및 스토리지 가상화 프로토타입 개발
. 개발 환경 : Docker v20.10.17, Kubernetes v1.25.2, Rook v1.9.10, Davfs2 (real device's file system client) v1.6.1, Ceph (server file system) v16.2.10, Nginx-Ingress v1.3.0, helm v3.9.2</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

. Ingress Controller 및 웹 기반의 파일 관리 시스템을 통한 스토리지 가상화 프로토타입 개발

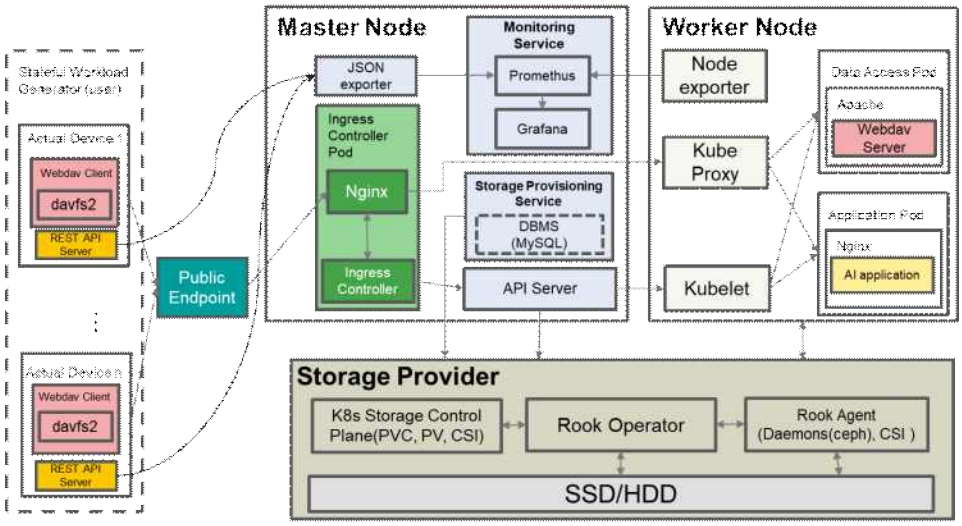


그림 280. 스토리지 가상화 프로토타입 구조

(3) 실-가상 디바이스 간의 데이터 공유 및 동기화 모델 설계

. 실디바이스는 라즈베리파이를 통해 모델링하고, 쿠버네티스의 파드를 통해 가상 디바이스를 모델링 하였음

. 스토리지 동기화는 Webdav 서버를 통해 데이터를 수행 할 수 있도록 설계 함

. 실디바이스에서 가상 디바이스의 webdav 서버로 mount를 요청하고 webdav 서버에서는 user와 password를 확인 후 webdav 서버에 접근이 가능하도록 mount하고 디렉토리를 공유하여 디바이스와 가상 디바이스간의 데이터를 공유함

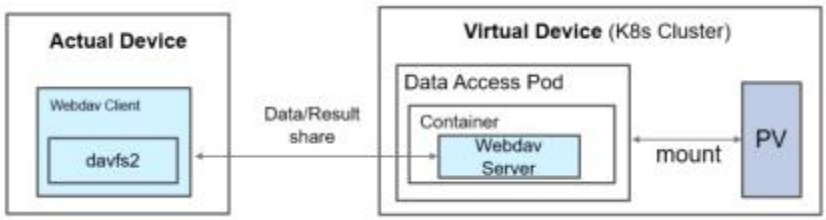


그림 281. 스토리지 동기화 기술 전체 구조도

- 연구 개발 결과 (성능 평가 혹은 당해연도 연구 결과물)

(1) 컨테이너 환경의 최신 표준 IO 인터페이스 동향 조사 및 특징 분석

. 실 디바이스는 라즈베리파이로 구성하고, 가상 디바이스는 쿠버네티스 시스템을 구축하여 스토리지 가상화 개발 환경을 구축

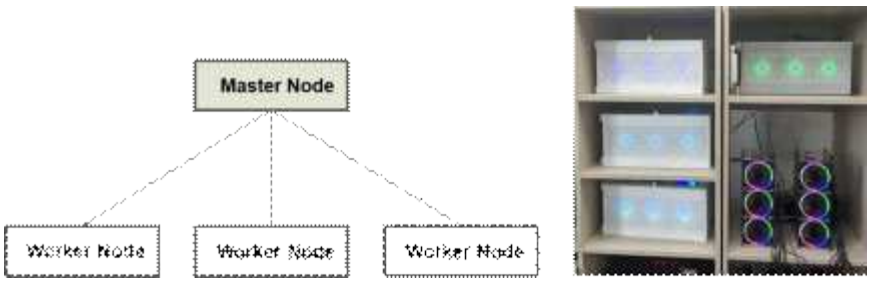


그림 282. 스토리지 가상화 개발 환경 구축 사진

수행기관	개발 목표	개발내용 및 범위

표 28. 소프트웨어 설치 스펙

No	Software Name	Version
1	Docker:	v20.10.16
2	Kubernetes:	v1.22.9
3	Rook:	v1.9.10
4	Ceph:	v16.2.10
5	Davfs2:	v1.6.1
6	Nginx Ingress	v1.3.0
7	Helm	v3.9.2
8	Pvc-autoresizer	v0.5.0
9	Prometheus-json-exporter	v0.5.0
10	Kube-prometheus-stack	v0.58.0

(2) 동적 자원 할당 기술 및 스토리지 가상화 프로토타입 개발

. 실디바이스의 동적 볼륨 프로비저닝을 요청 시 스토리지 가상화가 구현되는 시간 측정

. 실 디바이스 1대일 경우 약 30.18초, 5대의 경우 약 36.59초, 10대 경우 약 39.48초의 latency를 보였으며, 프로비저닝할 디바이스의 규모에 따라 선형적으로 증가하는 경향을 보임

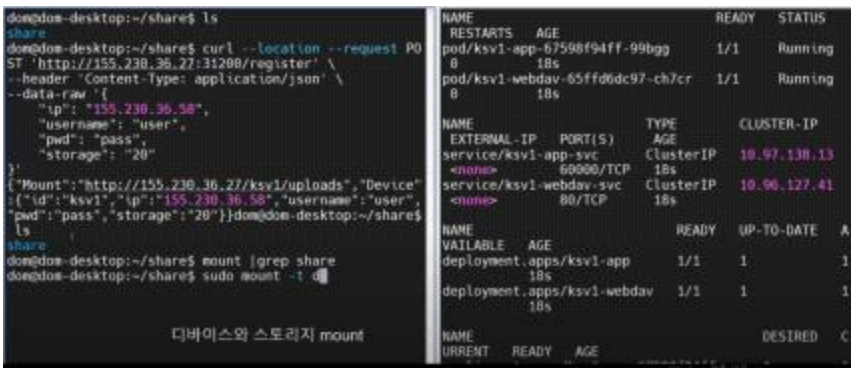


그림 283. 동적 볼륨 프로비저닝 기술 데모 결과

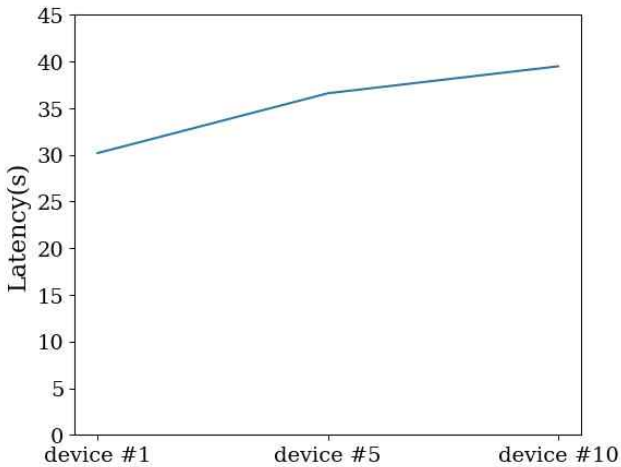
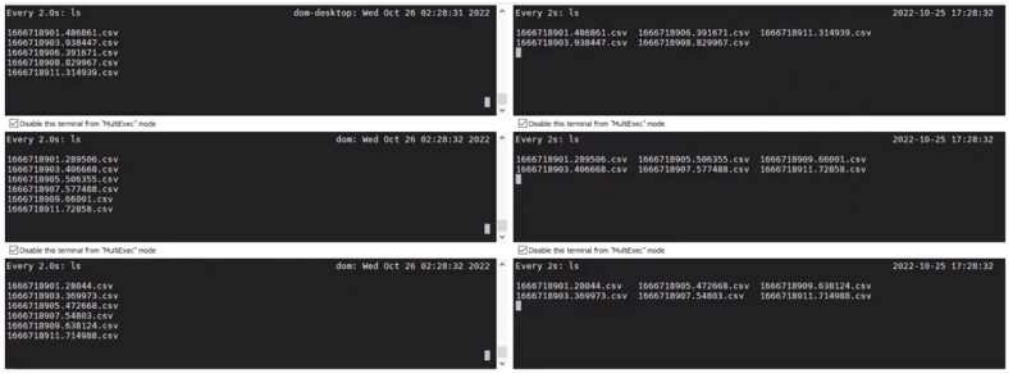
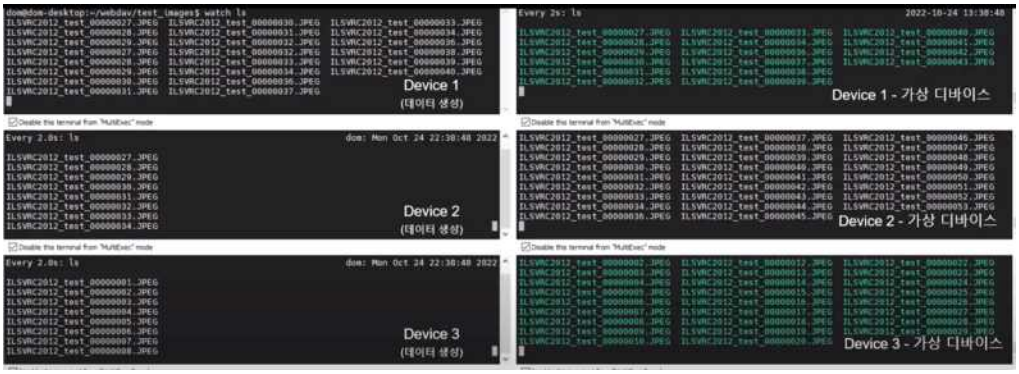
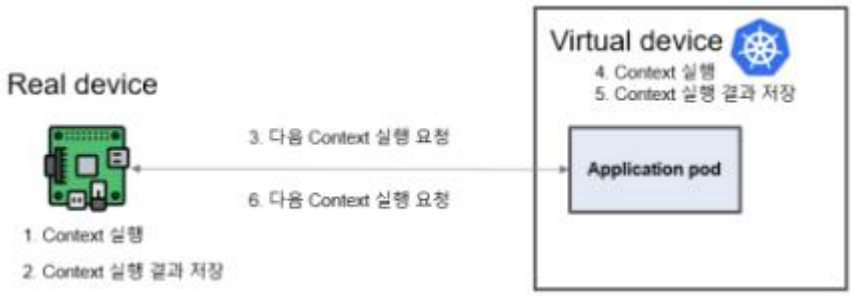
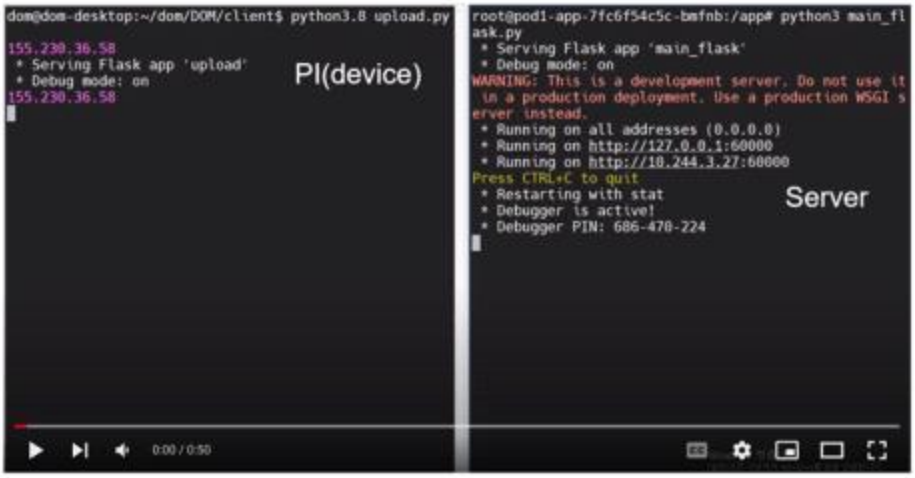


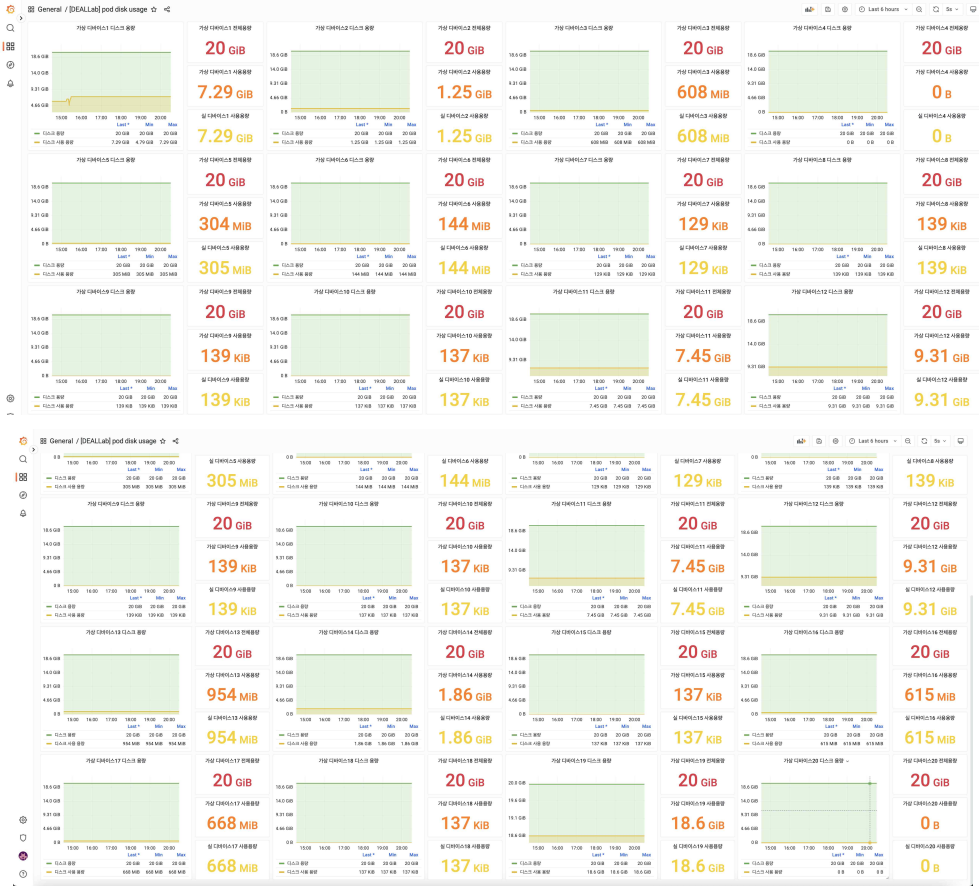
그림 284. 동적 볼륨 프로비저닝 지연 시간

(3) 실-가상 디바이스 간의 데이터 공유 및 동기화 모델 설계

. 실-가상 디바이스간의 단순 데이터 동기화 확인 실험 진행 : 텍스트(CVS)파일 공유, 이미지 공유, 동영상 공유

수행기관	개발 목표	개발내용 및 범위
		 그림 285. 텍스트(CVS) 파일 공유  그림 286. 이미지 파일(jpg) 공유 그림 287. 동영상 파일(mp4) 공유 . 실-가상 디바이스 간의 데이터 공유 및 동기화 실험을 위해 AI 모델 기반의 추론 응용을 수행하고, 실-가상 디바이스의 스토리지의 데이터 공유 기능을 검증

수행기관	개발 목표	개발내용 및 범위
		 그림 288. 실-가상 디바이스의 응용 실행을 통한 데이터 동기화 기술 흐름도  그림 289. 실-가상 디바이스의 응용 실행 환경 동기화 기술 실행 화면

수행기관	개발 목표	개발내용 및 범위
		 그림 290. 실-가상 디바이스 환경의 스토리지 모니터링 화면
경상대 (위탁2)	• 엣지사이버물리스템을 통한 디바이스의 자율성 증강 기술 연구	- 기술 개요 · EdgeCPS 환경에서는 다양한 규격(저성능/중성능/고성능)을 가진 자율형 디바이스가 혼재할 것으로 예상됨 · 본 위탁 연구에서는 대표적인 자율형 디바이스인 UAV(드론)을 활용해 고객에게 상품을 배송하는 시나리오를 가정하고, 미션 수행 중 발생 가능한 2가지 상황으로서 1) 저사양 UAV를 위한 연산 Offloading 기술 연구, 2) 고신뢰 다중 UAV의 데이터 전송을 위한 Network protocol 기술 연구를 수행함 - 연구 개발 내용 (1) 저사양 UAV를 위한 연산 Offloading 기술 (자원 증강 프레임워크 기술) . 아래 그림에서 UAV는 배송지(Drone Base)에서 고객(Recipient)까지 EdgeCPS와의 연결성에 기반해 자율 운행함.

수행기관	개발 목표	개발내용 및 범위

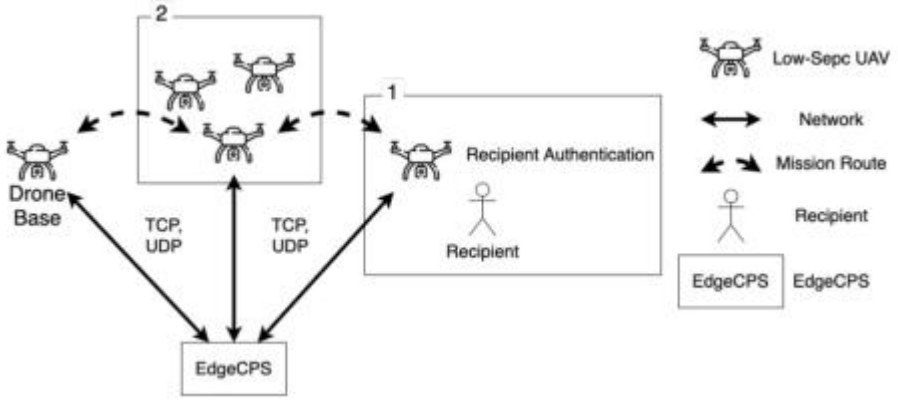


그림 291. EdgeCPS 기반 저사양 UAV의 배송 과정

. 이 과정에서 UAV의 대표적인 임무(Task)는 비행(목적지까지의 이동과 귀환)과 상품 배송이며, 배송 과정에는 고객의 신원 확인과 같은 영상 추론 임무가 발생한다. 영상 추론의 경우 고성능 CPU 자원과 GPU 자원의 활용이 동반되나, 저성능 UAV의 경우 이에 대한 역량 부족으로, EdgeCPS로부터의 자원 증강이 필요함

. 본 연구의 자원 증강 프레임워크 (Resource Augmentation Framework)는 UAV의 성능에 따라 디바이스에서 수행할 추론 부분과 EdgeCPS에서 수행할 추론 부분을 분할해 실행하는 것을 지원함

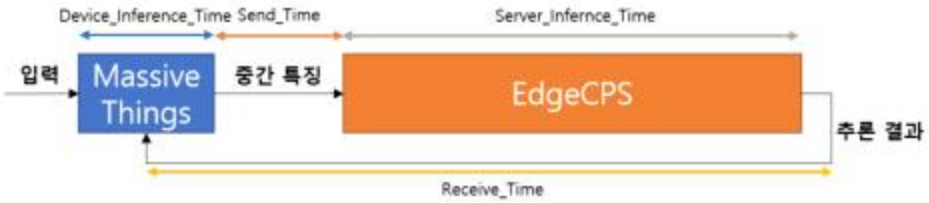


그림 292. 디바이스와 EdgeCPS 간 추론 최적화를 위한 분할 실행

. 또한 디바이스에서의 추론과 오프로딩 프로세스를 순차적으로 진행하면서, 디바이스의 자원을 최대한으로 사용하지 못하는 문제가 발생하여, 추론 프로세스와 오프로딩 프로세스를 파이프라인으로 연결하는 멀티 프로세싱을 진행하였음

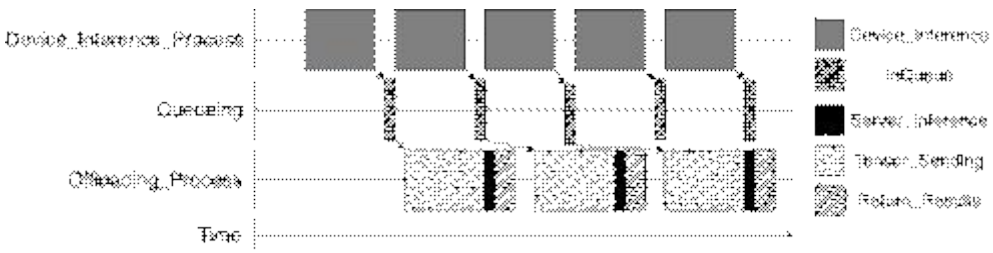
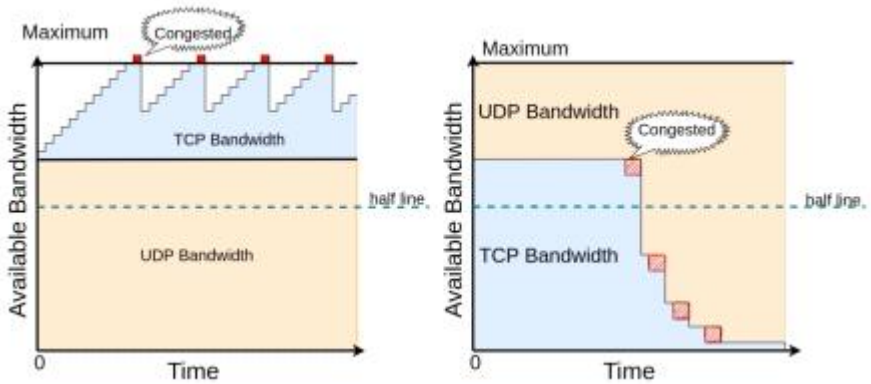
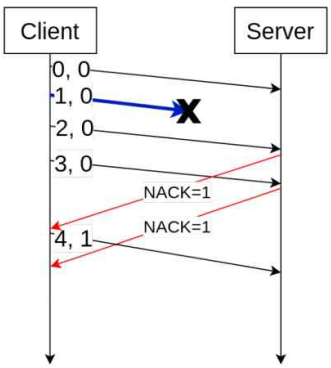
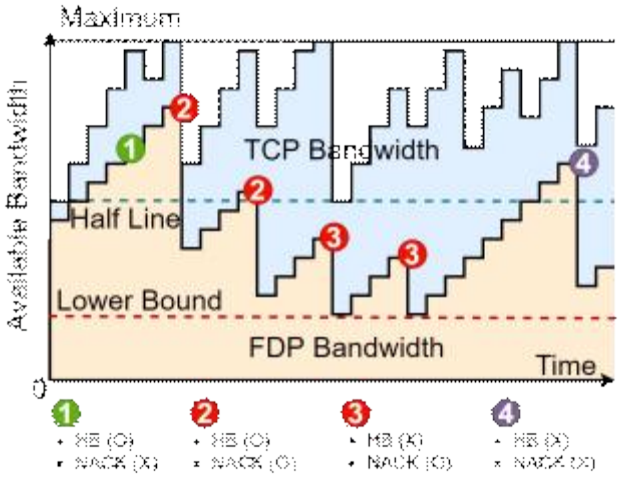


그림 293. 디바이스 추론과 EdgeCPS 추론의 병렬 프로세싱

(2) 고신뢰 다중 UAV의 데이터 전송을 위한 Network protocol 기술 연구

. 상기 UAV를 활용한 배송 시나리오에서 UAV 디바이스가 비행과 관련된 임무와 배송(고객 확인)과 관련된 임무를 수행할 때, 각각의 임무에 대응되는 EdgeCPS 상의 서비스를 위한 프로토콜이 상이함 (비행-TCP, 영상 추론을 통한 고객 확인-UDP)

수행기관	개발 목표	개발내용 및 범위
		. 임의의 영역(ex. 도시)에 다수의 UAV가 존재하는 환경에서 네트워크 대역폭 자원은 유한하며, 혼잡한 네트워크 상황에서 TCP 프로토콜의 congestion control로 인해 TCP 서비스의 대역폭이 부족하게 됨  그림 294. 혼잡한 네트워크에서 TCP와 UDP의 대역폭 소비 패턴 . 본 연구에서는 혼잡 상황 시 UDP 프로토콜도 congestion control을 수행하는 FDP (Fair Datagram Protocol)을 제안함. FDP에서 클라이언트와 서버 간에 packet loss 시 서버는 명시적으로 NACK을 전송하고, 클라이언트는 NACK 수신 시 전송속도를 반으로 감소함  1: **function** CHECHPACKETHEADER(header) 2: **if** *this.seq* = *header.seq* **then** 3: **if** *this.nack_seq* = *header.nack_seq* **then** 4: **return** OK 5: **else** 6: **return** SAME_NACK 7: **end if** 8: **end if** 9: **return** NEW_NACK 10: **end function** 그림 295. FDP의 동작 원리 및 NACK 발행 알고리즘 - 연구 개발 결과 (1) 자원 증강 프레임워크를 통한 분할 추론 성능 평가 . 동일한 추론 워크로드에 대해 Standalone(디바이스에서 전체 추론), BottleNet++ (기존 연구), Partitioning AI (본 연구 #1 / 디바이스와 EdgeCPS 간 분할 추론). Lightweight Encoder (본 연구 #2), Parallelism과 Light Encoder를 접목(본 연구 #3)시켜가며 실행 . 본 연구의 방법론 #1, #2, #3은 Stand alone에 비해 3.7배, 5.26배, 6.26배의 성능 향상 효과가 있으며, BottleNet++에 비해서도 유의미한 성능 향상

수행기관	개발 목표	개발내용 및 범위
		 그림 296. 자원 증강 프레임워크의 분할 추론 성능 비교 (2) 혼잡 네트워크 내에서 다수의 UAV 데이터 전송 대역을 보장하기 위한 FDP(Fair Datagram Protocol) 성능 평가 . FDP의 동작 시나리오 및 이를 검증하기 위한 시뮬레이션 환경 구축 (ns-3)  그림 297. FDP의 동작 시나리오 . 기존의 UDP/TCP 혼재 환경에서의 TCP 대역폭에 비해, FDP/TCP 혼재 환경에서의 TCP가 더 많은 대역폭 확보 (UAV 40대가 TCP 40Mbps, UDP 4Mbps 전송 가정)

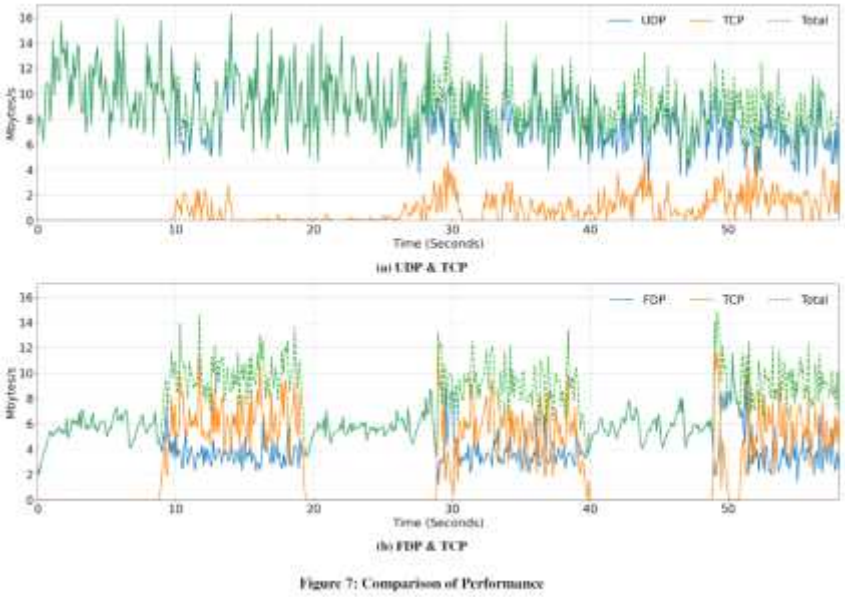
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>


그림 298. 기존 UDP 대비 FDP의 성능 (TCP 대역폭 확보 수준)
</td></tr>
<tr><td>경희대
(위탁3)</td><td>• 엣지사이버물리스템의 예측불가능한 환경 극복을 위한 멀티모달 데이터 모사 알고리즘 개발</td><td>
- 기술 개요
· 지리적으로 떨어진 사물 간의 상호작용은 기본적으로 거리에 따른 네트워크 지연에 영향을 받으며, 그중 촉감을 통한 상호작용은 다른 감각(시각, 청각 등) 대비 가장 네트워크 지연에 민감한 감각으로서, 촉감에 대해 실감적인 상호작용이 가능하다면, 다른 감각에 대해서도 가능한 것으로 평가되고 있음

· 본 위탁 연구에서는 그림과 같이 지리적으로 떨어진 양 단에 각각 배치된 EdgeCPS 플랫폼이 각자의 공간 속 사물에 대한 디바이스 트윈을 형성하고, 이들 디바이스 트윈을 통해 제1환경 (로컬)의 사물이 예측불가능한 제2환경(원격) 환경의 사물에 대해 시촉감을 이용한 관측과 제어 시 네트워크 지연에 따른 영향 분석과 이를 최소화하는 기술을 연구함

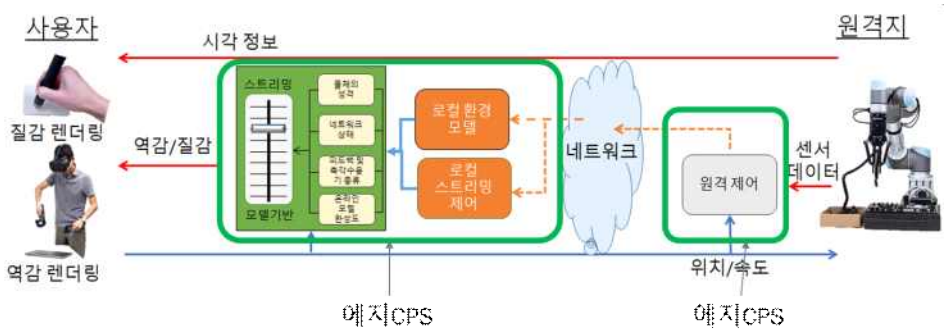


그림 299. EdgeCPS를 통한 양방향 멀티모달 상호작용 개념

- 연구 개발 내용
(1) 질감 모델링 방법 연구
. 지리적으로 떨어진 원격 공간에 존재하는 사물의 시촉감 정보를 획득하기 위해서는 영상 획득 장치(카메라)와 질감 획득 장치(햅틱 장치)가 필요함
. 햅틱 장치를 통해 힘(Force), 속도(Speed), 가속도(Accel) 측정이 가능하며, 획득된 값이 실제 어떤 재질과 일치하는지 비교하기 위해서는 사전에 구축된 질감 데이터베이스와의 비교해 질감을 추정함
</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

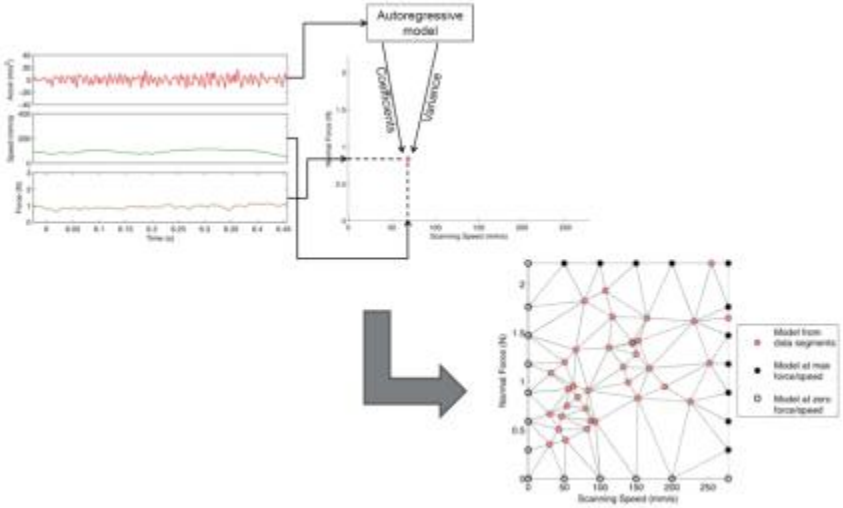


그림 300. 원격지의 힘/속도/가속도 측정값을 질감 데이터와 비교해서 추정

(2) EdgeCPS 기반의 온라인 원격 질감 모델링 방법 확장

. 상기 방법을 온라인으로 수행하기 위해서는 아래 그림의 기존 방법에 대해 적색 표시된 부분의 확장이 필요하며, 수정된 부분은 양 단의 EdgeCPS 플랫폼에서 수행 가능

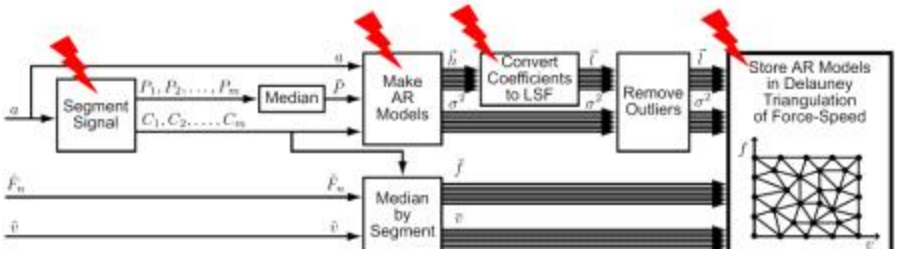


그림 301. 온라인 원격 질감 모델링 방법

(3) 온라인 원격 질감 모델링 환경 설계

. 실제 EdgeCPS를 통해 온라인 원격 질감 모델링을 위해 장치 구성 (3D Systems의 Touch와 TouchX를 각각 로컬과 원격 장치로 활용하고, 양 단의 EdgeCPS 플랫폼에서 응용 구성

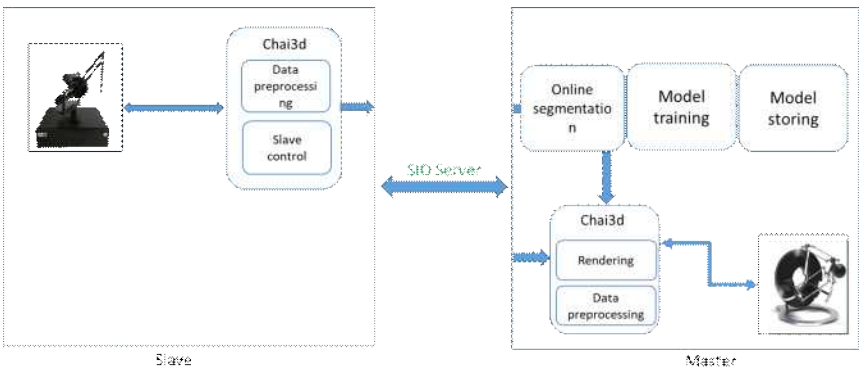
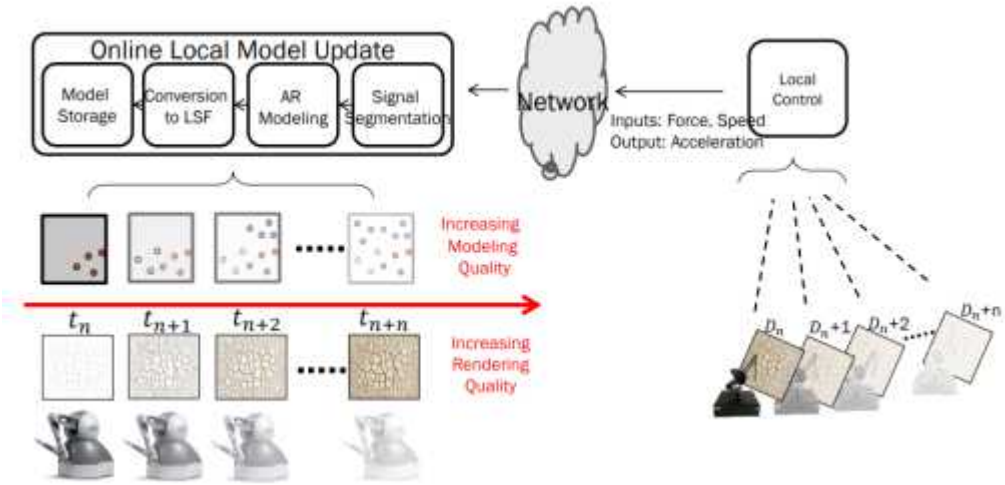
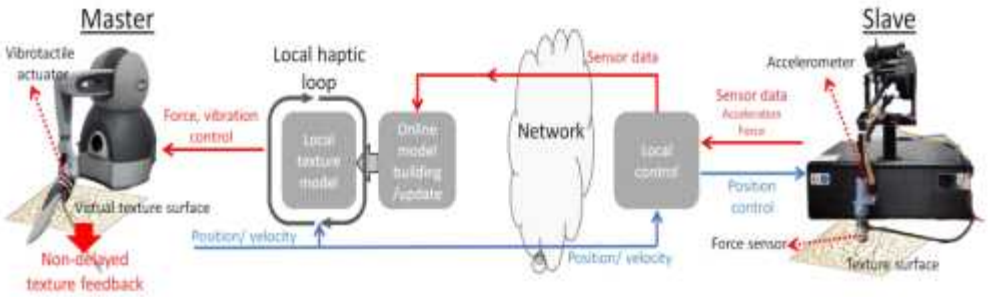


그림 302. 온라인 원격 질감 모델링 장치 설계

- 연구 개발 결과 (성능 평가 혹은 당해연도 연구 결과물)

(1) Model Mediated Teleoperation (MMT) 테스트베드 구축

. 로컬 EdgeCPS 플랫폼에서 동작하는 온라인 모델 업데이트 모듈 및 양 단의 햅틱

수행기관	개발 목표	개발내용 및 범위
		장치로 구성되는 MMT 알고리즘 구현 (그림 144) 및 테스트베드 구축  그림 303. Model Mediated Teleoperation 알고리즘  그림 304. 온라인 원격 질감 모델링 테스트베드 (2) 실시간 촉감 추정 알고리즘 성능 평가 . 원격 장치의 제어를 통해 초반에는 스트리밍을 통해 표면 질감 획득하고, 약 7.5초 이후에는 모델링된 재질 정보만으로 텔레오퍼레이션이 가능함을 확인

수행기관	개발 목표	개발내용 및 범위

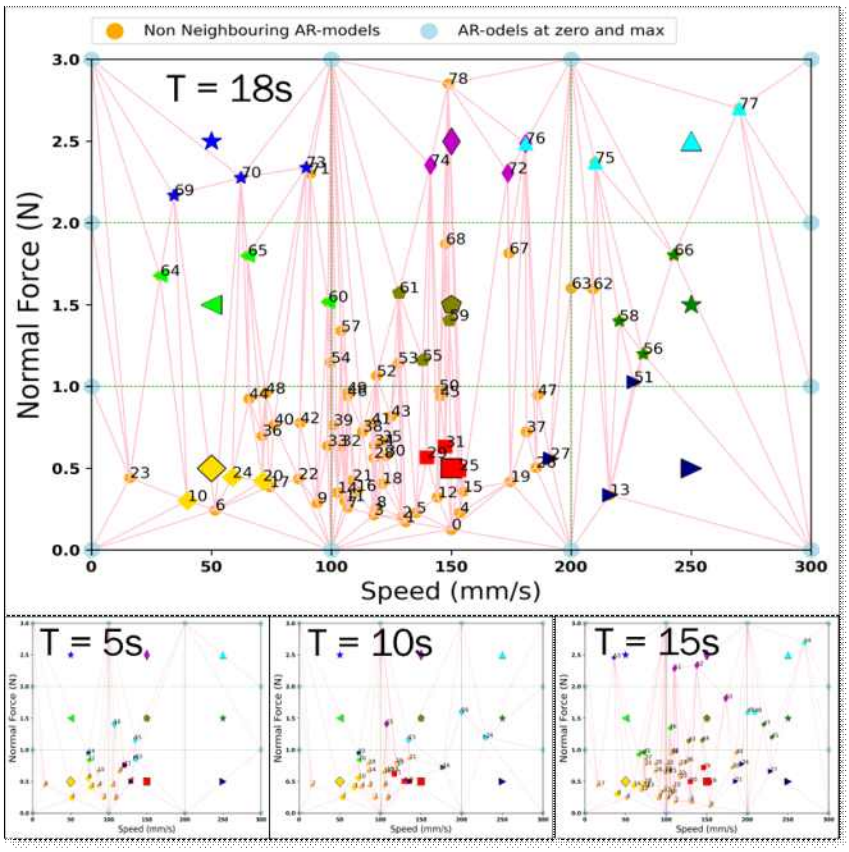


그림 305. 온라인 원격 질감 구축 과정

. 4개의 서로 다른 재질 데이터를 대상으로 실제 측정(스트리밍)한 질감과 로컬 EdgeCPS에 생성된 재질 모델을 통해 렌더링한 질감의 유사도가 높음을 확인

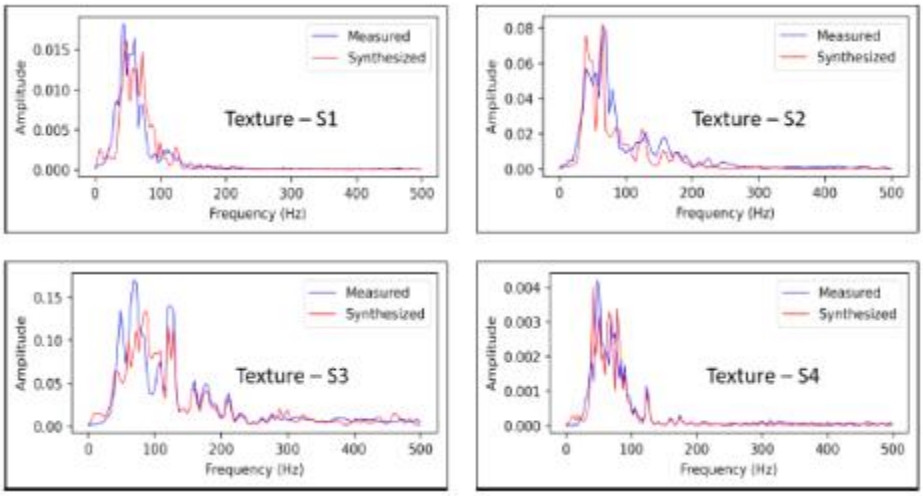


그림 306. 4개의 재질 데이터에 대한 스트리밍과 모델 기반 렌더링 성능 비교

. 성능 지표인 GFC Score의 경우 80이상일 경우 인지적으로 유사하다고 볼 수 있으며, 그림과 같이 4개의 재질 데이터에 대한 스트리밍과 모델 기반 렌더링 성능이 GFC Score 기준 7.5초 이후 80 이상이 됨을 확임. 이는 EdgeCPS 기반 MMT 시스템에서 스트리밍 기반 기법과 모델 기반 렌더링 기법의 선택 기준으로 활용 가능

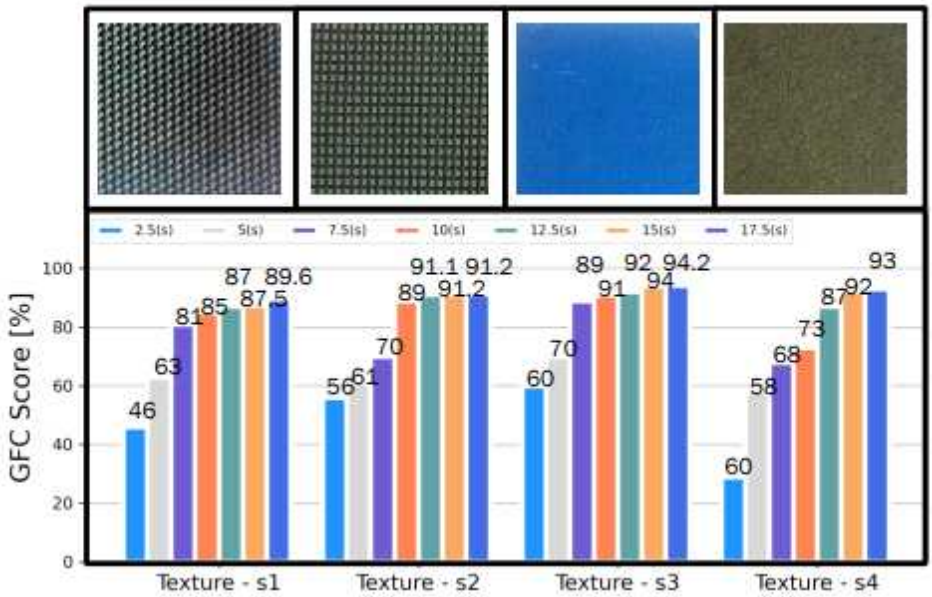
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>


그림 307. 4개의 재질 데이터에 대한 시간별 GFC 값 측정
</td></tr>
<tr><td>인하대
(위탁4)</td><td>• 서비스 요구사항을 고려한 지식그래프 노드간 가중치 도출 알고리즘 연구</td><td>
- 기술 개요
· 연구 정의 : 최적의 AI 응용 서비스 지원을 위해서 학습기 및 추론기의 실행 시에 측정된 성능 정보를 바탕으로 정보 DB를 구축하고, 구축된 정보를 이용하여 그래프 구축기를 개발하여 임의의 시스템에서 최적의 수행환경을 제공할 수 있는 지식 검색 및 추천 알고리즘 개발

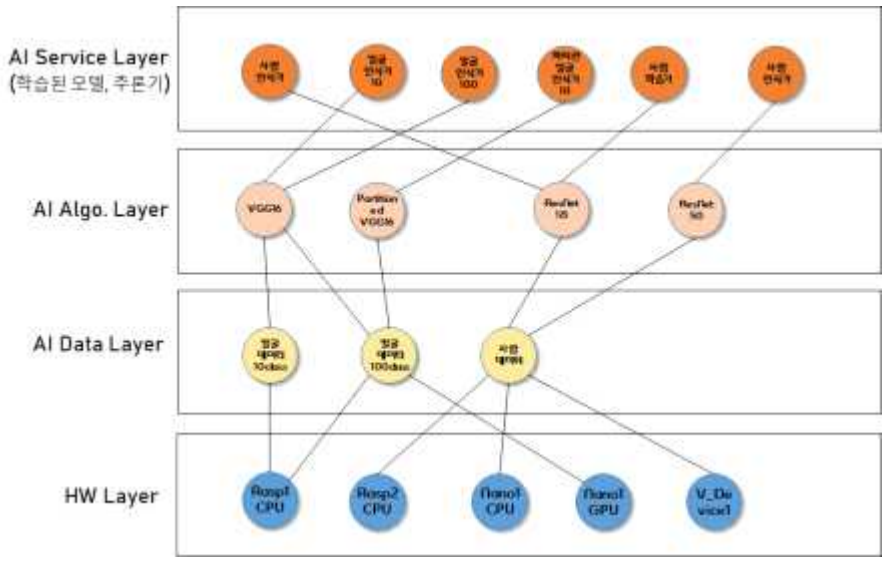
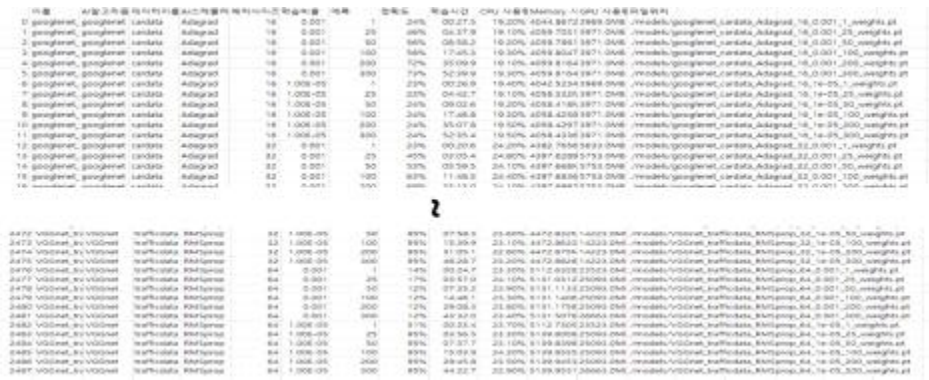
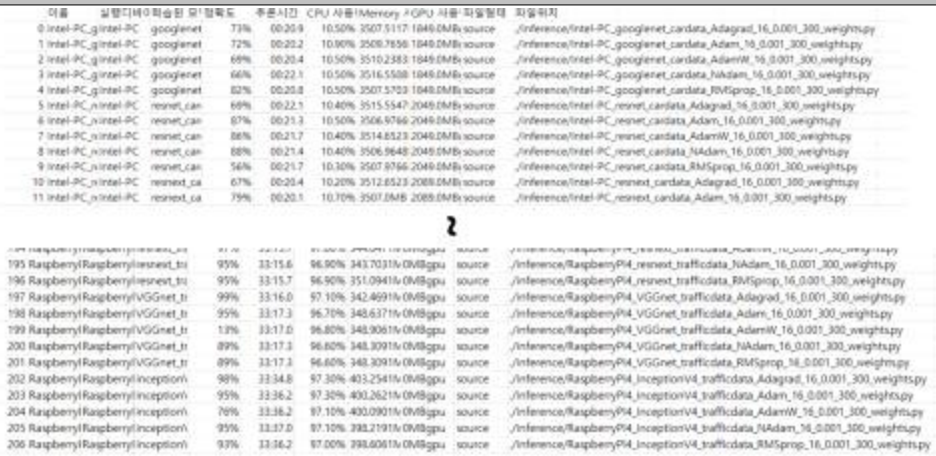
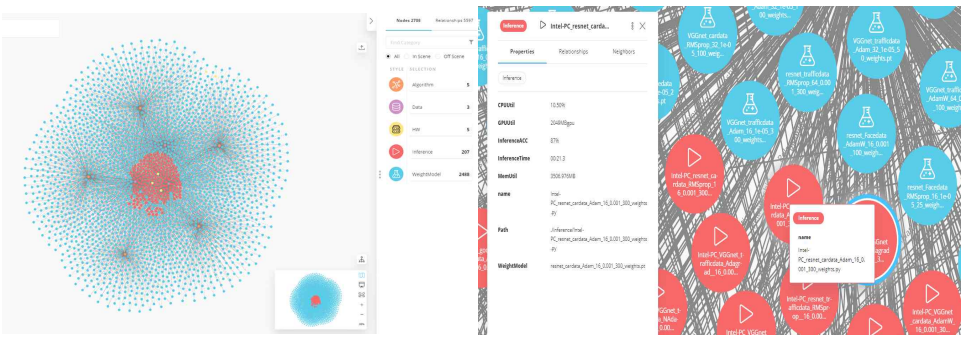
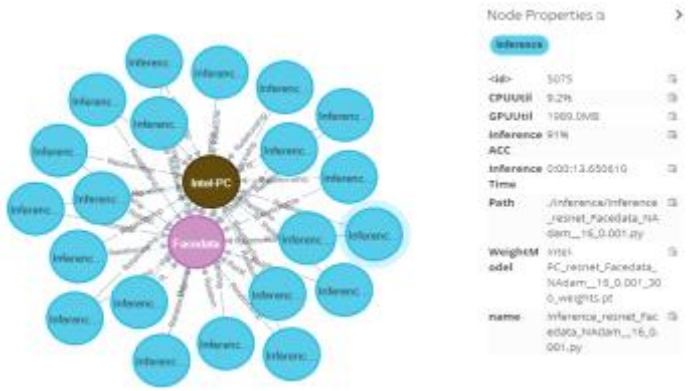
- 연구개발 내용
(1) 정보 공유 모델의 고도화
. 체계적인 정보 공유 시스템 구축을 위해서 HW Layer, AI Data Layer, AI Algorithm Layer, AI Service Layer의 4개 층으로 구성함


그림 308. 정보공유 시스템의 계층
. EdgeCPS 플랫폼에 최적의 AI 응용 지원을 위한 지식 그래프의 노드 및 관계에 대한 속성을 재정의함
</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		● 데이터 • DataName_#ClassNum_Capacity → Name: FaceData_30_0.0125, Cars_7_0.0357, TrafficSign_20_0.195 → Property • 데이터이름 • 클래스수 • 데이터총갯수 • 데이터타입 (image, signal, text, etc) • 데이터크기 (GB) • 데이터경로 ● 학습된 모델 • AI_DataName_AIScheduler_BS_LR_Epoch.pt → Name: VGGnet_Facedata_Adagrad__16_0.001_300_weights.pt → Property: • 이름 • AI알고리즘 • 데이터이름 • AI스케쥴러 • 배치사이즈 • 학습비율 • 에폭 • 정확도 • 학습시간 • CPU 사용량 • Memory 사용량 • GPU 사용량 • 파일위치 ● 추론기 • DeviceName_AI_DataName_AIScheduler_BS_LR_Epoch.py → Name: Intel-PC_VGGnet_Facedata_Adagrad__16_0.001_300_weights.py → Property • 이름 • 실행디바이스 • 학습된 모델 이름 • 정확도 • 추론시간 • CPU 사용량 • Memory 사용량 • GPU 사용량 • 파일형태: source, binary (e.g., docker image) • 파일위치 ● 하드웨어 • DeviceName → Name: Intel-PC → Property • 이름 • CPU(Model, Num, Freq) • GPU (Model, Num, Freq) • Mem(Type, Freq, Capacity) • Storage(Type, Capacity) ● AI 알고리즘 • AI.py → Name: Resnet.py → Property • 이름 • 입력타입 • 모델크기 • 모델경로 6 그림 309. 그래프 노드 및 관계 속성 정의 (2) EdgeCPS 지원 그래프 기반의 정보 시스템 구축 . 정보 시스템 구축 시 아래와 같은 조합으로 가중치 모델 구축 → 하드웨어 : 3종 (Intel-PC, Raspberry PI4, Jetson nano) → AI알고리즘 : 5종 (VGGNet, ResNet, GoogleNet, InceptionNet, ResNext) → 데이터셋 : 3종 (Face_Data, Car_Data, Traffic_sign_data) → 학습 parameter : 4종 (scheduler, Adagrad, Adam, AdamW, NAdam, RMSprop) → Epoch : 6종 (1, 25, 50, 100, 200, 300) → Batch size : 3종 (16, 32, 64) → Learning rate : 2종 (1e-3,1e-5) . 가중치 모델 구축 시 그래프 관계에 필요한 성능 요구사항 (모델 학습 소요 시간, 학습된 모델의 정확도, CPU utilization, GPU utilization, Memory utilization) 추출

수행기관	개발 목표	개발내용 및 범위
		. 가중치 모델을 기반으로 Intel-PC, Raspberry PI4, Jetson nano 3종의 하드웨어에서 추론기 생성 · 추론 시 그래프 관계에 필요한 성능 요구사항인 추론 소요 시간, CPU utilization, GPU utilization, Memory utilization 추출 및 추출된 성능 요구사항을 바탕으로 지식 그래프 구축 시 필요한 그래프 구축기 개발 · 지식 그래프 구축기에는 지식 그래프 생성을 위해 필요한 Cyper 쿼리문 자동 생성 모듈인 MakeCyper, 생성된 Cyper 쿼리문 기반 노드 생성, 제거, 업데이트용 모듈인 MakeNode, Cyper 쿼리문 기반 관계를 생성, 제거, 업데이트용 모듈인 MakeRelation 개발 · 개발된 지식 그래프 구축기를 사용하여 추출된 정보와 가중치 모델로 구성된 지식 그래프 구축 (3) SW명세 질의를 통한 노드 속성 기반 정보 추천 시스템 개발 : 정확도, 추론시간 등을 고려한 Approval Voting 기반 추천 알고리즘 · 개발된 지식 그래프 구축기 중 질의 변환기(MakeCyper)를 사용해 SW 명세에서 추추된 키워드로 cyper 질의문 생성 및 검색 기술 개발 · 지식 그래프에서 검색하여 생성된 서브 그래프의 노드, 속성을 기준으로 Approval Voting을 적용하여 평점을 매김 · 속성 값이 평균을 넘는 경우 1, 아니면 0 값을 부여하고 해당 노드의 값은 평균을 내애서 노드에 대한 순위를 결정함. - 연구개발 결과 (1) 총 2487개의 가중치 모델을 구축했으며, 가중치 모델 1개당 300epoch 기준으로 학습했으면, 평균 학습시간은 100분정도 소요되었으며, 총 용량은 1.5TB 정도임 (가중치모델의 모든 성능 정보 --> metafile.csv로 데이터화)  그림 310. metafile.csv 내용 (2) Intel-PC, Rasberry PI4, Jetson Nano, OrangPI4, Kandas 5종의 이기종 기기에서 총 206개의 가중치모델기반의 추론기를 구축했음 (inference_metafile.csv로 데이터화)

수행기관	개발 목표	개발내용 및 범위
		 그림 311. inference_metafile.csv 내용 (3) 모델의 성능 정보와 지식 그래프 구축기를 사용하여 AI 응용 서비스를 위한 지식 그래프를 구축함  그림 312. AI 응용 서비스를 위한 지식 그래프 (4) 하드웨어, 정확도, 학습데이타, 추론시간 등을 고려한 Approval Voting 기반 추천 알고리즘 · SW 명세 키워드를 이용한 질의 변환기인 MakeCyper로 질의어 생성함 생성된 Cypher: MATCH (n:HW{name:'Intel-PC'})-[]-(m:Inference)-[]-(k:Data{name:'Facedata'}) where toInteger(left(m.InferenceACC,2))>80 RETURN m 그림 313. MakeCyper로 생성된 질의어 · 생성된 질의어를 통한 지식 그래프에서 서브 그래프를 생성하고 순위를 정함

수행기관	개발 목표	개발내용 및 범위
		 그림 314. 생성된 질의어를 통한 지식 그래프

ETRI (주관)

. 저저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 핵심 기술 개발

. 엣지사이버물리 시스템을 위한 대규모 디바이스 제어 기술 개발

전인걸

UT달라스 (공동4)

. EdgeCPS 기반 메타버스 환경 구축 및 성능 실험

경북대 (위탁1)

. 스토리지 증강을 위한 컨테이너 기반 가상화 기술 연구

경상대 (위탁2)

. 엣지사이버물리스템을 통한 디바이스의 자율성 증강 기술 연구

경희대 (위탁3)

. 엣지사이버물리스템의 예측불가능한 환경 극복을 위한 멀티모달 데이터 모사 알고리즘 개발

인하대 (위탁4)

. 서비스 요구사항을 고려한 지식그래프 노드간 가중치 도출 알고리즘 연구

(4) (2단계) 2차년도(2023년도) 개발 내용 및 범위

[패브릭 메모리]

수행기관	개발 목표	개발내용 및 범위
ETRI (주관)	① 페타바이트급 주소 공간 관리 기술 연구	• 384TB급 패브릭 메모리 시스템 구축 (정량목표) - 대규모 패브릭 메모리 시스템 구축이 가능함을 입증하기 위해, 본 연구과제에서 개발한 기술을 바탕으로 384TB 규모의 메모리를 갖는 가상머신 생성 - Linux QEMU/KVM를 대규모 메모리를 지원하도록 확장 · 단일 메모리 영역 용약 8TB 한계 → NUMA 지원하도록 EMP 확장. 4TB씩 96개 NUMA 노드 생성하여 384TB 달성 · QEMU 내부 BIOS 구현상 이슈로 384TB 규모의 가상 메모리 모듈 장착 불가 → 가상 메모리 모듈 당 용량을 16GB에서 128GB로 확장하여 384TB 규모 장착 가능케 함 · 가상머신에 대한 EMP의 메타데이터 크기 축소를 위해 가상머신 첫 2MB 영역을 local memory로 고정시키고, 전체 영역의 subblock 크기를 최대치인 128KB로 설정 · 5-level EPT 사용을 위해 커널 및 QEMU 버전 업그레이드: Rocky Linux 9.2용 커널 5.14.0-284.30.1.el9_2.x86_64, QEMU 8.1.2 - KVM 주요 메타데이터 크기 분석 · rmap: reverse mapping 정보. 페이지당 8바이트씩 두 개. 1TB 당 4GB 필요 · gfn track: 쓰기 추적 정보. 페이지당 2바이트씩 두 개. 1TB당 1GB 필요 · rmap과 gfn track만으로도 384TB 메모리 영역을 위해 1,920GB 필요 · shadow page table 등은 local cache 용량에 비례하는 것으로 추정됨 - KVM 주요 메타데이터 관리를 위해 DCPMM을 활용한 컴퓨팅 노드 메모리 확장 · 1TB DRAM(64GB*16EA): EMP 메타데이터, 페이지테이블, 로컬 캐시 등으로 사용 · 1TB DCPMM(128GB*8EA): KVM 1번 memslot의 rmap과 gfn track 관리 · 1TB DCPMM(128GB*8EA): KVM 2번 memslot의 rmap과 gfn track 관리 - 384TB 규모의 확장 메모리 용량 확보를 위해 NVMe Over Fabric 기술 활용 · 컴퓨팅 노드: 7.68TB U.2 SSD * 24EA · 메모리 노드1: 7.68TB U.2 SSD * 24EA (NVMe Over Fabric으로 연결) · 메모리 노드2: 15.36TB U.2 SSD * 8EA (NVMe Over Fabric으로 연결) · NVMe Over Fabric 사용을 위해 MLNX_OFED_LINUX-5.8 설치 - 하드웨어 구성 · 컴퓨팅 노드 ◦ CPU: Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz X 2개 (총 112코어) ◦ DRAM: 1024GB (64GB*16EA) ◦ DCPMM: 2048GB (128GB*16EA) ◦ 메모리확장용 고성능 SSD: Intel SSD D7-P5510 (7.68TB) *24EA ◦ 인터커넥트 네트워크: Infiniband HDR · 메모리 노드1 ◦ CPU: Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz X 2개 (총 112코어) ◦ DRAM: 1024GB (32GB*32EA) ◦ 메모리확장용 고성능 SSD: Intel SSD D7-P5510 (7.68TB) *24EA ◦ 인터커넥트 네트워크: Infiniband HDR · 메모리 노드1 ◦ CPU: Intel(R) Xeon(R) Gold 6448Y CPU @ 2.10GHz X 2개 (총 128코어) ◦ DRAM: 1024GB (32GB*32EA) ◦ 메모리확장용 고성능 SSD: Intel SSD D7-P5520 (15.36TB) * 8EA ◦ 인터커넥트 네트워크: Infiniband HDR - 가상머신 구축 및 부팅 · 가상머신 설정: CPU: 64코어, 메모리: 384TB · 부팅 중 System RAM 402653183MB (=384TB) 생성 확인 · 정량 목표 달성: 48TB의 메모리를 장착한 가상머신 생성 성공

1) 본 보고서에서 paging과 swapping은 같은 의미로 사용함.

2) "Large-scale Graph Processing on Commodity Systems: Understanding and Mitigating the Impact of Swapping" MEMSYS 2023

수행기관	개발 목표	개발내용 및 범위

```
[    0.000000] NODE_DATA(80) allocated [mem 0x144003ffd5000-0x144003fffffff]
[    0.000000] NODE_DATA(81) allocated [mem 0x148003ffd5000-0x148003fffffff]
[    0.000000] NODE_DATA(82) allocated [mem 0x14c003ffd5000-0x14c003fffffff]
[    0.000000] NODE_DATA(83) allocated [mem 0x150003ffd5000-0x150003fffffff]
[    0.000000] NODE_DATA(84) allocated [mem 0x154003ffd5000-0x154003fffffff]
[    0.000000] NODE_DATA(85) allocated [mem 0x158003ffd5000-0x158003fffffff]
[    0.000000] NODE_DATA(86) allocated [mem 0x15c003ffd5000-0x15c003fffffff]
[    0.000000] NODE_DATA(87) allocated [mem 0x160003ffd5000-0x160003fffffff]
[    0.000000] NODE_DATA(88) allocated [mem 0x164003ffd5000-0x164003fffffff]
[    0.000000] NODE_DATA(89) allocated [mem 0x168003ffd5000-0x168003fffffff]
[    0.000000] NODE_DATA(90) allocated [mem 0x16c003ffd5000-0x16c003fffffff]
[    0.000000] NODE_DATA(91) allocated [mem 0x170003ffd5000-0x170003fffffff]
[    0.000000] NODE_DATA(92) allocated [mem 0x174003ffd5000-0x174003fffffff]
[    0.000000] NODE_DATA(93) allocated [mem 0x178003ffd5000-0x178003fffffff]
[    0.000000] NODE_DATA(94) allocated [mem 0x17c003ffd5000-0x17c003fffffff]
[    0.000000] NODE_DATA(95) allocated [mem 0x180003fb40000-0x180003fb6afff]
[    0.000000] Using crashkernel=auto, the size chosen is a best effort estimation.
[    0.000000] Reserving 512MB of memory at 2544MB for crashkernel (System RAM: 402653183MB)
[    0.000000] Zone ranges:
[    0.000000]   DMA      [mem 0x0000000000001000-0x0000000000ffffff]
[    0.000000]   DMA32    [mem 0x0000000001000000-0x00000000ffffffff]
[    0.000000]   Normal   [mem 0x0000000100000000-0x000180003fffffff]
[    0.000000]   Device   empty
```

그림 315. 가상머신 부팅 화면

• 프로세스간 분리 메모리 메타데이터 공유 기법 개발

- 본 과제의 분리 메모리 시스템은 가상머신을 기반으로 개발되어 있었으며, 두 가상머신이 메모리를 공유하는 것은 가정하지 않음. 이 경우 가상머신 시작시 한 번의 메모리 영역 생성이 일어나고 가상머신 종료시 그 메모리 영역이 삭제되는 것만이 일어남
- 네이티브 응용은 메모리 영역 생성이 실행 중간에 일어날 수도 있고, fork가 되면서 여러 프로세스가 메모리 영역을 공유하는 일도 있을 수 있으며, 실행 도중에 메모리 영역이 삭제 되기도 함. 이런 시나리오를 지원하기 위해서는 분리 메모리 시스템의 메타데이터의 생성, 공유, 삭제가 모든 시점에 가능하도록 다시 설계해야 함
- 본 과제에서는 아래와 같은 구조로 분리 메모리 메타데이터 구조를 설계
  - · 메타데이터의 디렉토리를 설계. 디렉토리는 배열 형식으로, 각 엔트리는 포인터 하나만을 포함하는 8바이트 크기. 메타데이터를 접근할 때는 디렉토리를 접근하여 포인터를 얻어낸 뒤 간접적으로 접근
  - · 메모리 영역을 생성할 때 메타데이터를 바로 생성하지 않고, 해당 엔트리에 해당하는 주소 영역을 접근할 때 메타데이터를 생성한 뒤, 생성한 메타데이터의 포인터를 디렉토리에 기록. 디렉토리 각 엔트리의 초기값은 NULL 포인터
  - · 각 메타데이터에 참조카운터(reference count)를 추가. 하나의 메타데이터를 여러 디렉토리에 속할 수 있음. 즉, 여러 메모리 영역에 속할 수 있으며, 몇 개의 메모리 영역에 속하였는지를 참조카운터로 표기
- 위와 같은 메타데이터 구조의 장점을 아래와 같음
  - · 메모리 영역 생성 속도가 빨라짐. 서브블록당 32~64바이트 크기의 메타데이터를 생성하고 초기화하는 기존 방식에 비해, 서브블록당 8바이트 크기의 디렉토리만을 생성하면 되며, 초기값이 NULL, 즉 0값이므로 vzalloc() 등을 통해 생성할 수 있으므로 디렉토리 생성 시간도 매우 단축됨
  - · 메타데이터 단위의 생성, 공유, 삭제가 가능. 디렉토리는 메모리 영역별로 존재하지만 메타데이터는 여러 메모리 영역에 공유될 수 있음. 이를 통해 포크(fork) 시에 메타데이터를 공유하는 방식으로 데이터 복제 없이 처리 가능

수행기관	개발 목표	개발내용 및 범위

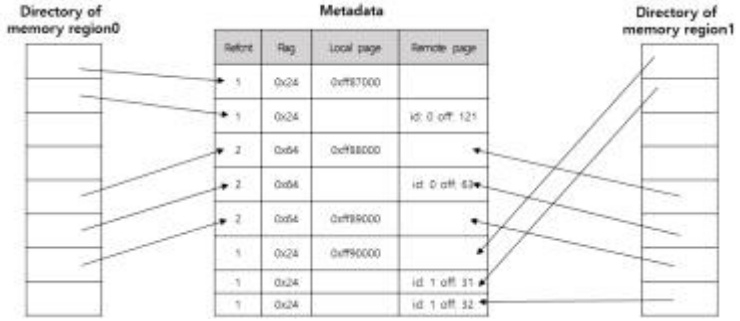


그림 316. 메타데이터 공유를 고려한 메타데이터 구조 설계

• 분리 메모리 환경에서 포크(fork) 지원을 위한 쓰기시복사(CoW, Copy-on-Write) 처리기술

- 프로세스 생성(fork)시 프로세스 메모리 공간의 CoW 지원 기술 설계 및 구현 완료
- CoW 지원은 포크 처리 지연시간을 최소화할 수 있어 성능상 이점이 있으나, 분리 메모리 시스템에서는 프로세스가 사용하는 메모리 공간이 로컬 메모리와 원격 메모리에 흩어져 있기 때문에 이를 관리할 수 있는 추가 기능 설계 및 구현이 필요함
- 분리 메모리 시스템에서 쓰기 요청이 발생했을 때, 쓰기 요청이 발생한 페이지에 대해 관련 프로세스 및 메모리 위치(로컬/원격)를 기준으로 5가지 케이스로 분류함
  · 1) 현재 프로세스와 다른 프로세스 주소공간에 공유하여 매핑되어 있는 경우
  · 2) 현재 프로세스에만 매핑되어 있는 경우
  · 3) 현재 프로세스가 아닌 다른 프로세스들에만 매핑되어 있는 경우
  · 4) 매핑된 프로세스가 없고 로컬 메모리에 있는 경우
  · 5) 매핑된 프로세스가 없고 원격 메모리에 저장되어 있는 경우
- 각 상황에 맞도록 mmu_notifier 및 쓰기 페이지 폴트를 활용하여 CoW를 지원하기 위한 동작 루틴을 설계하고 구현함
  · 1) 2)는 mmu_notifier를 통해, 3)-5)는 쓰기 페이지 폴트 루틴 내부에서 처리
  · 아래 그림은 각 경우에 대한 CoW 처리 루틴의 상세 설계도를 도식화한 것임

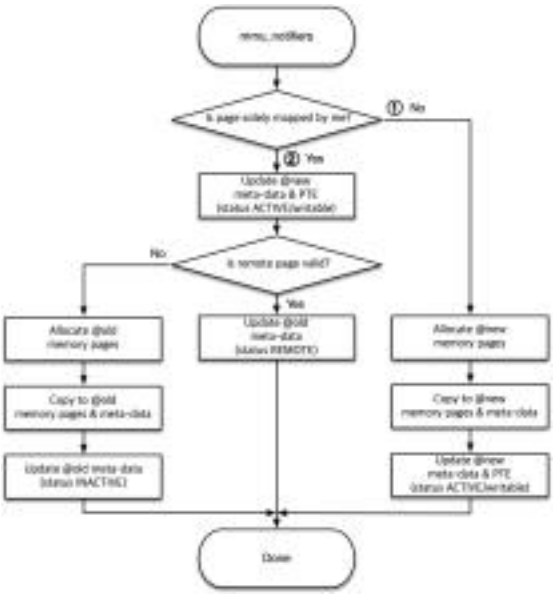


그림 317. mmu_notifier를 통한 CoW 처리 루틴

수행기관	개발 목표	개발내용 및 범위

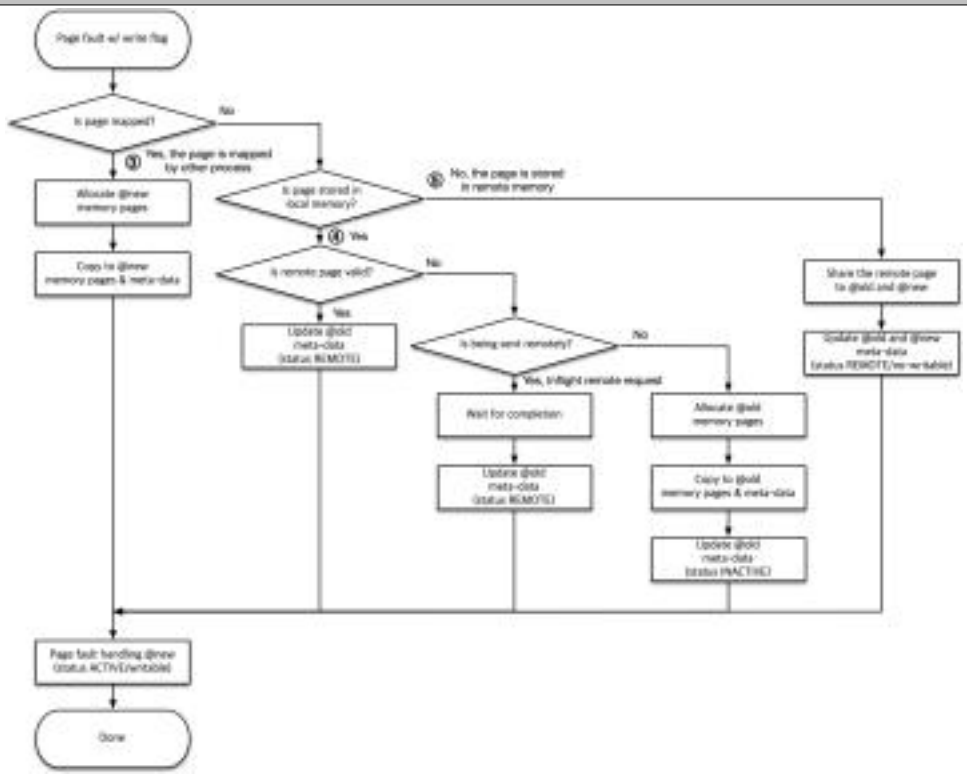


그림 318. 쓰기 페이지 폴트를 통한 CoW 처리 루틴

• 분리 메모리 환경에서의 성능 개선 기법 설계

- 분리 메모리 환경에서 원격 메모리로의 통신 성능 개선 기술 연구

· 공간적 지역성을 활용하여 큰 단위로 페이지들을 관리하고 I/O를 수행할 수 있도록 서브블록 기법을 제안함. 해당 기법은 RDMA 통신의 소프트웨어 오버헤드가 고정적으로 포함된다는 사실을 기반으로 큰 단위로 RDMA 통신 I/O를 수행하는 기법임. 이를 구현하는 방법에는 RDMA 프로토콜의 scatter-gather 기법이 존재하지만, 해당 기법은 scatter-gather 엔트리에 페이지 단위로 주소를 기입해야 하므로 세부적인 관리가 필요함. 따라서, 본 연구에서는 그림 319과 같이 리눅스 시스템의 컴파운드 페이지(compound page)를 활용하여 물리적으로 연속된 페이지를 할당하여 사용하고, 해당 연속된 페이지 단위로 I/O를 수행하는 방법을 제안함

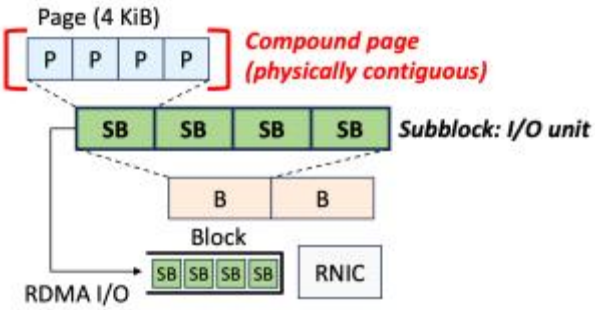


그림 319. 서브블록 기법

· 큰 단위의 I/O를 수행하게 되면 해당 페이지폴트에 필요한 페이지와 그렇지 않은 프리페치 페이지가 함께 전송되는데, 이에 대한 처리를 수행하게 되면 필요 페이지를 처리하는 시간에 비해서 많은 시간이 걸리게 됨. 따라서, 필요 페이지를 처리하고 프리페치 페이지에 대해서는 비동기식으로 처리하는 방법이 필요함. 본 연구에서는 두 가지 최적화 기법인 필요 서브블록 우선처리 기법 (CSF – Critical Subblock First), 필요 페이지 우선처리 기법 (CPF – Critical Page First)을 통해서 해당 문제를 해결함

수행기관	개발 목표	개발내용 및 범위
		· 그림 320는 CSF/CPF를 위한 I/O 큐와 각 기법의 적용으로 인한 페이지폴트 처리 지연시간의 감소를 표현한 그림임. 이때, CPF의 경우 I/O단위의 완료 확인이 아닌 그보다 작은 페이지 단위의 완료 확인이 필요함. 따라서, 데이터를 전송받는 메모리 버퍼의 각 페이지에 마지막 word에 서명을 써두고 해당 서명이 바뀌었으면 페이지가 전송이 완료되었음을 확인함. 그림 321은 필요 페이지 우선처리 기법에서 페이지의 전송완료를 확인하는 알고리즘을 나타냄

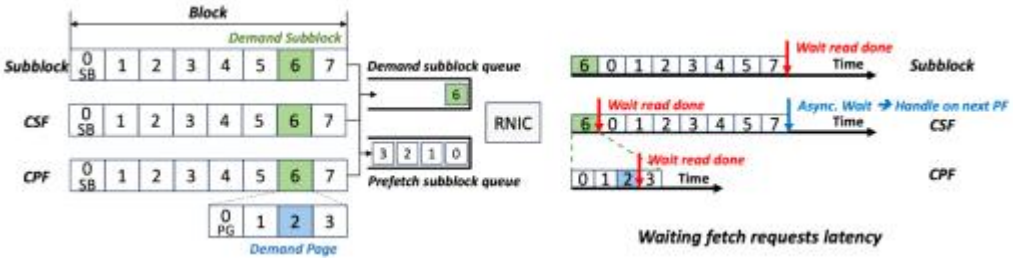


그림 320. CSF/CPF를 위한 I/O 큐 및 페이지 폴트 지연시간

Algorithm 1 Fetching a target page in DEHype

Input: *page*: target page to fetch in a subblock
Output: How the fetch is processed (CSF or CPF)

```
procedure FETCHING TARGET PAGE(page)
    retry = 0
    addr = the address of the last word in page
    for retry < RETRY_COUNT do
        value = read a word at addr
        if value <> signature then
            return Handled with CPF
        end if

        Yield CPU()
        retry++
    end for

    /* Fallback to the slow path */
    Wait for the I/O work completed()
    return Handled with CSF
```

그림 321. CPF의 페이지 전송완료 확인 알고리즘

- 분리 메모리를 지원하는 하이퍼바이저 개선 기술 연구

· 가상화 시스템의 하이퍼바이저가 분리 메모리 환경을 인지하지 못하여 발생하는 문제를 해결함. 분리 메모리 환경에서는 가상머신의 메모리에 매핑되어있는 호스트 메모리가 원격 메모리와의 잦은 통신으로 인해서 자주 언맵되는 경우가 발생함. 잦은 언맵은 리버스맵을 순회하고 동기화 문제로 인해서 성능 저하를 야기시킴. 따라서, 해당 문제를 최소화하기 위해서 물리적으로 연속적인 페이지에 대한 언맵을 수행하고, 해당 언맵 수행시 리버스맵의 리스트 헤드가 연속적임을 활용하여 한번에 배치로 언맵을 수행하는 기법을 제안. 그림 322은 배치 언맵 기법을 나타내는 그림으로 연속적인 리스트 헤드를 0으로 클리어시킴으로써 언맵을 수행

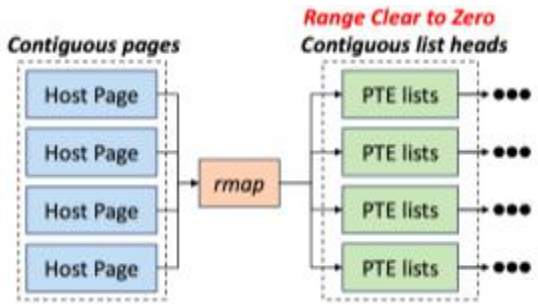


그림 322. 배치 언맵 기법

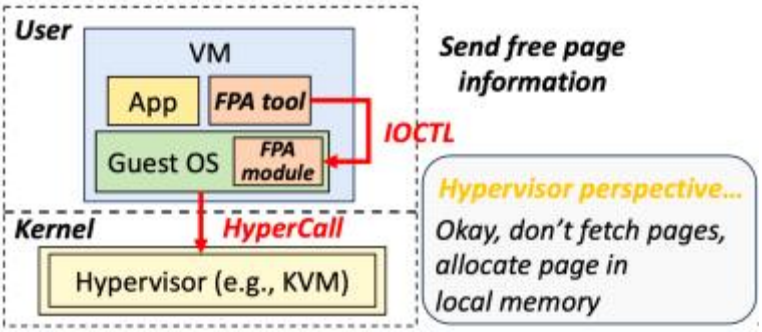
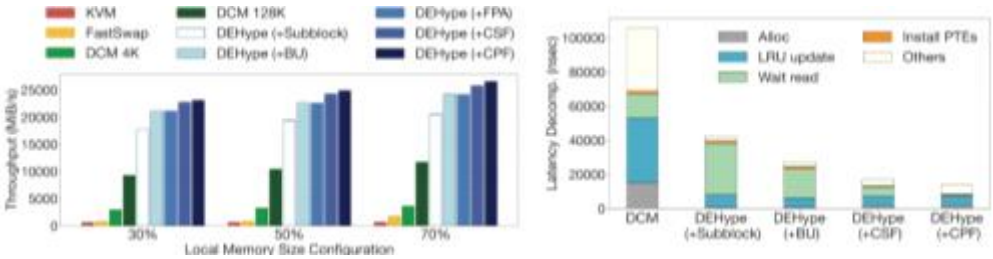
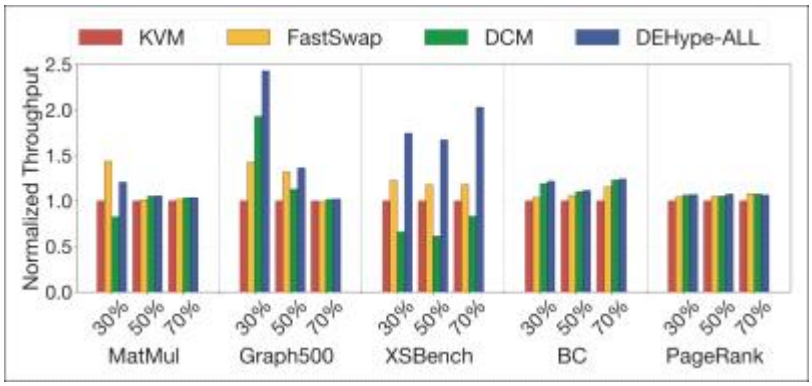
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>· 가상머신이 바라보는 메모리 뷰와 하이퍼바이저가 바라보는 메모리 뷰가 다름으로 인해서 발생하는 문제가 있음. 가상머신 내부에서는 프리 페이지이지만 하이퍼바이저 입장에서는 기존에 사용했던 페이지라면 원격 메모리에 존재하고 있을 때, 로컬 메모리로 페치를 수행해야함. 하지만, 해당 페이지가 프리 페이지이면 페치를 수행하지 않고 바로 로컬 메모리에 할당하면 되기 때문에 쓸모없는 데이터 페치로 인한 네트워크 대역폭의 소모와 지연시간이 소모됨. 따라서, 본 연구에서는 가상머신에서 하이퍼바이저로 프리 페이지에 대한 정보를 보냄으로써 이 문제를 해결함. 그림 323와 같이 프리 페이지 정보에 대한 GPA를 얻어내고, 해당 GPA를 HyperCall로 하이퍼바이저에 전송하여, 하이퍼바이저는 해당 GPA에 해당하는 페이지는 접근하더라도 페치하지 않도록 하여 문제를 해결함
 

그림 323. 프리페이지 정보 공유 기법</td></tr>
<tr><td></td><td>② 최신 인터커넥트를 고려한 메모리 중심 컴퓨팅 시스템 연구</td><td>• (1) CXL 디바이스 지원
• 분리 메모리 환경에서의 성능 개선 기법에 대한 성능 평가 수행
- 성능 평가를 위한 벤치마크 설정 및 대조군 설정
· 분리 메모리 환경 성능 개선 기법을 평가하기 위해서 다양한 메모리 접근 패턴 및 메모리 사용량을 가지는 벤치마크를 이용해서 성능 평가 수행. 그림 324는 본 실험에서 사용한 STREAM, Matmul, Graph500, XSBench, BC, PageRank의 소개 및 메모리 사용량을 나타낸 표임
<table><tr><th rowspan="2">Benchmark</th><th rowspan="2">Description</th><th rowspan="2">Memory Usage</th><th colspan="3">Local Memory</th></tr><tr><th>30%</th><th>50%</th><th>70%</th></tr><tr><td>STREAM</td><td>vector multiply & add</td><td>112G</td><td>34G</td><td>56G</td><td>78G</td></tr><tr><td>MatMul</td><td>matrix multiplication</td><td>40G</td><td>12G</td><td>20G</td><td>28G</td></tr><tr><td>Graph500</td><td>breath-first search</td><td>26G</td><td>8G</td><td>13G</td><td>18G</td></tr><tr><td>XSBench</td><td>monte carlo neutron transport</td><td>244G</td><td>73G</td><td>122G</td><td>171G</td></tr><tr><td>BC</td><td>betweenness centrality</td><td>71G</td><td>21G</td><td>35G</td><td>50G</td></tr><tr><td>PageRank</td><td>ranking web pages</td><td>71G</td><td>21G</td><td>35G</td><td>50G</td></tr></table>그림 324. 본 실험에 사용된 벤치마크
· 성능 비교를 위해서 아래와 같은 대조군을 설정하고 해당 시스템의 성능 평가 수행함
* KVM: cgroup2를 이용해서 메모리 자원의 리소스를 제한하고, 백엔드가 RAM 디스크인 스왑 인터페이스를 사용하여 분리 메모리 시스템의 naive 디자인 설정
* FastSwap: 2020년 EuroSys에서 발표된 분리 메모리 시스템으로 프론트 스왑 인터페이스를 이용해서 원격 메모리와 통신 수행
* DCM: 2019년 TC에 게재된 분리 메모리 시스템으로 본 연구의 베이스라인

- 분리 메모리 시스템의 성능 및 최적화 기법 성능 기여 영향 분석
· 그림 325는 마이크로벤치마크 (STREAM) 상에서의 성능 평가 및 페이지폴트 지연시간을 분석한 그래프임. Subblock 기법 (서브블록, 큰 I/O 단위를 사용)으로 관련 연구 중 가장 성능이 뛰어난 DCM 대비 91.5%의 성능향상을 나타내었고, BU 기법 (배치 언맵)으로 18.7%의 성능 향상, CSF/CPF 기법 (필요 데이터 우선 처리)으로 최대 9.1% 성능 향상을 확인할 수 있음</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		· 해당 성능 향상이 일어나는 이유를 확인하기 위해서 각 최적화 기법을 사용하였을 때 페이지폴트 지연시간을 분석함. 서브블록을 통해서 메타데이터를 서브블록 단위로 관리하여 관리 비용을 줄였고, 큰 I/O단위로 RDMA통신을 수행하여 페치를 기다리는 시간을 줄임을 확인함. CSF/CPF를 통해서는 필요한 데이터를 우선적으로 처리하는 방법을 사용하여 페치를 기다리는 시간을 66.8%/93.7% 가량 줄이고 필요 페이지의 페이지 폴트 처리 지연시간을 줄임을 확인함  그림 325. 마이크로벤치마크에서의 성능 및 페이지폴트 지연시간 분석 · 그림 326는 제안한 분리 메모리 시스템의 성능 (DEHype-ALL)을 다양한 벤치마트에서 다른 분리 메모리 시스템과 비교한 결과를 나타내는 그래프임. 각 성능 결과는 KVM의 결과에 정규화하여 표현한 정규화 성능치임. Matmul과 XSBench의 경우 응용의 메모리 접근 패턴이 랜덤 메모리 접근 패턴이므로 공간적인 지역성을 활용하는 DCM보다는 FastSwap의 성능 향상이 더 큰 것을 확인할 수 있음. 본 연구 역시 DCM을 베이스라인으로 동작하는 시스템이기 때문에 성능의 하락이 있을 수 있으나, 제안한 여러 가지 최적화 기법을 통해서 성능의 하락을 최소화하고, FastSwap보다 더 성능이 좋을 것을 확인할 수 있음  그림 326. 다른 분리 메모리 시스템과의 성능 비교 · 그림 327은 제안한 분리 메모리 시스템의 최적화 기법이 성능에 끼치는 영향을 확인하기 위해 각 최적화 기법에 따른 성능 변화를 나타낸 그래프임. 각 성능 결과는 DCM의 결과에 정규화하여 표현한 정규화 성능치임. Running 구간에서의 성능을 확인하면 거의 대부분의 성능 향상이 서브블록으로 인한 네트워크 지연시간 감소 및 메타데이터 관리 시간 감소로 인한 성능 향상 확인함 (DCM대비 32.4% 성능 향상). 다른 최적화 기법도 성능에 좋은 영향을 준다는 것을 확인 가능. Loading 구간에서의 성능을 보면, FPA 기법 (프리페이지 정보 하이퍼바이저 공유)으로 5.5%의 성능 향상을 확인할 수 있음

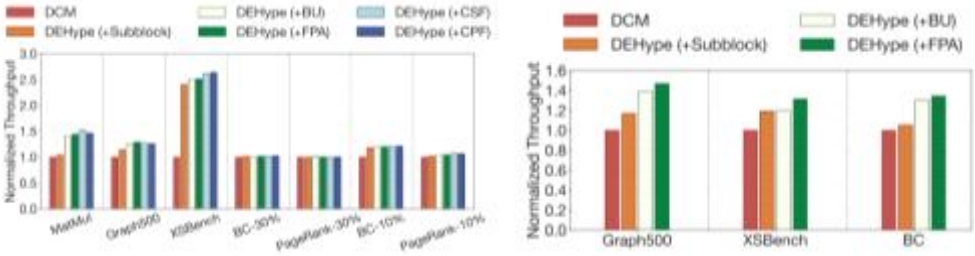
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td rowspan="2"></td><td></td><td>


그림 327. 최적화 기법의 성능 영향 분석 (왼쪽: run, 오른쪽: loading)

· 그림 328은 로컬 메모리의 크기에 따른 분리 메모리 시스템의 성능을 나타내는 그래프로, BC와 XSBench 두 가지 벤치마크의 수행시간으로 성능을 표현함. BC 벤치마크의 성능 그래프를 보면 로컬 메모리의 크기가 적을 때, FastSwap의 성능이 크게 떨어지는 것을 확인 할 수 있음. 이는 LRU를 구성하는 DCM과 본 발명과는 다르게 필요한 페이지를 로컬 메모리에 저장하는 방식이 아니라 발생하는 문제임. 또한, XSBench의 성능 그래프를 보면 DCM에서 성능의 하락이 많이 있음을 알 수 있음. 이는 랜덤 메모리 접근 패턴인 XSBench에서 큰 크기의 데이터 블록을 사용하면서 발생하는 문제임. 두 가지 결과 모두 본 연구에서 제안하는 DEHype의 성능이 다른 분리 메모리 시스템 대비 뛰어나다는 것을 확인 가능. BC의 경우 FastSwap 대비 최대 351%, DCM 대비 최대 21.5% 성능 향상, XSBench의 경우 FastSwap 대비 최대 89.8%, DCM 대비 최대 188% 성능 향상을 확인

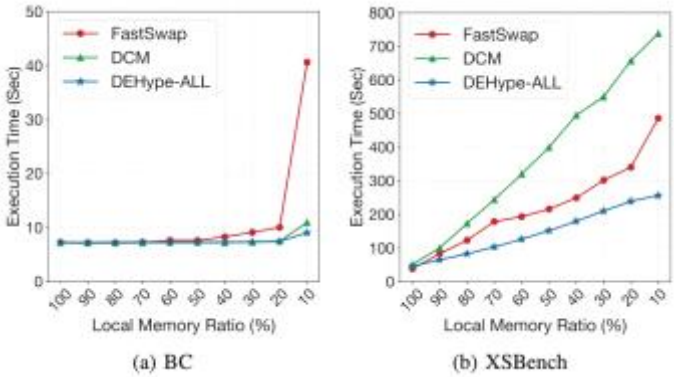
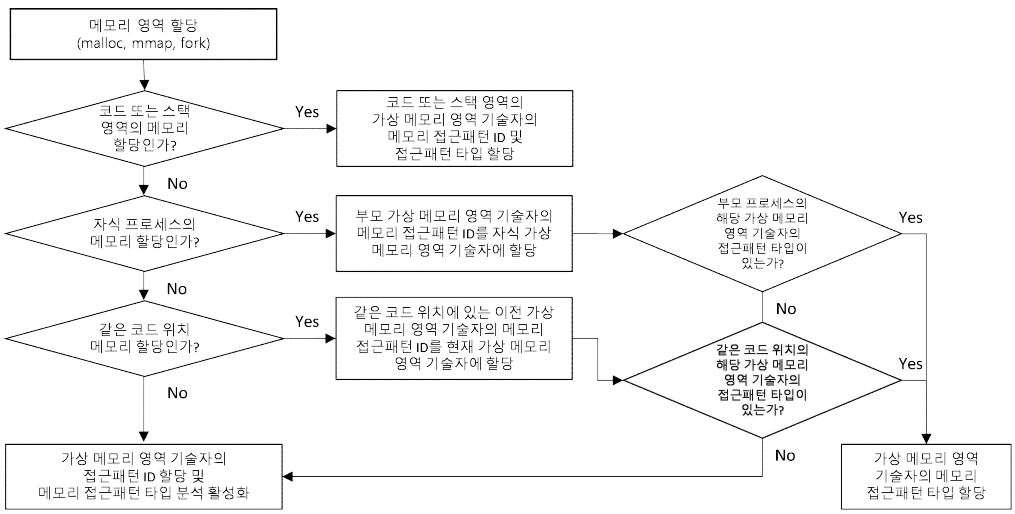
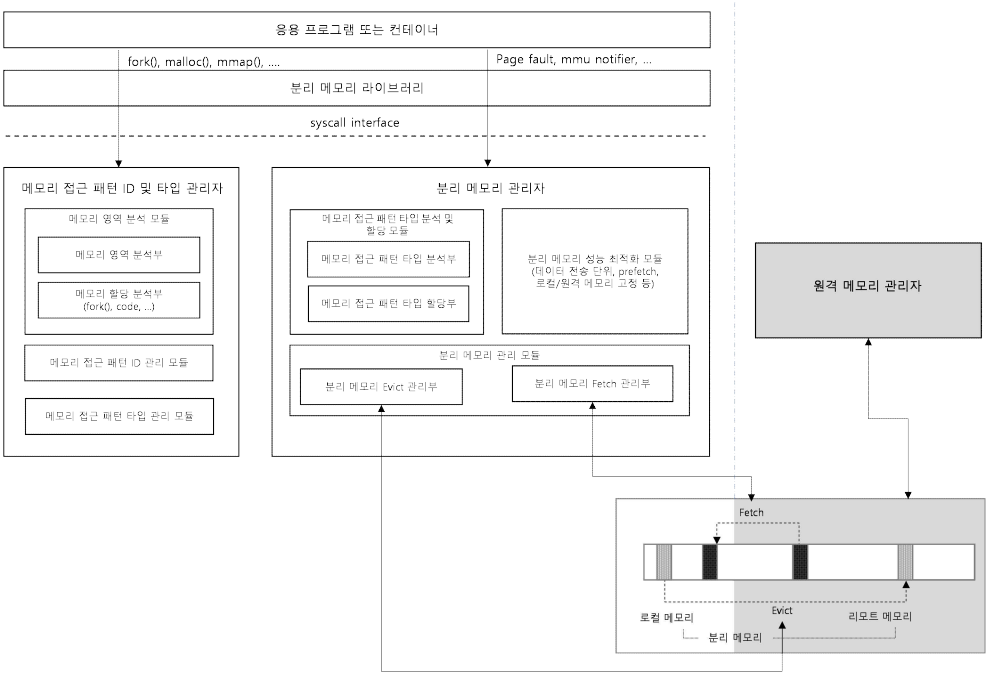
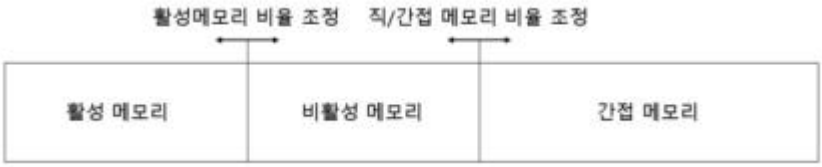
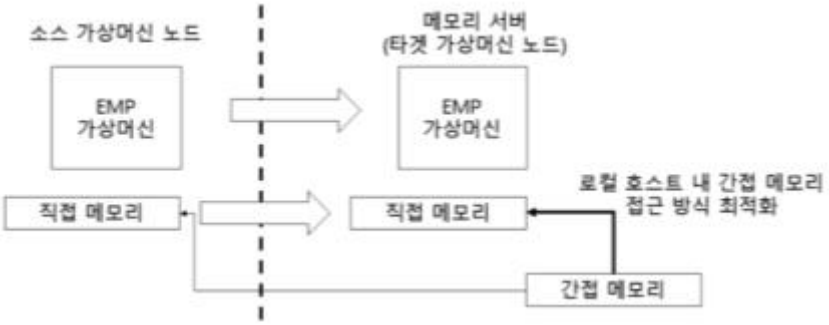
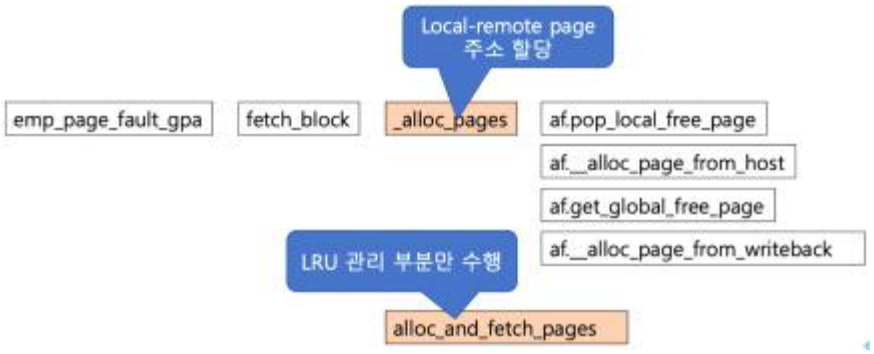


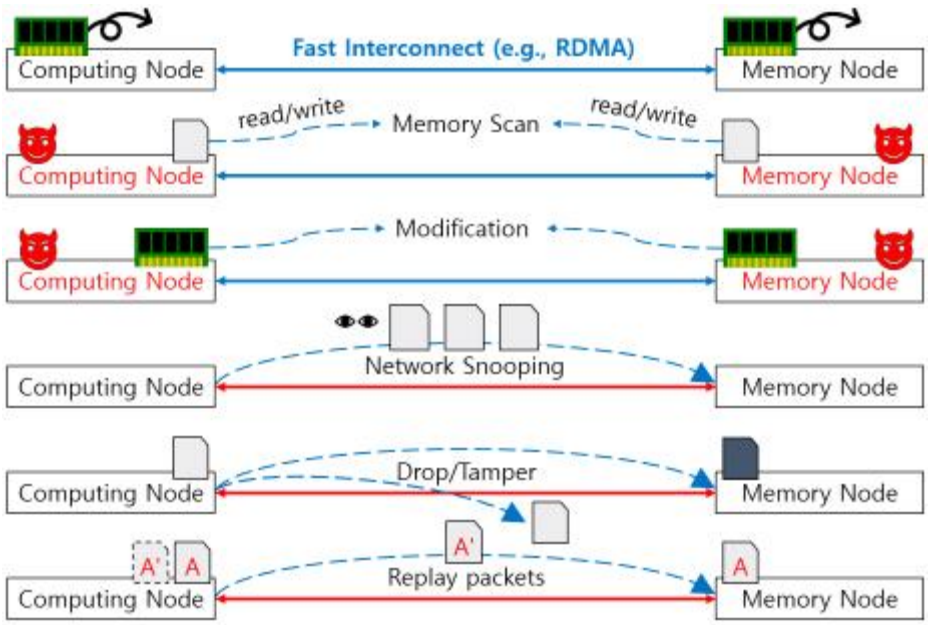
그림 328. 로컬 메모리 크기에 따른 분리 메모리 성능 평가
</td></tr>
<tr><td>③ 혼성·다계층 메모리 중심 컴퓨팅 시스템 메모리 공유 기술 연구</td><td>
• 응용의 메모리 접근 패턴을 인식하는 혼성·다계층 분리 메모리 관리 기술 개발
- 기술 개발의 필요성
· 분리 메모리 시스템에서는 응용이 원격 메모리에 있는 데이터를 필요로할 경우, 원격 메모리의 데이터를 로컬 메모리로 전송한 후에야 연산이 계속될 수 있음. 이때 로컬 메모리 접근에 비해 상당히 긴 시간이 필요하므로, 원격 메모리 접근 빈도가 응용 성능에 결정적 요인이 됨. 따라서, 분리 메모리 시스템에서는 원격 메모리 접근 빈도를 낮추는 것이 중요함
· 응용 프로그램의 메모리 접근 패턴은 프로그램이 메모리에 어떻게 접근하는지를 나타내는 패턴임. 메모리 접근 패턴을 파악할 수 있으면, 분리 메모리 시스템이 이를 활용하여 원격 메모리 접근 빈도를 낮출 수 있음

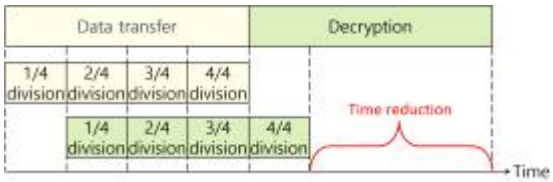
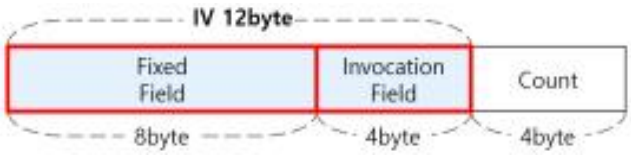
- 가상 메모리 영역 기술자 기반 응용의 메모리 접근 패턴 인식
· 본 연구에서 응용의 메모리 접근 패턴 인식 방안은 각 가상 메모리 영역 기술자가 관리하는 영역별로 나누어서 함
· 응용이 메모리를 할당받는 단위와 가상 메모리 영역 기술자가 관리하는 단위가 동일하지는 않음. 하지만, 분리 메모리 시스템이 통상적으로 구현되는 운영체제 단위에서 응용이 메모리를 할당받는 단위를 파악하기 어려우며, 응용이 메모리를 할당받는 단위는 작을 수
</td></tr>
</table>

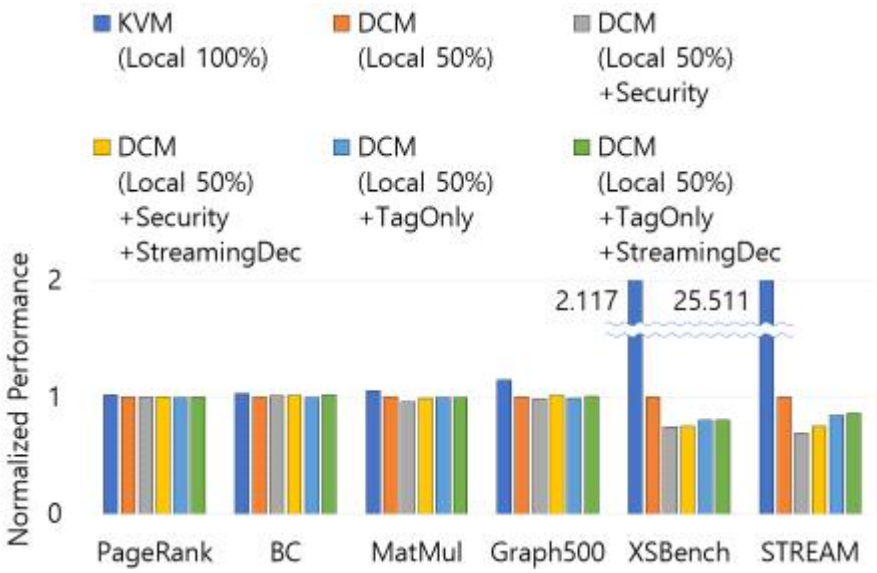
수행기관	개발 목표	개발내용 및 범위
		도 있기 때문에 메모리 할당 단위로 관리를 할 경우 관리 부담이 커지게 되며, 응용의 메모리 할당을 담당하는 라이브러리가 비슷한 성질(할당 요청 크기, 배열을 위한 메모리 영역인지에 대한 여부 등)의 데이터를 하나의 가상 메모리 영역 기술자에 속하도록 할 수 있으며, 다른 종류(코드 영역, 전역 데이터 영역, 힙 영역, 스택 영역)의 메모리들은 서로 다른 가상 메모리 영역 기술자에 속함 · 따라서, 가상 메모리 영역 기술자 단위로 메모리 접근 패턴을 파악하는 것은 충분히 다른 데이터들의 접근 패턴을 분리하면서도 메모리 접근 패턴 분석을 효율적으로 할 수 있는 방법임 - 응용의 메모리 접근 패턴 인식하는 혼성·다계층 분리 메모리 관리 기술 · 본 연구에서는 메모리 할당(malloc, mmap)과 포크(fork) 등 새로운 가상 메모리 영역 기술자가 생길 수 있는 함수들을 호출될 때 가상 메모리 영역 기술자에게 메모리 접근 패턴 ID와 메모리 접근 패턴 타입을 지정함. · 메모리 접근 패턴 타입은 코드 또는 스택 영역의 메모리와 같이 메모리 접근 패턴을 알 수 있는 경우는 메모리 접근 패턴 ID 부여시 함께 지정됨. · 자식 프로세스의 가상 메모리 영역 기술자의 경우 부모 프로세스의 메모리 접근 메모리 패턴을 계승함. · 메모리 접근 패턴 ID 부여시 메모리 접근 패턴을 알 수 없는 경우에는 페이지 폴트(page fault)가 발생하거나 mmu notifier에 의해 페이지 테이블의 변화를 감지했을 때, 메모리 접근 패턴을 분석하여 메모리 접근 패턴 타입을 부여함. · 메모리 접근 패턴 타입을 알고 있는 경우, 페이지 폴트(page fault)가 발생하거나 mmu notifier에 의해 페이지 테이블의 변화를 감지했을 때, 메모리 접근 패턴에 따른 원격 메모리의 데이터를 fetch하는 등의 분리 메모리 관리를 수행하고, 더불어서 메모리 접근 패턴 타입을 분석하며 분리 메모리 성능 최적화를 수행함.  그림 329. 응용의 메모리 접근 패턴 ID 및 타입 할당 - 응용의 메모리 접근 패턴 인식하는 혼성·다계층 분리 메모리 시스템 · 응용의 메모리 접근 패턴을 인식하는 분리 메모리 시스템은 메모리 접근 패턴 ID 및 타입 관리자와 분리 메모리 관리자, 원격 메모리 관리자로 구성됨. · 메모리 접근 패턴 ID 및 타입 관리자는 메모리 할당(malloc, mmap)과 포크(fork) 등 새로운 가상 메모리 영역 기술자가 생길 수 있는 함수들이 호출될 때, 가상 메모리 영역 기술자에게 메모리 접근 패턴 ID와 메모리 접근 패턴 타입을 지정함. · 분리 메모리 관리자는 페이지 폴트(page fault)가 발생하거나 mmu notifier에 의해 페이지 테이블의 변화를 감지했을 때, 원격 메모리의 데이터를 fetch하는 등의 분리 메모리 관리를 수행하고, 더불어서 메모리 접근 패턴 타입을 분석하며 분리 메모리 성능 최적화

수행기관	개발 목표	개발내용 및 범위
		를 수행함 · 메모리 접근패턴 ID 관리 모듈은 메모리 접근 패턴이 비슷할 것으로 예상될 수 있는 가상 메모리 영역 기술자들을 묶어서 접근 패턴을 분석하고 각 가상 메모리 영역 기술자들에게 하나의 메모리 접근 패턴 ID를 부여함. 따라서, 여러 가상 메모리 영역 기술자들이 하나의 메모리 접근 패턴 ID를 부여 받을 수 있으며 이 경우 ID가 같은 가상 메모리 영역 기술자들이 관리하는 영역들은 메모리 접근 패턴이 같다고 가정함. 즉, 하나의 메모리 영역에서 특정 메모리 접근 패턴 타입이 감지되었다면 해당 메모리 영역과 메모리 접근 패턴 ID가 동일한 메모리 영역들은 같은 메모리 접근 패턴을 가진다고 가정하는 것임. 이를 통해 일부 메모리 영역들에서만 메모리 접근 패턴이 분석이 되어도 더 많은 메모리 영역들에 대해 효율적인 분리 메모리 관리 정책을 사용할 수 있음  그림 330. 응용의 메모리 접근 패턴 인식하는 혼성·다계층 분리 메모리 시스템 구조도 • 가상머신 이주 최적화 기법: 메모리 제공 노드로의 가상머신 이주 구현 - 분리 메모리 환경에서 가상머신 이주를 위한 동적 자원 할당 · 가상머신 이주를 통해서 분리 메모리 환경에서의 가상머신이 메모리 제공 노드에서 동작할 수 있음. 따라서, 그림 331과 같이 해당 시스템에서 로컬 메모리 영역을 직접 메모리, 기존 원격 메모리 영역을 간접 메모리라고 정의하고, 각 메모리의 메모리양을 동적으로 조절 기능 구현  그림 331. 동적 자원 할당 기능 구현 - 분리 메모리 환경에서 가상머신 이주 기반 직/간접 메모리 동적 통합 관리 · 분리 메모리 환경에서 응용의 메모리 접근 양이 많고, 랜덤 메모리 접근 패턴의 형태를 가지게 된다면, 분리 메모리 환경 구조보다는 하나의 머신에서 응용을 수행하는 것이 이익. 따라서, 메모리 제공 노드로 가상머신 이주를 수행할 필요성이 생김. 따라서, 그림 332과 같이 원격 메모리로 사용되던 영역을 동적으로 일시적인 주소 공간 구성을 지원하여 더욱 빠른 메모리 접근이 가능한 메모리 관리 모델 구현. 또한, 가상머신이 메모리 제공 노드에서 다시 다른 호스트로 이주할 때 기존의 분리 메모리 시스템 모델을 유지할 수

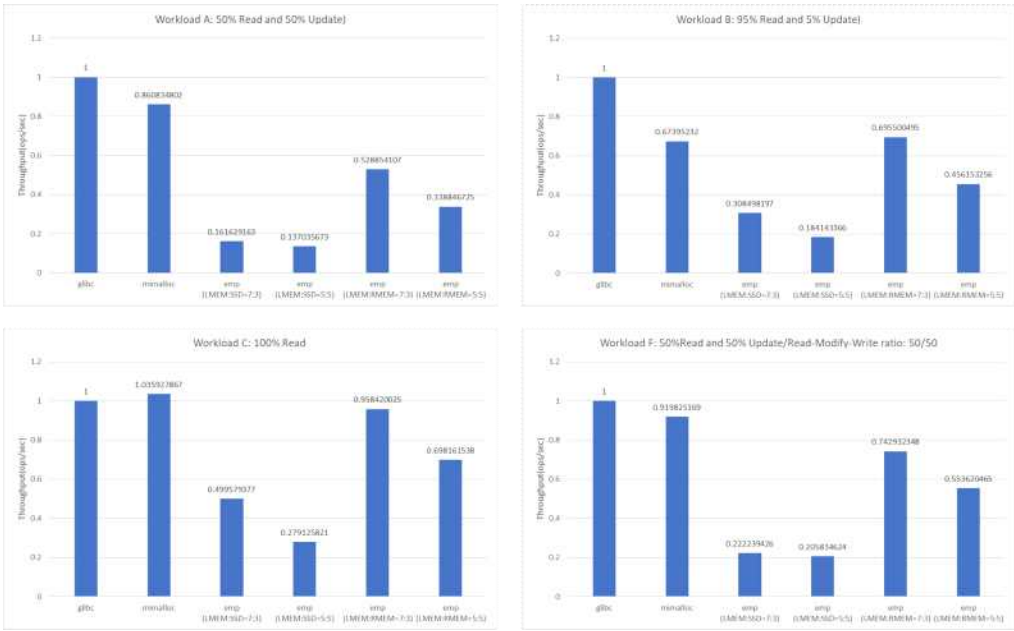
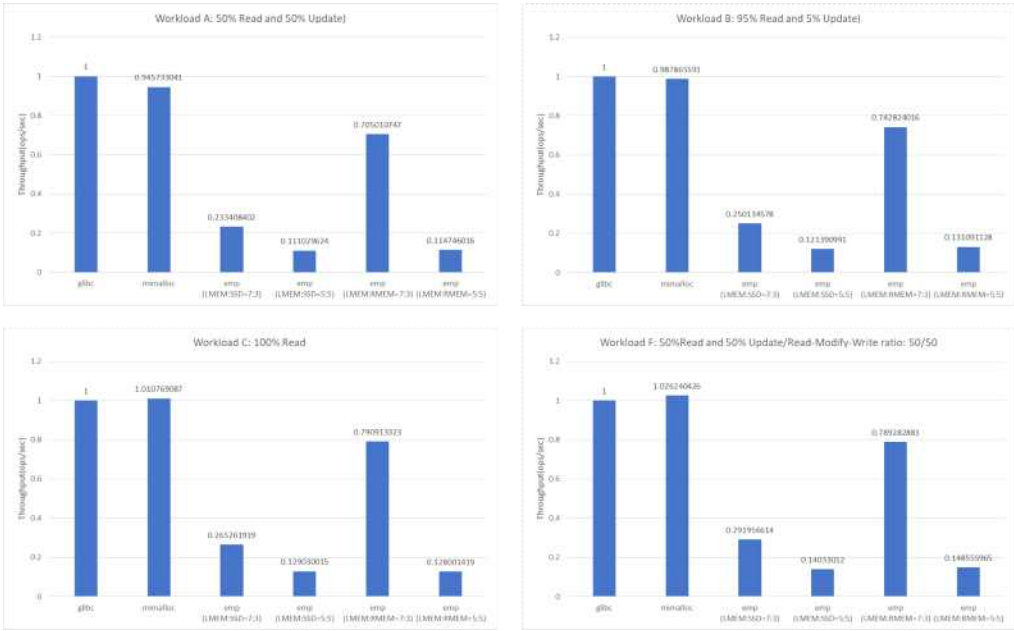
수행기관	개발 목표	개발내용 및 범위
		있도록 로컬 메모리와 원격 메모리 간의 분리 지원  그림 332. 동적 직/간접 메모리 통합 관리 · 해당 기법을 구현하기 위해서 분리 메모리 시스템 자체에서 메모리 제공 노드의 메모리 영역을 관리해야 함. 따라서. 해당 영역에 대한 정보를 char 디바이스로 저장하고 분리 메모리 시스템에서 해당 메모리를 인지하도록 함. 또한, 가상머신이 메모리 제공 노드로 이주하면 기존 메모리 제공 노드의 메모리는 공유 메모리의 속성으로 분류하고, 해당 메모리에 접근하는 경우 데이터를 페치하는 대신 직접 매핑하여 사용하도록 함. 이를 통해서 직접 메모리로의 복사 시간 및 용량을 절약 가능. 그림 333은 메모리 제공 노드 메모리를 직접 제어하기 위한 방법을 나타내는 그림. 메모리 페이지를 할당하는 과정에서 기존 로컬 메모리의 프리페이지를 활용하는 대신, 공유 메모리에 저장된 주소 자체를 할당함. 이후 데이터 페치 부분을 생략하고 LRU 관리 메커니즘 수행  그림 333. 메모리 제공 노드 메모리 직접 제어 방법 · 가상머신이 메모리 제공 노드에 머물다가 다시 다른 호스트로 이주하게 될 때, 공유 메모리로 분류되어 사용되고 있던 메모리 제공 영역은 그대로 두고, 가상머신만 이주시키는 방법을 사용. 해당 영역은 간접 메모리로 간주되므로 함께 이주시키지 않음으로써 이주 오버헤드를 줄임 - 메모리 제공 서버 기능 고도화 및 가상머신 이주 시 문제 해결 · 기존 메모리 제공 노드의 메모리 관리 데몬을 고도화함. 각 프로세스마다 하나의 메모리 영역을 담당하고, 이는 RDMA 포트로 구분되고 있는데, 동적으로 메모리 제공 노드의 제공 메모리 영역이 필요할 때마다 프로세스를 추가하여 원하는 용량과 포트를 넣어 제공 메모리 추가 기능 구현. 메모리 제공 노드는 서버의 IP와 포트로 분리 메모리 시스템에서 인지 가능하며, 분리 메모리 시스템에서도 해당 메모리 제공 노드의 정보의 일관성을 유지함 · 가상머신 이주 기능과 고도화 기능을 구현할 때 발생하는 문제들을 해결함. * 가상머신 이주 이후 페이지 폴트가 발생할 때 해당 페이지에 대한 동기화 문제 해결 * LRU 체인으로 인한 메모리 회수 시 리버스 맵에서 발생하는 문제를 해결 * 가상머신 이주 도중 가상머신 이주 대상 호스트에서 원격 메모리로의 데이터 쓰기가 발생하는 경우 문제 해결. 문제가 발생하지 않도록 가상머신 이주 대상 호스트의 로컬 메

수행기관	개발 목표	개발내용 및 범위
		모리 크기를 동적으로 증가 시켜서 해당 데이터 쓰기가 발생하지 않도록 함 • 분리 메모리 환경의 보안 제공 기법 연구 - 분리 메모리 환경의 보안 위협의 종류 분석 · 물리적 공격(Physical Attack): 메모리 등의 부품을 물리적으로 탈취하여 데이터를 가져가는 것. 분리 메모리 시스템에서는 컴퓨팅 노드 뿐만 아니라 메모리 노드에 대한 물리적 공격도 데이터 탈취를 가능하게 하는 요소가 됨 · 소프트웨어 공격(Software Attack): 악의적인 공격자가 서버의 관리자 권한을 탈취하여 데이터를 탈취하거나 변경할 수 있으며, 부채널 공격(Side-Channel Attack)을 사용해 데이터 접근 패턴을 파악하여 암호 등의 데이터를 유추하는 것도 가능. 분리 메모리 시스템에서는 메모리 노드에도 데이터가 저장되고 저장된 데이터가 컴퓨팅 노드에 의해 다시 사용되므로, 메모리 노드에 대한 소프트웨어 공격으로도 데이터 탈취나 변경, 부채널 공격 등이 가능 · 네트워크 공격(Network Attack): 기존 시스템에서는 응용이 명시적으로 내보내는 데이터만이 네트워크를 통해 전송되나, 분리 메모리 시스템에서는 컴퓨팅 노드의 메모리 사용량에 따라 운영체제나 하이퍼바이저 수준에서 데이터를 선별하여 네트워크를 통해 메모리 노드로 전송. 또, 컴퓨팅 노드에서 사용할 데이터가 메모리 노드에 있는 경우에도 역시 네트워크를 통해 컴퓨팅 노드로 전송. 이 과정에서 네트워크 패킷 전체를 스캔하여 데이터를 탈취하는 네트워크 스누핑(Snooping) 공격과 네트워크 패킷을 누락시키거나 패킷 데이터를 변조하는 스푸핑(Spoofing) 공격이 가능. 네트워크 전송 데이터에 대한 무결성 검증을 무력화시키기 위해 이전에 전송된 정상 패킷의 데이터를 사용해 새로 전송되는 패킷을 변조함으로써 시스템에 혼란을 일으키는 리플레이(Replay) 공격도 가능  그림 334. 분리 메모리 시스템에서의 위협 모델 - 분리 메모리 환경 보안 강화를 위한 실용적인 위협 모델(Threat Model)과 신뢰 컴퓨팅 기반(Trusted Computing Base) 설정 · 분리 메모리 보안과 관련한 기존 연구들은 특수 하드웨어를 추가하거나 네트워크 스위치 및 카드를 프로그램 가능한 제품을 써야 하는 등의 제약이 있었음 · 본 연구에서는 일반적인 분리 메모리 시스템 모두에 적용 가능한 보안 기법을 개발하기 위해 소프트웨어 기반의 보안 제공 기법을 연구 · 위협 모델(Threat Model): 본 연구의 위협 모델은 분리 메모리 시스템에서 추가되는 보안 위협만을 포함. 즉, 메모리 노드에 대한 물리적 공격과 소프트웨어적인 공격, 그리고 네트워크 공격이 포함됨. 구체적으로는 메모리 노드의 하드웨어를 물리적으로 탈취당했을 때

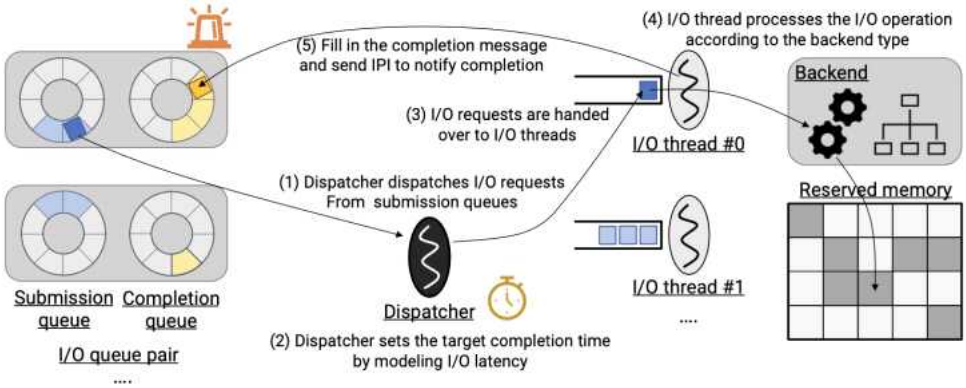
수행기관	개발 목표	개발내용 및 범위
		도 원본 데이터가 노출되는 것을 막고, 메모리 노드의 관리자 권한이 악의적인 사용자에게 넘어갔을 때에도 컴퓨팅 노드에서 실행 중인 응용의 데이터에 대한 기밀성과 무결성이 보장되며, 네트워크 수준에서 데이터 탈취나 변조가 일어났을 때도, 원본 데이터는 노출되지 않으며 변조된 데이터가 컴퓨팅 노드에서 사용되지 않도록 만드는 것을 목표로 함 · 신뢰 컴퓨팅 기반(Trusted Computing Base): 컴퓨팅 노드의 CPU, 메모리, 운영체제 전체. 분리 메모리 시스템 적용으로 인해 추가되는 보안 위협에 대해서만 다루기 때문에, 컴퓨팅 노드 자체의 보안은 보장되는 상황을 가정함 - 분리 메모리 환경 보안 강화 기법 설계 및 성능 최적화 기법 · AES-GCM 알고리즘을 기반으로 컴퓨팅 노드에서 메모리 노드로 보내는 데이터를 암호화하며, 메모리 노드에서 컴퓨팅 노드로 오는 데이터에 대해 복호화 및 무결성 검증을 수행 · 성능 최적화를 위해 사전 계산값을 이용. 암호화 키를 라운드 숫자에 맞추어 확장하는 키 확장(Key Expansion)을 메모리 영역 생성 시에 1번 수행하고 저장된 값을 반복해서 사용. 본 연구의 암호화와 복호화 단위는 서브블록. 서브블록은 작은 크기이므로 키 확장 시간이 전체 암호화 시간에서 차지하는 비중이 7% 정도 됨. 이에 대한 성능 최적화 가능 · 복호화는 컴퓨팅 노드에서 필요한 데이터를 메모리 노드로부터 가져오는 과정에서 수행되므로 critical path에 속함. 복호화에 대한 성능을 최적화하기 위해 스트리밍 복호화 기법을 제시. 메모리 노드로부터의 데이터 전송과 복호화를 중첩(overlap)시키는 기법  그림 335. 스트리밍 복호화 기법 - 분리 메모리 환경 보안 강화를 위한 암호화 메타데이터 설계 · 분리 메모리 기법은 메모리가 부족한 상황을 전제하므로 보안 강화를 위한 메타데이터의 크기도 최소화할 필요가 있음 · 암호화 및 복호화에 필요한 초기화 벡터(Initialization Vector)는 총 16바이트가 필요. 그런데 하위 4바이트는 16바이트 블록의 ID에 해당하므로 저장할 필요 없음. 나머지 12바이트 중 8바이트를 해당 서브블록의 메타데이터 주소값으로 지정. 메타데이터는 복호화 과정에서 필요하므로 그 주소값을 알아내기 쉬움. 따라서 나머지 4바이트를 invocation field로 지정하여 4바이트만 저장 · 4바이트의 invocation field는 overflow가 날 수 있고, overflow가 나면 초기화 벡터가 재사용되는 것이므로 보안에 문제가 됨. (하나의 암호화 키에 대해서 초기화 벡터는 절대 재사용되어서는 안 됨) 이를 방지하기 위해 응용 혹은 가상머신의 최대 실행 시간을 10년으로 보고, 10년 동안 overflow가 나지 않도록 강제함. 4바이트 변수가 10년 동안 overflow 나지 않으려면 10년/2^{32}=0.07초 이하의 간격으로 카운트가 올라가지 않으면 됨. 따라서 0.07초 이하의 간격으로 암호화를 수행할 경우, busy-waiting을 통해 시간 간격을 0.07초 이상으로 늘려줌. 그런데 하나의 서브블록이 두 번 암호화가 되려면, 그 사이에 victim으로 선정되어 메모리 노드로 보내진 뒤, 다시 컴퓨팅 노드에서 접근을 해야 하므로, 실질적으로 시간 간격이 0.07초 이상인 경우가 대부분. 본 연구에서 수행한 실험에서 두 암호화 시간의 간격이 0.07초 이하인 경우는 발견되지 않음  그림 336. 분리 메모리 시스템 보안을 위한 초기화 벡터 디자인

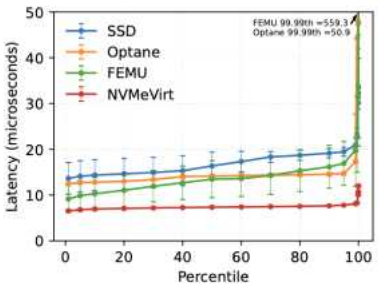
수행기관	개발 목표	개발내용 및 범위
		· 무결성 검증에 필요한 태그의 경우, NIST 문서에 따르면 32비트에서 128비트까지를 사용할 수 있으나, 크기가 작은 태그를 사용할 경우 forgery attack이 가능하므로 키 사용 횟수에 제한을 둠. 하지만 태그를 신뢰 컴퓨팅 기반(TCB) 내에 둘 경우 forgery attack이 불가능하므로, 본 연구에서는 태그를 컴퓨팅 노드에 두는 대신 최소 크기인 32비트를 사용함 · 결론적으로, 보안 강화를 위해 필요한 메타데이터는 서브블록당 8바이트가 됨. 초기화 벡터 중 저장 부분 4바이트와 태그 4바이트가 필요 - 실험 및 결과 · 실험 환경: 컴퓨팅 노드는 Intel Xeon Gold 6230 CPU 2개와 375GB의 메모리를, 메모리 노드는 Intel(R) Xeon(R) Gold 5215M CPU 2개와 375G의 메모리를 장착한 것을 활용. 네트워크는 인피니밴드 ConnectX-4를 활용. 분리 메모리 시스템을 사용한 가상머신 위에서 실행되는 응용 성능 측정. 가상머신의 vCPU는 8개를 할당하고, 컴퓨팅 노드의 메모리는 각 워크로드의 워킹셋의 50%로 설정 · 워크로드: BC, PageRank, Graph500, MatMul, XSBench, STREAM. 이중 BC, PageRank, Graph500은 그래프 분석 응용이고, MatMul과 XSBench는 과학계산응용, STREAM은 넓은 메모리 공간을 반복해서 읽는 마이크로 벤치마크 · 보안 적용시 성능 결과: 분리 메모리 시스템 적용으로 인한 성능 부하가 거의 없는 4개의 응용(BC, PageRank, Graph500, matmul)의 경우에는 보안 기법 적용에 따른 성능 부하도 거의 없음. XSBench와 STREAM은 각각 25.6%와 30.6%의 추가 성능 부하를 보임. 스트리밍 복호화 적용시, STREAM에서의 성능 부하가 25.1%로 줄어듦. 스트리밍 복호화 적용시 6개 응용의 평균 성능 부하는 8.5% · 무결성 검증만 적용시 성능 결과: 데이터에 따라서는 공개되어도 상관 없는 경우가 있음. 그러나 무결성 검증은 시스템 안정성 및 관리자 권한과 직결될 수 있으므로 필수적. 이에 따라 무결성 검증만 적용하는 경우에 대해서도 실험함. 보안 기법 적용시에 큰 추가 성능 부하를 보였던 XSBench와 STREAM의 추가 성능 부하가 각각 20.0%와 15.9%로 낮아짐. 스트리밍 복호화까지 적용시 추가 성능 부하가 19.6%와 14.2%로 낮아졌으며, 전체 평균은 5.7%  그림 337. 보안 기능 추가에 따른 분리 메모리 시스템 성능 부하
	④ 대규모 메모리 기반 데이터 분석 플랫폼 개발	**• [정량 목표] 대규모 그래프 분석 가속 연구** - 개념: 1 trillion edge 규모의 그래프 분석 응용에 대한 연구 - 시스템 구성: 1 trillion edge는 데이터 규모가 크기 때문에 데이터 저장에 필요한 고용량 저장장치, 효율적인 데이터 처리가 가능한 고성능 연산장치 및 대용량 메모리를 갖춘

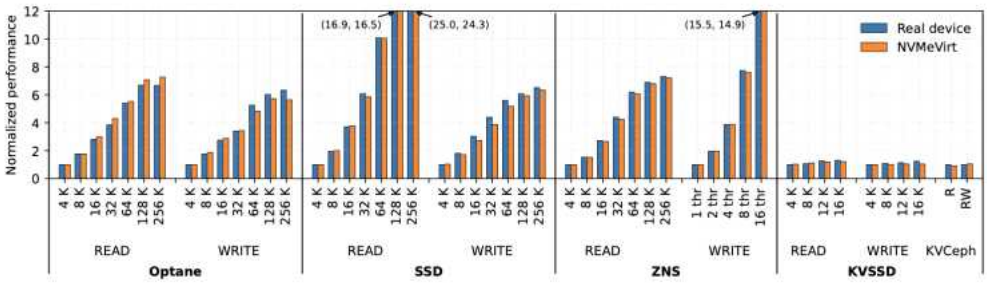
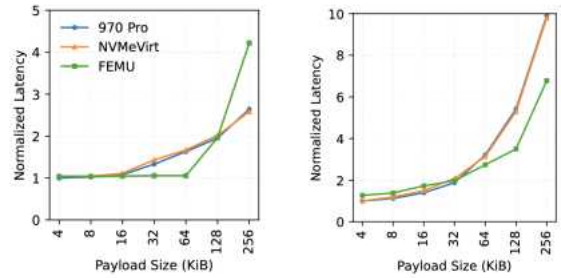
수행기관	개발 목표	개발내용 및 범위
		시스템 구성이 필요함 · CPU: AMD EPYC 9654 CPU @ 2.4GHz x 2개 (총 192코어, 총 384 쓰레드) · DRAM: 1.5TB · 그래프 저장용 SSD: Intel SSD (7.68TB) X 4개 · 그래프 분석용 SSD: Intel SSD (7.68TB) X 8개 - 소프트웨어 구성 · 운영체제: CentOS 8 Stream, Linux 4.18.0-425el8 · 그래프 생성 플랫폼: TrillionG (SIGMOD'17) · 그래프 분석 플랫폼: Graphene (FAST'17) - 개발 내용 · TrillionG를 사용해서 Edge 1,000,727,379,968개, Vertex 4,294,967,296개를 갖는 그래프 데이터 생성 (약 1 trillion edge) . Graphene 플랫폼은 그래프 분석에 사용하는 I/O 장치의 개수와 데이터 처리 병렬 성능이 비례함. 시스템 내의 최대한 많은 디스크를 그래프 분석용으로 사용하고, 데이터를 균등하게 분할하여 각 디스크에 분산 배치함 . 데이터 처리 병렬성을 극대화하기 위해 I/O 성능을 실시간으로 모니터링 하고, I/O 병목이 발생하지 않으면서 대역폭을 최대한 활용할 수 있도록 I/O 컨텍스트 병렬화(concurent I/O context) 정도를 조절함으로써 성능 최적화 수행 . 그래프 메타데이터는 데이터에 비해 접근이 빈번하게 일어나기 때문에, 메타데이터는 메모리에 배치될 수 있도록 메모리 용량 확보 . 그래프 프로세싱 성능은 CPU 활용도(utilization)에 비례함. CPU 활용도를 모니터링하고, 활용도를 최대한으로 높일 수 있는 설정값을 탐색함. 실험적으로 얻어진 결과를 기반으로, 각 디스크에 분산 배치된 데이터를 다시 56개의 작은 데이터로 분할하여 CPU가 제공하는 병렬처리 성능을 극대화하도록 수정함 · 성능 실험: BFS 알고리즘 수행 결과, 559.247초 소모 [user@localhost bfs]$ sudo ./aio_bfs.bin 56 8 896 ~/Graphene/ ~/Graphene/ 200_trillion_beg 200_trillion_csr 896 $((1<<20)) 160 3 896 10 1 Format: /path/to/exe #row_partitions #col_partitions thread_count /path/to/beg_pos_dir /path/to/csr_dir beg_header csr_header num_chunks chunk_sz (#bytes) concurr_IO_ctx max_continuous_useless_blk ring_vert_count num_buffs source ./aio_bfs.bin 56 8 896 /home/user/Graphene/ /home/user/Graphene/ 200_trillion_beg 200_trillion_csr 896 1048576 160 3 896 10 1 Vertex count: 4294964497 col-ranger file is found iostat -x 1 -k > iostat_bfs.log& Before cd init Finished cd init @level: 0 time: 0.260s @level: 1 time: 265.399s @level: 2 time: 272.857s @level: 3 time: 20.191s @level: 4 time: 0.488s @level: 5 time: 0.051s Total time: 559.247 second(s) 그림 338. 1 trillion edge에 대한 BFS 알고리즘 수행 결과 • 대규모 메모리 기반 User-level EMP 성능 결과 - 대규모 메모리 기반 User-level EMP Benchmark 환경 · User-level EMP의 성능을 벤치마킹하기 위하여 인메모리기반의 키-값 저장소 프로그램인 memcached와 redis를 대상으로 YCSB 벤치마킹 도구를 사용함. · HW 및 OS ◦ HW: Intel(R) Xeon(R) Gold 6448Y ◦ OS: Linux 4.18.0-477.10.1.el8_8.x86_64 ◦ SSD : INTEL SSDPF2KX076TZ · YCSB ◦ Record count: 80,000,000 ◦ Operation count: 80,000,000 ◦ Workload Type: A(50% Read and 50% Update), B(95% Read and 5% Update), C(100% Read) , F(50%Read and 50% Update/Read-Modify-Write ratio: 50/50) · EMP 환경: EMP의 도너는 리모트 메모리와 SSD 2가지를 대상으로 하였으며, 이들을 대상으로 로컬 메모리는 워크로드에서 요구되는 메모리 사용량의 50%와 70%의 메모리를 할당하였음

수행기관	개발 목표	개발내용 및 범위
		◦ Local memory size: 50%, 70% ◦ Memory Donor: Remote DRAM, SSD - User-level EMP Memcached Benchmark 결과 · 리모트 메모리를 도너로 사용하는 로컬 메모리 70%와 50%를 할당받은 EMP는 리눅스 대비 Workload Type A의 경우 각각 71%, 11%의 성능을 나타냄, Workload Type B의 경우 각각 74%, 13%의 성능을 나타냄, Workload Type C의 경우 각각 79%, 13%의 성능을 나타냄, Workload Type F의 경우 각각 79%, 15%의 성능을 나타냄  그림 339. User-levle EMP Memcached Benchmark - User-level EMP Redis Benchmark 결과 · 리모트 메모리를 도너로 사용하는 로컬 메모리 70%와 50%를 할당받은 EMP는 리눅스 대비 Workload Type A의 경우 각각 53%, 34%의 성능을 나타냄, Workload Type B의 경우 각각 70%, 46%의 성능을 나타냄, Workload Type C의 경우 각각 96%, 70%의 성능을 나타냄, Workload Type F의 경우 각각 74%, 55%의 성능을 나타냄  그림 340. User-level EMP Redis Benchmark

수행기관	개발 목표	개발내용 및 범위
아주 대학교 (공동1)	메모리 중심 컴퓨팅 시스템을 위한 기반 소프트웨어 기술 연구	**• 스마트 스토리지를 이용한 메모리 풀 구축 연구를 위한 NVMe 장치 에뮬레이션 연구** - 연구 배경 · 대규모의 메모리풀을 구축하는 것이 메모리 중심 컴퓨팅 시스템 실현을 위한 핵심기술 중 하나임. 메모리풀은 고용량 메모리 노드를 활용하는 방법, CXL 기반 장치를 이용하는 방법, 여러 노드의 유휴 메모리를 모아 풀을 구성하는 방법, SSD를 backing device로 활용하는 방법, 가상화 환경을 이용하는 방법 등 다양한 방법을 통해 구현이 가능함. · 단순히 대용량의 DRAM을 이용하는 방식은 구축비용(cost of installation)과 데이터의 영속성(persistence) 측면에서 큰 약점을 가지고 있음. 특히 CXL을 이용하여 호스트의 external device 형태로 시스템에 메모리를 추가 설치할 수는 있겠으나 DRAM의 저장용량 대비 높은 에너지 소모량 및 낮은 집적도는 대용량의 메모리풀을 구축하는 데 장애물로 작용하고 있음. · 이에 대한 대안으로 NAND 플래시 기반 스토리지를 활용하여 메모리풀을 구축하고자하는 연구들이 주목을 받고 있음. 즉, 일정 크기의 DRAM을 버퍼로 활용하고 실제 데이터는 SSD와 유사한 저장장치에 저장함으로써 매우 큰 메모리공간을 낮은 에너지 소비 및 비용으로 구축이 가능해짐. 하지만 이를 위해서는 플래시기반 스토리지의 블록에 기반한 인터페이스와 NAND의 비정형적인 동작 특성을 해결해야 하며, 이를 위해 KVSSD, ZNS, FDP, OpenChannel, Computational Storage Device 등 다양한 스토리지 장치 컨셉이 학계와 산업계를 통해 제안되고 있음. · 이러한 장치들은 대부분 콘소시엄을 통해 표준화 작업이 수행하고 내부용 프로토타입용 장치를 구현, 활용하여 장치의 기능을 검증하는 형태로 사용됨. 또한 구현이 대부분 NVMe 프로토콜을 하드웨어로 처리하고 있는 IP를 통해 접근되고, 펌웨어 형태로 개발되고 있음. 따라서 새로운 기술에 대한 평가나 새로운 기법에 대한 테스트, 기능 개발 등에 매우 많은 엔지니어적 노력이 필요하고 이는 효과적인 스토리지 도메인 연구의 큰 걸림돌이 되고 있음 · 이러한 스토리지 도메인 연구의 난점을 극복하기 위해 소프트웨어에 기반한 에뮬레이터를 활용하는 연구는 이전부터 꾸준히 제안되어 옴. 에뮬레이터를 구현하는 방법은 크게 디바이스 드라이버 수준에서 블록 요청을 에뮬레이션하는 기법(예: FlexDrive)와 호스트 가상화 기법을 통해 게스트 운영체제에 가상의 장치를 보여주고 여기로 들어오는 요청을 hypervisor 수준에서 에뮬레이션하는 기법(예: FEMU) 등이 현재에도 대표적으로 사용되는 에뮬레이션 방법임. 하지만 이러한 에뮬레이션 방법은 최신의 스토리지 I/O 모델을 제대로 지원하지 못하는 단점이 있음. · 디바이스 드라이버 수준에서 에뮬레이션을 하는 기법의 경우 호스트 운영체제의 block layer를 통해 들어오는 요청만을 에뮬레이션 할 수 있음. 따라서 운영체제의 오버헤드를 극복하고자 Intel SPDK 등을 통해 사용자 응용이 직접 디바이스에 I/O 요청을 내리는 user-driven I/O 모델에 정상적으로 대응이 불가능함. 또한 PCIe device가 바로 다른 target PCIe device에 접근하는 device-driven I/O 의 경우도 대상 device가 에뮬레이션 되는 수준의 한계로 인해 지원하기 어려움. 이 두 가지 I/O 모델은 메모리 중심 컴퓨팅 시스템에서 요구하는 고성능 저지연성을 위해 필수적으로 활용되는 I/O 모델임에도 불구하고 이들 모두 지원할 수 없음. · 가상화 기법을 이용하는 에뮬레이션의 경우 게스트 운영체제 내의 응용이 발생시키는 I/O는 호스트 hypervisor를 통해 처리할 수 있으므로 user-driven I/O 모델은 손쉽게 지원이 가능함. 하지만 가상화 기법의 도입으로 인해 상대적으로 높은 성능 오버헤드가 발생하고, 또한 에뮬레이션 하는 장치의 성능 변화폭이 매우 크게 발생할 수 있음. 스토리지 장치의 주요 I/O 성능이 operations의 tail latency에 크게 영향을 받는 상황을 고려할 때 가상화 기법을 이용한 에뮬레이션은 그 한계가 매우 치명적으로 작용할 수 있음. 또한 가상화로 인해 게스트 OS내의 물리적 메모리의 배치가 복잡하게 됨에 따라 device-driven I/O를 제대로 지원하기 매우 어려운 한계가 있음.

수행기관	개발 목표	개발내용 및 범위
		 그림 341. NVMeVirt의 구조 및 요청 처리 흐름도 - 연구 내용 · 본 연구는 현대적인 I/O 모델을 지원하지 못하는 기존 에뮬레이션 방식의 큰 단점을 보완한 NVMeVirt 기법을 제안함. · NVMeVirt는 Linux kernel 형태로 구현되어 있는 가상 NVMe 장치 에뮬레이터임. 후술하는 방식을 통해 커널에 로딩된 이후 다른 하드웨어를 포함한 시스템 전체에서 실제 장치와 동일한 형태로 동작하는 가상의 NVMe 장치를 제공함. 기본적인 가상 장치 프레임워 위에 일반 SSD, NVM SSD, KVSSD, ZNS SSD 등의 기능을 구현하였고, 실제 장치의 성능을 높은 · NVMeVirt는 NVMe 하드웨어의 인터페이스 부분을 에뮬레이션하는 접근 방식을 이용함. NVMe를 포함한 PCIe 장치들은 디바이스 내의 메모리 주소 공간에 맵핑되어 있는 PCIe configuration header를 통해 호스트 운영체제에게 자신의 설정 정보를 전달함. 운영체제는 PCIe configuration header를 통해 해당 디바이스의 상태와 설정 정보를 읽거나 표준화를 통해 지정된 값을 특정 field에 쓰는 방법을 통해 해당 디바이스에게 설정을 요청하게 됨. 디바이스는 요청을 처리한 후 역시 표준화 된 field를 통해 처리 결과를 반환할 수 있음. NVMeVirt는 Linux의 PCIe device hierarchy를 활용하여 가상의 PCI bus를 통해 configuration header를 넘겨주도록 구현되어 있음. 이 configuration header에는 디바이스의 상태를 알려주는 fields이외에도 NVMe 프로토콜에 정의되어 있는 BAR addresses도 포함되어 있음 · 실제 디바이스의 경우 호스트에서 configuration header에 값을 쓰는 operation은 PCI transaction 형태로 디바이스에게 전달되어 디바이스가 처리를 할 수 있으나 NVMeVirt에서 PCI bus를 통해 제공되는 configuration header에 대한 연산은 단순히 메모리에 값이 쓰이는 연산으로 configuration header에 값이 바뀌는 것을 notification 형태로 받기 어려움. 따라서 configuration header의 변경을 확인하는 전담 thread인 dispatcher를 통해 요청을 처리함 · Dispatcher는 configuration header 뿐만 아니라 NVMe 프로토콜 상 정의되는 submission queue/completion queue와 연결되어 있는 doorbell도 확인함. Doorbells은 NVMe BAR를 통해 위치가 호스트에 알려주는데, dispatcher는 BAR의 doorbells에 변화가 생기는 경우 해당 doorbell과 연결된 submission queue/completion queue에 I/O 요청이 발생함을 알아내고 처리를 시작함. · 에뮬레이션을 하는 대상 디바이스의 특징에 따라 다양한 I/O 요청이 발생하게되고, 또한 대상 디바이스의 performance model에 따라 각 operation의 처리 시간이 다양하게 됨. NVMeVirt는 dispatcher가 timing model을 통해 해당 요청의 완료 시간을 계산하고 만약 해당 요청과 관련된 I/O가 모두 처리된 이후 완료 시간이 경과되었으면 해당 요청에 대한 completion을 알려주도록 구현됨. Dispatcher가 I/O의 시간을 모델링하는 반면 실제 I/O (예: 데이터 복사, index 수정 등)는 I/O worker라는 별도의 threads를 통해 처리하도록 하는 확장성(scalability)있는 구조임.

수행기관	개발 목표	개발내용 및 범위
		· I/O worker에서의 데이터 복사는 실제 장치에서의 데이터 전송 특성을 최대한 모사하기 위해 버퍼간 데이터 이동을 memcpy형태가 아닌 DMA Engine을 통해 수행함. 만약 I/O 요청의 데이터가 device memory-mapped region에 있는 경우 memcpy로 데이터를 옮기기 위해서는 수 bytes 수준의 수많은 PCI transactions이 발생하여 데이터 전송 성능이 매우 느려지게 되고, 이는 PCI peer-to-peer model에서의 I/O 성능에 큰 영향을 미침. 이에 대한 대안으로 Intel의 I/O Advanced Technology (Intel IOAT)를 통해 제공하는 DMA engine을 통해 직접 데이터를 전송하도록 하여 이러한 PCI transaction으로 인해 발생하는 문제를 해결하였음 · 여러 종류의 장치의 성능을 모사하기 위해 두 종류의 성능 모델을 구현하였음. "Simple performance model"은 Intel Optane, PMEM 등 NVM 등 단순하지만 병렬화된 구조의 스토리지 장치의 성능을 모델링함. 주요 구현은 독립적으로 동작할 수 있는 n개의 unit이 주어진 payload를 병렬적으로 처리하는 매우 단순한 구조임. 하지만 NVM 기반 SSD나 KVSSD 등의 경우 이를 통해 실제 장치와 비슷한 성능을 모사할 수 있었음 · Garbage collection이 필요한 NAND flash기반 스토리지 장치는 FTL의 기능과 디자인에 따라 성능 모델이 매우 복잡해 질 수 있음. NVMeVirt에서는 최근 SSD에 적용된 것으로 알려진 "write-buffer"와 "One-shot programming"을 고려하여 성능을 모델링 하였음. 또한 PCIe channel, NAND channel, NAND plane 등 병렬적인 데이터 처리를 위해 현대의 SSD 내부에 도입되는 병렬적인 구조에 대해 각 channel의 bandwidth를 제한할 수 있는 모델을 도입함으로써 기존의 에뮬레이터보다 훨씬 더 실제 장치의 성능을 잘 모사할 수 있었음 · NVMeVirt에서 channel은 일정 시간 내에 일정 양의 데이터를 전송할 수 있는 object로 모델링 되었다. 각 channel은 단위 시간에 일정 크기의 데이터를 전송할 수 있는 권리를 표현하는 tokens을 가지고 있다. 예를 들어 10us동안 1 KiB를 전송할 수 있는 token 30개를 가지고 있는 channel은 3 GiB/sec의 성능을 가진다고 볼 수 있다. SSD를 구성하고 있는 각 component는 channel을 통해 데이터를 전송하려면 전송하려는 데이터 만큼의 tokens을 가지고 가고 그 만큼만을 단위시간 내에 전송할 수 있다. 만약 tokens이 없거나 부족한 경우 해당 operations은 부족한 만큼 성능이 degrade되도록 된다. 이와 같은 방식을 통해 channel의 bandwidth를 정확하게 제어할 수 있다. - 연구 결과 평가  그림 342. 에뮬레이션 된 장치의 성능 변동성 평가 · 위의 그림 342은 현재 가장 널리 이용되고 있는 가상화 기술에 바탕을 둔 FEMU의 성능 변동성과 NVMeVirt의 성능 변동성을 비교하였음. 가상화에 기반을 두는 경우 VMENTER/VMEXIT을 통해 매우 많은 성능 변화가 발생하며 또한 가상머신의 게스트 물리 메모리가 호스트에서 할당이 되었는지의 여부에 따라 매우 높은 성능 변화율을 보여 고성능 NVM 기반 SSD의 성능을 잘 에뮬레이션하기 어려운 성능을 보여줌. 하지만 NVMeVirt의 경우 직접 커널의 모듈 형태로 구현되어 매우 낮은 오버헤드와 성능 변동성을 보여주어 에뮬레이션 대상 장치의 성능을 충분히 잘 조절할 수 있음을 보임

수행기관	개발 목표	개발내용 및 범위
		 그림 343. 실제 장치와 NVMeVirt를 통해 에뮬레이션한 장치의 성능 차이 · 그림 343은 NVMeVirt의 성능모사 수준을 보여줌. 파란 bars는 해당 workload에서의 Intel Optane SSD, Samsung 970 Pro SSD, SK Hynix의 prototype ZNS SSD, 삼성의 KVSSD의 성능을 나타내며 주홍 bars는 해당 장치의 성능을 NVMeVirt를 통해 모사하여 보여주는 성능을 비교하여 보여줌. 서로 완전히 다른 형태의 NVMe 프로토콜 상에 구현된 스토리지 장치들이지만 NVMeVirt에서 제공하는 modular한 performance model 구조를 통해 원하는 장치의 성능을 매우 비슷하게 모사할 수 있어 전체 비교실험에서 성능 error의 harmonic means은 약 1.16%임.  (a) Read latency on various payload sizes (b) Write latency on various payload sizes 그림 344. 채널 성능 모델로 인한 장치 성능모델 정확도 비교 · 그림 344는 각 payload size에 대한 Samsung 970 Pro에서의 성능과 FEMU에서의 성능, NVMeVirt에서의 성능 변화 추이를 비교한다. NVMeVirt는 현대 SSD내부에 매우 광범위하게 적용되어 있는 병렬화 기법들의 핵심인 channel의 bandwidth를 제어하는 방법을 제공한다. SSD 내부에 여러 개의 모듈들이 완전히 독립적으로 동작할 수 있더라도 해당 모듈들이 공유하고 있는 channel로의 데이터 전송은 contention이 발생할 수 있다. NVMeVirt에서는 이를 token에 기반을 둔 bandwidth 제어 기법을 통해 실제 장치의 성능을 매우 높은 정확도로 모사할 수 있다. 하지만 기존의 에뮬레이션 기법(FEMU)은 이러한 contention을 고려하지 않아 실제 장치보다 비현실적인 성능 값을 보여줌을 확인할 수 있다. · 그 이외에도 Megatron DeepSpeed의 checkpointing의 성능 특징, PCI peer-to-peer operation을 통한 GPUDirect의 성능 평가, MariaDB와 PostgreSQL의 성능 비교 실험 등을 통해 NVMeVirt를 이용해 다양한 최신 스토리지 환경을 쉽게 구성할 수 있음을 보여주었고 이를 이용해 다양한 스토리지 관련 연구를 수행할 수 있음을 보여주었다.
성균관 대학교 (공동2)	비휘발성 메모리 기반 스케일러블 인메모리 저장소 연구	**• 비휘발성 메모리 기반 스케일러블 인메모리 저장소 연구** **- 연구 배경 및 목표** · 원격 대용량 분리형 메모리 (Disaggregated Memory)를 파일-백드 페이지 (File-Backed Page) 용으로 활용하기 위한 확장형 분리형 페이지 캐시 모듈 구현 · ISCA 2009에 발표된 "Disaggregated Memory for Expansion and Sharing in Blade

수행기관	개발 목표	개발내용 및 범위

Servers" 논문에 의하면, 코어의 개수 증가율을 메모리 용량 증가율이 따라가지 못해 매년 코어 당 메모리 용량은 30%씩 감소하고 있음.

· 응용 프로그램 또는 개별 쿼리가 사용하는 메모리 용량은 일정하지 않고 시간에 따라 큰 격차가 있음.

· Peak 메모리 사용량을 고려하여 메모리를 개별 노드에 추가 설치하면 메모리 활용률이 크게 저하됨.

· 반면, 평균 메모리 사용량을 고려하여 메모리를 충분하지 않게 설치하면 메모리 스왑이 발생하고 이로 인해 IO가 발생하여 성능이 저하됨.

· ISCA 2009 논문은 이 문제를 해결하기 위해 익명 페이지들을 원격 분리형 메모리에 저장하는 방법을 제안함.

· 그러나, 데이터센터에서 수행되는 최근 응용들은 익명 페이지보다 대용량 입력 파일로 인한 메모리 부족도 심각함. 기존 분리형 메모리 연구들은 파일-백드 페이지에 대한 고려를 하지 않음.

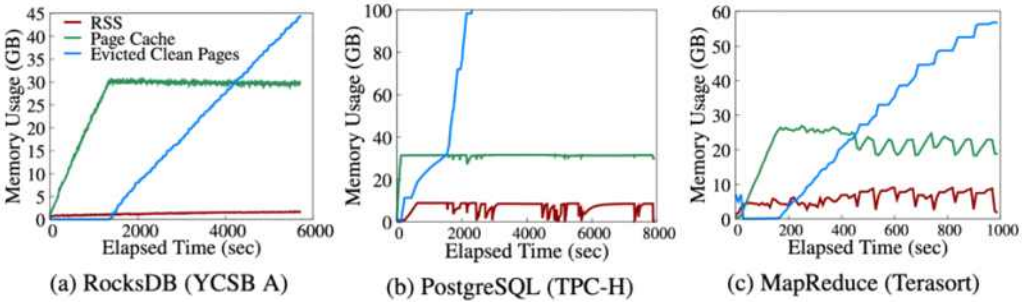


그림 345. Memory Footprint over Time

· 위 실험은 RocksDB 키-밸류 스토어, PostgreSQL DBMS, 그리고 Map-Reduce 등 대용량 데이터를 처리하는 응용들을 수행할 때 익명 페이지와 파일-백드 페이지들이 메모리를 사용하는 경향을 측정한 결과임.

· 붉은 색으로 표시된 메모리 사용량이 익명 페이지들이며, 익명 페이지가 메모리 압력(Memory Pressure)에 미치는 영향은 크지 않음을 알 수 있고, 입력 파일에서 읽은 페이지들의 용량이 월등히 큼을 확인할 수 있음.

· 페이지 캐시에서 쫓겨난 페이지는 파일 시스템에서 다시 읽을 수 있어 익명 페이지와 달리 디스크에 swap할 필요가 없음. 그러나, 페이지 캐시 미스는 이후 다시 디스크 IO를 발생시키기 때문에 큰 분리형 메모리를 활용할 수 있는 환경에서는 파일-백드 페이지도 분리형 메모리에 저장함으로써 디스크 IO를 감소시키고 성능을 향상시킬 수 있음.

• BalloonStasher: 파일-백드 페이지를 위한 분리형 메모리 시스템

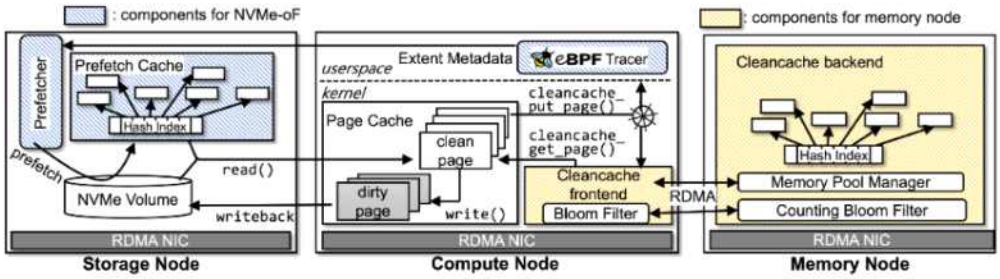


그림 346. BalloonStasher 아키텍처

- 본 연구에서 개발한 BalloonStasher는 원격 분리형 메모리 노드와 분리형 스토리지 노드를 함께 고려하는 파일-백드 페이지를 위한 분리형 메모리 시스템임.
- 위 그림에서 노란색으로 표시된 구성 요소들은 분리형 메모리 노드를 위한 모듈들이고 파란색으로 표시된 구성 요소들은 스토리지 노드를 위한 모듈들임.

수행기관	개발 목표	개발내용 및 범위

(1) **분리형 메모리 노드 모듈**

- 분리형 메모리 노드로 Page Cache를 확장하는 경우, Linux Kernel의 페이지 캐시에서 버려지는 File-backed Page를 Disaggregated Memory 노드로 RDMA 전송함.

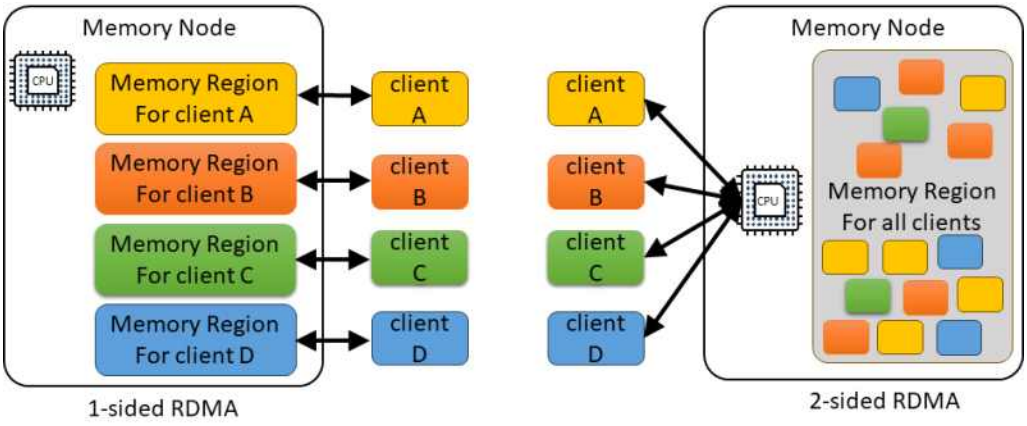


그림 347. 1-sided vs 2-sided 디자인

- 1-sided RDMA를 사용하여 분리형 메모리를 활용하기 위해서는 Infiniswap과 같이 개별 클라이언트에게 메모리 영역을 독점 할당해야 함. 메모리 접근 지연 시간이 짧다는 장점이 있으나, 개별 클라이언트가 필요로 하는 메모리 양이 동적으로 빠르게 변화할 때 적절하게 대응하지 못한다는 단점이 있음.
- 2-sided RDMA를 사용하여 분리형 메모리를 사용하면, 1-sided RDMA에 비해 메모리 접근 지연 시간이 길다는 단점이 있음. 그러나, 2-sided RDMA를 사용하여 전송함으로써 메모리 노드의 CPU가 개입하여 캐시 교체를 수행하고, 이를 통해 현재 busy한 클라이언트가 더 많은 메모리 공간을 활용할 수 있기 때문에, 클라이언트들의 메모리 요구량에 빠르게 대응할 수 있다는 장점이 있음.
- 1-sided RDMA를 사용하면서 2-sided RDMA와 동일한 방식으로 캐시 교체 정책을 수행하여 두 방법의 장점만을 취할 필요가 있음.

(2) **분리형 스토리지 노드 모듈**

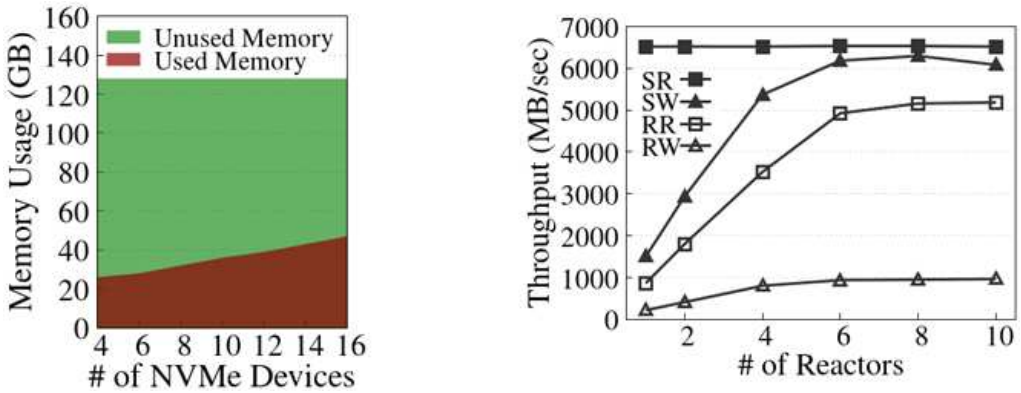
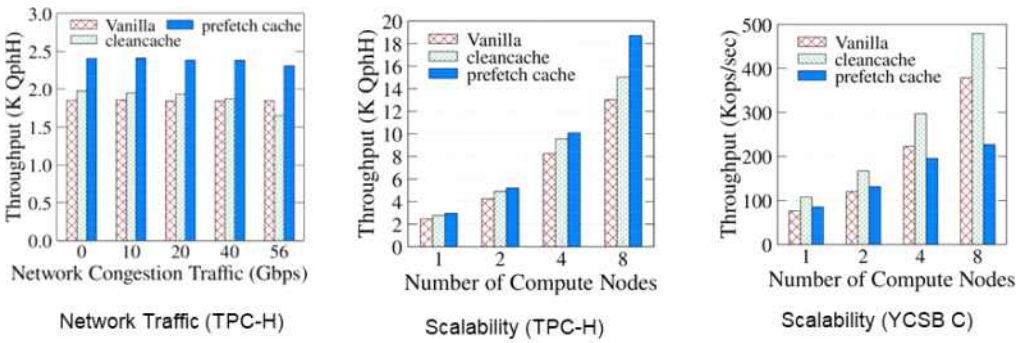


그림 348. 분리형 스토리지 노드 자원의 낮은 활용도

- 위 실험 그래프는 삼성전자와 Inspur가 공동으로 개발한 NVMe-oF 기반 분리형 스토리지 관리 시스템인 PoseidonOS의 자원 활용율을 보여줌. NVMe 디바이스 1개당 2GB 정도의 DRAM을 사용함. 또한, CPU core는 6개 이상 활용하지 않음을 확인할 수 있음. 즉, 자원 사용율을 높이기 위해 제안된 분리형 시스템에서 새로운 형태의 자원 활용율 저하 문제가 발생함.
- 분리형 스토리지의 메모리를 활용하기 위해 스토리지 노드의 DRAM을 캐시로 사용하는

수행기관	개발 목표	개발내용 및 범위
		방법을 생각할 수 있으나, NVMe-oF의 특성상 계산 노드에 파일 시스템 캐시 또는 사용자 레벨 캐시가 존재하므로, 스토리지 노드 캐시는 계산 노드의 캐시와 동일한 페이지를 갖게 되어 활용율이 높지 않을 수 있음. - 이 문제를 해결하기 위해 본 연구에서는 계산 노드에서 버려지는 페이지 정보를 얻어 PoseidonOS로 전송하고, PoseidonOS는 버려진 페이지를 NVMe volume으로부터 읽어서 Prefetch Cache에 저장하는 구현을 함. 페이지를 계산 노드로부터 스토리지 노드로 전송하지 않고 NVMe 볼륨으로부터 Prefetching하는 이유는 NVMe volume에 이미 있는 페이지를 다시 네트워크로 전송하는 비효율을 피하기 위해서임. - Disaggregated Memory Node와 Disaggregated Storage Node를 함께 쓰는 환경에서 cleancache에서 버려지는 페이지를 Memory Node로 전송할 지 아니면 Storage Node로 전송할 지 판단하기 위해, 버려지는 페이지들의 주소가 연속되어 있는지 아닌지를 확인하여 연속된 페이지들은 Storage Node가 캐싱하도록 하여 이들을 Sequential Read하고 Prefetch하도록 함. 반면 Random하게 읽는 페이지들은 Memory Node로 전송하게 함. - 페이지를 읽을 때는 버려진 페이지가 Memory Node에 캐싱되어 있을지 Storage Node에 있을 지 알 수 없기 때문에, 불필요한 Memory Node 접근이 발생할 수 있음. 이를 최소화하기 위해 Bloom Filter를 분산 환경에서 사용하 있도록 개선함. - 아래 실험에서는 Compute Node와 Storage Node만 사용하는 환경에서 스토리지 노드의 메모리를 활용하지 않는 Vanilla의 성능과, Cleancache를 통해 버려지는 페이지를 RDMA로 모두 전송하여 캐싱하는 cleancache 설정, 그리고 스토리지 노드가 페이지를 NVMe volume에 이미 가지고 있다는 점에 착안하여 버려지는 페이지의 메타데이터만 전송하고 NVMe로부터 해당 페이지들을 읽어서 prefetch cache에 저장하는 성능을 비교한 실험임.  그림 349. 메모리 노드 캐시 vs 스토리지 노드 prefetch 캐시 성능 비교 - Sequential IO가 많이 발생하는 TPC-H의 경우 Prefetch Cache의 성능이 월등히 우수함. 반면 Random IO가 많이 발생하는 RocksDB YCSB 워크로드의 경우 Prefetch Cache는 Random IO로 인해 Vanilla보다도 성능이 저하됨. (3) **분리형 메모리-스토리지 Steering** - 분리형 메모리와 스토리지 노드가 함께 사용되는 환경에서 계산 노드는 버리는 파일-백드 페이지를 메모리 노드로 보낼지 스토리지 노드로 보낼지 결정해야 함. - 스토리지 노드는 메모리 노드와 달리 버려진 페이지를 디스크에서 읽는 방식을 취하므로 random IO를 피하기 위해 연속된 페이지 (즉, extent)가 버려질 때 해당 extent를 prefetch하는 것이 바람직하며, 메모리 노드는 random하게 읽은 페이지가 버려질 때 해당 페이지를 메모리 노드로 보내 second chance를 주는 것이 바람직함. - 아래 실험은 스토리지 노드와 메모리 노드가 모두 있을 때 IO pattern에 따라 버려진 페이지를 Storage Node로 보낼지 Memory Node로 보낼지 판단하는 기법 Steering(Pattern)과 Random하게 결정하는 Steering(Random), 그리고 Vanilla의 성능을 비교한 실험임.

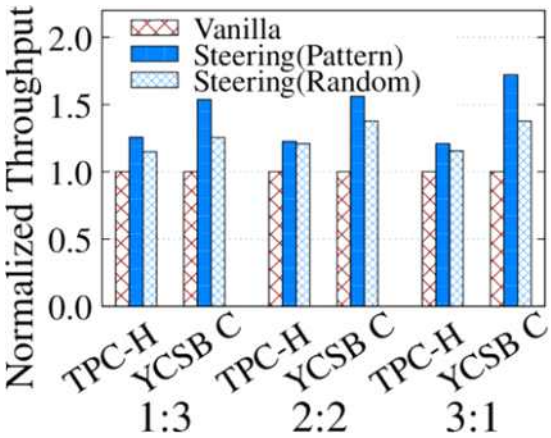
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>- 워크로드는 Sequential Read를 많이 하는 TPC-H와 Random Read를 많이 하는 YCSB C 워크로드의 비율을 1:3, 2:2, 3:1로 변화시켰음. IO Pattern에 따라 Steering을 하는 기법은 Vanilla에 비해 항상 성능이 높음.
 

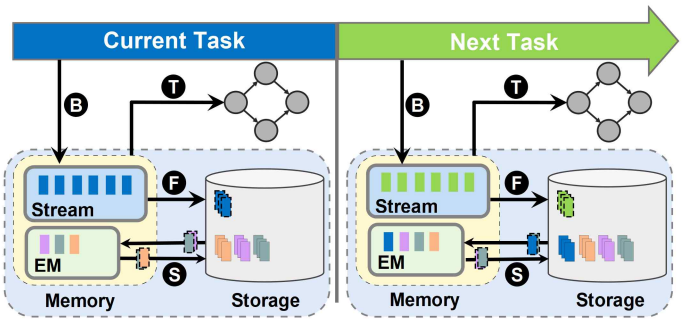
그림 350. Steering 정책 비교</td></tr>
<tr><td>UNIST
(공동3)</td><td>다계층 메모리 기반 분산 인공지능 플랫폼 연구</td><td>• 분산 플랫폼 기반 Continual Learining 기법 연구
- 기술 개발의 필요성
▪ 본 연구에서는 분산 플랫폼 기반 지속 학습(Continual Learning, CL) 기법 연구의 일환으로 on-device 지속학습을 위한 자원 효율적인 CL 플랫폼을 개발하여 자원 효율적인 지속 학습을 가능하게 하였음.
▪ 지속 학습(Continual Learning)은 기존 딥러닝을 지속적으로 학습하기 위한 방법론.
▪ 이러한 지속 학습의 가장 중요한 해결 과제 중 하나는 치명적 망각(Catastrophic Forgetting) 문제. 치명적 망각 문제는 딥러닝 신경망이 새로운 지식을 점진적으로 학습할 때, 과거에 학습된 데이터에 대한 성능(정확도)이 급락하는 것을 지칭함.
▪ 이러한 망각 문제를 해결 방법 중 가장 효과적인 방식으로 새로운 데이터와 이전 데이터를 학습 과정에 포함하는 에피소드 메모리(Episodic Memory, EM)을 이용한 기억 방식.
 

그림 351. 계층적 에피소드 메모리(HEM)의 구조 및 실행 단계.
▪ 본 연구의 선행 연구로써 CarM(DAC'22)은 계층적 에피소드 메모리(Hierarchical Episodic Memory, HEM) 구조를 이용하여 효율적인 에피소드 메모리 활용을 달성.
▪ 그림 351에서는 이러한 계층적 에피소드 메모리를 활용하여 지속 학습을 수행을 도식화한 것임. 현재 단계에서 학습한 데이터를 다음 단계에서도 학습이 가능하도록 구성하고, 한정된 메모리 자원을 효율적으로 활용하기 위해서 메모리와 저장장치 간의 데이터 스왑을 통해 높은 정확도를 유지.</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

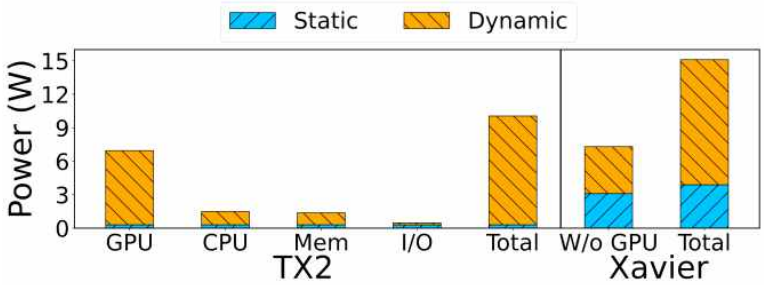


그림 352. HEM을 사용할 경우의 주요 시스템 구성 요소의 전력 소비 (NVIDIA TX2, and Xavier).

- 그림 352은 on-device 지속 학습이 분산 플랫폼인 엣지 플랫폼을 대상으로 했을 경우를 상정해 NVIDIA에서 출산 Jetson TX2와 Jetson Xavier 모델에 대해서 학습을 진행하고, 그에 따른 에너지 소모량을 측정한 그래프.
- 그림 352에서 볼 수 있듯, 가장 많은 에너지 소모를 일으키는 엣지 장비 내 컴포넌트는 GPU이고, GPU 사용 시간이 엣지 플랫폼의 에너지 소비 효율에 가장 큰 영향을 미치는 것을 볼 수 있음.
- GPU 사용 시간에 가장 영향을 많이 미치는 요인은 모델 자체의 학습 시간(iteration 시간)과 모델의 학습의 필요한 반복 횟수(number of epochs)임. 모델 자체의 학습 시간은 모델 자체의 고유적인 것이나, 학습에 필요한 반복 횟수는 학습 효율을 높일 경우, 효율성을 달성할 수 있음.
- 본 연구에서는 이러한 HEM 기반 시스템의 on-device 확장을 위한 기반 연구 및 효율적인 시스템을 개발함.

- Miro: 자원 효율적인 on-device 계층적 메모리 지속 학습 플랫폼

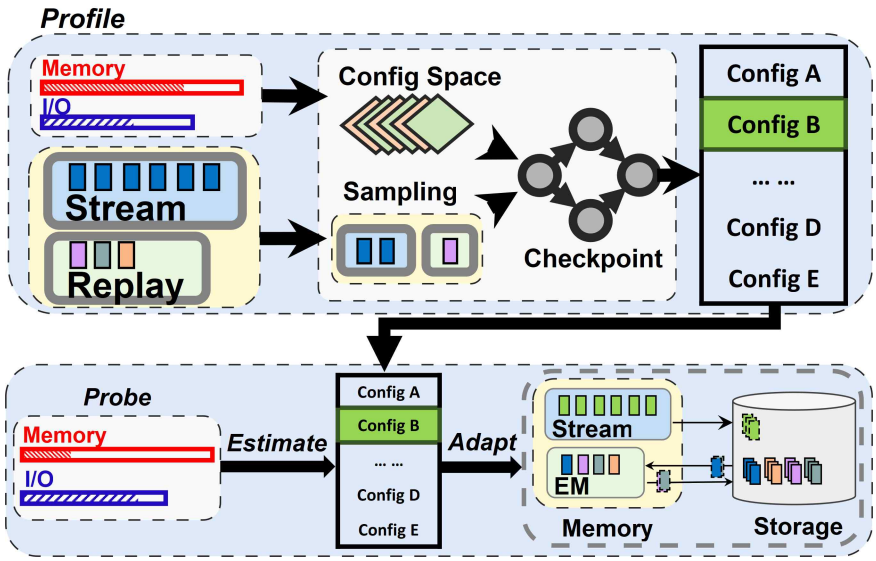
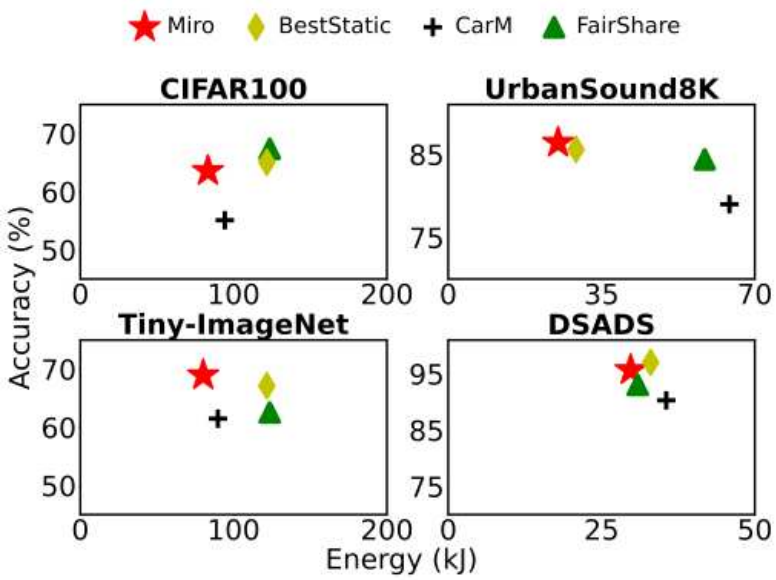
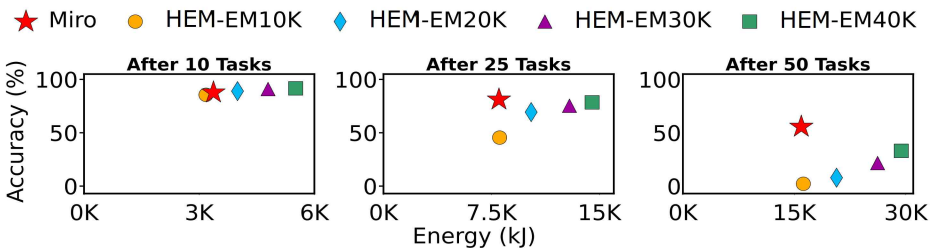
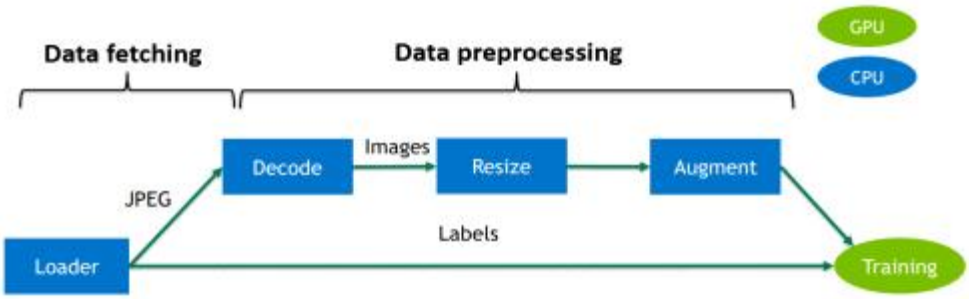
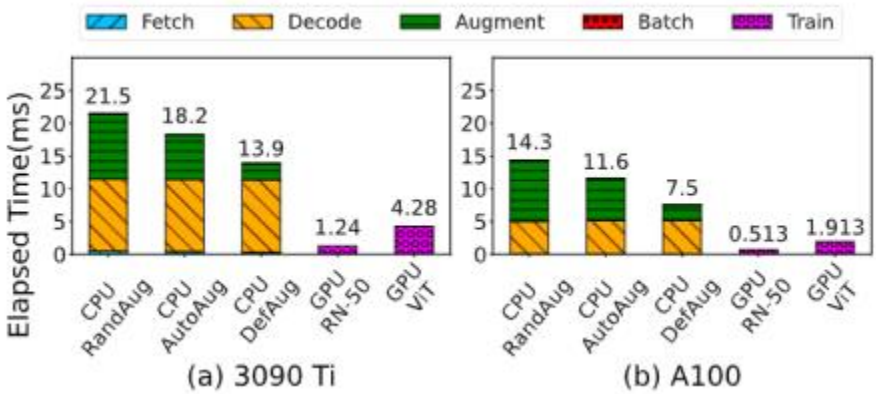


그림 353. Miro 런타임 시스템 구조도.

- Miro의 시스템 구조는 3가지로 구성됨. (그림 353)
- Miro의 작업은 1. 프로파일 단계, 2. 최적 구성(configuration) 설정, 그리고 3. 자원 모니터링 및 변경된 자원에 대한 최적 구성 재적용으로 구성됨.
- 프로파일 단계에서 Miro는 작은 오버헤드(overhead)로 가능한 구성의 성능을 추정하고, 유틸리티 스코어와 컷라인(cutline)을 사용하여 최적의 구성을 식별. 학습하는 동안 Miro는 프로브(probe) 상태에서 시스템 자원을 모니터링하고 변경 사항이 감지되면, 최적 구성을 다시 측정(estimate)한 후, 적응(adapt) 단계에서 새 구성을 적용.

수행기관	개발 목표	개발내용 및 범위
		▪ Miro의 프로파일 단계는 실제 시스템 처리에서 7~10%의 오버헤드만 발생하며, 이는 Miro가 제공하는 이점에 충분히 상쇄됨. - Miro 시스템의 성능 평가 결과  그림 354. 여러 데이터 세트와 모델에서의 Miro의 성능 비교. ▪ 그림 354에서는 Miro의 일반화 가능성을 위해 이미지 분류, 오디오 분류, 그리고 인간 활동 인식(Human Activity Recognition, HAR) 세 가지 종류, 네 가지 데이터 세트에 대해서 실험함. ▪ Miro와 비교하기 위한 비교군으로는 1) 모든 작업을 분석한 결과 가장 좋은 구성인 BestStatic, 2) 계층적 에피소드 메모리(HEM)을 사용한 선행 연구인 CarM, 그리고 3) Miro에서 수행하는 메모리 버퍼를 동적으로 변경하는 것이 아닌 균등하게 배분한 FairShare. ▪ 본 연구에서 목표로 하는 자원 효율적인 지속 학습의 성능은 적은 에너지를 사용하고, 높은 정확도를 획득할수록 높다고 볼 수 있음 (그래프 상에서 좌상단으로 갈수록, 적은 에너지와 높은 정확도를 획득한 것임). Miro는 테스트한 4개의 데이터 세트에서 모두 소비된 에너지 당 더 높은 정확도를 획득함.  그림 355. 대형 데이터 세트에서의 Miro의 성능. ▪ 그림 355에서는 실제 지속 학습이 이뤄지는 환경을 설정하여, 대형 데이터 세트인 ImageNet-1K (146GB)에 대해서 Miro의 성능을 측정한 값임. 실험은 146GB의 데이터 세트를 50개의 작업(task)으로 분할함. ▪ 실험의 진행은 선행 연구였던 CarM의 구성에서 고정된 에피소드 메모리들을 기준으로 하였고, 그림 355에서 볼 수 있듯, Miro는 정확도와 에너지 소비량을 크게 개선함. 특히, 이 격차는 작업이 후반으로 갈수록 더 커짐. 50 작업 수행 이후인 가장 오른쪽 그래프를 보면, 다른 실험 대조군들은 정확도가 현저하게 떨어지지만, Miro의 경우

수행기관	개발 목표	개발내용 및 범위
		런타임 조정을 통해서 정확도 손실을 크게 줄인 것을 볼 수 있음. • 다계층 메모리 기반 인공지능 데이터 관리 기법 연구 - 기술 개발의 필요성 ▪ 본 연구에서는 데이터 전처리가 중요한 이미지 관련 인공지능 모델을 대상으로 동적 데이터 증강 단계를 병렬화하여 전체 DNN 학습 속도를 높임. ▪ 합성곱 신경망 데이터 로딩 파이프라인은 이용하는 모델마다 다르며, 대부분의 인공지능 플랫폼에서 사용자가 쉽게 수정할 수 있도록 지원함. ▪ 인공지능 학습에 사용되는 원본 이미지는 480x480 크기의 JPEG 이미지로써 이미지 해독 과정(Decoding)을 거쳐야 이미지 처리가 가능함. 이러한 이미지들은 원본 그대로 학습되지 않고 이미지 학습 과정에 해로운 극단적인 아웃라이어(outlier)를 제거하기 위해서 여러 과정을 거침. 기본적인 방법으로는 이미지의 크기를 변경시키고, 색조를 변경하는 등의 이미지 변환 이후에 학습을 위한 크기로 잘라내는(crop) 과정을 거치게 됨. 이 과정은 그림 356에서 볼 수 있듯이, 데이터 불러오기(data fetching)와 데이터 전처리(data pre-processing)으로 구분됨.  그림 356. 기본적인 데이터 로딩 파이프라인 (2019년 NVIDIA GTC 발표 자료). ▪ GPU를 이용한 모델 학습과 학습을 위한 데이터 전처리 과정은 순차적으로 이루어져 있으므로 CPU에서의 데이터 전처리가 늦어지면서 GPU의 활용율이 떨어져 전체 학습 성능 저하로 이어지는 현상을 발견.  그림 357. 데이터 증강 알고리즘과 모델별 CPU와 GPU에서의 샘플당 처리시간. ▪ 그림 357은 RandAugment와 AutoAugment를 대표적인 ResNet 50 모델과 이미지 학습에서 가장 큰 성능을 차지하는 ViT 모델에 대해서 CPU와 GPU에서의 샘플당 처리시간을 분석한 그래프임. ▪ 그래프에서 보기와 같이, GPU의 활용율을 높이기 위해서는 RTX 3090 GPU에서 ResNet 50의 경우는 18개와 15개의 CPU가 필요함. 이는 CPU와 GPU의 비율이 통상적으로 5.3 대 1 정도인 것을 감안할 때, 학습 과정에서 병목은 필수 불가결함.

수행기관	개발 목표	개발내용 및 범위
		▪ 이런 문제를 해결하기 위해 본 연구에서는 CPU-GPU를 통합적으로 사용하는 공유 메모리 기반 DNN 학습 성능 최적화 기법을 개발함. - FusionFlow: 세분화된(fine-grained) 협력적 병렬화(coorerative-parallel)를 통한 DNN 데이터 로딩 시스템 ▪ 본 연구에서 우리는 CPU와 GPU에 대해 협력적으로 Intra Batch 병렬 처리를 활용하여 데이터 증강 속도를 높이고 균형을 맞추는 DNN 프레임워크인 FusionFlow를 설계함. FusionFlow는 세 가지 주요 설계 통찰력을 기반으로 구성됨. ▪ 첫째, FusionFlow에서는 학습 미니 배치(mini-batch)를 더 작은 타이니 배치(tiny-batch)들로 수직으로 분할하여 CPU 또는 GPU에 타이니 배치들을 할당하여 병렬로 처리하는 Intra-Batch 병렬 처리(Intra-P)를 수행함. 더 작은 배치들을 병렬로 처리하는 Intra-batch 병렬 처리(Intra-P)를 활용하여 기존 데이터 로딩 워크플로(workflow)의 한계를 극복하고 CPU와 GPU 모두에서 데이터 증강 작업을 협력적으로 조정(schedule)할 수 있도록 구현. 임박한 미니 배치에 우선순위를 지정하고 컴퓨팅 자원(resource)을 할당함으로써 선형 속도 향상을 얻을 수 있고 아주 작은 병렬화 오버헤드(overehead)만 발생시킴. ▪ 둘째, FusionFlow에서는 유휴 GPU 사이클(idle GPU cycle)을 최대한 활용하여 빠른 속도로 타이니 배치(tiny-batch)를 처리하여서 종합적으로 미니 배치(mini-batch) 계산 시간을 줄였다. GPU 커널 구성(kernel construction) 및 메모리 할당에 상당한 오버헤드를 유발할 수 있는 GPU 작업(task) 생성을 모든 타이니 배치마다 하지 않도록 사전(ahead-of-time) 작업 할당을 사용하여서 GPU kernel 구성 및 생성으로 인한 전체 워크플로우의 오버헤드를 감소시킴. 이것은 런타임 스택(runtime stack)을 우회하기 위해 가능한 모든 데이터 증강 파이프라인(pipeline)조합을 미리 생성하고 호스트와 GPU 간의 GPU 메모리 사용량 및 메모리 복사 작업을 최소화함. ▪ 셋째, FusionFlow는 미니 배치가 먼(remote) GPU에 속하더라도 현재 반복(iteration)의 미니 배치들을 준비 완료할 수 있도록 워커(worker)를 동적으로 조정(schedule)할 수 있도록 시스템을 구성함. 학습은 모든 GPU에서 현재 미니 배치를 완료하지 않고는 다음 반복(iteration)으로 이동할 수 없기에 다음 반복을 위해 미니 배치를 준비하는 것은 도움이 되지 않음. 따라서 FusionFlow는 GPU에 할당된 로컬(loacl) CPU 워커(worker)들을 재조정(rebalance)하여 글로벌 도메인(global domain)에서 느린 데이터 준비를 촉진시킴. ▪ FusionFlow는 로컬 매니저(local manager), 글로벌 매니저(global manager) 및 워커(worker)로 구성됨. 시스템 전반에 걸쳐 효과적인 Intra-Batch 병렬 처리 실행을 위해 이러한 구성 요소를 조정하는 통신 프로토콜(protocol)을 활용했고, 들어오는 각 미니 배치(mini-batch)에 대해 로컬 매니저(local manager)는 더 작은 타이니 배치(tiny-batch)로 나누어 Intra-Batch 병렬 처리를 할 수 있도록 하고 작업 스케줄링(scheduling)을 수행함. ▪ 글로벌 매니저(global manager)는 로컬 매니저(local manager)의 상태를 추적하고 진행 상황을 조정함. 글로벌 매니저의 주요 임무는 동적 리소스 재배치를 통해 데이터 병렬 학습에서 스트래글러(straggler)를 완화하는 것임. 반복(iteration) 중에 스트래글러가 식별되면 글로벌 매니저는 스트래글러 영향(straggler effect)을 완화하기 위해 점진적으로 작업자를 재배치함. 이를 통해 진행 속도가 느린 로컬 매니저에는 반복이 끝날 때까지 보류 중인 작업을 추가로 병렬화하기 위해 더 많은 워커가 부여되는 구조로 이루어져 있음. ▪ FusionFlow 시스템에서는 이러한 작업자 증강의 경우, 오예측의 위험을 없애는 방향으로 워커 스케일링(worker scaling)을 수행하고 워커를 재배치하게 구현됨.

수행기관	개발 목표	개발내용 및 범위

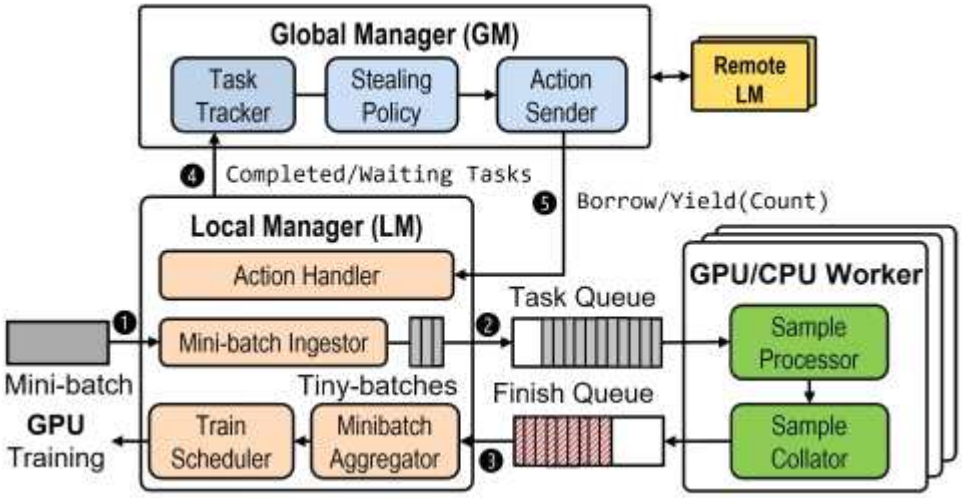


그림 358. FusionFlow 시스템 구조도

- 그림 358은 FusionFlow 시스템의 시스템 구조를 도식화함. 가장 먼저 DNN 프레임워크에서 정의한 순서에 따라 미니 배치들을 하나씩 받아오고 (1번), 세분화된 Intra-Batch 병렬화 작업을 생성하여 워커들에게 전달하고 (2번), 처리된 작업들을 미니 배치로 모아 모델 학습을 진행. (3번)
- 로컬 매니저는 각각의 다른 로컬 매니저와 유사한 속도로 작업을 하기 위해 CPU 워커들을 확장하거나 축소하는 작업 처리기(action handler)를 통해서 작업을 진행.
- 이러한 작업의 효율적 처리를 위해서 글로벌 매니저는 주기적으로 Intra-Batch 병렬 작업(task) 처리에 대한 진행 상황을 보고하고 (4번), 워커 확장 및 축소에 대한 지침을 받음. (5번)

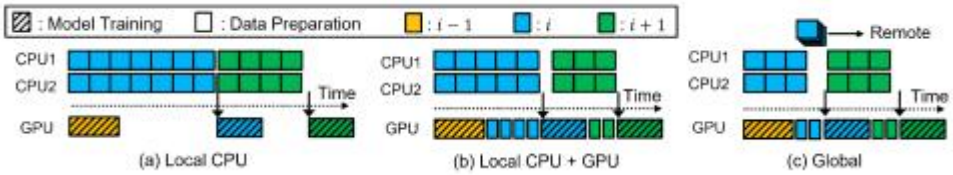


그림 359. 배치 작업을 진행하는 방식에 따른 모델 학습 작업 타임라인 개조도.
(a) CPU만을 활용한 배치 작업 처리,
(b) CPU와 GPU를 활용한 배치 작업 처리,
(c) 외부 CPU를 추가적으로 활용할 경우 배치 작업 처리.

- 그림 359에서는 배치 작업 처리의 향상이 어떻게 이루어지는지를 도식화 함. (a) Local CPU만을 사용하여 Intra-Batch를 처리하는 경우, (b) CPU-GPU가 협력적으로 Intra-Batch 처리하는 경우, 그리고 (c) 다른 로컬 매니저의 워커로 확장 처리가 추가된 경우.
- (c) 의 경우가 훨씬 효과적으로 데이터 전처리 유휴 시간을 줄일 수 있음을 직관적으로 확인 가능함.

- FusionFlow 시스템의 성능 평가 결과

수행기관	개발 목표	개발내용 및 범위

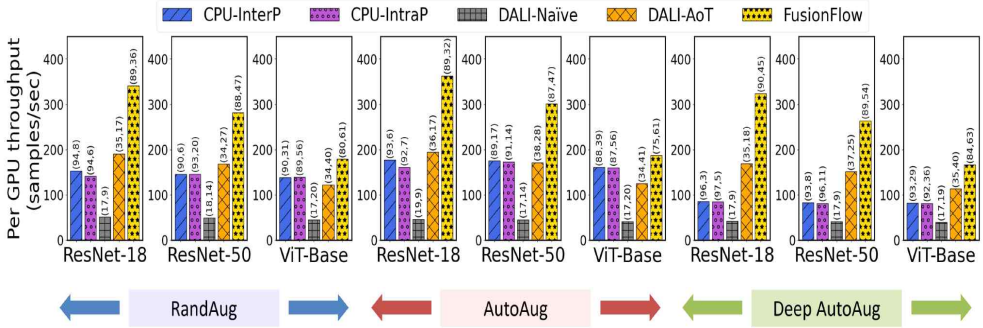


그림 360. 단일 GPU에서 ResNet-18, ResNet-50 및 ViT의 학습 처리량. 순서대로 RandAugment, AutoAugment 및 DeepAutoAugment에서 처리량.

- 그림 360는 서로 다른 모델과 데이터 증강 알고리즘에 대한 세 가지 병렬 처리 방법의 처리량을 보여줌. 비교의 용이성을 위해 처리량(throughput)은 단일 GPU의 처리량(samples/sec)으로 표기. 또한, 그래프 상단에 (CPU, GPU 활용률)을 표기함. FusionFlow와 비교하기 위한 두 개의 비교군은 다음과 같음.
- CPU-InterP는 CPU에서 Intra-Batch 병렬화를 수행하는 PyTorch의 기본 방법으로 본 연구에서 베이스라인(baseline)으로 삼은 방법. CPU-IntraP의 경우, CPU에서 Intra-Batch 병렬화를 수행하는 것을, DALI-Naive의 경우, GPU에서 Intra-Batch 병렬화를 유휴 로컬 GPU 사이클을 이용하여 수행하지만, FusionFlow의 주요 최적화 기술을 적용하지 않은 상태를, DALI-AoT의 경우, GPU에서만 Intra-Batch 병렬화와 FusionFlow에서 사용된 사전 작업 할당(ahead-of-time task allocation)을 적용한 상태.
- 모든 경우에서, FusionFlow는 다른 비교군보다 높은 성능을 보여주는 것을 볼 수 있고, 특히 ResNet-18의 경우, 베이스라인에 비해서는 105-281%, DALI-AoT의 경우보다 78.5-91.7% 성능 향상을 보여줍니다. 또한, 상대적으로 모델의 GPU 학습 시간이 긴 ViT 모델의 경우에도 베이스라인에 비해 29.7%, 16.1%, 그리고 102%의 성능 향상을 RandAugment, AutoAugment, 그리고 DeepAutoAugment에서도 성능 향상을 보임.
- 또한, 데이터 전처리에 시간이 감소하고, 유휴 GPU를 활용한 데이터 전처리로 GPU 사용률이 높은 폭으로 상승하는 것을 볼 수 있음.

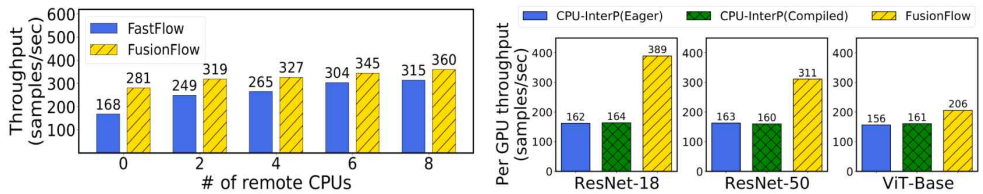
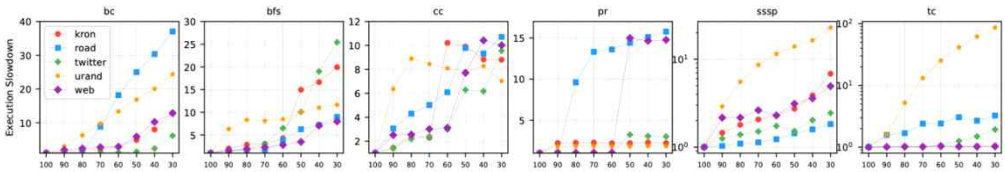


그림 361. FusionFlow의 확장성을 비교하기 위한 실험. 왼쪽은 외부 CPU를 활용하는 최신 CPU 기반 기술인 FastFlow와 FusionFlow의 비교, 오른쪽에서는 향상된 프레임워크인 PyTorch 버전 2에서의 FusionFlow의 처리량.

- 그림 361은 FusionFlow를 다양한 확장성을 평가하긴 위해서 실험은 진행한 것임.
- 왼쪽 그래프는 현재 CPU 기반 데이터 로딩 프레임워크 중 외부 CPU를 활용하여 성능을 높인 FastFlow (VLDB 2023)와 FusionFlow를 비교. 처리량(throughput)은 단일 GPU의 처리량(samples/sec)으로 표기하였고, 외부 자원인 CPU 수를 늘리면서 변화를 비교함.
- 그림 361에서 볼 수 있듯이, FusionFlow는 외부 CPU를 50-60% 적게 사용하면서 동일한 처리량을 달성할 수 있었고, 외부 CPU를 활용하는 디자인과도 상보적으로 작용할 수 있다는 것을 보여줌.
- 또한, 그림 361에서 오른쪽 그래프의 경우 대표적 딥러닝 학습 프레임워크인 PyTorch의 경우, PyTorch 버전 2에서 비약적인 학습 처리량 향상을 이루었고, 이러한 향상은

수행기관	개발 목표	개발내용 및 범위
		GPU에서의 모델 학습 속도를 높였기에, 본 연구와 상보적으로도 작동할 수 있고, 실제로 PyTorch 버전 1.8을 사용한 성능보다 높은 처리량을 보임을 확인할 수 있었음. 이러한 실험들을 통해서 FusionFlow가 단독으로써의 성능뿐 아니라 다른 시스템들과 상보적으로 성능 향상을 이뤄낼 수 있다는 것을 확인함.
웁살라 대학교 (공동4)	대규모 그래프 분석 가속 기술 연구	• **Graph Processing on future memory systems** - 본 연구는 그래프 데이터 크기와 시스템 주 메모리 크기(DRAM) 간의 간격이 계속해서 확대될 것으로 전망함. - 이 문제를 해결하기 위해, 본 연구는 상용 시스템과 상용 저장 장치를 활용하여 대규모 그래프 처리를 지원하는 데 중점을 둠. · 이러한 시스템의 필요성은 하드웨어 변경 없이(하드웨어 변경은 디자인/검증 비용이 많이 들며 그래프 알고리즘 변경에 불리함) 더 나은 그래프 처리를 제공하는 것임. (1) 본 연구는 응용 프로그램 투명한 방식을 목표함. · 리눅스 커널의 가상 메모리 및 paging/swapping[1)]을 활용하여 DRAM보다 큰 그래프를 지원함. · Swap 장치로는 소비자용 저장 장치 (Samsung PCIe 4.0 NVMe)를 사용함 · 일반적인 시스템 하드웨어를 가정함. - 발표된 연구[2)]는 두 단계로 보고함: · 1. 현재의 그래프 처리를 위한 시스템 병목 현상 분석. · 2. 현재 병목 현상을 해소하기 위한 제안. - 발표된 연구 결과 외에도, 더 많은 Swap 장치 및 다른 시스템 구성에 대한 시스템 동작에 대한 추가 분석을 수행함. • 현재 시스템 병목 현상에 대한 분석 - 상용 SSD 장치를 사용하는 최근 시스템에서 Paging(또는 Swapping)과 관련된 오버헤드를 분석하기 위해 수행함. - Linux 커널 6.3.0에서 실험을 수행하였으며, PCIe 3.0 버스를 통해 시스템에 연결된 Samsung 980 Pro 1TB NVMe SSD를 Swap 장치로 사용함. - cgroups 기능을 사용하여 DRAM 사용량을 100%에서 30%로 제한함. - GAP 벤치마크 스위트(6개의 그래프 알고리즘) 및 해당 스위트에서 제공하는 5개의 입력 그래프를 사용함.  그림 362. 각 그래프 알고리즘-그래프 입력 데이터의 전체 메모리 사용량 대비 DRAM에 할당하는 메모리의 비율 (x축)에 따른 성능 변화 - 성능 저하 결과는 그림 362에서 보여줌. · 성능 저하는 알고리즘 및 입력 그래프 유형에 따라 서로 다른 영향을 받음. - 추가 분석을bfs 알고리즘을 web 그래프 입력에 집중함.

수행기관	개발 목표	개발내용 및 범위

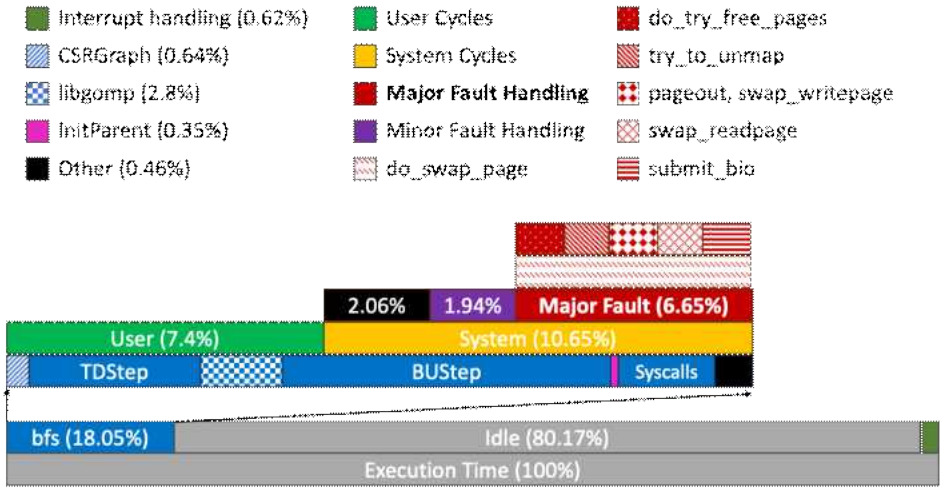


그림 363. BFS 알고리즘과 Web 입력 데이터의 실행시간 세분화.

- bfs:web 실행 시간의 세분화를 측정하여 그림 363에 flame graph 형식으로 나타냄.
  · perf record 및 cpu-clock 소프트웨어 이벤트를 사용하여 측정되었으며, 이는 정확한 idle time (유휴 시간)을 캡처할 수 있는 유일한 이벤트였음.
  · 시스템이 유휴 상태에서 80%의 시간이 소비함.
  · 응용 프로그램을 실행하는 데는 전체 실행 시간의 18%가 소비됨.
    (1) User space에서 7.4%의 시간이 소비됨.
    (2) 커널에서의 소비는 10.7%이며, 그 중 6.7%는 page fault를 처리하고 Swap 장치로 page fault를 요청하는 데 소비됨.
  · 결론: 시스템이 너무 많은 시간을 유휴 상태에서 소비하고 있음.

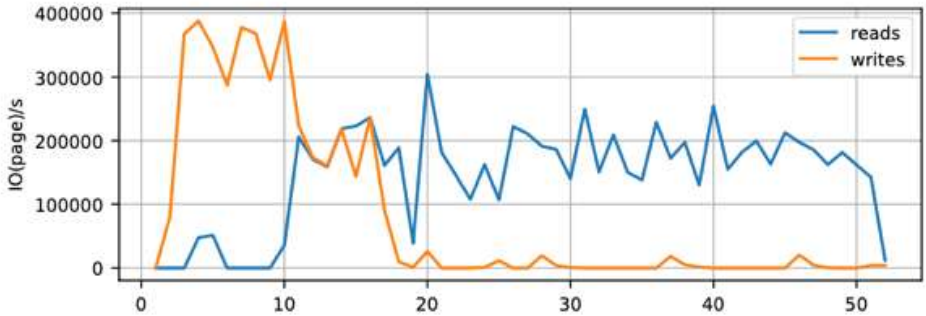


그림 364. BFS-web의 전체 실행 시간 중 paging에 의한 SSD로의 읽기/쓰기 페이지 수

- bfs:web 실행 중에 시간에 따른 Swap 장치의 입 출력을 측정하였음. (그림 364)
  · 프로그램 실행에 따라 총 세 단계의 입 출력 패턴을 확인함.
    (1) 쓰기 집중 단계: 사용 가능한 메모리가 부족하여 초기 임page-out 이 발생하는 단계.
    (2) 읽기/쓰기 단계: Page-out을 계속해서 수행하고 데이터를 사용하려는 page-in 요청이 발생하는 상태. 응용 프로그램이 대량의 데이터에 액세스하기 시작하지만 시스템은 아직 evict할 page들의 page-out을 실행하는 단계 .
    (3) 읽기 집중 단계: 데이터가 재사용되어 page-in 되지만 주로 읽기 전용 액세스임.
- Swap 장치의 성능 특성을 확인하기 위해 SSD의 성능을 평가하였음. (그림 365)

수행기관	개발 목표	개발내용 및 범위

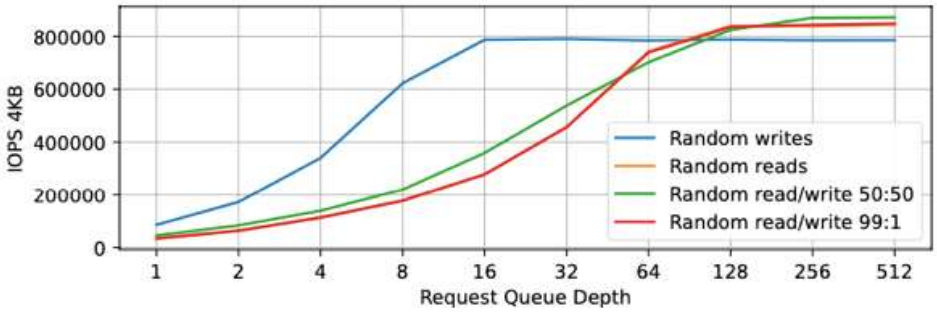


그림 365. SSD의 요청 수 대비 대역폭 성능

· Swap 장치의 성능(그림 365)과 시간에 따른 paging(그림 364)을 비교한 결과, swap 장치 대역폭이 충분히 활용되지 않고 있음을 확인함.
· SSD는 처리량을 극대화하기 위해 I/O Request Queue (입출력 요청 큐)에 충분한 요청이 필요함(그림 365)

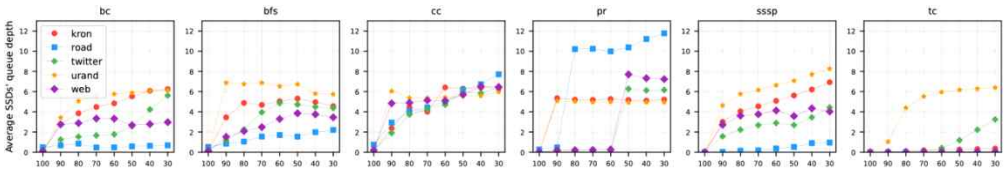


그림 366. 각 그래프 알고리즘-입력 별 paging/swapping에 의한 SSD 요청의 양

- 각 그래프 알고리즘 및 입력 그래프 실행 시 paging에 의한 입출력 요청 큐 깊이를 측정하였음(그림 366)
  · Paging은 SSD 의 무작위 접근 대역폭을 포화 시키기에 충분한 IO 요청을 생성하지 못함

• 병목 현상 해소를 위한 제안
- 이전 섹션의 결과는 다음과 같이 요약됨:
  · 1. 시스템이 너무 많은 시간을 유휴 상태에서 소비하고 있음
  · 2. Swap 장치 대역폭이 충분히 활용되지 않음 (swap 장치 대역폭을 높이려면 더 많은 요청이 필요)
- 시스템에서 CPU 활용도를 향상시키고 swap 장치에 더 많은 입출력 요청을 생성하기 위해, 본 연구는 thread overcommitting을 제안함.
  · 물리적 CPU 코어 당 더 많은user thread를 실행.
  · 프로그램 실행 시 OpenMP 환경 변수를 사용하여 제어할 수 있음. (커널/프로그램 수정이 필요하지 않음)
- 그림 367에서 확인할 수 있듯이 overcommit factor (OCF)를 1(기준)에서 16로 증가시키면 기준 대비 성능이 평균 1.65배 향상됨.
  · 최대 4.18배 향상 (bc:web)

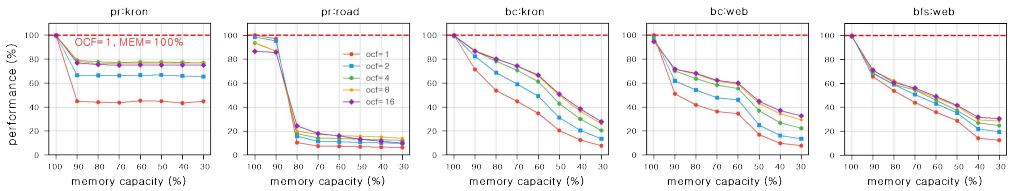


그림 367. BFS-web, BC-web의 그래프 전체 메모리 대비 DRAM에 할당된 양 (x축)에 따른 성능 변화

- OCF를 증가시킴에 따라 swap 장치에서 대기 중인 요청 수가 증가하는 것이 관찰됨. (그림 368)
  · 더 많은 page fault가 생성되어 swap 장치 대역폭을 더 잘 활용함.

수행기관	개발 목표	개발내용 및 범위

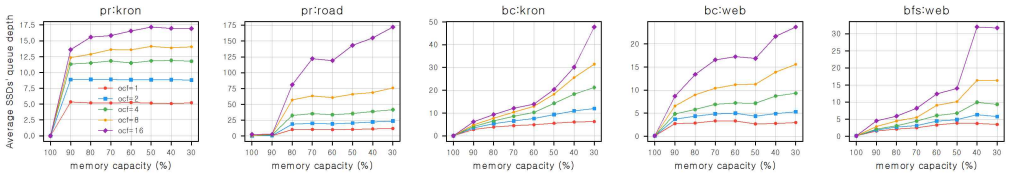


그림 368. OCF 증가에 따른 SSD 요청 개수의 증가

- 동시에 처리되는 page-in 수를 증가시킴으로써 유휴 시간이 감소함. (그림 369)
  · 그래프의 빨간 막대는 유휴 시간을 나타냄. 전체 데이터의 30%를 DRAM에 할당 할 수 있을 때, OCF를 증가시키면(막대 그룹 내에서 오른쪽) 유휴 시간이 크게 감소함.

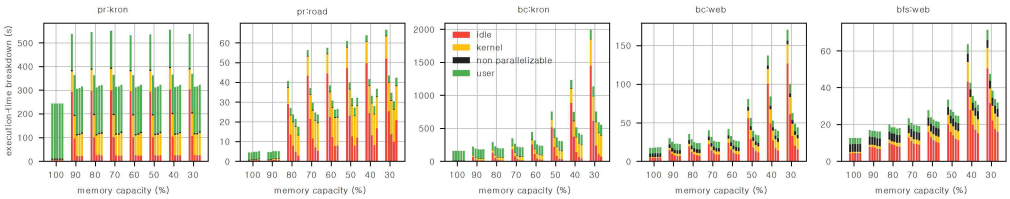


그림 369. 각 막대는 좌우로 OCF=1에서 OCF=16까지 실행 시간을 세분화함.
OCF=16일 때 (pr:road 는 OCF=8)유휴 시간을 최소화 할 수 있음.

• 추가 분석 결과
- 세 가지 추가 연구를 진행함:
  · 1. 시스템 성능에 더 많은 swap 장치의 영향
  · 2. 시스템 성능에 더 큰 overcommitting의 영향
  · 3. 더 많은 swap 장치에서 readahead 구성의 영향
• 추가 분석 결과는 PCIe 4.0을 지원하는 다른 시스템에서 수행되었으며, 코어의 수가 다르므로 이전 섹션의 결과와 차이가 날 수 있음.

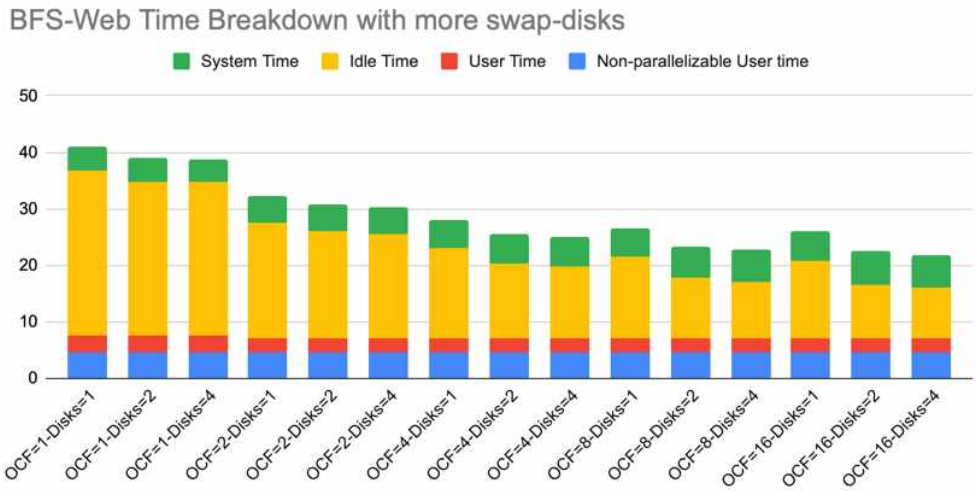
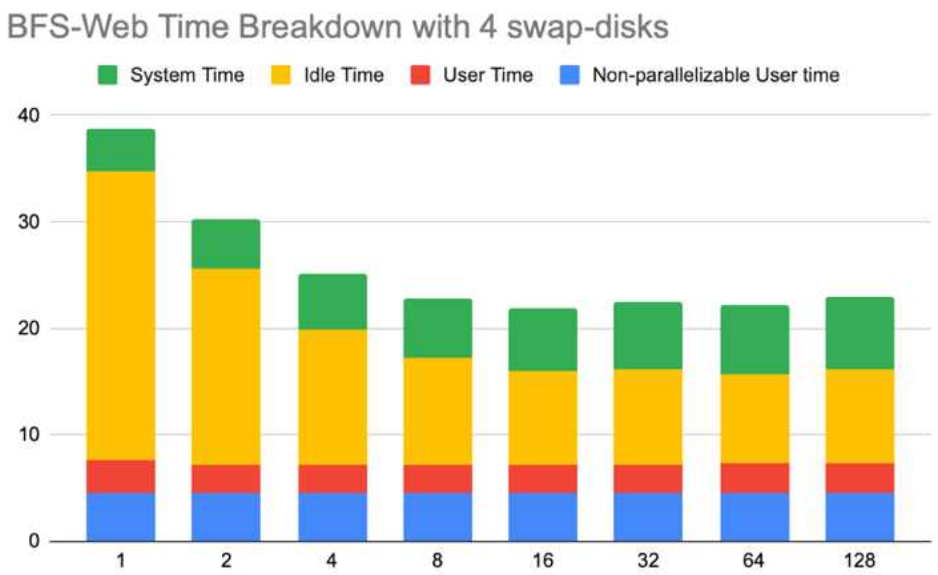


그림 370. OCF 별 swap 디스크의 개수 변화에 따른 실행 시간 변화

• 더 많은 swap 장치를 사용하면 시스템 성능을 향상시킬 수 있는지 확인하고자 함. (그림 370)
- 더 많은 swap 장치의 효과는 작았음.
  · Paging에 의해 생성되는 요청의 개수가 비교적 적고, 이러한 적은 요청의 개수는 단일 SSD나 여러 SSD나 비슷한 성능을 보여줌.
  · 단일 SSD에서 16개 요청까지는 perfect scaling에 가까운 성능을 보여줌. 16개를 초과하는 요청이 존재 할 경우 여러 SSD와 성능 차가 발생하기 시작하였음.

수행기관	개발 목표	개발내용 및 범위
		· SSD 1대의 대역폭 대비 SSD 4대의 대역폭 차이가 다음과 같이 관측 됨. (1) 16개 요청: 10% (2) 32개 요청: 24% (3) 64개 요청: 51% (4) 128개 요청: 104% · Paging의 요청 개수가 비교적 적어서 swap device의 개수에 따른 성능 변화가 크지 않았음. - 더 높은 OCF로 갈수록 실행시간이 감축되는 추세가 관찰됨. • 더 많은 swap 장치와 함께 더 큰 OCF에 대한 추가 연구를 수행함. (그림 371) - 더 높은 OCF(최대 128)를 탐색하였고, OCF=16 이후에는 유휴 시간이 감소하지 않음. - 오히려 시스템 시간이 증가 하는 것이 관찰됨. 이는 커널에서 공유 데이터에 대한 contention이 추가로 발생하는 것으로 예상됨.  그림 371. 4개 swap 디스크를 활용할 때 더 큰 OCF에 따른 실행 시간 변화. OCF=16 초과에서 성능 개선은 없으며, system time (커널에서 소비되는 시간)이 증가하는 것이 관측 됨. • 더 높은 OCF 및 더 많은 swap 장치에서 readahead의 효과를 조사 하였음. (그림 372) - 더 높은 OCF에서 page fault 수가 증가. - 본 연구에서 readahead (다음 8개 페이지를 미리 읽음)가 메모리에서 더 많은 경합을 일으켜 성능이 저하될 것이라 가설을 세움. - 하지만 실험 결과 더 높은 OCF에서 page fault의 수가 약간 증가하더라도 readahead의 혜택이 훨씬 더 크다는 것을 확인 함.

수행기관	개발 목표	개발내용 및 범위

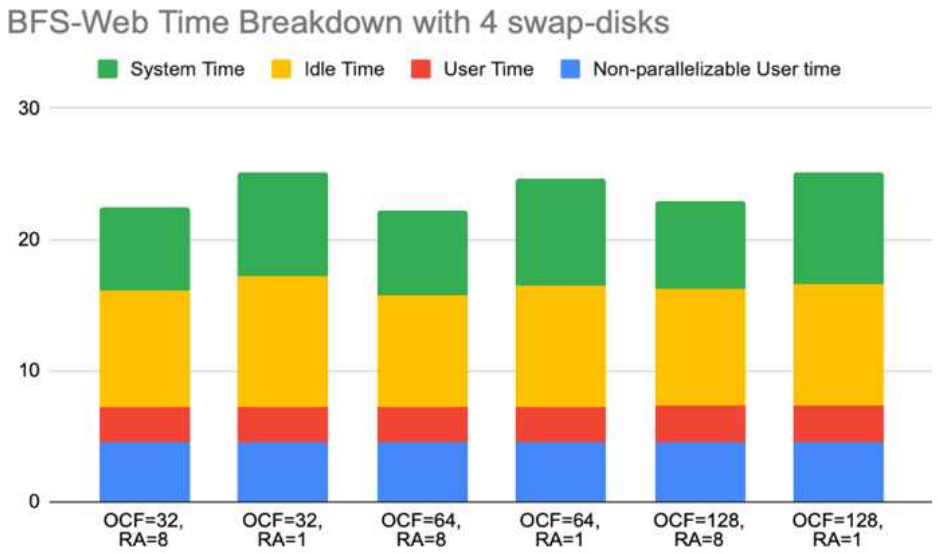


그림 372. Readahead (RA)에 유/무에 따른 실행 시간 변화. Readahead에 의해 8개 페이지를 (RA=8) 들여 오는 것이 readahead가 없을 때 (RA=1) 보다 짧은 실행 시간을 보여줌.

- OCF=128, RA8 및 RA1 비교 시 page fault가 2.8배 적게 발생함

• 결론: 현재 paging을 하는 시스템에서 가장 큰 병목은 커널과 swap device에서 발생함
- 유휴 및 커널 시간은 응용 프로그램이 실질적인 연산을 하지 못하고 있는 시간임.
- 그림 373에 표시된 것처럼 전체 실행 시간 중 시스템(커널) 부분은 여전히 매우 높음
- 커널 page fault 처리 경로를 최적화하거나 성능을 더 향상시키기 위해 특수 처리 logic이 필요한 시점임.

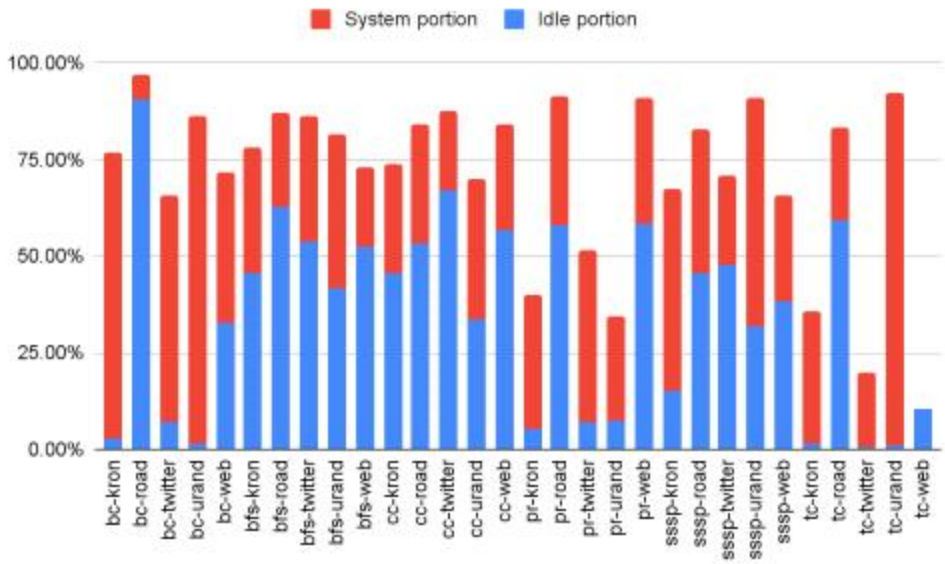


그림 373. 전체 실행 시간 중 커널 (system) 또는 유휴 상태 (idle)에서 보내는 시간의 비중. OCF=16, 총 메모리 대비 30%만 DRAM에 할당, swap disk는 1개를 사용했을 때의 결과.

수행기관	개발 목표	개발내용 및 범위

ETRI (주관)

- 대규모 메모리 중심 컴퓨팅 핵심 기술 연구
- 메모리 중심 컴퓨팅 기반 소프트웨어 통합 설계 기술 연구

고광원

아주대학교(공동1)

- 메모리 중심 컴퓨팅 시스템을 위한 기반 소프트웨어 기술 연구

성균관대학교(공동2)

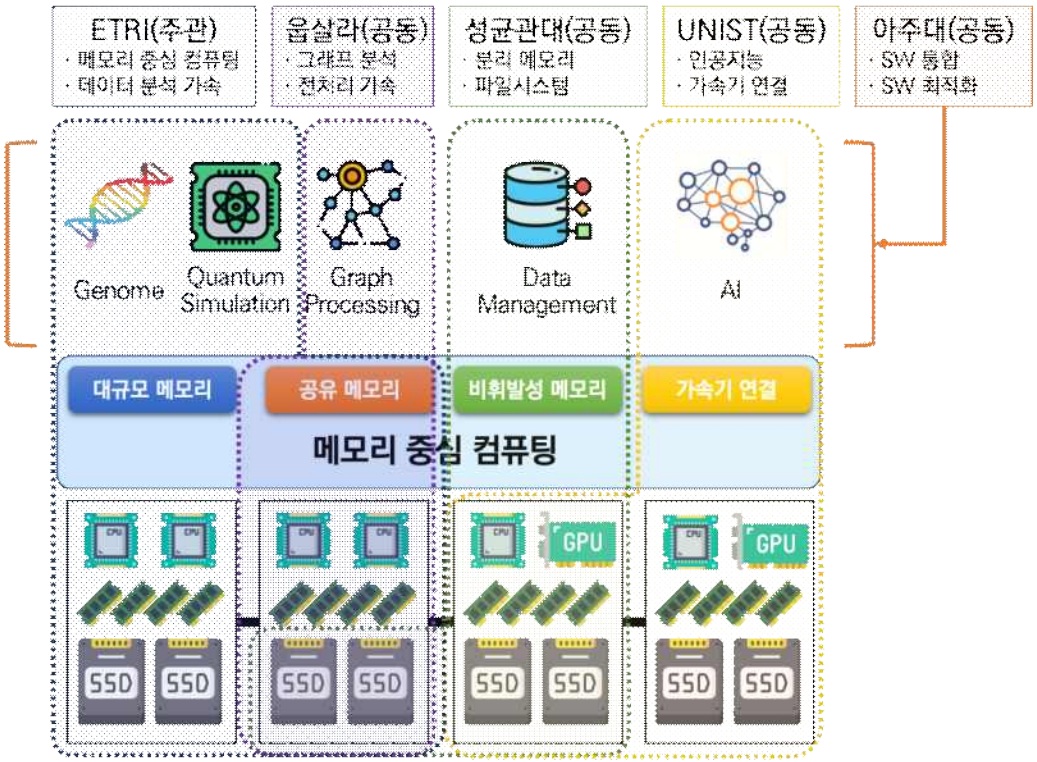
- 비휘발성 메모리 기반 스케일러블 인메모리 저장소 연구

UNIST(공동3)

- 메모리 중심 컴퓨팅 기반 인공지능 플랫폼 연구

웁살라대학교(공동4)

- 대규모 그래프 분석 가속 기술 연구



[EdgeCPS]

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td>ETRI
(주관)</td><td>③저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 프로토타입 개발</td><td>

• 저자원 디바이스의 자원 증강 프레임워크 개발

- 기술 개요
  · 저자원 디바이스 단말의 연산 성능 한계극복을 위해 실제 디바이스에 존재하지 않는 자원(GPU, FPGA, Storage등)을 증강 프레임워크를 통해 가상화된 자원을 제공함으로써 성능 한계를 극복하는 디바이스리스 컴퓨팅을 제공하는 연구

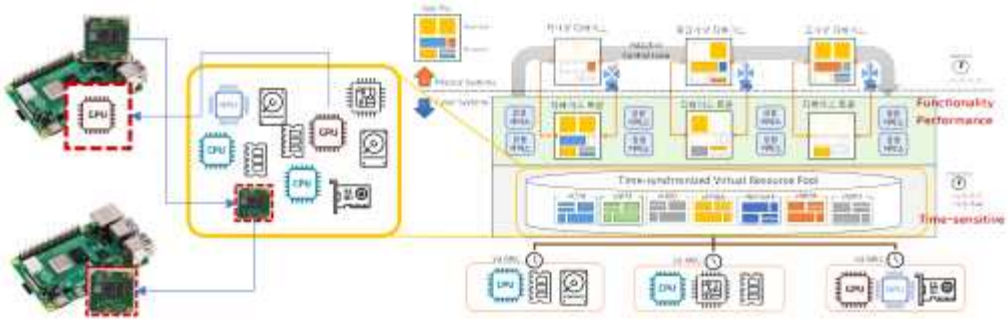


그림 374. 저자원 디바이스 자원 증강

- 당해연도 연구 목표 및 범위
  (1) 디바이스 자원 증강 프레임워크 및 오프로딩 서비스 관리 기술 연구
  (2) 디바이스리스 컴퓨팅을 지원하기 위한 디바이스-서버간 공통 라이브러리 기술 연구
  (3) 소프트웨어 정의 자원(Software Defined Resource) 관리 기술 연구

- 세부 연구 개발 내용
  (1) 디바이스 자원 증강 프레임워크 및 오프로딩 서비스 관리 기술 연구
  . 저자원 디바이스 자원 증강을 엣지 서버를 통해 가상화된 고자원 디바이스를 생성하여 응용에 대한 오프로딩을 수행하는 방식에서, 가상화된 디바이스 자원을 실디바이스에서 직접 활용하는 형태로 재정의함
  . 자원 오프로딩을 위해서는 디바이스리스 컴퓨팅 프레임워크를 사용할 수 있는 SW 구조 변화가 요청되며, 증강을 지원하는 개별 자원은 서비스 형태로 증강 기능을 제공하도록 설계함
  . 증강을 지원 받는 디바이스는 프레임워크 사용을 위한 공통 라이브러리를 기반으로 생성된 응용 SW 형태로 개발되어야 하며, Python 기반의 응용을 기반으로 설계되었으며, 실-서버 모두 동일한 응용을 사용하는 환경을 가정하고 있음

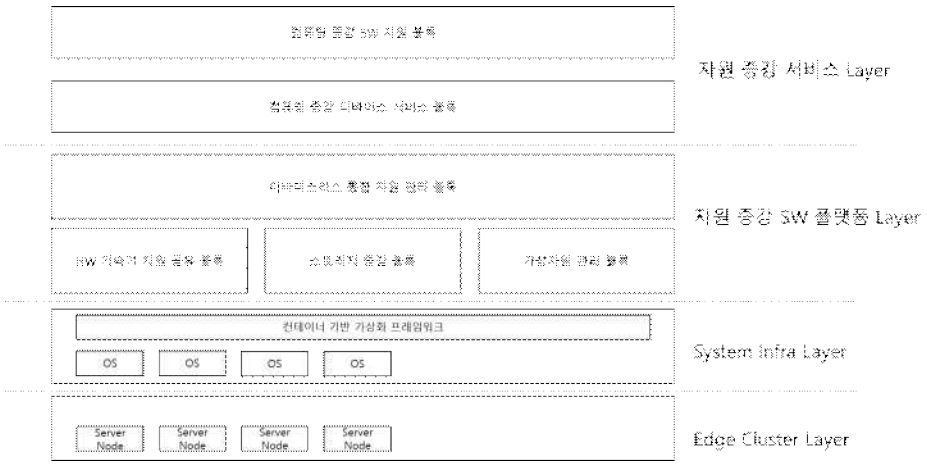
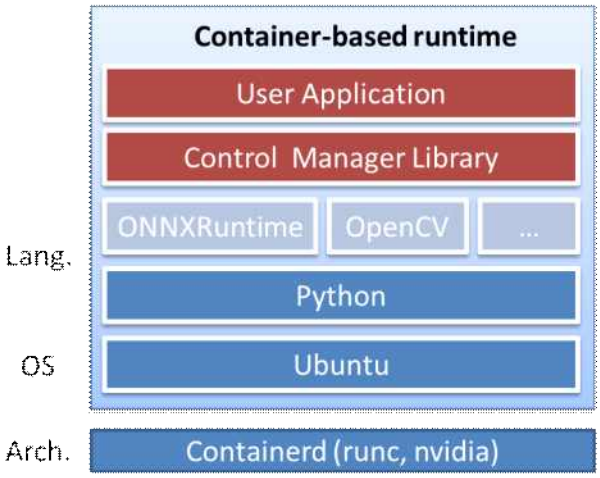
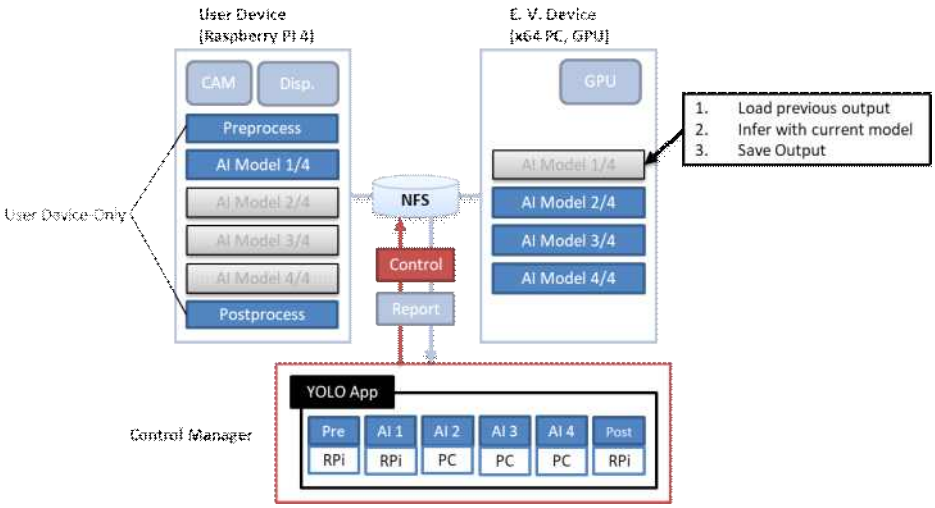
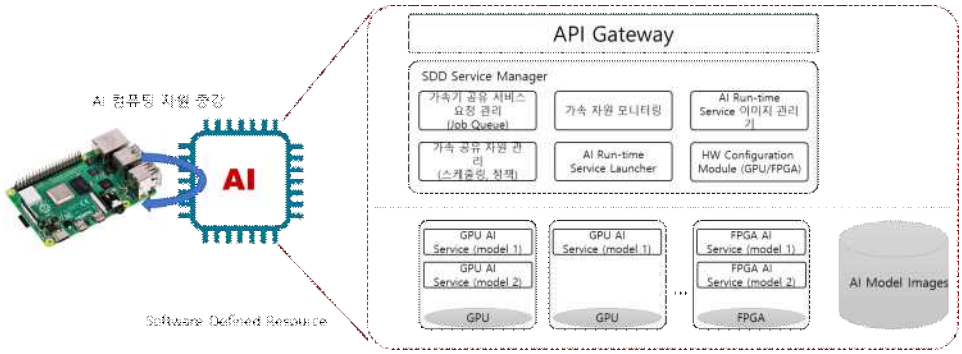


그림 375. 디바이스리스 컴퓨팅 프레임워크 시스템 구조도

</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		(2) 디바이스리스 컴퓨팅을 지원하기 위한 디바이스-서버간 공통 라이브러리 기술 연구 . 서비스를 수행하는 사용자 기기인 실 디바이스에서 요청하는 증강 자원은 결국 응용 서비스 내의 특정 기능을 실행하는 경우 요청되게 되며, 이는 증강이 가능한 형태의 공통 API를 통해 지원 가능함 . 공통 API를 통해 개발된 응용 SW는 서버에서 동일한 코드로 동작하며, 디바이스/서버의 실행 환경에 따라 증강 여부를 결정할 수 있게 구성되어야 함 . 디바이스리스 컴퓨팅 프레임워크에서는 Python을 기반으로하는 공통 API를 사용하고 있으며, 스토리지 자원은 디바이스/응용 기준의 증강을 제공하고, 컴퓨팅 증강은 AI 응용 서비스를 개발 환경을 가정하고 있음  그림 376. 자원 증강 지원 디바이스리스 컴퓨팅 응용SW 구조  그림 377. 응용 관리자를 통한 기능 단위의 증강 지원 . 디바이스리스 컴퓨팅을 지원하는 응용 SW는 공통 라이브러리를 통해 SW에 정의된 기능을 수행하며, 각 기능은 컨트롤 매니저를 통해 사용자 요구사항에 따른 성능 검증을 진행하며, 기능 모듈의 실패(자원 부족, 예외상황 발생)을 감지하여, 프레임워크를 통한 컴퓨팅 증강을 수행하게 됨 . 증강 요청이 발생한 경우 서버측 모듈은 동일한 SW의 실행 시작점을 설정하고,

수행기관	개발 목표	개발내용 및 범위
		실행된 시작점의 기능을 자원 증강을 통해 수행하게 되며, 응용 모듈 실행시 요구되는 데이터는 스토리지 증강 모듈을 통해 공유되도록 설계됨 (3) 소프트웨어 정의 자원(Software Defined Resource) 관리 기술 연구 . 소프트웨어 정의 자원은 디바이스에게 저자원을 극복하기 위해 제공되는 서버 자원의 특수한 형태로, 주로 가속기 자원을 의미함 . AI 서비스에서 요구되는 대표적인 고성능 가속기 자원은 GPU, FPGA, NPU 등의 자원이 있으며, HW의 특성에 따라 전체 혹은 파티셔닝된 공간에 따라 HW를 사용할 수 있게 됨 . 응용 서비스 측면의 HW 가속기는 가속 기능을 제공하는 디바이스 드라이버나 전용 라이브러리를 통해 사용되고 있기에 SW 측면에서의 가속기 자원은 특정 기능을 정해진 요구사항 내에 수행이 된다면 추상화된 HW를 사용하는 것과 동일하게 동작됨 . 디바이스리스 프레임워크에서는 이러한 형태의 가속 자원의 활용을 소프트웨어 정의 자원이라고 정의하고, 그림 378와 같이 AI 컴퓨팅 자원을 제공하는 형태로 활용되고 있음 . 소프트웨어 정의 자원을 사용하기 위해서는 디바이스 및 전용 라이브러리 형태로 컴퓨팅 기능을 제공해야 하며, 당해 년도에는 GPU 및 FPGA를 통한 소프트웨어 정의 자원을 제공하고 있음  그림 378. 소프트웨어 정의 자원 (AI 컴퓨팅 자원) - 연구 결과 시연 . 시연 구성은 아래의 그림은 디바이스리스 컴퓨팅 프레임워크를 통해 등록된 디바이스에서 증강 요청에 따른 서버 자원의 동작 과정에 대한 시연 결과를 나타내고 있음 . 시연에서는 위탁 연구를 통해 개발한 스토리지 증강 프레임워크를 중심으로 디바이스가 등록되며, 등록된 디바이스와 연결되 스토리지 자원을 통해 데이터가 공유되고 있음 . 공유된 데이터는 실-디바이스 환경에서 수행과 서버 자원을 통해 수행되는 시나리오로 시연을 진행 . 실 디바이스(라즈베리파이4) 6대에서 디바이스리스 컴퓨팅 프레임워크에게 6개의 스토리지 증강을 요청하고, 증강된 스토리지를 통해 공유된 데이터를 기반으로 컴퓨팅 증강을 진행 . 시연에서 각 디바이스에서는 연산 증강 요청이 없는 사전 계산 결과를 개별 디바이스의 로컬 영역에 저장하게 되며, 저장된 결과는 스토리지 증강에 의해 서버 환경과 공유되어 관리 됨 . 연산 증강이 요청되는 구간이 발생할 경우 디바이스의 응용 SW는 서버에 연산 증강을 요청하게 되며, 요청된 증강 연산은 서버에서 증강된 자원을 통해 수행된후

수행기관	개발 목표	개발내용 및 범위
		계산결과를 다시 공유된 스토리지 영역게 기록하고, 이를 디바이스에서 최종 확인하는 시나리오로 진행하였음

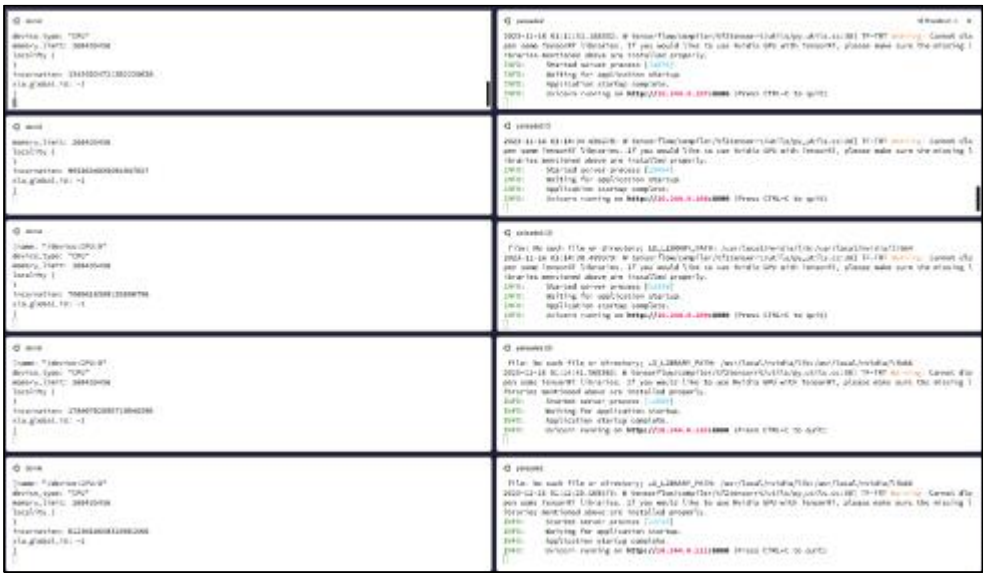


그림 379. 애플리케이션 마이그레이션 기능 데모 시연

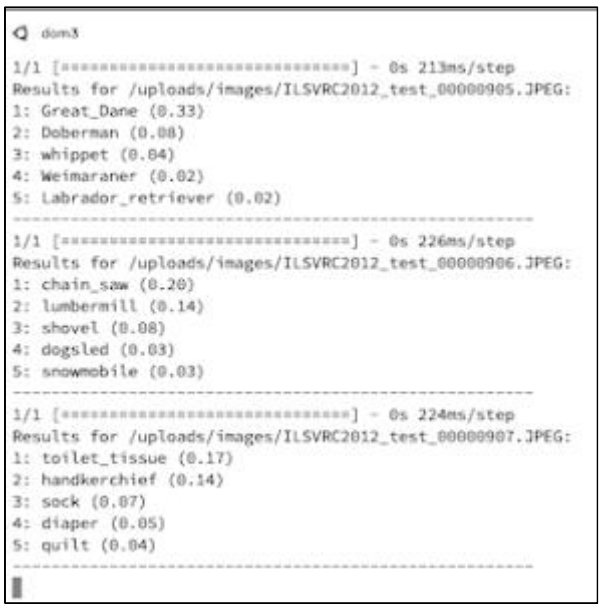


그림 380. 실디바이스(라즈베리파이)의 로컬 작업

수행기관	개발 목표	개발내용 및 범위

```
-------------------------------------------------------
1/1 [==============================] - 0s 17ms/step
/uploads/output/ILSVRC2012_test_00000194.txt
Results for /uploads/images/ILSVRC2012_test_00000194.JPEG:
1: king_crab (0.25)
2: spiny_lobster (0.12)
3: barn_spider (0.07)
4: crayfish (0.02)
5: wolf_spider (0.02)
-------------------------------------------------------
1/1 [==============================] - 0s 17ms/step
/uploads/output/ILSVRC2012_test_00000195.txt
Results for /uploads/images/ILSVRC2012_test_00000195.JPEG:
1: passenger_car (0.59)
2: electric_locomotive (0.11)
3: streetcar (0.02)
4: dock (0.01)
5: amphibian (0.01)
-------------------------------------------------------
1/1 [==============================] - 0s 19ms/step
/uploads/output/ILSVRC2012_test_00000196.txt
```

그림 381. 연산 증강 요청에 따른 서버 작업

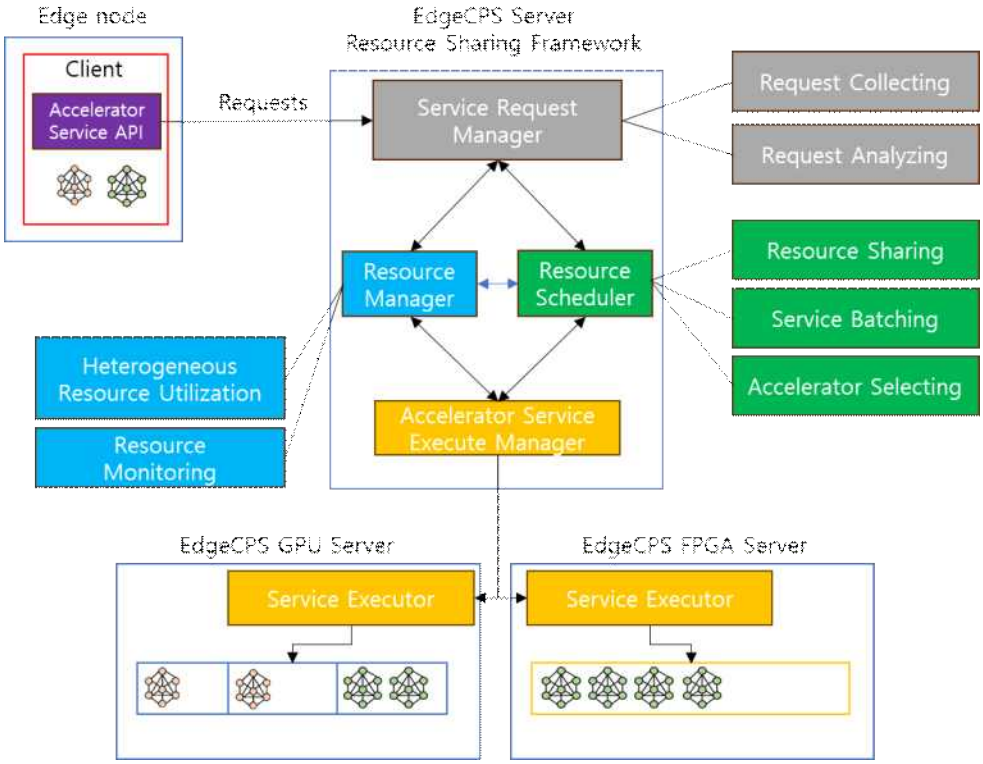
• FPGA 가속기 자원 프로비저닝 기술 연구

- 기술 개요
  . EdgeCPS 플랫폼에서는 저자원 디바이스의 성능 한계를 극복하도록 지원하기 위하여 디바이스 트윈을 정의하고 엣지 서버에서 사용하는 다양한 HW 자원들의 가상화 및 공유모델을 통하여 저자원 디바이스의 연산능력을 높일 수 있지만, 단일 엣지 서버의 가속기 자원은 한정되어 있음.
  . 이를 해결하기 위해 본 연구에서는 대표적인 HW 가속 장치인 GPU 자원 활용의 제한적인 문제를 해결하여 다중 노드 기반의 GPU 자원을 공유하기 위한 프레임워크를 연구를 진행하였음.
  . 3차년도에서는 FPGA 가속기 자원 프로비저닝 기술 연구를 통해 가속기 자원 공유 프레임워크에 GPU 뿐 아니라 FPGA 등 다양한 HW 자원들의 가상화를 통해 저자원 디바이스의 성능 한계를 극복하도록 지원하는 것이 목표
  . 각 디바이스에서 최적으로 수행할 수 있는 AI 응용 서비스 제공을 위하여 컴퓨팅 분할 시스템과 AI 응용에 필요한 지식을 공유할 수 있는 시스템 구축을 목표

- 당해연도 연구 목표 및 범위
  (1) FPGA 가속기 프로비저닝 기술 개발
  (2) 다중 노드 기반의 이기종(GPU/FPGA) 가속기 공유 프레임워크 개발
  (3) 다중 노드 기반의 이기종(GPU/FPGA) 가속기의 효율적인 공유 기술 연구
  (4) 가속기 활용을 위한 컴퓨팅 분할 기반 EdgeCPS-AI 지식 공유 모델 연구

- 세부 연구 개발 내용
  (1) FPGA 가속기 프로비저닝 기술 개발
  . 하나의 FPGA 가속기에 여러 마이크로 서비스가 공유할 수 있도록 지원 FPGA 가속기 자원 공유를 위한 시스템 SW 기술 개발 수행
  . 컨테이너 환경에서 FPGA를 활용하기 위한 가속 인터페이스를 개발하여 가속기 자원 공유 프레임워크에 이기종(GPU/FPGA) 가속기 자원 공유를 위한 기반 기술 확보

  (2) 다중 노드 기반의 이기종(GPU/FPGA) 가속기 자원 공유 프레임워크 개발
  . 마이크로 서비스가 수행 될 서버의 이기종(GPU/FPGA 가속기 자원을 여러 응용이 공유할 수 있는 자원 공유 프레임워크 개발
  . 다중 노드 기반의 이기종 가속기 자원 공유 프레임워크 동작 시나리오
    - 마이크로 서비스가 요청하는 가속 연산 요구사항 수집
    - 현재 클러스터에서 가용한 이기종 가속기 자원 수집

수행기관	개발 목표	개발내용 및 범위
		- 요구사항 분석을 통한 이기종 가속기 자원 공유 모델 생성 - 자원 스케줄러에 따른 이기종 가속기 자원 할당 및 가속 연산 수행 . Service Request Manager - 마이크로 서비스의 가속 연산 요구사항 수집 - 마이크로 서비스의 가속 연산 요구사항 분석 . Resource Manager - 자원 활용률을 높이기 위한 이기종 가속기 자원의 파티션 수행 및 관리 - 클러스터 내의 이기종 가속기 자원 모니터링 . Resource Scheduler - 클러스터 내의 이기종 가속기 HW 자원 공유 스케줄링 - 자원 활용률을 높이기 위한 동일 서비스의 일괄처리 스케줄링 - 가속 연산 수행할 이기종 가속기 HW 선택 스케줄링 . Accelerator Service Execute Manager - 클러스터 내의 이기종 가속기 동작 수행 및 관리  그림 382. 이기종 가속기 공유 프레임워크 (3) 다중 노드 기반의 이기종(GPU/FPGA) 가속기의 효율적인 공유 기술 연구 . 이기종 가속기 자원 사용률을 높이기 위한 자원 스케줄링 기술 연구 . 이기종 가속기 자원 특성을 고려한 서비스 배치 기술 연구 (4) 가속기 활용을 위한 컴퓨팅 분할 기반 EdgeCPS-AI 지식 공유 모델 연구 . EdgeCPS 플랫폼에 적용 가능한 AI 지식 공유 모델 설계 및 구현 - 10개의 노드 (Device Node, TrainingData Node, AINetwork Node, WeightedModel Node, PartitionWeightGroup Node, PartitionedModel Node, PartitionInferenceGroup Node, ServiceModel:Trainer Node, ServiceModel: Inferencer Node, ServiceModel: Partition_Inferencer Node) - 5개의 관계 (CreatedWith, UsedIn, ExecutedOn, SeparatedFrom, ConsistOf)

수행기관	개발 목표	개발내용 및 범위

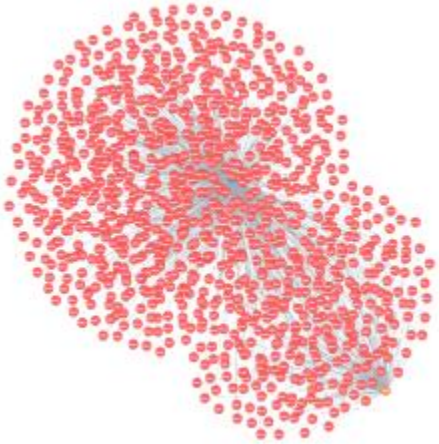


그림 383. 2,762개의 노드 중 일부 그래프만을 표현

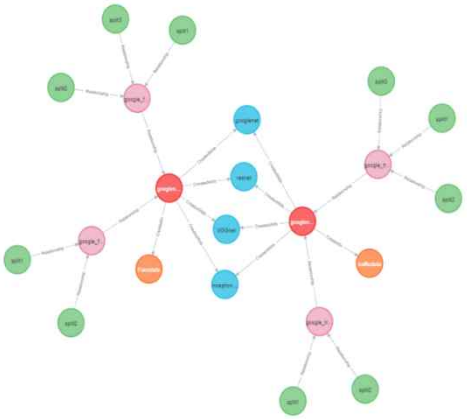


그림 384. 지식그래프에 쿼리를 통한 결과 표현

- 총 노드 수: 2,762개
- 총 관계 수: 5,891개
- Device: 9 (Intel Desktop-PC:1, JetsonNano:1, OrangePi4: 2, KhadasVIM3: 2, RaspberryPi4: 3
- TrainingData: 3 (FaceData, CarData, TrafficSignData)
- AINetwork Model: 5 (VGGNet, ResNet, GoogleNet, InceptionNet, ResNetX)
- TrainedWeightModel: 2,487 (Scheduler=Adagrad, Adam, …, Epoch=25, 50, …, BatchSize=16, 32, …, LearningRate=1e-3, le-5, …)
- PartitionedWeightModel: 10
- Service Model: 207

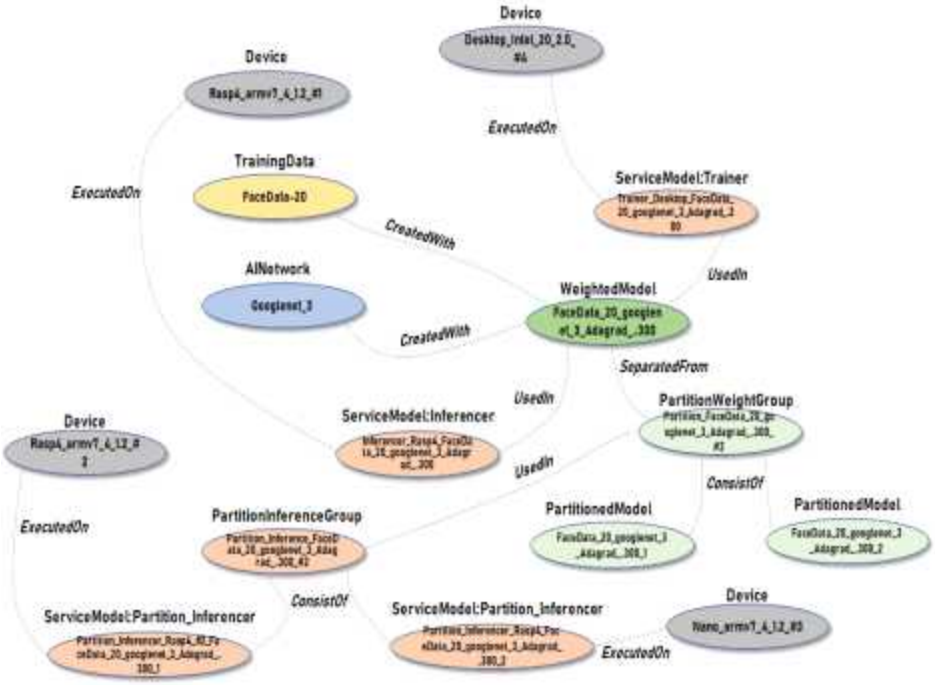
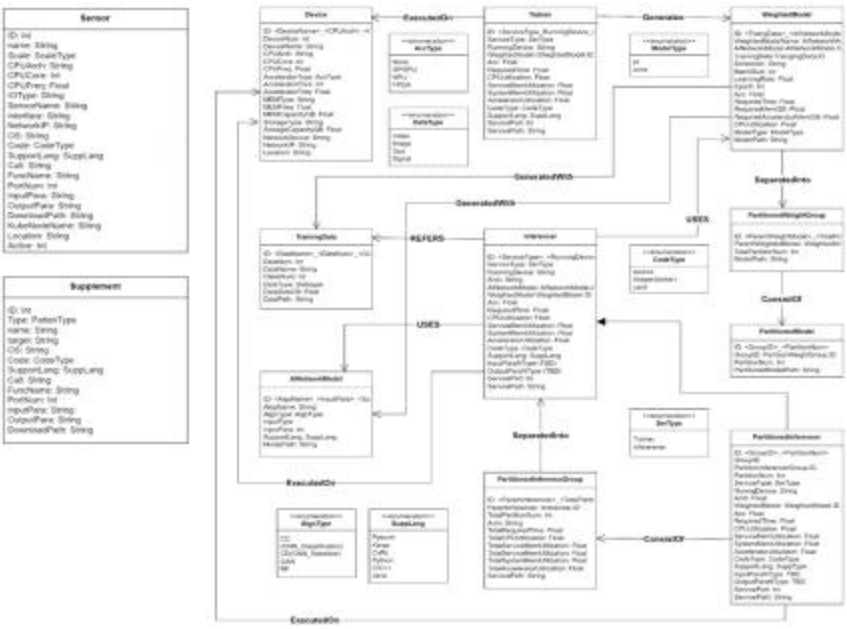
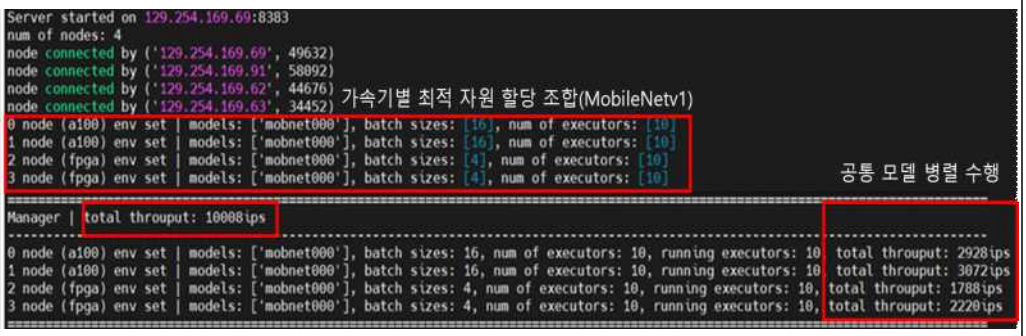
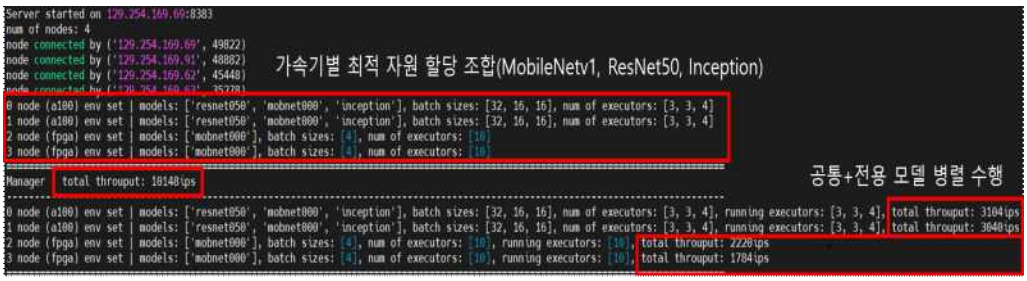
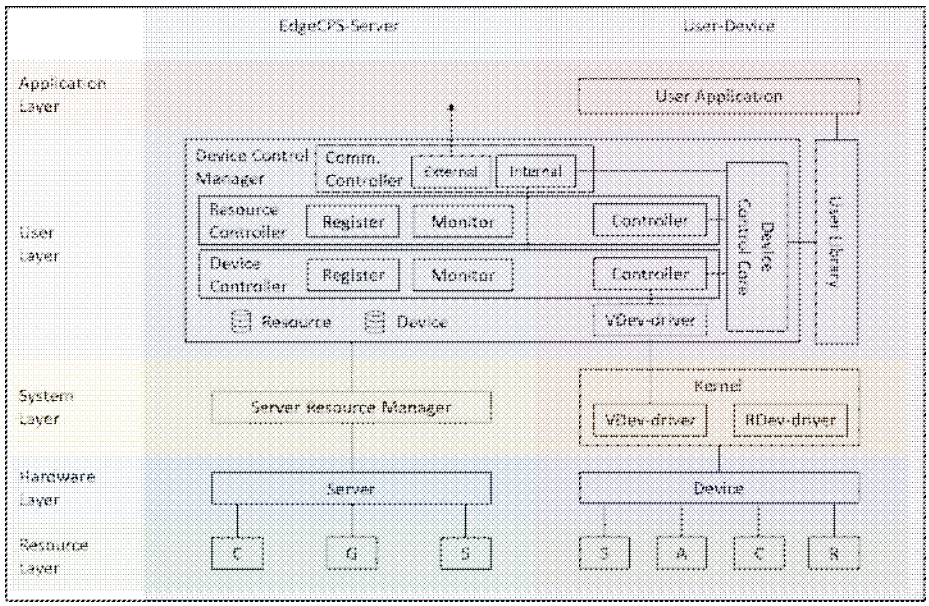


그림 385. 10개의 노드와 5개의 관계로 구성된 지식 그래프

수행기관	개발 목표	개발내용 및 범위
		 그림 386. 지식 그래프의 구조를 메타모델 형태로 표현 . 컴퓨팅 분할 지원을 위한 학습 모델 균등 분할기 구현 - 컴퓨팅 분할을 위해서 학습된 모델 분할: pytorch로 학습된 모델과 ONNX로 학습된 모델 - 분할 개수: 2개 ~ 4개 (분할갯수는 사용자가 원하는 만큼 더 분할할 수 있도록 프로그램이 설계되어 있음) - 분할 지점 파악: 각 레이어 파라미터를 파악하여 최대한 균등하게 분할할 수 있는 알고리즘이 포함 (캡쳐된 프로그램에서 19개의 레이어에서 8, 11, 13 지점을 선정하여 4개로 분할됨) - 컴퓨팅 분할: 2개, 3개, 그리고 4개로 분할하여 분할된 학습된 모델을 이용하여 추론기를 생성하고, 생성된 추론기를 Edge device에 분산 배치하여 실행하면서 시스템 성능 측정 (CPU Utilization, Power(W), SendPack, RecvPack, PageFault, ExecutionTime(sec), Program Mem Usage(MB), System Total Mem (GB), System Used Mem (GB), System Available Mem (GB), Memroy Rate(%)) . 그래프 기반의 지식 공유 시스템 구현 - 지식 공유 시스템: 지식 공유 시스템은 RESTAPI 형태로 된 입력인터페이스와 출력인터페이스, 그래프 기반의 지식 공유를 관리하는 가중치 모델 분할기, EdgeCPS 응용 정보 등록기, 쿼리분석기, EdgeCPS 응용 정보 처리기, EdgeCPS 응용 구성 추천기로 구성되어 있으며, 이 시스템을 활용하여 EdgeCPS 플랫폼 상에서 동작하는 AI 응용 프로그램 작성에 필요한 자료들을 제공함 (여기서 자료는 학습용 데이터, 학습된 모델 (가중치), 추론기 (분할된 추론기 포함) 등을 의미함) - Gaussian Normal Distribution 기반의 Top-K 추천기: 지식 공유 시스템 내에 있는 추천기는 학습된 모델을 위한 추천과 추론기를 위한 추천이 있으며, 학습된 모델과 추론기 추천을 위해서 성능 매트릭스를 정의함 - 연구 결과 시연 **[정량 목표 - 가속기 공유 모델 2종]** . 이기종(GPU/FPGA) 가속기 자원 특성(전력사용량 대비 처리량) 비교 - 이기종 가속기 지원하는 공통 추론 모델(MobileNetv1)에 대한 전력 사용량 대비 처리량 비교 - FPGA: 최대 성능 수행 시 전력 소비량 45Watt 이고, 전력 대비 처리 처리량은 60 Images/Watt

수행기관	개발 목표	개발내용 및 범위
		- GPU(A100): 최대 성능 수행 시 전력 소비량 240Watt 이고, 전력 대비 처리 처리량은 22 Images/Watt - MobileNetv1 추론 응용을 사용할 때 FPGA가 GPU의 2.7배의 에너지 효율이 좋음을 보임 - FPGA 가속 연산을 사용함으로써 클러스터 서버 전체의 에너지 소비량을 줄이고 비용을 절감할 수 있을 것으로 예측 그림 387. 이기종 가속기 자원 특성(전력사용량 대비 처리량) 비교(MobileNetv1 추론 모델) . 이기종(GPU/FPGA) 가속기의 최적 자원 할당 연구 - 가속기 별 공간정보와 동일모델 추론 성능에 따른 디바이스 할당 조합 생성 - NVIDIA A100: 40GB - Xilinx U250 FPGA: 64GB - MobileNetv1 모델(GPU/FPGA 공통지원)의 이미지 추론 성능: 초당 이미지 추론수(image per sec) 측정 - Batch: 하나의 추론 응용이 한번에 추론을 수행 할 이미지 개수(같은 추론 응용을 여러개 수행하는 것보다 공간이 절약되면서 성능은 비슷) - MPS(Multi-Process Service): 단일 가속기(GPU/FPGA)에서 다수의 추론 응용이 동작할 수 있는 가속기 공유 기술 - A100 GPU: {batch:16, MPS:10}, FPGA: {batch:4, MPS:10} 일 때 최적 자원 할당 조합 확인 그림 388. 이기종(GPU/FPGA) 가속기의 최적 자원 할당 조합(MobileNetv1 추론 모델) . 이기종 가속기 사용 단일 추론 연산 동작 수행 확인(GPU/FPGA 공통 모델 지원) - GPU/FPGA 공통 모델: MobileNetv1

수행기관	개발 목표	개발내용 및 범위
		- 가속기 별 한정된 공간에 따른 최대 서비스 수행 개수 결정 - 자원 활용률 높이기 위한 가속기 당 일괄처리 서비스 개수 도출 - 가속기별 최적 자원 할당 조합에 따른 일괄처리 서비스 배치 및 수행  그림 389. 이기종 가속기 사용 단일 추론 동작 수행 확인(GPU/FPGA 공통 모델 지원) . 이기종 가속기 사용 다중 추론 연산 동작 수행 확인(GPU/FPGA 공통 모델 지원) - GPU/FPGA 공통 모델: MobileNetv1 - GPU 지원 추론 모델: ResNet50, Inception - GPU 가속기의 경우 3가지의 모델의 독립 수행을 위하여 HW 파티셔닝 기술인 MIG(Multi-Instance GPU) 사용하여 하나의 GPU 공간을 3개로 분할 - 가속기별 지원 추론 모델에 대한 최적 자원 할당 조합에 따른 일괄처리 서비스 배치 및 수행  그림 390. 이기종 가속기 사용 다중 추론 동작 수행 확인(GPU/FPGA 공통 지원 추론 모델 + GPU 지원 추론 모델) **• 이종 디바이스를 위한 가상 자원 동적 생성 기술 연구** - 기술 개요 . 저자원 디바이스 단말에 대한 한계극복을 위해 가상화된 공간에서 자원 공유의 탄력성을 제공하여 학습/추론 등의 고성능 AI 연산을 가능하게 하는 디바이스리스 컴퓨팅 실현을 위한 가상화 프레임워크 연구 수행 - 당해연도 연구 목표 및 범위 (1) 메타데이터 기반 자원 가상화 인프라 개발 . Device-To-Edge 실현을 위한 디바이스리스 컴퓨팅의 기본 개념을 구성하기 위해서는 EdgeCPS 플랫폼에 연결되는 다양한 디바이스들에 대한 연결, 자원 관리, 자원 가상화, 자원 동적 생성 기술 등이 필요함 . 컴퓨팅/비컴퓨팅, 시스템 동적/정적, 센서/엑츄에이터 등 산발적이고 이타적인 정보들을 통합 관리하기 위한 메타데이터 기반 자원 관리 기술을 개발하고 증강을 위한 기초 모델을 설계함 (2) 자원 증강 실현을 위한 대규모 실/가상 디바이스 장치관리기 개발

수행기관	개발 목표	개발내용 및 범위
		. 사용자 관점에서의 유기적인 EdgeCPS 응용 서비스 제공을 위한 SW 실행 기반을 제공하고, 저자원 디바이스 성능 및 자원을 증강하기 위한 가상 자원 인프라 및 자원 관리 기술을 연구함 . 실 디바이스와 연동되는 장치관리기를 개발하여 10,000개 이상의 대규모 디바이스 관리가 가능함을 보임으로써, 디바이스리스 프레임워크의 초기 프로토타입을 제시하고자 함 - 세부 연구 개발 내용 (1) 메타데이터 기반 실/가상 디바이스 동시 지원 자원 가상화 인프라 연구 . EdgeCPS 플랫폼에 연결되는 다양한 성능 및 형태의 대규모 디바이스를 동일하게 관리하기 위해서는 추상화된 메타데이터 형태로 구성해야 할 필요성이 있음 . 각 디바이스의 구성을 연결성을 띄는 디바이스와 디바이스가 포함하는 자원으로 구분하여 메타데이터를 설계하고 각 모듈이 동일한 형태로 관리되도록 클라우드와 연계를 진행함 . 메타데이터는 실 디바이스가 모사된 가상 디바이스와 1:1로 매핑되며, 증강을 위한 타 디바이스의 연동을 지원할 수 있도록 구성함 . 해당 정보들을 통합적으로 관리하는 장치관리기를 구성하고, 추후 서버 자원과의 연계를 위한 대단위 설계를 진행함  그림 391. 실/가상 디바이스 동시 지원 자원 가상화 인프라 구조 (2) 디바이스 자원 관리를 위한 장치관리기 연구 및 설계 . 자원 증강과 함께 디바이스리스 컴퓨팅을 실현하기 위해서는 EdgeCPS 플랫폼에 연결되는 장치 및 자원들에 대한 관리가 필수적임 . 자원의 통합 관리를 위해 컴퓨팅 성능을 갖는 디바이스와 디바이스에 연결되는 센서/액추에이터(Sensor/Actuator)로 분리된 구조의 연결 관리기 세부 설계를 진행함 . CPU/GPU/MEM 등의 같은 시스템 내부 자원들도 디바이스에 연결되는 물리적 자원과 동일한 형태로 관리할 수 있도록 구성함 . 디바이스별로 통신을 위한 매퍼(Mapper)를 제공함으로써, 디바이스리스의 자원 증강에 대응하고 외부 자원에 접근할 수 있도록 연결관리기 모듈을 설계함

수행기관	개발 목표	개발내용 및 범위

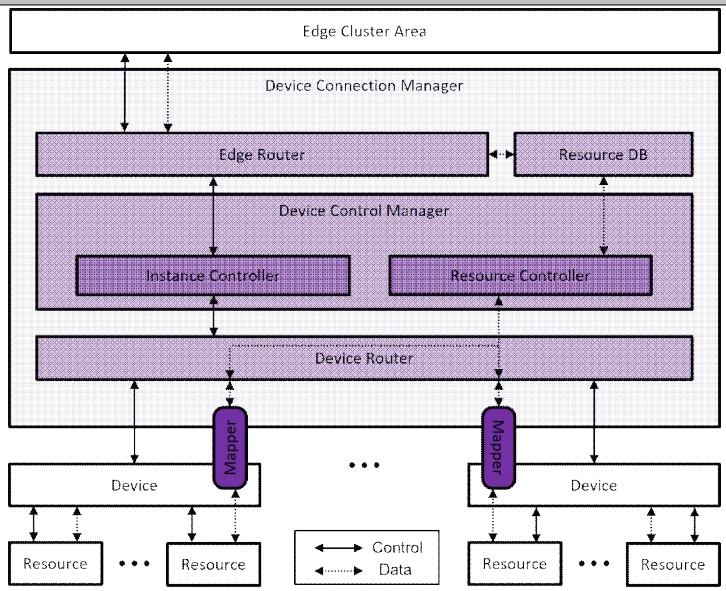


그림 392. 디바이스리스 장치관리기 세부 설계 및 연동 구조

- 연구 결과 시연 **[정량 목표 – 가상 자원 관리 성능 10000개]**
  . 메타데이터 기반 자원 가상화 인프라 개발
    - 실/가상 디바이스의 메타데이터를 기반으로 디바이스의 연결 및 데이터 관리, 동기화를 담당하는 가상화 인프라를 구축하고 모니터링 할 수 있는 기능을 제공함
    - RESTful POST/GET/PUT/DELETE(CRUD) API를 활용하여 메타데이터 및 자원에 손쉽게 접근할 수 있도록 프로토타입을 개발함
    - 디바이스에 연결된 센서/엑츄에이터 자원 정보 및 CPU/GPU/MEM등 의 시스템 메트릭 정보도 동일한 형태의 추상화된 메타데이터로 관리할 수 있도록 설계를 진행함

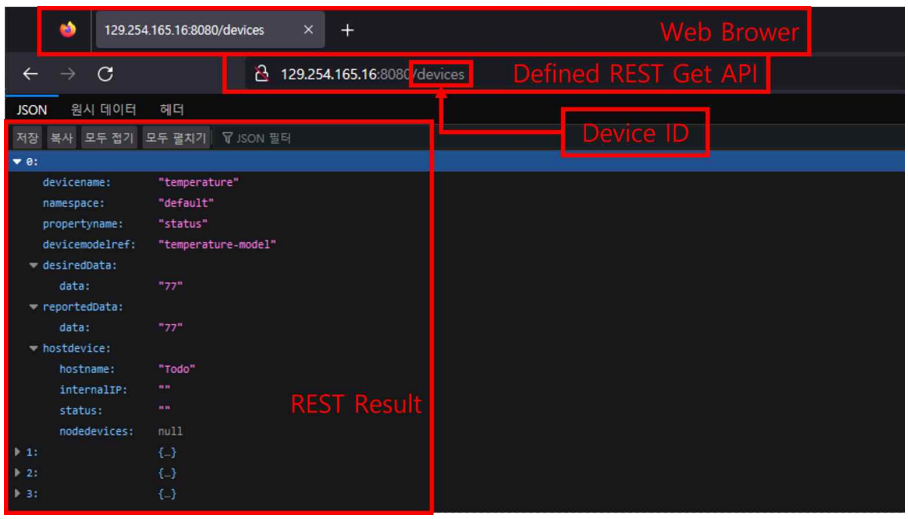


그림 393. 자원 가상화 인프라 기반 장치 메타데이터 확인

  . 자원 증강 실현을 위한 대규모 실/가상 디바이스 장치관리기 개발
    - EdgeCPS 플랫폼상에서 구동되는 장치관리기를 통해 10,000개의 실/가상 비컴퓨팅 디바이스가 동시에 관리되는 서비스 및 테스트베드를 구축함
    - 디바이스리스 장치관리기는 디바이스의 연결부터 시스템에 연결된 장치의 메타데이터, 증강을 위한 디바이스별 맵핑 정보, 장치별 자원 정보, 외부 센서에서 생성된 센싱 정보까지 통합 관리를 관제함

수행기관	개발 목표	개발내용 및 범위
		- 웹 모니터링 도구를 통해 각 기능에 따른 장치 관리 현황을 즉각적으로 확인할 수 있음

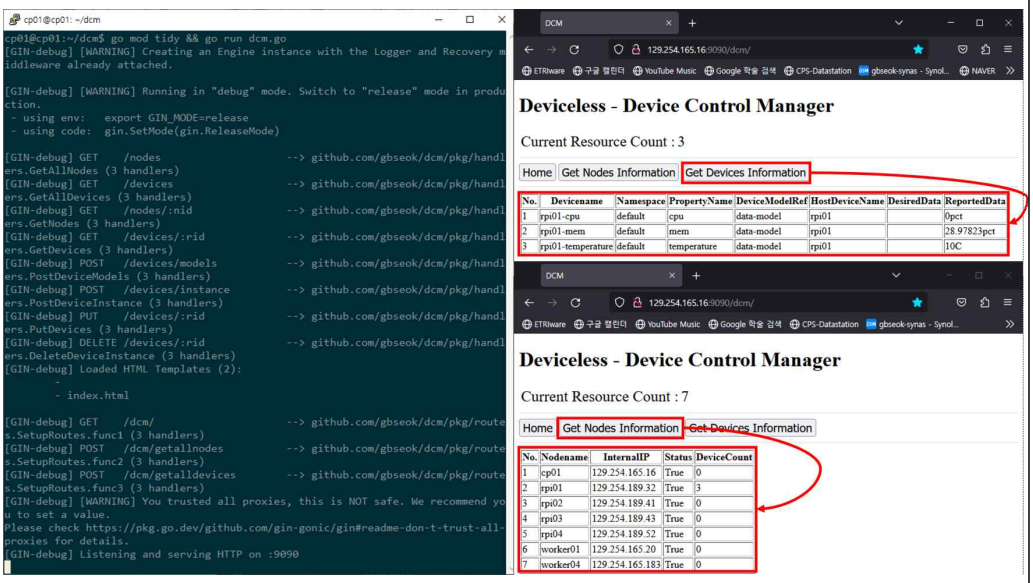


그림 394. EdgeCPS 플랫폼 기반 장치관리기 실행 및 웹 모니터링 도구

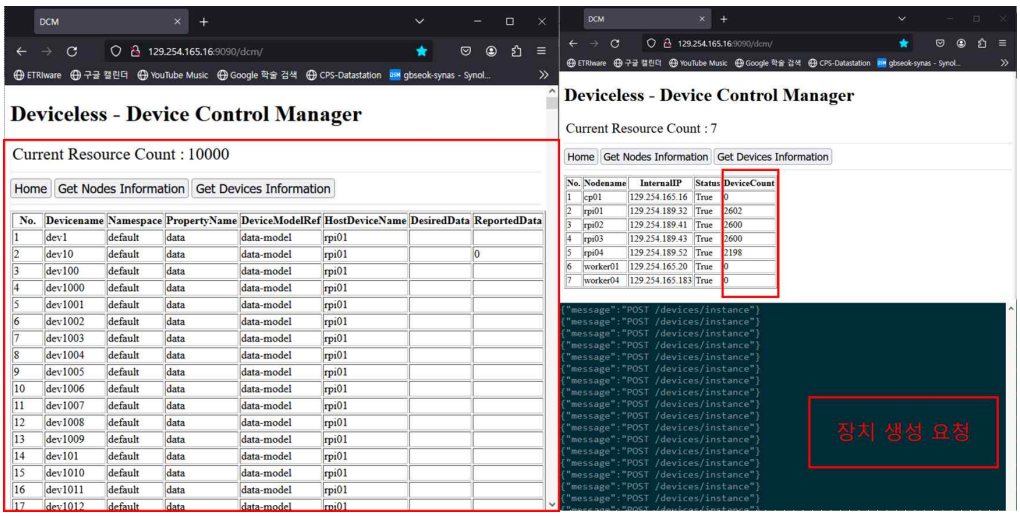


그림 395. 장치관리기 기반 실/가상 디바이스 생성 및 제어

- 장치 정보에 매핑된 메타데이터의 업데이트를 통해, 외부 자원이나 센서에 접근할 수 있는 인터페이스를 제공함으로써 자원 증강을 지원하기 위한 기본 설계 및 연구 환경을 검증함

수행기관	개발 목표	개발내용 및 범위

```
cp01@cp01:~/dcm$ sh scripts/curl/demo/put-devices.sh 77
+ value=77
+ curl -XPUT -H Content-type: application/json -d {
  "twins": [
    {
      "desired": {
        "metadata": {
          "type": "string"
        },
        "value": "77"
      },
      "propertyName": "data",
      "reported": {
        "value": "77"
      }
    }
  ]
} localhost:9090/devices/dev10
{"message":"Put /devices"}cp01@cp01:~/dcm$
```

그림 396. 장치관리기 기반 외부 자원 증강 및 데이터 접근

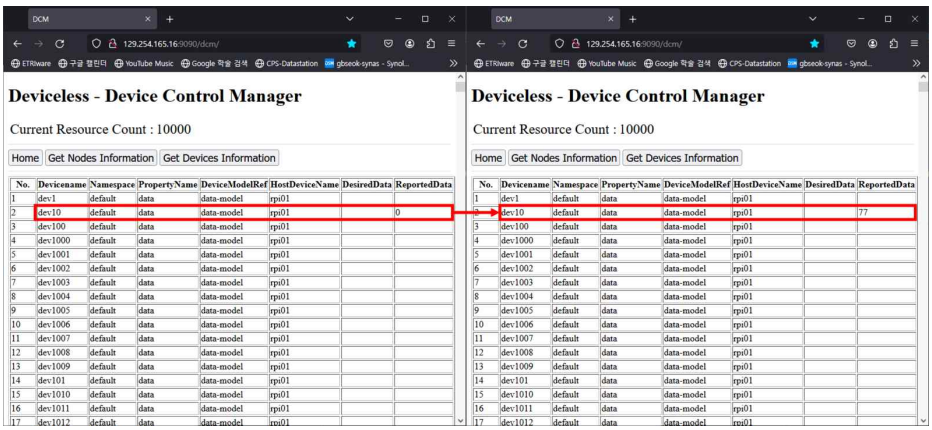
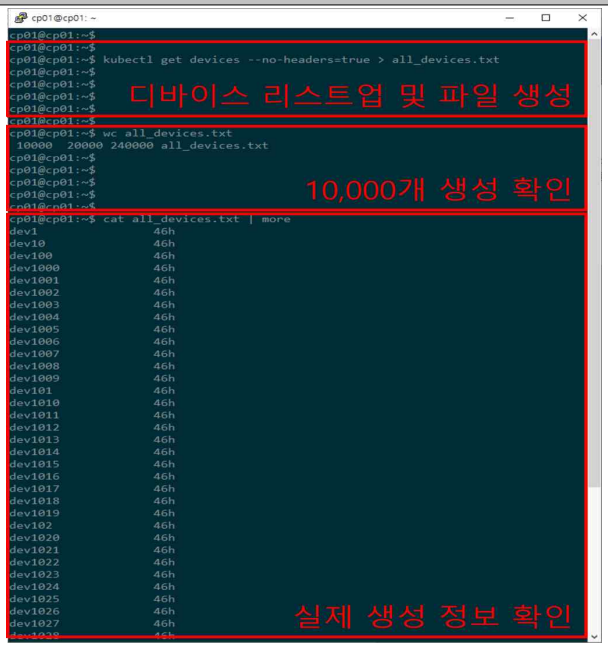


그림 397. 장치관리기 기반 자원 및 디바이스 증강 프로토타입 검증

- Device-Edge-Cloud의 계층적인 형태로 구성된 EdgeCPS 플랫폼 환경에서 클라우드를 기반으로 한 장치관리기를 검증과 실/가상 디바이스 10,000개 제어를 통한 대규모 자원 관리를 기능을 실험하고 검증을 진행함

수행기관	개발 목표	개발내용 및 범위
		 그림 398. 10,000개 디바이스 메타데이터 생성 검증
	④엣지사이버물리 시스템을 위한 대규모 디바이스 제어 플랫폼 프로토타입 개발	**• 실-가상 디바이스간 컴퓨팅 자원/서비스 연계 기술 연구** - 기술 개요 · 연구 정의: EdgeCPS 영역 내의 대규모 디바이스들의 디바이스 트윈에 대해 로컬/원격 EdgeCPS 간 모델/데이터를 증강하고, 이를 모니터링 및 제어하기 위한 실-가상 디바이스간 컴퓨팅 자원/서비스 연계, Inter-EdgeCPS의 디바이스 증강을 위한 데이터 전처리, 실디바이스의 원격 활용을 위한 디바이스 미러링 기술 연구 수행 - 당해연도 연구 목표 및 범위 (1) 컴퓨팅 자원/서비스 연계 기술의 이해관계자 관점 및 요구사항 정의 (2) 사용자 친화적 EdgeCPS 서비스 명세 모델링 언어 개발 (3) 모델 기반 EdgeCPS 워크플로우 서비스 오케스트레이션 도구 개발 - 세부 연구 개발 내용 (1) 컴퓨팅 자원/서비스 연계 기술의 이해관계자 관점 및 요구사항 정의 . EdgeCPS 플랫폼의 컴퓨팅 컴퓨팅 자원/서비스 연계 기술의 이해관계자들을 정의하고, 각 관점에서 고려하는 컴퓨팅 자원, 서비스 개발/운영 전문성 수준을 분류함 . 도출 된 4개의 이해관계자 관점 중 서비스 개발/운영 전문성 수준이 낮은 비즈니스 프로세스 매니저와 서비스 워크플로우 매니저를 위해 자원/서비스 연계 기술의 높은 사용자 편의성 제공이 필요하며, 이를 위한 두 개의 주요 기능 요구사항(1. 모델 기반의 시각적인 EdgeCPS 서비스 명세 2. 서비스 운영을 위한 컴퓨팅 자원 자동 할당 및 실행)을 도출함.

수행기관	개발 목표	개발내용 및 범위

표 29. 모델 기반 컴퓨팅 자원/서비스 연계 기술의 이해관계자 정의

이해관계자 (stakeholder)	정의	관심 대상	서비스 개발/운영 전문성 수준
비즈니스 프로세스 관리자 (Business process manager)	EdgeCPS 서비스로 달성하고자 하는 요구사항을 명세하고, 각 요구사항을 달성할 개괄적인 프로세스를 계획 및 관리하는 주체	비즈니스 프로세스 명세, 요구사항 명세	낮음
서비스 워크플로우 관리자 (Service workflow manager)	비즈니스 프로세스를 소프트웨어 기반 서비스로 구축하고 관리하는 주체	소프트웨어 기반 업무 명세, 업무 제어 흐름 명세	낮음
마이크로서비스 제공자 (Microservice provider)	서비스 워크플로우를 구성하는 최소 단위 업무를 마이크로 서비스로 소유하고 제공하는 주체	서비스 인터페이스 구현, 서비스 인스턴스 실행	높음
디바이스 클러스터 관리자 (Device cluster manager)	서비스를 운용할 자원을 정의하고 관리하는 주체임. 이는 다수의 컴퓨팅 자원 (클라우드 서버, 엣지 서버, 모바일 기기 등)을 소유 및 제공하는 주체	디바이스 및 자원 관리, 클러스터 운영	높음

(2) 사용자 친화적 EdgeCPS 서비스 명세 모델링 언어 개발

. EdgeCPS를 위한 서비스 명세 모델링 언어를 개발하기 위해, 산업에서 활발히 사용 중인 서비스 명세 플랫폼들의 온톨로지에 기반하는 상향식 모델링 언어 개발 방법론을 아래 그림과 같이 정의함. 해당 방법론은, 플랫폼 의존적인 온톨로지 도출, 플랫폼 독립적인 온톨로지 도출, EdgeCPS 서비스 명세 모델링 언어 정의 단계를 가짐

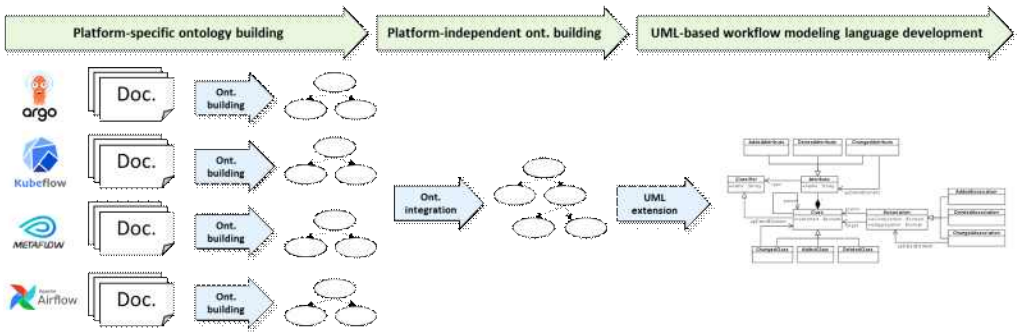


그림 399. EdgeCPS 서비스 워크플로우 모델링 언어를 위한 상향식 개발 방법론

. 아래 그림과 같이 4개의 상용 서비스 워크플로우 플랫폼 Argo workflow, Kubeflow, Metaflow, Airflow의 공식 문서로부터 온톨로지를 추출하고, 시맨틱 매핑을 통해 플랫폼 의존적인 온톨로지들을 통합함으로써 EdgeCPS 서비스 명세 언어 개발을 위한 통합 온톨로지를 구축하고 공개함.

수행기관	개발 목표	개발내용 및 범위

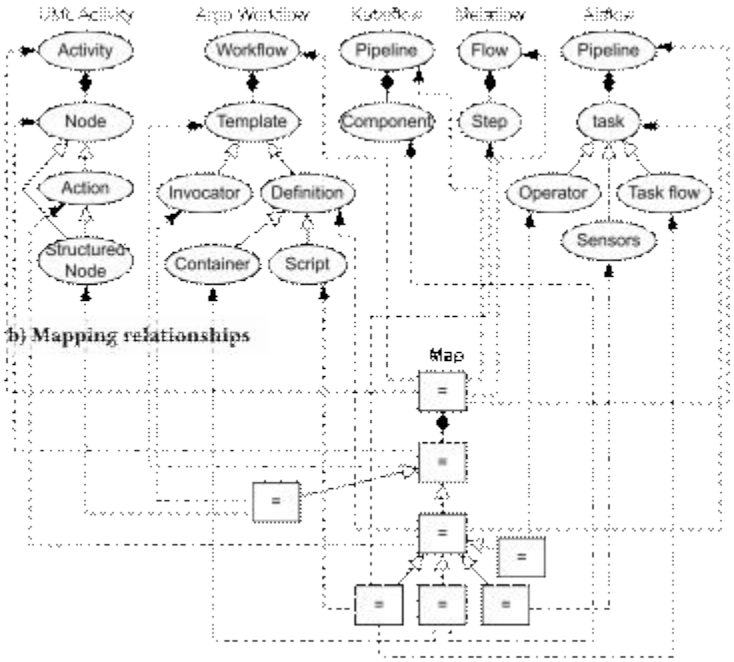


그림 400. EdgeCPS 서비스 워크플로우 모델링 언어를 위한 온톨로지 통합

. 상향식 방법론으로 도출한 온톨로지에 기반하여 EdgeCPS 서비스를 위한 모델링 언어를 개발함. 모델링 언어는 UML의 메타모델을 확장한 아래 그림의 모델링 언어 프로파일로 정의함

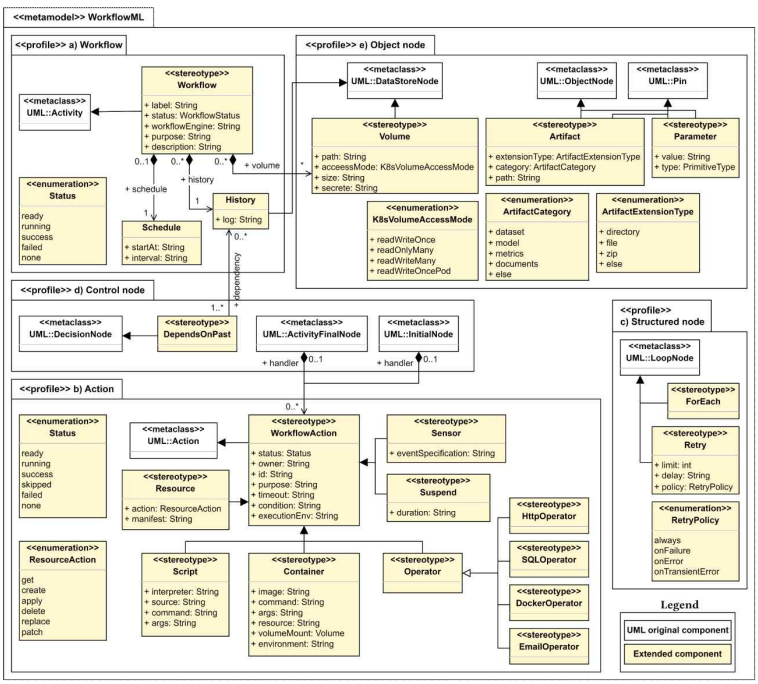


그림 401. EdgeCPS 서비스 워크플로우 모델링 언어의 메타모델 프로파일

. 개발된 모델링 언어를 평가하기 위해, 모델링 도구 프로토타입을 아래 그림과 같이 개발하고 오픈소스 도구로 공개함. 또한 프로토타입 도구를 이용하여 본 모델링 언어가 EdgeCPS 사용자로하여금 코딩 없이 모델에 기반하여 서비스를 쉽게

수행기관	개발 목표	개발내용 및 범위
		명세하도록 하는 표현력을 평가함

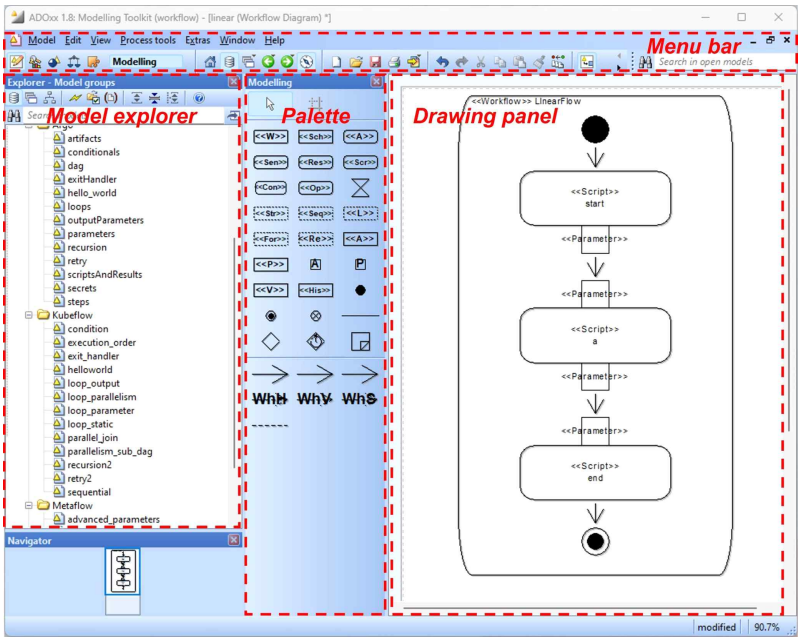


그림 402. EdgeCPS 서비스 워크플로우 모델링 도구 프로토타입

(3) 모델 기반 EdgeCPS 워크플로우 서비스 오케스트레이션 도구 개발

. 개발한 EdgeCPS 서비스 명세 모데릴 언어에 기반하여 컴퓨팅 자원/서비스 연계 기술을 제공하는 서비스 오케스트레이션 도구의 개념 아키텍처를 아래 그림과 같이 설계함

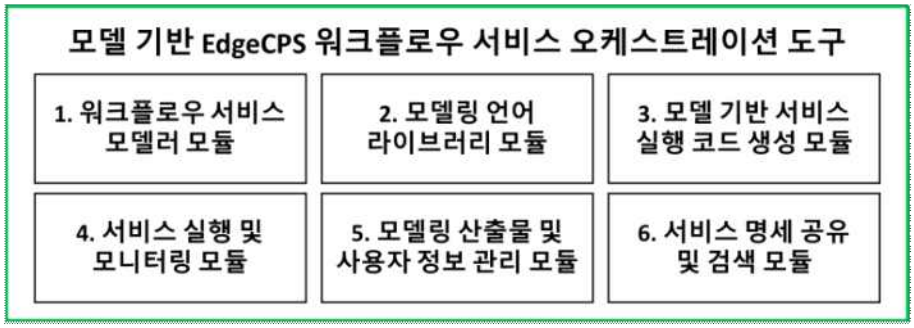
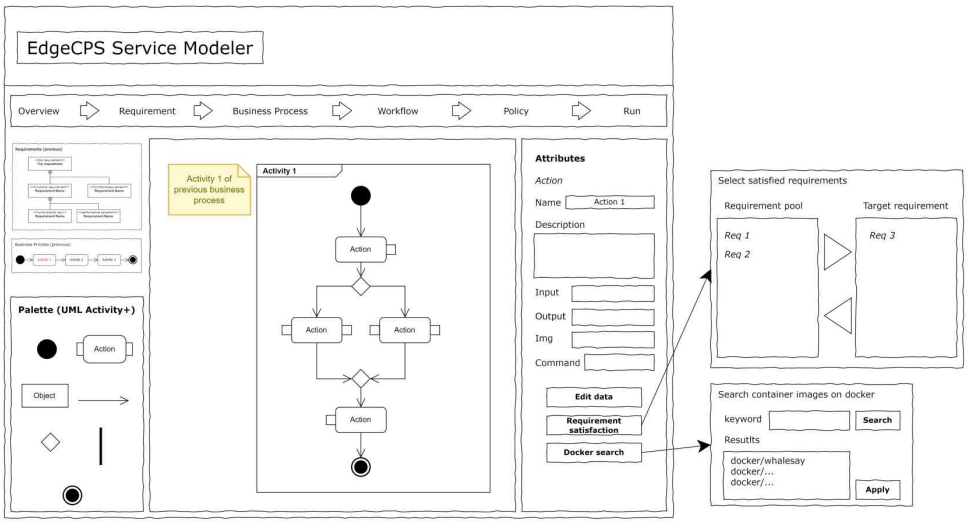
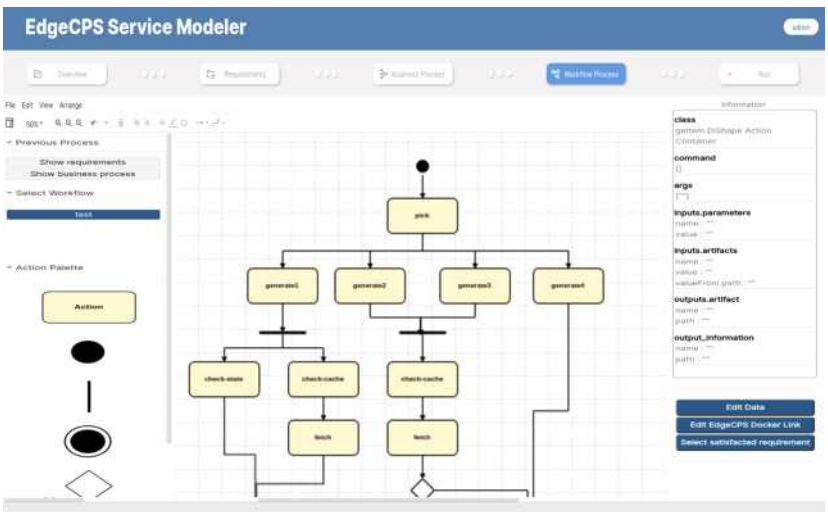


그림 403. 모델 기반 EdgeCPS 워크플로우 서비스 오케스트레이션 도구 아키텍처

. EdgeCPS 서비스 오케스트레이션 도구의 사용자 인터페이스를 아래 그림과 같이 설계함. 아래의 그림은 정의한 EdgeCPS 서비스 명세 언어에 기반한 서비스 모델링 화면의 인터페이스임

수행기관	개발 목표	개발내용 및 범위
		 그림 404. EdgeCPS 서비스 오케스트레이션도구 UI 설계 1 (서비스 워크플로우 모델링) . 모델 기반 EdgeCPS 워크플로우 서비스 오케스트레이션 도구를 아래 그림과 같이 웹기반 어플리케이션으로 개발 . EdgeCPS 서비스의 비즈니스 프로세스 매니저와 서비스 워크플로우 매니저는 도구를 이용하여 요구사항, 비즈니스 프로세스, 서비스 워크플로우를 시각적으로 명세함으로 서비스를 쉽게 개발할 수 있음  그림 405. 웹 기반 EdgeCPS 서비스 오케스트레이션 도구 UI 1 (서비스 워크플로우 모델링) . 사용자는 도구를 통해 시각적 모델링으로 명세한 서비스를 EdgeCPS 플랫폼에서 실행 시킬 수 있음. 이는 내부적으로 서비스 코드 자동 생성, 컴퓨팅 자원 할당, 서비스 실행, 모니터링 및 시각화를 수행함. 서비스 실행 결과를 아래 그림의 인터페이스에서 시각적으로 확인할 수 있음.

수행기관	개발 목표	개발내용 및 범위

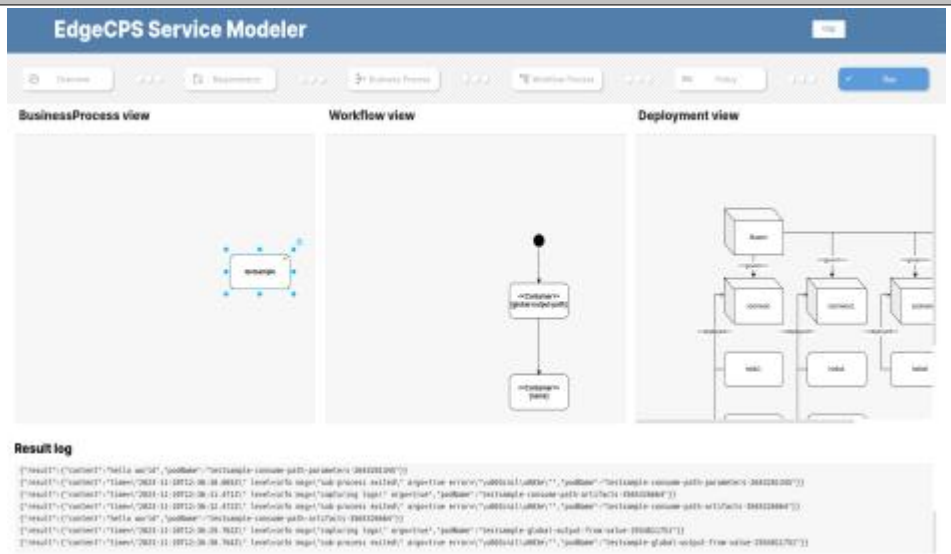


그림 406. 웹 기반 EdgeCPS 서비스 오케스트레이션 도구 UI 2 (서비스 실행 및 모니터링)

• Inter-EdgeCPS의 디바이스 증강을 위한 데이터 전처리 기술 연구

- 기술 개요
  · 연구 정의: EdgeCPS 영역 내의 대규모 디바이스들의 디바이스 트윈에 대해 로컬/원격 EdgeCPS 간 모델/데이터를 증강하고, 이를 모니터링 및 제어하기 위한 실-가상 디바이스간 컴퓨팅 자원/서비스 연계, Inter-EdgeCPS의 디바이스 증강을 위한 데이터 전처리, 실디바이스의 원격 활용을 위한 디바이스 미러링 기술 연구 수행

- 당해연도 연구 목표 및 범위
  (1) EdgeCPS 플랫폼 간 디바이스 증강을 위한 실-가상 디바이스의 동기화 구조 설계
  (2) Inter-EdgeCPS간 엣지 디바이스 데이터 전송 및 동기화 실증
  (3) Inter-EdgeCPS간 디바이스 상호작용 프로토타입 구현

- 세부 연구 개발 내용
  (1) EdgeCPS 플랫폼 간 디바이스 증강을 위한 실-가상 디바이스의 동기화 구조 설계
  . EdgeCPS에서 현실과 동기화된 가상공간(MetaTwin)을 구현을 지원하기 위한 실-가상 디바이스 동기화 플랫폼 설계

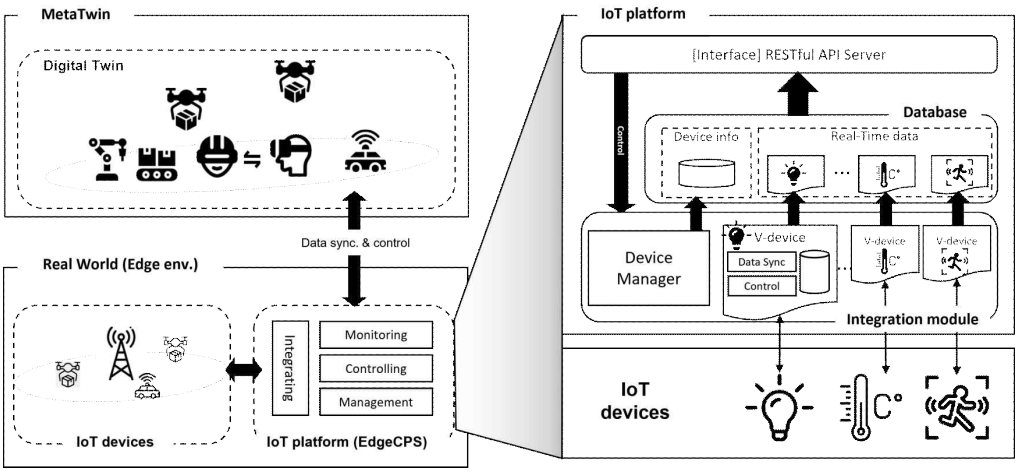
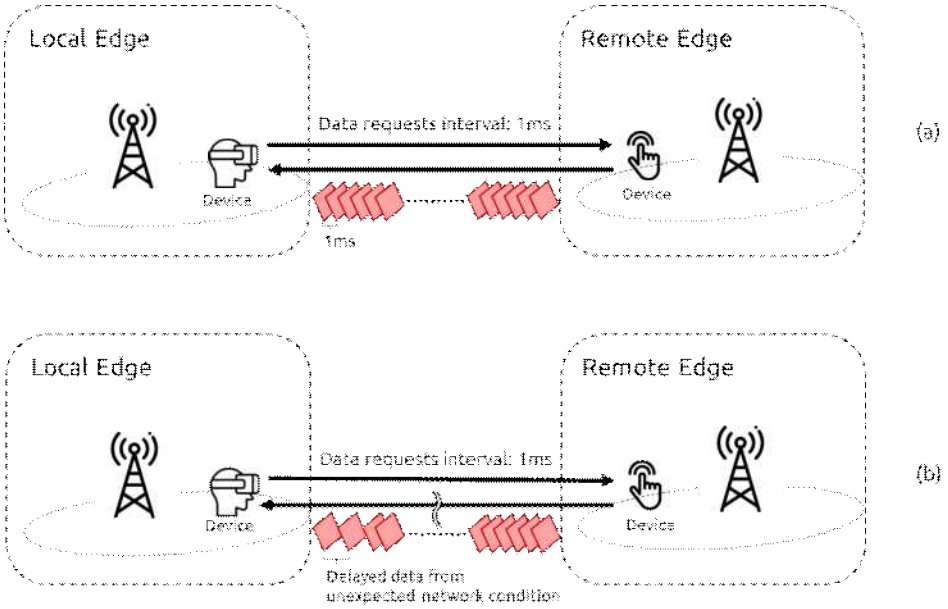


그림 407. EdgeCPS에서 현실-가상공간 동기화를 위한 설계 및 IoT 플랫폼 세부 구조

수행기관	개발 목표	개발내용 및 범위
		(2) Inter-EdgeCPS간 엣지 디바이스 데이터 전송 및 동기화 실증 . 통합된 Inter-EdgeCPS 간 디바이스 증강을 실시하기 위해 로컬 EdgeCPS에 실제 존재하는 디바이스를 실시간 관리하고 디바이스의 실시간 정보에 대한 외부 요청에 응답하는 역할을 개발함 . 원거리 엣지간 통합이 실시되었을 때 리모트 엣지에서 로컬 엣지로의 디바이스 제어 및 로컬 엣지의 디바이스를 가상 디바이스 형태로 리모트 엣지와 공유할 수 있도록 함 . 가상 디바이스는 엣지에 존재하는 실제 물리 디바이스를 가상화한 형태로, 물리 디바이스와 실시간 데이터 동기화를 수행함. 동기화된 데이터는 IoT plaform이 관리하는 실시간 시계열 데이터베이스로 전송됨. 가상 디바이스는 외부의 요청에 의해 실제 디바이스를 제어할 수 있음 . Device Manager는 엣지 클러스터에 존재하는 IoT 디바이스들을 가상 디바이스를 이용해 통합하고 제어 및 관리함. Device Manager는 IoT 디바이스들에 대한 실시간 정보를 실시간 통합 및 업데이트하며, 외부에서 IoT platform으로 요청된 엣지 디바이스 제어 요청을 가상 디바이스를 활용하여 처리함. . 엣지 내·외부에서 디바이스에 대한 동기화 및 제어를 용이하게 할 수 있도록 RESTful Api 서버를 위치시켜 해당 서버를 통해 모든 디바이스 데이터 및 제어 요청과 수행이 이루어질 수 있도록 하였음. 또한 이를 위한 Json 형식의 데이터 포맷을 설계함 (3) Inter-EdgeCPS간 디바이스 상호작용 프로토타입 구현 . 설계된 IoT platform을 ETRI 내부 엣지 클러스터에 구현하여 VR기기를 통해 접속해 로컬-원격 EdgeCPS 간 스마트 전구 동기화를 실시함. 그림 408. 로컬-원격 EdgeCPS 간 디바이스 증강 **• 실디바이스의 원격 활용을 위한 디바이스 미러링 기술 연구** - 기술 개요 · 연구 정의 : 지리적으로 한정된 엣지 환경 내에 이종 규격을 갖는 다수의 자율 사물들이 규격에 관계 없이 개별 사물에 할당된 원활한 임무 수행을 위하여 EdgeCPS는 자원 및 기능 증강을 제공하며, 이를 실현할 수 있는 아키텍처 및 이를 기반으로 데이터를 고속 처리할 수 있는 기법 연구 - 당해연도 연구 목표 및 범위 (1) 엣지간 저지연 서비스를 위한 가상 디바이스 미러링 구조 및 절차 분석

수행기관	개발 목표	개발내용 및 범위
		(2) 지리적으로 멀리 떨어진 엣지간 디바이스 미러링 구조 및 절차 기술 개발 (3) Inter-CPS 간 디바이스 증강 기술을 확장하여 디바이스 미러링 플랫폼 설계 - 세부 연구 개발 내용 (1) 엣지간 저지연 서비스를 위한 가상 디바이스 미러링 구조 및 절차 분석 . 지리적으로 떨어진 엣지간 데이터 동기화를 실시할 때 물리적 거리로 인해 필연적으로 발생하는 데이터 지연을 최소화하기 위해 가상 디바이를 활용하는 가상 디바이스 미러링 기술의 구조 및 절차를 설계함 . 지리적으로 떨어진 엣지 영역간 통신 및 상호작용은 거리 및 예측할 수 없는 오류에 따른 네트워크 지연이 발생함. 예를 들어, 엣지간 촉감 데이터를 전송하는 경우, 거리 및 예측할 수 없는 네트워크 상태로 1ms 이상의 딜레이가 발생할 경우 사용자는 위화감을 느끼게 되며, 이를 해소하기 위한 방안 분석  그림 409. 엣지간 데이터 전송 : (a) 엣지간 데이터의 온전한 전송, (b) 딜레이가 발생해 위화감을 조성 (2) 지리적으로 멀리 떨어진 엣지간 디바이스 미러링 구조 및 절차 기술 개발 . 지리적으로 멀리 떨어진 엣지간 요청 및 응답에 대해 ITU-T에서 제정된 Tactile Internet(촉감: 1ms, 시각: 10ms, 청각: 100ms) 기준을 만족시기기 위해 디바이스 미러링 기술을 개발 . 양측 엣지 클러스터에 존재하는 고성능 서버에 동기화된 가상 디바이스를 배치하고 이를 통한 데이터 동기화 정의 . 고성능 엣지 서버를 통해 데이터 전달/동기화를 빠르고 안정적으로 수행하며, 원격 엣지의 실 디바이스에 대한 가상 디바이스를 양측 에지 서버에 위치시켜 실시간 동기화를 수행함

수행기관	개발 목표	개발내용 및 범위

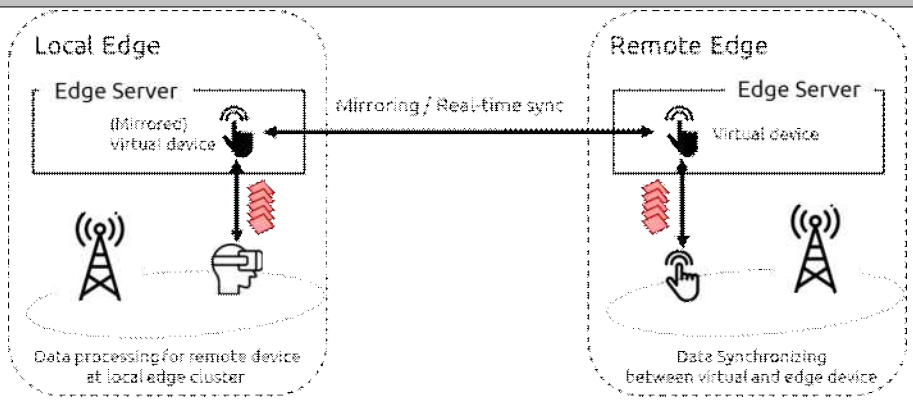


그림 410. 엣지간 가상 디바이스를 활용한 동기화

(3) Inter-CPS 간 디바이스 증강 기술을 확장하여 디바이스 미러링 플랫폼 설계

. 디바이스 미러링 플랫폼은 로컬 엣지에 존재하는 디바이스와 원격 에지의 미러링된 가상 디바이스를 관리하는 것을 목표로 함. 디바이스 미러링 플랫폼은 디바이스 명세 관리부와 가상 디바스 관리부로 구성됨

. 디바이스 명세 관리부(송수신부): 가상 디바이스의 재원 및 실시간 동기화를 위한 정보를 관리함.가상 디바이스 미 러링이필요한 경우 원격에 위치한 디바이스의 명세 관리부로부터 명세를 제공받아 복제된 가상 디바이스를 만들고가상 디바이스들끼리 동기화 시킴

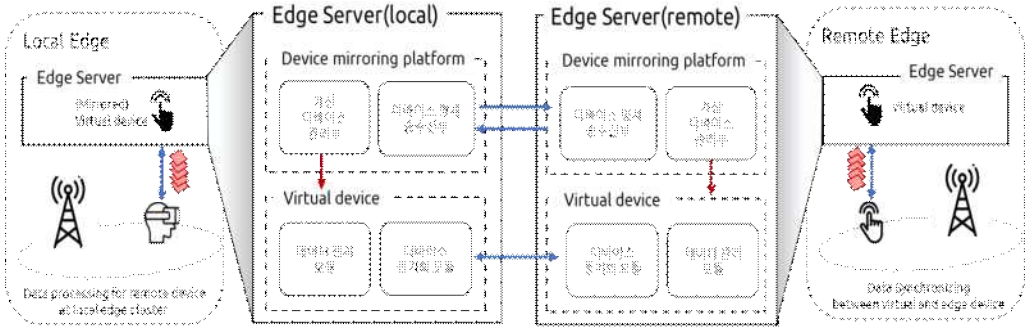


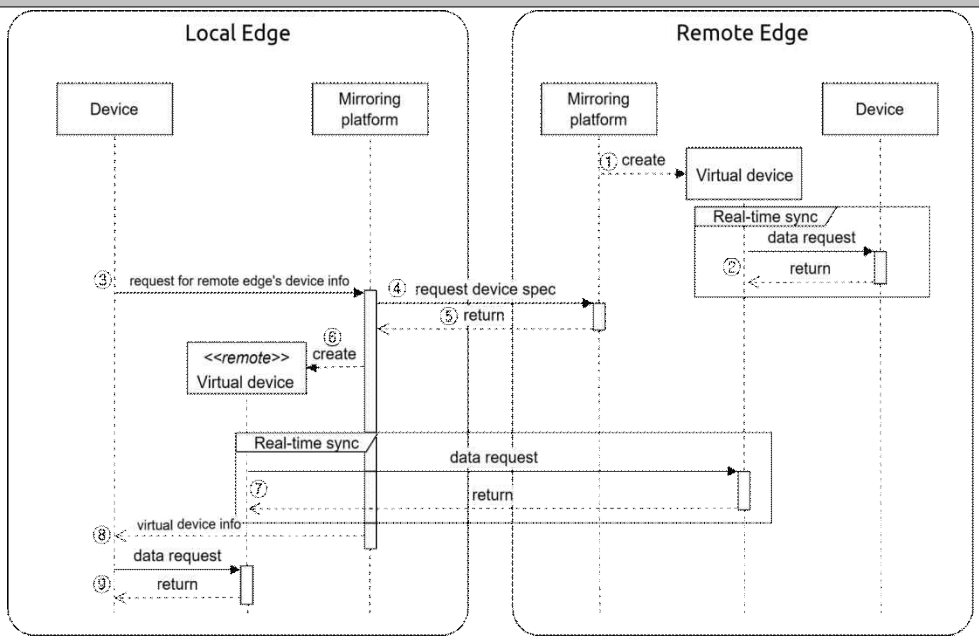
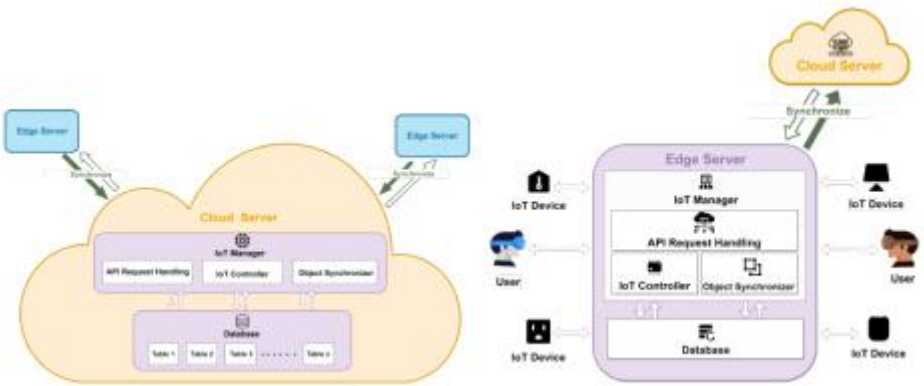
그림 411. 디바이스 미러링 플랫폼 및 이를 위한 가상 디바이스의 구조

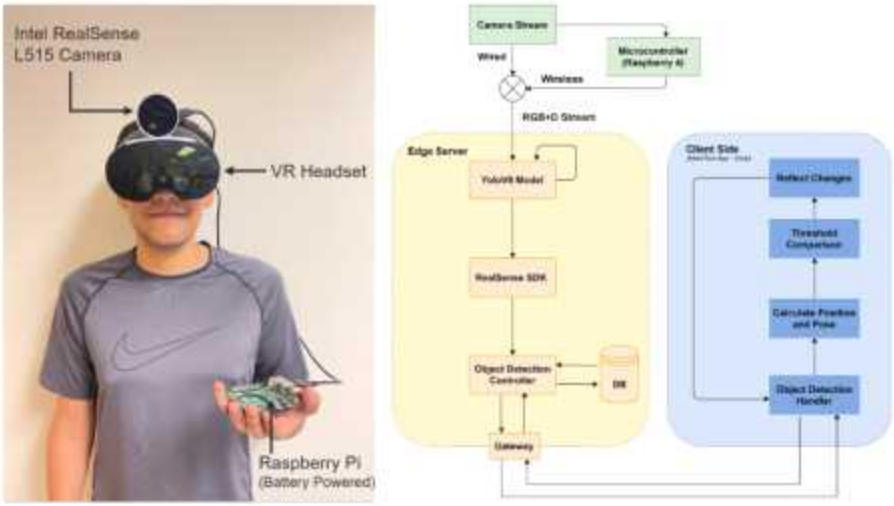
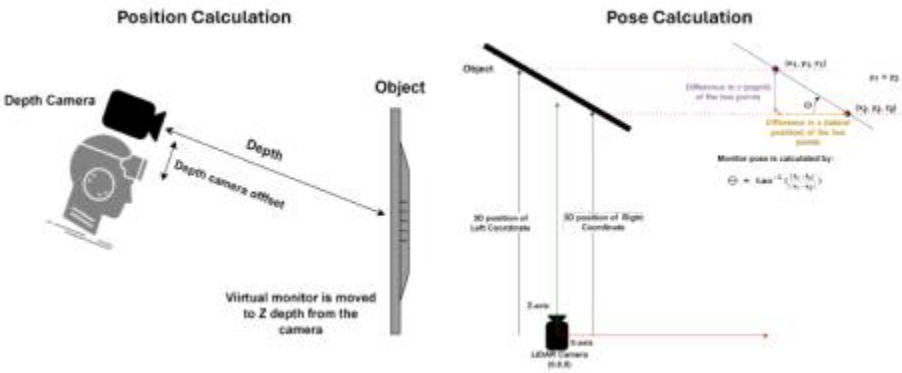
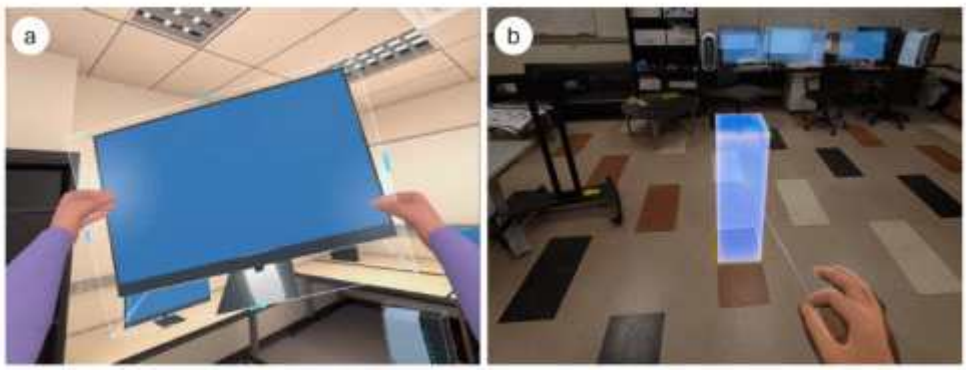
. 아래 그림은 가상 디바이스 생성 및 미러링 절차를 도시하며 가상 디바이스 관리부: 가상 디바이스 명세를 기반으로 가상 디바이스의 생성, 삭제, 업데이트를 수행함

. 또한 가상 디바이스에 대한 세부 추가 설계를 함으로서 엣지간 가상 디바이스 동기화를 수행하고, 가상 디바이스 내부에서 예상치 못한 네트워크 지연을 보완할 수 있도록 함

. 디바이스 동기화 모듈: 실제 디바이스의 정보를 가상 디바이스, 미러링 디바이스와 동기화함

. 데이터 관리 모듈: 실제 디바이스, 또는 원격 가상 디바이스로부터 전달받은 정보를 관리하도록 구성함

수행기관	개발 목표	개발내용 및 범위
		 그림 412. 디바이스 미러링을 통한 엣지간 동기화 절차
UT달라스 (공동4)	• 근거리 가상공간 사용자들간 멀티모달 상호작용 기술 연구	- 기술 개요 · 연구 정의 : 본 연구과제는 가상공간 환경을 개발하고, 다양한 테스트 실험환경을 구축하여 사람들이 물리적 공간에 한정되지 않고 다양한 협업을 할 수 있게 하는 가상환경 기반 EdgeCPS 플랫폼을 개발하는 것을 목표로 아래의 연구 목표 및 내용의 연구개발을 수행함 - 연구 개발 내용 (1) 근거리 가상공간 사용자들간 멀티모달 상호작용을 위한 요소 기술 개발 - 연구 개발 결과 (성능 평가 혹은 당해연도 연구 결과물) (1) 근거리 가상공간 사용자들간 멀티모달 상호작용을 위한 요소 기술 개발 . EdgeCPS 영역 간 네트워크 및 클라우드 기술 개발 - 로컬 디바이스와 사용자간 통신 및 신관리를 위한 엣지 서버 및 다양한 IOT 장비 연결을 위한 클라우드 서버 통신 기술 개발  그림 413. EdgeCPS 영역 간 네트워크 및 클라우드 연계 기술 . EdgeCPS 환경 내 객체 인식 기술 개발 - 다양한 객체의 인식, 움직임 등을 위한 객체 위치 연산 및 객체 포즈 연산 기술 개발. 이를 통해 물리적 객체의 움직임을 자동으로 감지하고 정확한 위치를 파악하여

수행기관	개발 목표	개발내용 및 범위
		접속된 모든 사용자들에게 동기화 함  그림 414. EdgeCPS 영역 내에서 HMD를 통한 객체 인식 파이프라인 . EdgeCPS 영역에 대한 증강현실 통합 기술 개발 및 캘리브레이션 고도화 - 가시화 모듈 통합 및 좀 더 정확한 실세계/가상세계 매핑을 위한 새로운 캘리브레이션 기법 개발 및 적용  그림 415. 포지션 및 포즈 계산 기법 . EdgeCPS 기반 가시화 편집 기술 개발 - EdgeCPS 환경 내에서 사용자가 손으로 자유롭게 씬 편집을 제공하는 편집 기술 개발  그림 416. 공간 내에서의 가시화 모듈 편집 기술

수행기관	개발 목표	개발내용 및 범위
		. IOT 관리 및 리소스 공유 기술 개발 - 다양한 IOT 장비 관리를 위한 공유 기술 개발
경북대 (위탁1)	• 스토리지 증강을 위한 동적 프로비저닝 기술 연구	- 기술 개요 · 연구 정의 : EdgeCPS 플랫폼에 연동된 대규모 디바이스들의 스토리지 증강을 안정적으로 지원하기 위한 동적 프로비저닝 정책을 설계하고, 동적 볼륨 자원 프레임워크 프로토타입을 구현함 - 연구 개발 내용 (1) 쿠버네티스 환경의 컨테이너 스토리지 가상화 기술 동향 및 특징 분석 (2) IO 특징과 스토리지 자원 특성에 따른 프로비저닝 정책 설계 (3) 저자원 디바이스 응용 특성에 따른 스토리지 증강 성능 평가 - 연구 개발 결과 (성능 평가 혹은 당해연도 연구 결과물) (1) 쿠버네팅스 환경의 컨테이너 스토리지 가상화 기술 동향 및 특징 분석 . 컨테이너 기반 가상화를 지원하는 Kubernetes의 스토리지 기술에 대한 비교 수행 - Rook-Ceph : Ceph, Cassandra, NFS와 연계 가능한 클라우드 네이티브 스토리지 서비스이며, 서로다른 PVC를 하나의 PV에 매핑할 수 있는 특징이 있음 - OpenEBS : 노드 단위로 스토리지를 관리하고 단순한 구성으로 빠른 구성이 장점이나, 여러 PVC를 하나의 PV로 관리하지 못하여 스테이트풀 활용에 한계가 있음 - Long-horn : 노드 별로 사용가능한 스토리지를 관리하고 있으며, 블록 단위로 관리하는 파일을 관리하고 있으며, PCV 공유를 지원하지 않음 - Ondat : 상용 소프트웨어 정의 클라우드 네이티브 스토리지 솔루션으로 볼륨 공간 최적화 및 풀을 통한 프로비저닝을 제공함하여 빠른 스토리지 공유 성능을 제고함

수행기관	개발 목표	개발내용 및 범위

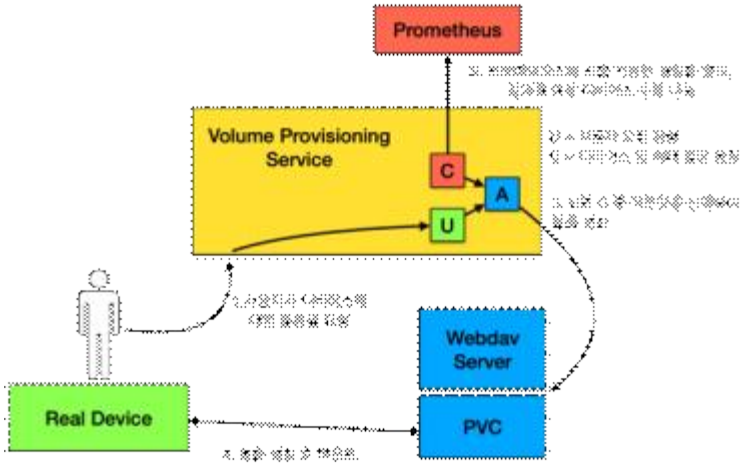
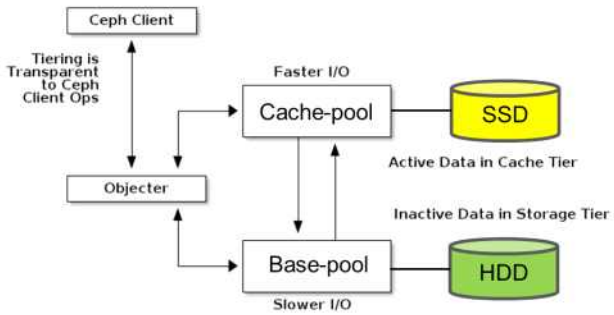
표 30. 스토리지 솔루션 기능 비교

Feature	Rook/Ceph	OpenEBS	Longhorn	StorageOS	Portworx	GlusterFS
Provisioning	Thin Provisioning	Thin Provisioning	Thin Provisioning	Thin Provisioning	Thin Provisioning	Thin/Thick Provisioning
Replication	Multi-site	Synchronous	Synchronous	Synchronous	Synchronous, Asynchronous	Synchronous. Asynchronous
Snapshot	Incremental Snapshot	Incremental Snapshot	Recurring Snapshot, Off-site Snapshot	Incremental Snapshot	Incremental Snapshot	File and Block Snapshot
Backup and Restore	Incremental Backup, Disaster Recovery	Velero Plugin	Incremental Backup, Object Storage Backup, Disaster Recovery	Native Backup / Restore	PX Store, Geo Replication	Geo Replication
Monitoring	Prometheus Plugin	Prometheus Plugin	BEST API (Prometheus, Graphite, Telegraf text format)	Prometheus Plugin	Prometheus Plugin	Prometheus Plugin
Dashboard	Ceph Dashboard	Grafana Dashboard	Longhorn UI, Grafana Dashboard	StorageOS UI, Grafana Dashboard	PX Central, Grafana Dashboard	Heuristic UI, CLI

(2) IO 특징과 스토리지 자원 특성에 따른 프로비저닝 정책 설계

. 물리 스토리지 특성 및 응용 I/O 특성을 동적으로 반영하기 위한 Cache-Tiering 기술 개발

- (Cache-tiering) SSD/HDD 간의 Tiering 체계 구축하여 HDD ↔ SSD간 데이터 마이그레이션 수행
- Cache-pool: 물리 SSD를 사용하는 OSD에 개체 저장을 위한 논리 파티션으로 cache tier 구현
- Base-pool: 물리 HDD를 사용하는 OSD에 개체 저장을 위한 논리 파티션으로 기본적으로 CephFS를 통해 데이터가 저장

수행기관	개발 목표	개발내용 및 범위
		 그림 417. 실디바이스 기반 스토리지 프로비저닝 정책 구현 흐름도  그림 418. Cache-tiering 정책 설계 (3) 저자원 디바이스 응용 특성에 따른 스토리지 증강 성능 평가 . 동적 볼륨 프로비저닝을 지원하는 스토리지 가상화 프레임워크 개발 - 개발 환경 : Docker v20.10.17, Kubernetes v1.25.2, Rook v1.9.10, Davfs2 (real device's file system client) v1.6.1, Ceph (server file system) v16.2.10, Nginx-Ingress v1.3.0, helm v3.9.2

수행기관	개발 목표	개발내용 및 범위

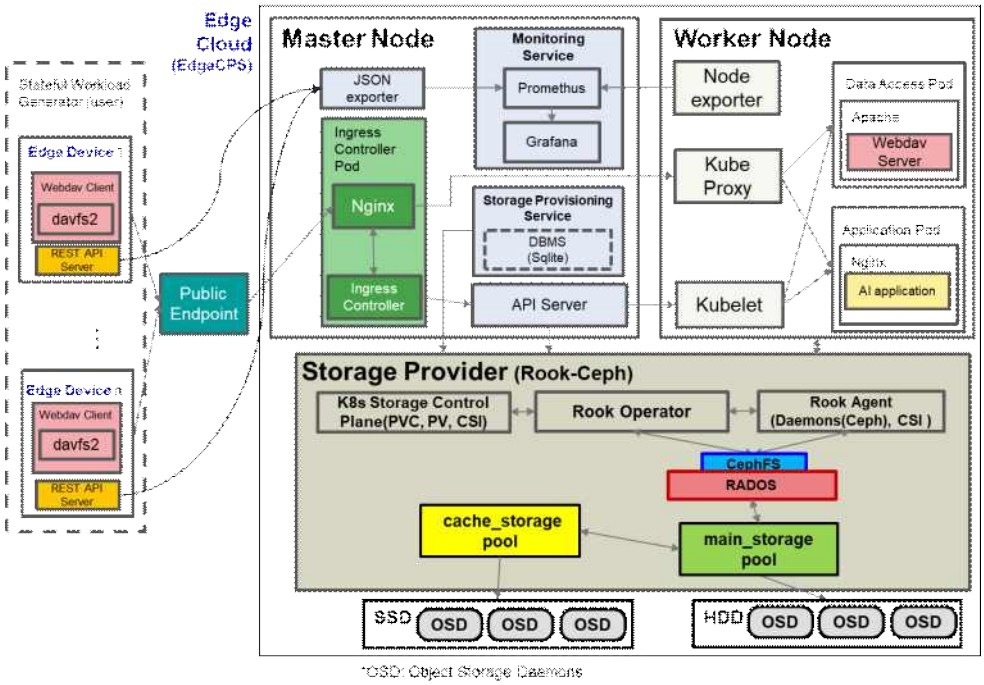


그림 419. 동적 프로비저닝 지원 스토리지 가상화 프레임워크

. 동적 볼륨 프로비저닝은 프레임워크에 기등록된 디바이스 사용자의 볼륨 증강 요청에 따라 서버 노드에 스토리지 공간이 생성되며, 생성된 볼륨은 사용자 디바이스에 직접 마운트되는 구조로 개발

. 디바이스와 연동된 볼륨은 서버측 스토리지의 임계 용량을 넘을 경우 자동으로 볼륨의 크기를 증가시켜, 디바이스의 증강을 지원하며 실 디바이스의 스토리지 자원이 남아있을 경우 실-가상 스토리지는 동기화된 상태로 관리 됨

. 스토리지 볼륨 프로비저닝 요청에 따른 성능 평가 수행 결과 증강할 스토리지의 요청 수에 영향을 받지 않고 지연시간 90초를 기준으로 10초 내외의 오차범위로 일정한 경향을 보임

. 스토리지 볼륨의 증강 용량에 따른 성능 평가 수행 결과 증강할 스토리지 볼륨 크기에 상관없이 90초를 기준으로 10초 내외의 오차범위 내에서 일정한 성능을 보였음

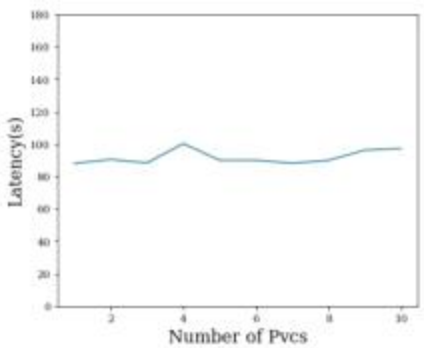


그림 420. 스토리지 수에 따른 자동 증강 기술의 성능 평가 결과

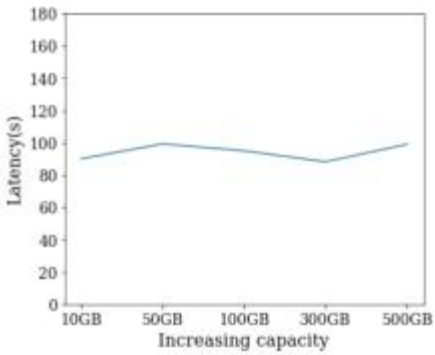
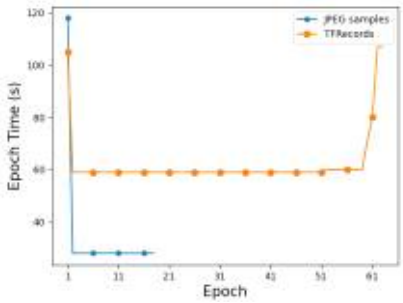
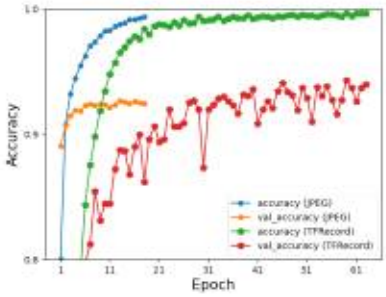
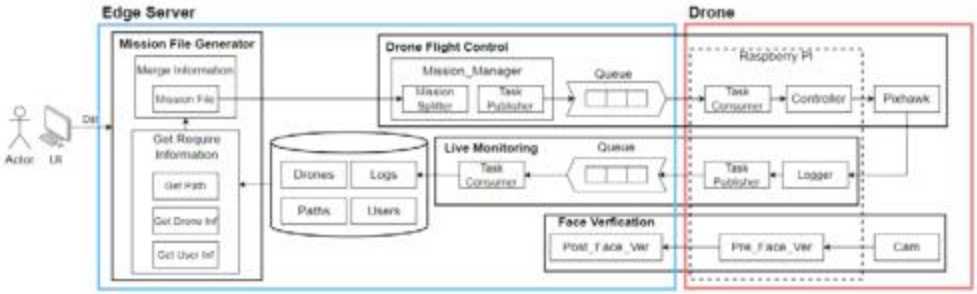


그림 421. 증가용량에 따른 스토리지 자동 증강 기술의 성능 평가 결과

. 응용특성에 따른 스토리지 증강 성능 평가를 위해 Tensorflow를 통한 성능 평가 실험을 진행하였고, 프레임워크에서 사용하는 CephFS를 통한 학습 연산시 TFRecord 포맷의 입력 시간에 대한 성능 평가를 실시하였음

. 데이터셋의 포맷에 따른 성능 평가를 통해 fetch stall이 발생하는 첫 에포크에서

수행기관	개발 목표	개발내용 및 범위
		TFRecord 사용시 에포크 시간이 줄었으며, 데이터 셋이 모두 캐싱된 이후에는 Jpeg 데이터 셋을 사용할 경우 에포크 시간이 2배 가량 감소하였음 . 정확도면에서는 Jpeg 데이터 셋을 사용하는 경우 더욱 빠르게 증가였으며, 학습 시간였기 더 좋은 성능을 보였음 . 로컬 스토리지를 통하 성능 평가에서는 한 스텝에서 입력 시간이 차지하는 비율이 약 43.4%로 fetch stall이 존재하는 것으로 보이며, 원격 스토리지를 이용한 스토리지 확장을 제공할 경우, fetch stall로 인해 학습 작업 부하의 성능이 크게 저하될 것으로 예상됨  그림 422. 데이터셋 포맷에 따른 에포크 성능 비교  그림 423. 데이터셋 포맷에 따른 정확도 비교
경상대 (위탁2)	• EdgeCPS 기반의 자원 제약 디바이스의 분할 실행 기술 연구	- 기술 개요 · 드론이 미션을 수행하는 환경은 다양한 외부 변수들로 인해 비행 중에 위험을 야기함. 외부 변수들은 근접하여 비행하는 다른 드론들, 돌풍, 빌딩풍, 비, 낙뢰 등 돌발적이고 예측하기 힘들다는 특징이 있음. 이러한 다양한 미션 수행의 변수들을 회피하기 위해서는 외부 환경을 실시간으로 센싱하고 드론의 진행경로를 수정하는 알고리즘이 요구됨. 본 연구에서는 EdgeCPS 기반의 GCS 이 외부 변수에 반응하여 실시간으로 드론의 경로를 변경하는 능동적 시스템을 설계하는 것을 목표로 함 - 연구 개발 내용 (1) 저/중/고성능 AuT의 자원 증강 검증 (2) EdgeCPS 대시보드 동기화를 통한 미션 수행 - 연구 개발 결과 (성능 평가 혹은 당해연도 연구 결과물) (1) 저/중/고성능 AuT의 자원 증강 검증 . 최근 활용도가 높아지고 있는 무인 항기(UAV)인 드론 기반 배송 시스템은 1) 임무 자동화, 2) 주문자 검증, 3) 성능 한계 극복의 3가지 요소가 만족되어야 함. 본 연구에서는 드론에 장착된 카메라를 이용하여 얼굴인식을 실시하고 이를 인증 과정에 추가하고, 배송 서비스의 확장성을 고려하여 EdgeCPS 기술을 도입함 . 시스템은 아래 그림과 같이 Edge Server와 드론으로 나뉘고 각 컴포넌트는 기능에 따라 임무 생성기(Mission File Generator), 드론 비행 제어(Drone Flight Control), 실시간 모니터링 (Live Monitoring) 그리고 얼굴 검증(Face Verification)으로 나뉨

수행기관	개발 목표	개발내용 및 범위
		 그림 424. EdgeCPS 임무 자동화 시스템 설계도 - RP는 연산을 수행하고 Pixhawk에게 비행 명령을 내리는 저사양 컴퓨팅 장치임. Pixhawk는 RP로부터 받은 비행 명령에 따라 드론의 비행을 제어하고 GPS 정보를 수집함. Edge Server는 UI를 통해 사용자의 요청에 맞게 비행 임무를 생성하고 드론과 통신하여 임무를 부여하고 고연산이 필요한 작업에 대해 자원을 증강해주는 역할을 함 - Database를 활용하여 드론 정보, 실시간 좌표, 사용자 정보 등 배송에 필요한 정보를 저장하며 RabbitMQ의 Queue를 활용하여 드론과 Edge server간 생성된 Task를 순차적으로 전달함. 임무 생성 모듈은 배송에 필요한 드론 이름, 고도, 도착지 좌표 그리고 주문자 정보와 사용자가 지정한 좌표로 갈 수 있는 경로를 DB에서 가져와 임무 파일을 생성함. 또한, 얼굴인식 과정에서 DB의 주문자의 얼굴 정보를 이용하여 얼굴검증을 실시함 - 실증을 위해 드론에 물품 적재를 위한 적재함을 설계하고 구현하였음. 배송물품으로 인해 발생되는 응력해석을 위해 Autodesk Inverter Professional을 활용함. 3D 프린팅의 재료로 쓰이는 PLA 필라멘트의 물리적 특성에 관한 물성치를 기반으로 시뮬레이션을 통해 적재함의 경량화와 구조적 강도 사이에 타협점을 찾아 최종 프로토타입을 결정하였음 - 드론 배송의 실증을 위해 Web UI를 개발하고 Web UI를 통해 드론에게 임무 하달 및 배송 완료를 확인하였음. 본 Web UI를 활용해 1) 드론에게 임무 하달, 2) 임무 목적지로 실 드론 이동 및 Web UI 동기화 확인, 3) 목적지 도착 후 얼굴 인식, 4) Web UI를 이용한 임무 복귀 의 4단계 시나리오를 단계별 검증하였음

수행기관	개발 목표	개발내용 및 범위

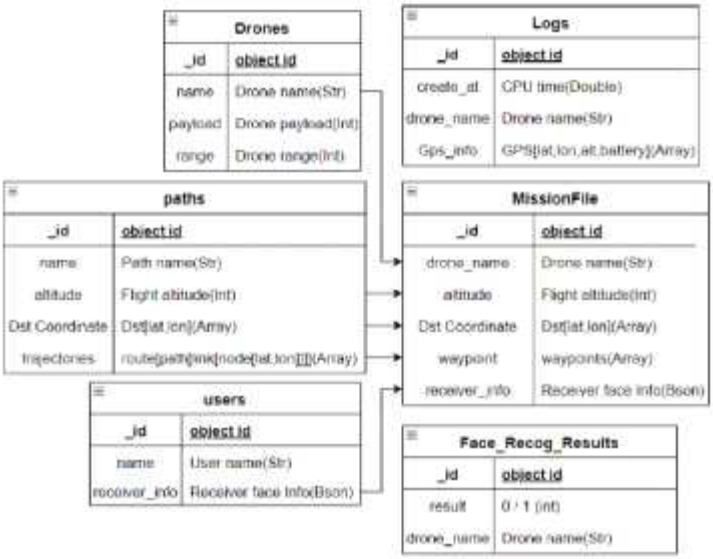


그림 425. 데이터베이스 스키마 설계도

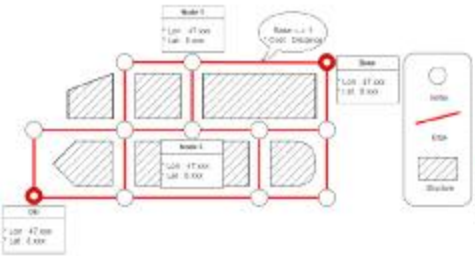
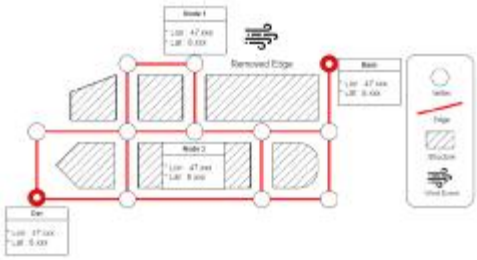
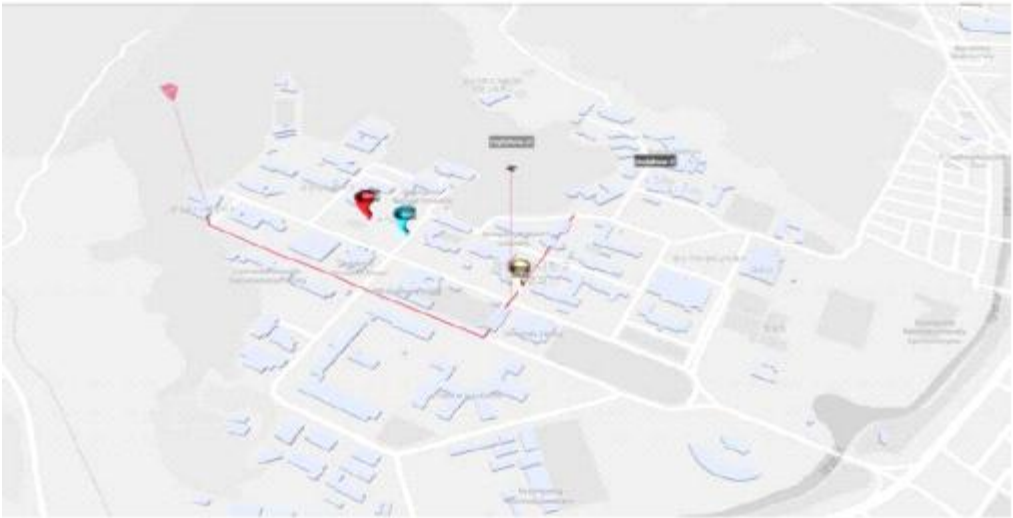
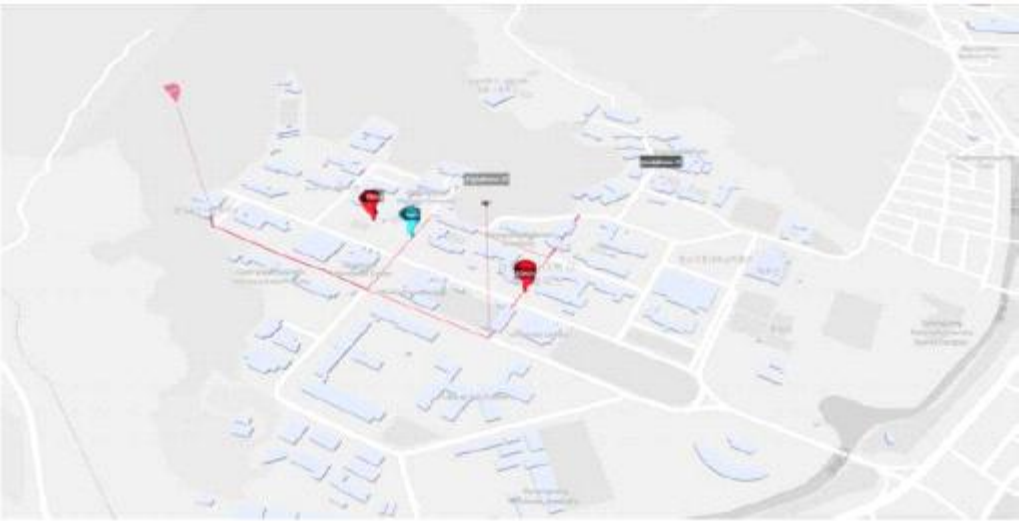
그림 426. 임무 자동화 시스템 시연

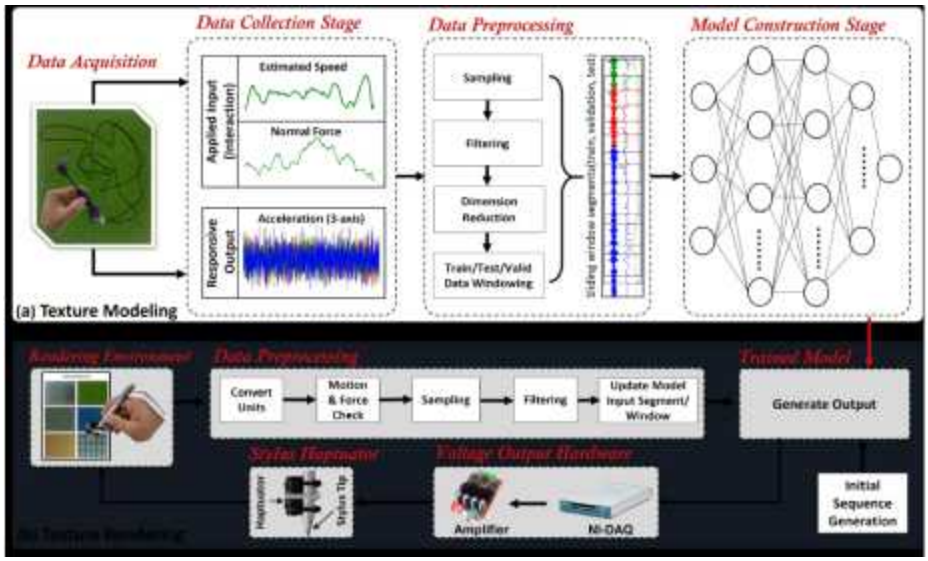
(2) EdgeCPS 대시보드 동기화를 통한 미션 수행

. 기존 비행경로 생성은 빌딩풍과 같은 동적인 외부 변화를 고려하지 않아 효율적인 임무 수행이 어려움. 또한 저/중/고성능 AuT 규격에 따라 실제 운행이 가능한 환경 및 고도가 다르기 때문에 AuT의 제원을 고려한 실시간 환경 기반 동적인 배송 경로를 가져야 할 필요성이 있음

. 따라서 본 연구에서는 배송 경로를 그래프 형태로 정의하고 배송 영역에 빌딩풍 정도를 변화시키고 이에 따라 동적으로 배송 경로를 변화시키는 시뮬레이션 시스템을 개발함

- 비행경로 결정 시스템은 빌딩풍 시뮬레이션 데이터를 기반으로 이벤트의 영향을 받는 간선을 식별하고 목적지까지의 후보 경로를 수정

수행기관	개발 목표	개발내용 및 범위
		 그림 427. 기존 AuT의 비행경로 및 노드  그림 428. 빌딩풍 시뮬레이션 결과에 따라 업데이트된 그래프  그림 429. 시뮬레이션 Web UI: 7m/s의 빌딩풍(노란색)의 빌딩풍이 10m/s로 바뀐 경우  그림 430. 시뮬레이션 Web UI: 비행 경로가 바뀐 모습
경희대 (위탁3)	• 로컬 모델 기반의 디바이스 데이터 증강 기술 연구	- 기술 개요 · 지리적으로 멀리 떨어진 엣지간 디바이스 제어를 실시할 때, 제어 주체에게 제어 대상 환경을 예측 불가능한 환경으로 가정하고, 예측불가능성을 극복하기 위한 방법을 EdgeCPS에 적용하는 것을 목표로 함

수행기관	개발 목표	개발내용 및 범위
		- 연구 개발 내용 (1) 원격지에서 비정형 멀티모달 데이터 수집을 위한 텔레오퍼레이션 기술 연구 (2) 스트리밍 기반의 데이터와 모델 기반의 데이터를 스위칭하는 기준 연구 (3) 로컬 EdgeCPS 상의 불충분한 모델에 대응하는 라이브러리를 EdgeCPS 상에 배치하여, 데이터 증강을 지원하는 방법론 개발 - 연구 개발 결과 (성능 평가 혹은 당해연도 연구 결과물) (1) 원격지에서 비정형 멀티모달 데이터 수집을 위한 텔레오퍼레이션 기술 연구 . 데이터 기반 로컬 모델을 점진적으로 완성하는 기술 개발 및 네트워크 문제가 발생했을 때 구축된 로컬 모델을 통해 원격 디바이스를 제어하는 기술 개발 - 기존 데이터 중심 모델링 방식과 머신러닝 기술을 결합한 푸리에 강화 트랜스포머 인코더 네트워크(FoTEN) 개발 - 기존 연구의 로컬 촉감 모델을 구축하는데 필수적으로 필요한 사전 준비 시간을 줄이기 위해 경량화된 네트워크 - 촉감 정보의 위상정보, 파라미터 확장성을 고려해 입력 차원을 자유롭게 설계함으로써 전처리 단계를 간소화  그림 431. 데이터 기반 로컬 모델링의 전체 구조 . 데이터 수집 하드웨어 제작 및 데이터 수집 - 촉감 모델을 구축하기 위한 데이터 수집 하드웨어 구축 및 이를 통한 여러 질감 데이터 수집 실시 - 3축 가속도계alc 외부 위치 추적기를 통해 질감 모델의 위치에 따른 데이터를 수집함 - 본 연구에는 아래 그림과 같이 6개의 텍스처 샘플을 준비해 데이터를 수집하고, 수집된 데이터는 DFT321 알고리즘을 사용하여 세 축에 대한 시간 정보와 스펙트럼 에너지를 단일 축에 매핑함 . 푸리에 강화 변압기 인코더 네트워크(FoTEN) - 트랜스포머 네트워크를 기반으로 푸리에 인코더 블록을 아키텍처에 통합하여 촉감 스펙트럼 영역에서 전체 신호의 글로벌 컨텍스트를 캡처할 수 있도록 설계함 - 제안된 FoTEN 아키텍처는 인코더 블록과 푸리에-인코더 블록이라는 두 개의 독립적인 블록으로 구성되며 자세한 내용은 아래 그림과 같음

수행기관	개발 목표	개발내용 및 범위

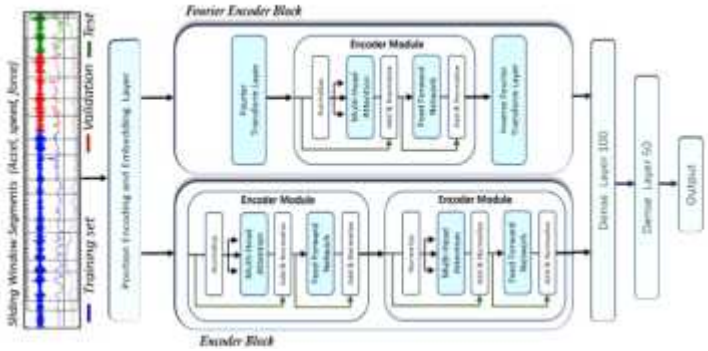


그림 432. 딥러닝 질감 모델링을 위한 FoTEN의 구조

. 제안 아키텍처의 평가를 위해 초방향적 텍스쳐 6종(인공 잔디, 천, 강철 메시, 플라스틱 메시, 고무 메시 및 판지)을 임의로 합성하고 촉감에 대한 실제 데이터와 예측 데이터를 비교함. 실제 신호 및 생성된 신호가 시간 영역 뿐 아니라 스펙트럼 영역에서도 잘 정렬되어 있는 것을 확인할 수 있었음

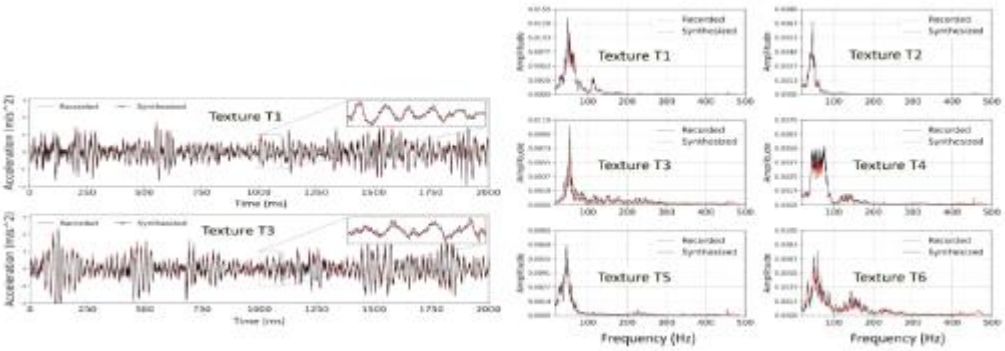


그림 433. 추정 가속도 신호 및 실제 가속도 신호, 시간축 및 주파수 축 비교

. 또한 지각적 성능을 평가하기 우해 여섯 명의 실험 참가자(남성 6명, 여성 1명; 평균 연령 29.3세)가 본 연구에서 제안한 FoTEN 방식과 DSTN 이 두 접근 방식으로 생성된 가상 텍스처를 해당 실체 텍스처와 0부터 100까지의 척도로 비교함. 비교 결과 본 연구에서 제시한 방법의 실감 정도가 더 높게 나타남

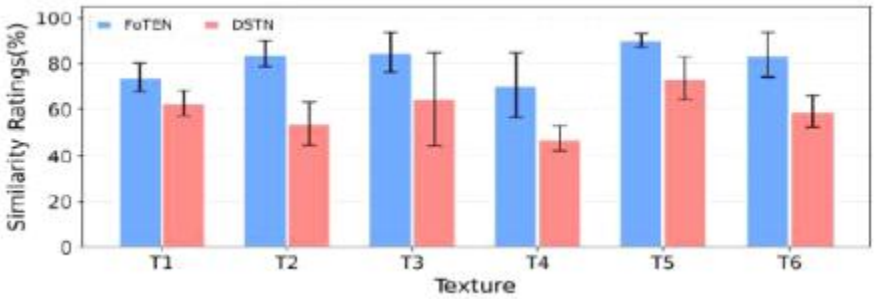


그림 434. 실제 데이터와 로컬 모델 기반 증강 데이터 간의 성능 비교

(2) 스트리밍 기반의 데이터와 모델 기반의 데이터를 스위칭하는 기준 연구

. 다음 환경 모니터링 파라미터를 이용해 현 스트리미이 데이터 및 로컬 모델 데이터의 질을 확인하고, 어떤 데이터가 좀 더 인지지석으로 Relable 한지 선정함

- 결과적으로 각 파라미터의 영향력 및 성격에 따라 다른 가중치를 주고, 스트리밍 데이터 및 로컬 모델 데이터의 상대적인 Reliablility를 추정하는 알고리즘을 개발함

수행기관	개발 목표	개발내용 및 범위

표 31. 환경 모니터링 파라미터

파라미터	특징
상호작용하는 문제의 성격	- 동적 물체인지 (스트리밍), 정적 물체인지 (모델) - 변형체(스트리밍) 인지 비 변형체인지 (모델)
현재 네트워크 지연 상황 및 대역폭	- 지연이 크거나 대역폭이 작으면 모델 기반으로 렌더링
전달되고 있는 촉감 신호와 관련된 촉각 수용체 종류	- 촉각 수용체의 시간 해상도가 높을수록 모델 기반 유리
현 촉감 모델의 완성도	- 모델 완성도가 낮으면 스트리밍 유리

(3) 로컬 EdgeCPS 상의 불충분한 모델에 대응하는 라이브러리를 EdgeCPS 상에 배치하여, 데이터 증강을 지원하는 방법론 개발

. EdgeCPS 클러스터로 구성된 디바이스를 통해 새로운 질감 물체와 접촉했을 때 아직 로컬에 촉감 모델이 생성되지 않는 접촉 초기에 햅틱 피드백을 제공하기 위해, 미리 많은 샘플들에 대한 질감 라이브러리를 구축하고 신호와 가상 유사한 질감 샘플을 선정하여 렌더링에 사용하는 방법론을 연구

. 방법론 구현을 위해 먼저 실제 가속도 데이터를 기반으로 보유한 질감 모델 중에 가장 유사하는 모델을 추정하는 알고리즘을 개발함

- 이를 위해 스랜스포머 네트워크를 사용한 분류 모델을 새로 구현하고 실시간 테스트를 위해 스트리밍 시뮬레이터를 만들어 햅틱 모델의 레이블을 추정하는 시뮬레이션을 검증함

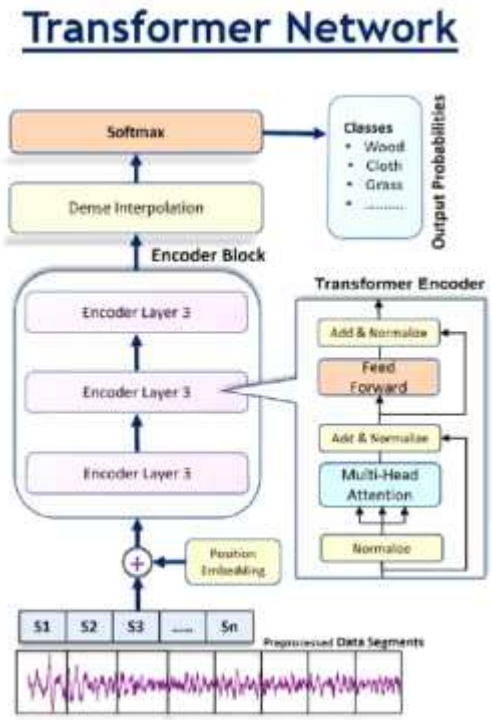


그림 435. 트랜스포머 네트워크 기반의 로컬 모델 생성 방법

수행기관	개발 목표	개발내용 및 범위

ETRI (주관)

. 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 프로토타입 개발
. 엣지사이버물리 시스템을 위한 대규모 디바이스 제어 플랫폼 프로토타입 개발

전인걸

UT달라스 (공동4)

. 근거리 가상공간 사용자들간 멀티모달 상호작용 기술 연구

경북대 (위탁1)

. 스토리지 증강을 위한 동적 프로비저닝 기술 연구

경상대 (위탁2)

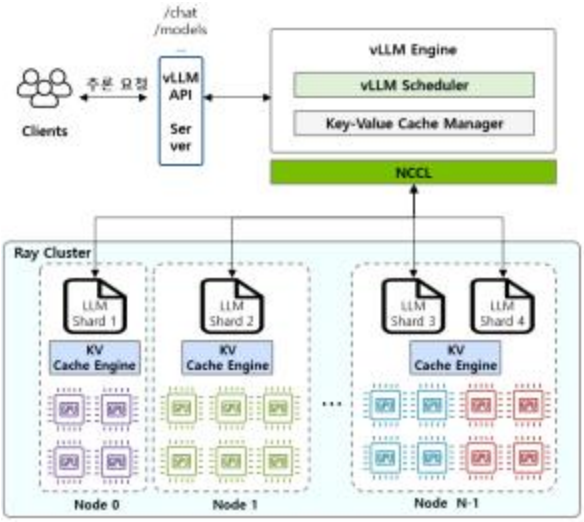
. EdgeCPS 기반의 자원 제약 디바이스의 분할 실행 기술 연구

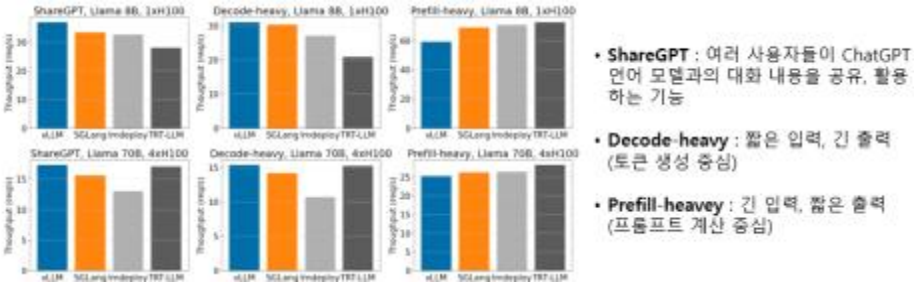
경희대 (위탁3)

. 로컬 모델 기반의 디바이스 데이터 증강 기술 연구

(5) (2단계) 3차년도(2024년도) 개발 내용 및 범위

[이종가속기AI컴퓨팅]

수행기관	개발 목표	개발내용 및 범위
ETRI (주관)	① AI 병렬컴퓨팅 구조 연구	**• GPU 기반 AI 분산/병렬 컴퓨팅 구조 연구** - 연구 개요 · 생성형 대규모 언어 모델(LLM)의 빠른 발전과 광범위한 채택으로 인해 LLM 워크로드가 다양한 애플리케이션에서 핵심 워크로드가 되고 있음 · 특히, 각 도메인에서의 추론 서비스 질 향상을 위해 빠른 추론이 중요해지고 있음 · LLM 추론을 위한 컴퓨팅 클러스터는 엄격한 서비스 수준 목표(SLO, Service-Level Objective)를 갖춘 다수의 쿼리를 처리하고 있음 · LLM 추론은 데이터센터 부하에서 빠르게 증가하는 비중을 차지하고 있음. 이러한 상황은 DGX A100, H100과 같은 전력 밀도가 높은 하드웨어가 이러한 작업을 지원하기 위해 배치되면서 전력, 에너지, 탄소 집약성을 높이는 결과를 초래하며, 이 문제를 효과적으로 해결하기 위해 해당 시스템의 에너지를 관리하기 위한 포괄적인 기술이 필요함. · LLM 추론 성능향상을 위해서 다중 GPU와 다중 컴퓨팅 노드를 활용한 분산/병렬추론을 수행할 때 발생하는 통신 병목을 줄이고자 활발한 연구가 진행되고 있음 · LLM 모델의 크기와 추론 요청이 증가할수록 추론에 필요한 GPU 개수와 컴퓨팅 노드의 수가 많아지며 이로 인해 추론 작업에 필요한 전력사용량이 급증하는 문제가 있음 · 추론 성능향상과 장비 구축비용 절감을 함께 도모하기 위해 이기종 하드웨어 장치 (GPU, NPU, FPGA 등)를 사용하여 LLM 추론 작업을 수행하는 연구들도 활발히 진행되고 있음 · 본 연구에서는 위에서 언급한 현 기술의 한계를 극복하고자 가장 범용적으로 사용되고 있는 GPU 기반 추론 성능을 향상 시키는 분산/병렬컴퓨팅 구조 연구를 수행하고자 함 - 다중노드, 다중 GPU를 활용한 LLM 분산 추론 프레임워크 vLLM · GPU 병렬분산 추론 도구인 vLLM 은 LLM 의 확장성과 효율성을 극대화하는 목표로 개발된 오픈소스 LLM 분산 추론 도구임. Apache-2.0 license 형태로 제공됨 · vLLM은 GPU 메모리 사용 최적화와 모델의 추론 속도 향상을 주요 목표로 설계됨 · 아래 그림은 다중 컴퓨팅 노드 환경에서 다중 GPU를 사용한 효율적인 LLM 분산 추론 기능을 지원하는 도구 중 가장 널리 사용되고 있는 vLLM 의 동작 개념도를 보임  그림 436 다중 컴퓨팅 노드 환경에서 이종 GPU를 사용한 LLM 분산 추론 동작 개념도 (vLLM) · vLLM 이 동작하는 클러스터는 LLM 추론을 수행하는 클러스터는 다수의 컴퓨팅 노드로 구성되며, 각 컴퓨팅 노드에는 GPU들이 장착되어 있음. 다중노드의 GPU 자원들은 Ray를 이용해서 통합 관리됨. Node 0과 Node 1처럼 동종 GPU 들이 다수 장착된 형태의 노드가 있으며, Node N-1처럼 이종 GPU들이 하나의 노드에 함께 장착된 형태의 노드가 있음. 사용자들의 추론은 vLLM이 제공하는 API 서버가 제공하는 REST (Representational

수행기관	개발 목표	개발내용 및 범위
		State Transfer) API의 URL을 이용해서 요청됨. vLLM 엔진은 vLLM 스케쥴러와 키-밸류 (Key-Value) 캐시 관리자로 구성됨 · vLLM의 특장점은 다음과 같음. 다중 GPU를 이용한 텐서 병렬 (Tensor Parallel)과 다중 노드를 이용한 파이프라인 병렬 (Pipeline Parallel)를 지원하여 모델과 파라미터를 효율적으로 분할 및 병렬처리함 · vLLM의 동적 메모리 관리 (Dynamic Memory Management) 기능은 GPU 메모리를 효율적으로 활용해서 더 큰 모델의 추론을 가능하게 함. 특히, 비활성 상태의 텐서를 GPU 또는 NVMe로 오프로딩하여 메모리 병목현상을 완화 시킴 · vLLM은 배치 사이즈 (batch size) 가 가변적인 요청을 비동기 처리해서 추론의 처리량 (throughput)을 극대화함 · vLLM은 체크포인트를 분산 환경에 최적화된 형태로 저장하고 로드하여 빠른 추론을 가능하게 함 · vLLM의 주요 구성요소는 다음과 같이 3가지로 구성됨. 1. Parallel Execution Engine (계산 효율성을 극대화하도록 설계된 분산 실행 엔진), 2. Inference Scheduler (입력 데이터 배치를 효율적 스케쥴링하여 GPU 활용도 극대화), 3. Memory Manager (GPU, CPU, NVMe를 포함한 다중 메모리 계층을 효율적으로 관리) · vLLM 은 GPU 메모리를 효과적으로 활용하여 LLM 추론 처리량이 우수함. 아래 그래프에서 ShareGPT 는 여러 사용자가 ChatGPT 언어 모델과의 대화 내용을 공유 및 활용하는 워크로드임. Decode-heavy는 사용자의 짧은 입력에 대해 긴 출력을 생성하는 토큰생성 중심의 워크로드임. Prefill-heavy는 사용자의 긴 입력에 대해 짧은 출력을 생성하는 프롬프트 계산 중심의 워크로드임. vLLM은 세 가지 워크로드 중 메모리 접근이 많이 발생하는 ShareGPT와 Decode-heavy 워크로드에서 높은 처리량을 보임.  그림 437 LLM 분산추론 SOTA 프레임워크 성능 비교 [1] vLLM, https://docs.vllm.ai/en/latest/ [2] SGLang, https://github.com/sgl-project/sglang [3] LMDeploy, https://github.com/InternLM/lmdeploy [4] TensorRT-LLM, https://github.com/NVIDIA/TensorRT-LLM · 동작 시험 : 아래는 3개 노드로 구성된 GPU 클러스터 환경에서 vLLM을 이용하여 Mixtral-8x7B-v0.1-Instruct 모델을 텐서 병렬 (4), 파이프라인 병렬(3)로 설정하여 분산추론하는 모습을 캡쳐한 화면임. dl012 노드는 A100 (40GB) 4장, dl013 노드는 A100 (80GB) 4장, dl025 노드는 A6000 (40GB) 4장으로 구성됨. 클러스터의 GPU 자원들은 분산처리 프레임워크인 Ray에서 관리되며, vLLM 은 Ray에서 제공하는 이기종 GPU들을 활용해서 LLM 분산/병렬추론을 수행함

수행기관	개발 목표	개발내용 및 범위

그림 438 vLLM을 활용한 다중노드 이종 GPU 분산 추론
시험결과-3노드 3종 GPU 활용

- LLM 추론의 성능 지표
  · LLM 추론 성능은 TTFT와 TBT로 측정함.
  · TTFT (Time To First Token) : 첫 번째 출력 토큰이 생성되는데 필요한 지연 시간
  · TBT (Time Between Tokens) : 각 새로운 출력 토큰이 생성될 때 필요한 지연 시간
- LLM 추론의 병렬화 종류
  · LLM 모델을 여러 GPU로 분할하여 추론을 수행하면, 성능을 향상시키고 더 큰 메모리 용량을 활용할 수 있음. LLM 추론은 일반적으로 파이프라인 병렬(Pipeline Parallelism, PP)과 텐서 병렬(Tensor Parallelism, TP)을 사용함
  · 파이프라인 병렬(PP)은 LLM 레이어를 GPU 간 분할하며, 레이어의 모든 연산자/텐서는 동일 GPU에 유지됨. 이 방식에서는 GPU 간 통신이 연속된 두 단계 사이에서만 이루어짐
  · 텐서 병렬(TP)은 각 레이어의 일부를 각각의 GPU에 할당하며, 레이어별로 모든 GPU에 대한 데이터 집계가 필요하기 때문에 고대역폭 통신이 요구됨
  · TP는 NVLink와 같은 고대역폭 인터커넥트를 통해 연결된 동일 서버 내의 GPU에서 더 좋은 성능을 발휘하며, PP는 서버 간 분산 환경에서 선호됨
- (문제 1) LLM 추론컴퓨팅의 이질성 극복을 위한 이종 GPU 관리의 필요성
  · LLM 추론 시 두 단계의 컴퓨팅이 발생함. (1단계) 프롬프트 계산 단계 (Prompt Computation) (2단계) 토큰생성 단계 (Token Generation). 프롬프트 계산 단계는 프롬프트의 모든 토큰을 한꺼번에 모델에 입력해서 병렬 처리를 수행하므로 GPU 계산 성능에 성능 제약을 받는 computation-bound 특성을 가짐. 토큰생성 단계는 앞선 토큰의 출력 결과를 입력으로 받아 순차적으로 새로운 토큰을 생성하므로 GPU 메모리 대역폭에 성능 제약을 받는 memory-bound 특성을 가짐. 따라서, 토큰생성 단계는 고성능 GPU의 계산 자원을 충분히 활용하지 못하는 문제가 발생하며 더 낮은 전력과 낮은 계산 성능을 가지는 장비를 이용해서 처리 가능. 아래 표는 LLM 추론컴퓨팅의 프롬프트 계산 단계와 토큰생성 단계의 이질적인 특성을 정리한 것임

표 32 LLM 추론컴퓨팅 단계별 특성

추론 컴퓨팅 단계	특성
프롬프트 계산 (Prompt Computation)	● 프롬프트의 모든 토큰을 한꺼번에 모델에 입력해 병렬 처리 ● GPU 계산 성능에 성능 제약 받음 (Computation-bound)
토큰 생성 (Token Generation)	● 앞선 토큰 출력 결과를 입력으로 받아 순차적으로 토큰 생성 ● GPU 메모리 대역폭에 성능 제약 받음 (Memory-bound) ● 고성능 GPU의 계산 자원이 충분히 활용되지 못하는 문제 (더 낮은 전력과 장비 구성 비용으로 실행 가능)

  · LLM 추론 시 컴퓨팅 특성이 다른 두 단계의 계산을 동일한 GPU에서 수행하면 (1) 추론 성능 저하 문제, (2) GPU 오버프로비저닝 문제가 발생함. GPU 오버프로비저닝 문제는 GPU 자원의 과도한 사용으로 장비 구축비용 및 전력사용량이 크게 증가하는 문제를 의미함.
- (문제 2) LLM 추론 시 발생하는 2가지 작업 부하 극복을 위한 이종 GPU 관리가 필요함.

수행기관	개발 목표	개발내용 및 범위

· (작업 부하 1) 추론 워크로드가 동적으로 변화하는데, 이를 잘 관리해서 서비스 해야 함. 동적으로 변화하는 추론 워크로드는 추론 요청의 길이, 추론 요청의 개수, LLM 서비스의 SLO 수준이 있음. (작업 부하 2) 자원 재분배 오버헤드가 동적으로 변화하는데, 이를 잘 관리해서 서비스 해야 함. LLM 추론 시 필요한 GPU의 계산 성능과 필요한 GPU의 개수가 달라지며 GPU 개수에 따라 모델 병렬화 수준이 동적으로 변화함. 아래 표는 LLM 추론 시 동적으로 변화하는 항목과 해당 항목의 작업 부하 종류를 정리한 것임

표 33 LLM 추론 작업 부하 종류와 동적 변화 항목

작업 부하 종류	동적 변화 항목
추론 워크로드	주요 요청 길이, 추론 요청 개수, LLM 서비스 SLO 수준
자원 재분배 오버헤드	필요 GPU 계산량, 필요 GPU 개수, 모델 변렬화 수준

- LLM 추론 성능향상을 위한 에너지-성능 프로파일링 기술 (DynamoLLM)

· 기존 연구 Stojkovic, Jovan, et al. "Dynamollm: Designing llm inference clusters for performance and energy efficiency." arXiv preprint arXiv:2408.00741 (2024)에서는 추론 서비스의 SLO 달성을 위해 LLM 모델의 추론은 고성능 GPU에서 실행되며, 이로 인해 추론 클러스터가 대량의 에너지를 소비하게 하고 과도한 탄소 배출을 초래하는 문제가 있음을 지적함

· 그러나, 추론 계산 특성의 이질성과 추론 워크로드의 변동성을 활용하여 에너지 효율성을 향상시킬 수 있는 기회가 존재함. 그러나 이러한 시스템 구성(예: 인스턴스 수, 모델 병렬화, GPU 주파수)이 서로 다른 에너지-성능 절충점을 만들어내는 방대한 탐색 공간을 생성함. 이러한 문제를 해결하기 위해, 기존 연구인 DynamoLLM 은 LLM 추론 환경을 위한 최초의 에너지 관리 프레임워크를 제안함.

· DynamoLLM은 서비스의 성능 SLO를 충족하면서 LLM 서비스를 위한 에너지와 비용을 최적화하기 위해 추론 클러스터를 자동으로 동적으로 재구성함. 서비스 수준에서 DynamoLLM은 에너지를 53%, 운영 탄소 배출을 38% 절감하며, 고객 비용을 61% 감소시키면서도 지연 시간 SLO를 충족하는 것으로 나타남.

· DynamoLLM에서는 에너지 효율성을 정량화하기 위해, 특정 지연 시간 SLO (Service Level Objectives)를 충족하면서 소비되는 에너지를 와트시(Wh) 단위로 측정함. 지연 시간에 민감한 작업의 경우, TTFT와 TBT를 엄격하게 지키도록 설계함.

· 아래 그래프에서 Baseline은 TP8에서 가장 높은 GPU 주파수로 실행되는 인스턴스의 자원 풀을 이용하는 경우를 의미함

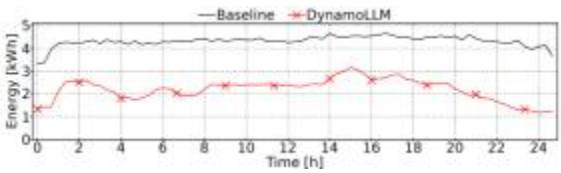


그림 439 Llama2-70B의 1일간 추론에 필요한 전력사용량 (출처 : DynamoLLM '24 논문)

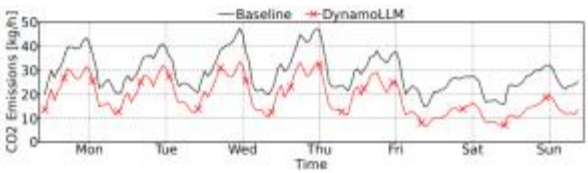


그림 440 Llama2-70B의 1일간 추론에 필요한 탄소 배출량 (출처 : DynamoLLM '24 논문)

수행기관	개발 목표	개발내용 및 범위

- LLM 추론 성능 향상을 위한 자원 재분배 오버헤드 (resharding overhead) 예측 기술
  · 아래 그림은 TP4에서 TP2와 TP8로 리샤딩하는 예를 보임. 낮은 수준의 병렬 처리에서 높은 수준의 병렬 처리로 이동하는 경우(TP4→TP8), 초기 상태(TP4)에서는 GPU 0-3이 각각 모델가중치의 4분의 1을 보유하고 있음. 최종 상태(TP8)에서는 모든 GPU가 모델가중치의 8분의 1을 보유해야 함. 따라서 처음 네 개의 GPU는 이미 필요한 상태를 가지고 있으며, 자신의 가중치 중 절반을 나머지 네 개의 GPU로 전송하기만 하면 됨. 이 네 개의 전송(GPU0→GPU4, GPU1→GPU5, ...)은 병렬로 이루어짐.

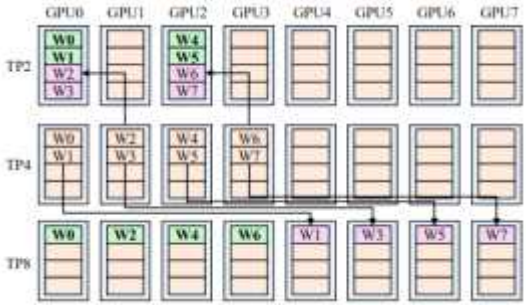


그림 441 TP4 모델을 TP2/TP8으로 re-sharding 하는 과정 예

  · 리샤딩에는 NVLink를 통해 모델가중치의 8분의 1을 전송하는 데 걸리는 시간이 필요하며, 이는 설정에 따라 약 50ms 소요됨.
  · 높은 수준의 병렬 처리에서 낮은 수준의 병렬 처리로 이동하는 경우(TP4→TP2)를 살펴보면, 최종 상태(TP2)에서는 두 개의 GPU가 각각 가중치의 절반을 보유해야 하며, 각 GPU는 처음에 가중치의 4분의 1을 보유하고 있으므로, 두 개의 GPU의 가중치를 하나로 병합해야 함. GPU1은 가중치 W2/W3을 GPU0으로 전송하고, GPU3은 가중치 W6/W7을 GPU2로 전송함. 이 두 전송은 병렬로 이루어지므로 총 리샤딩 시간은 모델가중치의 4분의 1을 전송하는 데 걸리는 시간이 되며, 이는 설정에 따라 약 100ms 소요됨.
  · 아래 표는 다양한 소스/목적지 페어에서 리샤딩 오버헤드를 보임. 일부 구성은 빠르게 다른 구성으로 전환되며(전환 시간 0), 일부 변경은 더 큰 오버헤드(전환 시간 4T, 여기서 T는 가중치의 1/8을 전송하는 데 걸리는 시간)를 발생시킴.

Src/Dst	TP2	4TP2	TP4	TP2+TP4	2TP4	TP8
TP2	0	4T	2T	2T	2T	T
4TP2	0	0	0	0	0	0
TP4	2T	2T	0	2T	2T	T
TP2+TP4	0	2T	0	0	T	T
2TP4	T	T	0	T	0	0
TP8	T	T	T	T	T	0

그림 442 모델가중치를 재분할(resharding)할 때 발생하는 전송 오버헤드

  · T는 모델가중치의 1/8을 이동하는 데 소요되는 시간 단위임. (예: NVIDIA DGX H100의 300GB/s NVLink 대역폭과 Llama2-70B 모델의 경우, T = 50ms)
  · 더 작은 텐서 병렬성(TP4→TP8)에서 더 큰 텐서 병렬성으로 확장하는 경우에는 동일한 처리량으로 요청을 계속 처리할 수 있음. 이 경우, 이전 인스턴스는 메모리 사용량을 증가시키지 않고 새로운 인스턴스로 가중치 일부를 전송함.
  · 반면, 현재 인스턴스가 더 큰 텐서 병렬성(TP8→TP4)에서 더 작은 텐서 병렬성으로 축소될 때는 낮은 처리량으로 작동함. 이전 인스턴스에서 사용된 일부 GPU는 추가 가중치를 수용해야 하므로, 새로운 요청을 처리할 수 있는 메모리 용량이 줄어들기 때문. 일반적으로, 모델가중치를 저장하는 데 필요한 GPU 메모리 사용량이 증가할수록, 들어오는 요청을 처리할 수 있는 메모리 용량 감소로 인해 처리량이 감소함.

수행기관	개발 목표	개발내용 및 범위

· 가중치가 올바른 메모리로 전송된 후에는 새로운 인스턴스를 실행하는 GPU를 동기화해야 함. vLLM과 같은 최신 추론 엔진은 이 작업을 수백 밀리 초에서 몇 초 안에 수행함. 그러나 이 기간에 인스턴스는 부하를 받을 수 없기 때문에 성능 저하가 발생함.

· DynamoLLM은 위의 성능 저하를 줄이기 위해, 새로운 인스턴스가 동기화를 진행하는 동안 이전 인스턴스가 요청을 처리하도록 허용하며, 인스턴스 재구성 여부를 결정할 때 이들의 전체적인 효율성에 미치는 영향을 고려함.

- LLM 추론에 필요한 전력사용량

· 요청된 모델의 계산 특성의 차이점은 LLM의 에너지 프로필에 직접적으로 반영됨. 아래의 표는 중간 크기의 추론 요청 (MM)을 중간 시스템 부하 (2K TPS)에서 실행할 때 다양한 LLM의 에너지 소비량을 보임.

Tensor Parallelism	TP2				TP4				TP8			
GPU Frequency (GHz)	0.8	1.2	1.6	2.0	0.8	1.2	1.6	2.0	0.8	1.2	1.6	2.0
Llama2-13B [32]	1.05	**0.99**	1.14	1.24	1.52	1.27	1.58	1.65	2.61	2.35	2.74	3.85
Mixtral-8x7B [39]	1.03	**0.98**	1.21	1.32	1.39	1.51	2.09	2.31	2.57	3.06	3.71	4.66
Llama2-70B [33]						4.23	**3.91**	4.08	5.34	4.39	4.56	5.44
Llama3-70B [34]						4.32	**4.28**	4.57	6.11	5.18	5.42	6.45
Mixtral-8x22B [38]									3.83	**3.23**	3.65	4.03
Falcon-180B [66]									9.56	**7.94**	8.57	10.34

그림 443 다양한 LLM 의 중간 크기(MM) 요청에 대한 Wh 단위의 에너지 소비량, 주파수 및 모델 병렬성, 중간 시스템 부하 (2K TPS)
(출처: DynamoLLM ‘24 논문)

· Llama2-13B와 Mixtral-8x7B와 같은 소형 모델은 어느 TP의 구성에서든지 유연하게 작동할 수 있으며, 단일 GPU에서도 실행 가능함. 이들의 최저 에너지 설정은 TP2에서 1.2GHz. 반면, Mixtral-8x22B와 Falcon180B 같은 대형 모델은 TP8에서만 실행할 수 있으며, 이들의 최저 에너지 설정은 TP8에서 1.2GHz.

· 매우 많은 파라미터를 가진 계산 중심 모델은 GPU 주파수와 모델 병렬성에 더 민감함. 따라서 높은 주파수와 높은 에너지 모드에서 작동해야 하는 경우가 많음. 비교적 적은 파라미터를 가진 희소 모델은 낮은 주파수와 낮은 모델 병렬성을 허용하며, 낮은 성능 모드에서도 성능 요구사항을 충족할 수 있음.

- LLM 추론 워크로드의 1주간 일일 변동 패턴 분석 결과

· LLM 추론 워크로드는 일반적으로 근무 시간 동안 피크를 이루고 밤이나 주말에는 낮아지는 전형적인 일일 변동 패턴을 보임. 아래 그림은 일주일 동안 두 워크로드의 초당 토큰 수 부하를 보임. 해당 부하는 각 워크로드의 최댓값에 대해 정규화되어 있음. Coding 워크로드는 매일 피크가 나타나고, 밤에는 부하가 낮아지며, 주말에는 부하가 훨씬 더 낮아지는 명확한 일일 패턴을 보임.

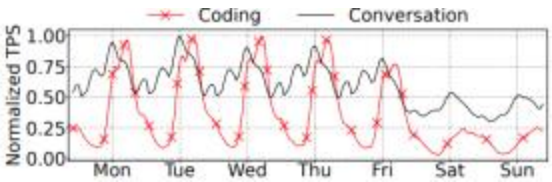
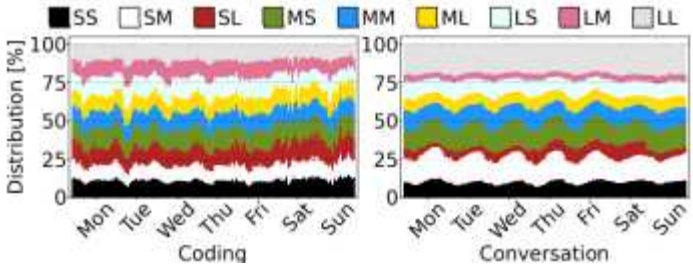


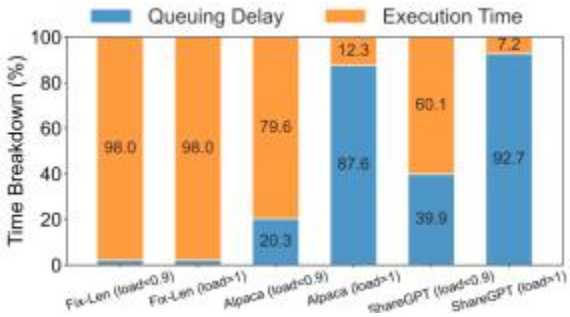
그림 444 코딩 및 대화 LLM 추론 워크로드에 대한 1주일간 워크로드 분석 결과

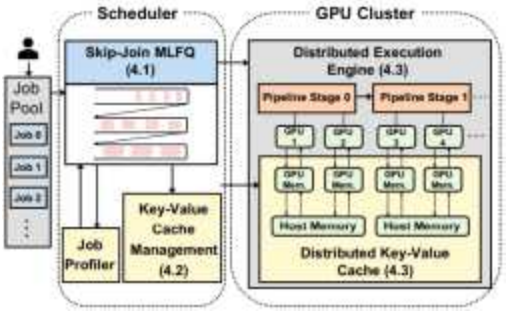
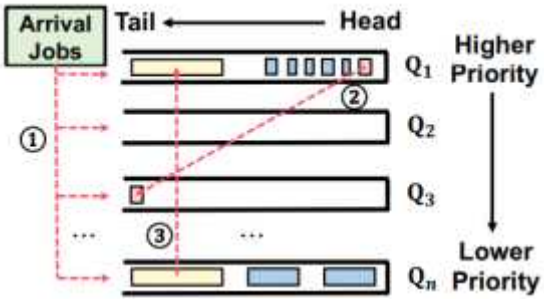
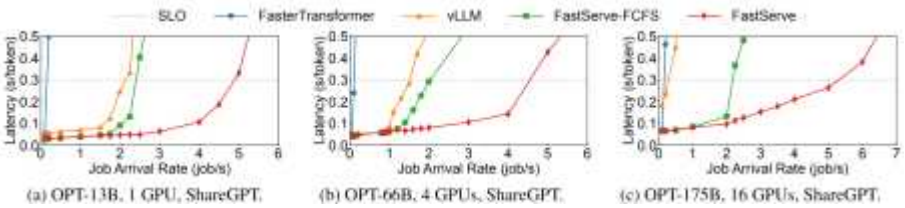
- Coding 서비스와 Conversation 서비스에 대한 워크로드 요청 분포

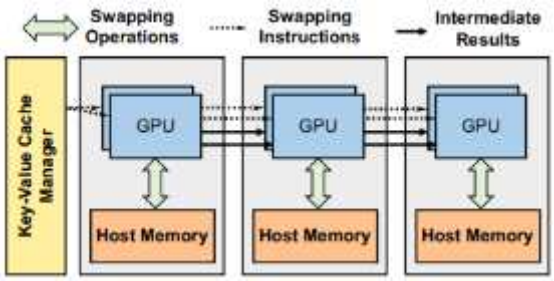
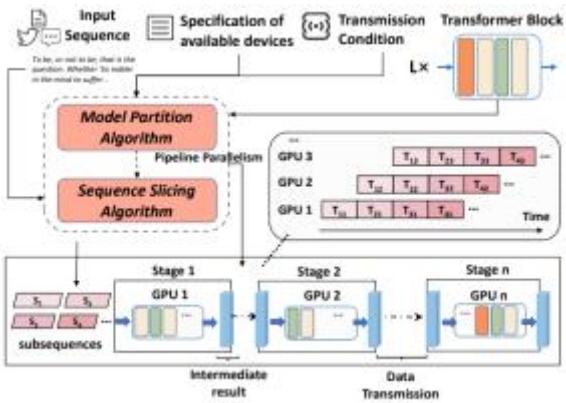
· Conversation 서비스는 보통 더 긴 출력과 짧은 입력을 가지는 반면, Coding 서비스는 그 반대의 경향을 보임.

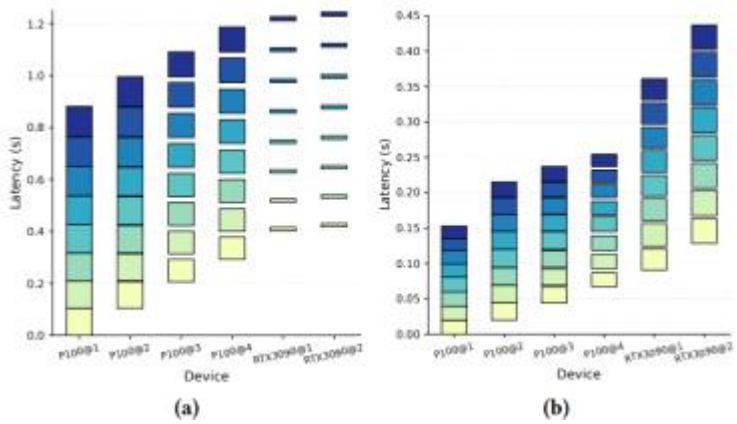
· 각기 다른 요청 유형은 최적의 에너지를 사용하기 위해 서로 다른 구성을 필요로 함. 따라서 시스템은 요청 유형별로 리소스를 분리하여 풀(pool)을 구성하고, 각 풀을 개별적으로 설정하며, 현재 요청 분포에 따라 풀의 구성을 동적으로 조정해야 함. 하지만 요청을 너무 적은 클래스로 분류하면 시스템을 세밀하게 조정하여 최적의 에너지 효율을 달성할

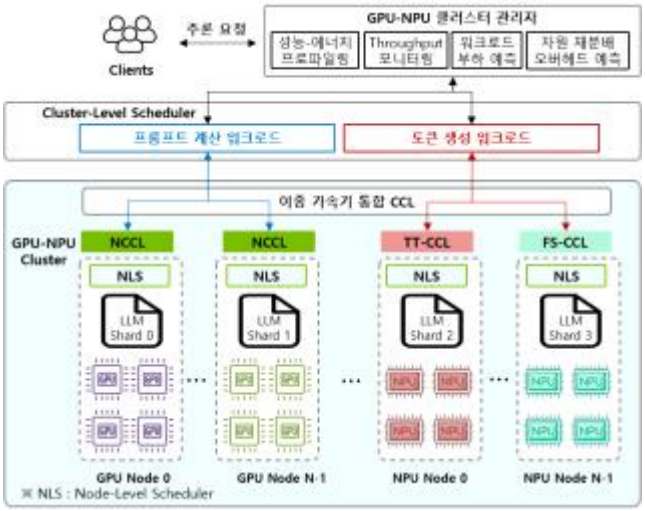
수행기관	개발 목표	개발내용 및 범위
		수 없음. 반대로 너무 많은 클래스로 분류하면 리소스가 분산(fragmentation)되어 에너지 효율이 저하될 수 있음. 따라서 시스템은 적절한 리소스 풀의 개수를 찾아야 함. · DynamoLLM에서는 과거 데이터를 사용하여 각 요청 유형이 요구하는 SLO(서비스 수준 목표) 조건(TTFT 또는 TBT 한계)과 계산 속성(계산 집약적 또는 메모리 집약적)에 따라 개별적으로 풀의 수를 설정함. 또한, 특정 요청 유형의 부하가 줄어들 경우, 단편화를 방지하기 위해 해당 풀을 더 긴 요청을 처리하는 인접한 풀과 병합함.  그림 445 LLM 추론 요청의 입출력 길이에 따른 워크로드 분포 - (라벨 의미, SS : Short Input, Short Output, SM : Short Input Medium Output, ...) - Wu, Bingyang, et al. "Fast distributed inference serving for large language models." arXiv preprint arXiv:2305.05920 (2023) 논문은 head-of-line (HOL)blocking과 추론시간이 긴 문제를 해결하고자 FastServe 시스템을 제안함. FastServe 는 LLM 추론의 자기회귀(augoregressive) 패턴을 활용해서 각 출력 토큰 단위로 선점(preemption)을 가능하게 함 · 특히, 선점형 스케쥴링 (preemptive scheduling)을 사용해서 지연시간을 최소화하며, Skip-Joing Multi-Level Feedback Queue 스케쥴러를 도입해서 LLM 추론의 semi information-agnostic 설정을 기반으로 입력 길이 정보를 활용해서 각 도착 작업에 적합한 초기 대기열을 지정함. 이후 작업이 들어간 대기열보다 높은 우선순위의 대기열은 건너뛰어 추론 효율성을 높임. · FastServe는 GPU 메모리와 호스트 메모리 간에 중간 상태를 사전 오프로드하고 업로드하는 GPU 메모리 관리 메커니즘을 설계하여 vLLM 대비 average requirements와 tail latency requirements 에 대해 최대 34.1배, 17.9 배의 성능향상을 보였음 · Head-of-line blocking(HOL blocking)이란 큐(queue) 또는 대기열에서 가장 앞에 있는 작업이 처리되지 못해 뒤에 있는 작업들도 처리되지 못하고 대기하는 현상을 의미. LLM 추론 시스템은 주로 run-to-completion 방식을 사용하는데, 이는 하나의 작업이 끝날 때까지 해당 자원을 독점적으로 할당하는 방식임. 이 경우 큐의 앞쪽에 있는 작업이 시간이 오래 걸릴 경우, 뒤에 있는 작업들이 처리되지 못하고 대기하게 되며, 한 작업이 전체 큐의 지연을 초래하게 되어 시스템의 응답 시간이 증가함. · FastServe 에서는 HOL blocking 문제를 해결하기 위해 선점 (preemption) 기술을 도입해서 작업이 완전히 끝날 때까지 기다리는 대신, 각 출력 토큰 단위로 작업을 중단하고 새로운 작업을 처리할 수 있도록 스케쥴링해서 HOL Blocking 문제를 완화해서 지연시간을 줄임 · LLM 추론은 ResNet과 같은 다른 딥 뉴럴 네트워크(DNN) 모델 추론과는 다른 고유한 특징을 가지고 있음. 일반적으로 DNN 추론 작업은 deterministic하고, 예측 가능성이 높으며, 추론 작업의 실행 시간은 주로 모델과 하드웨어에 의해 결정됨. 예를 들어, 동일한 ResNet 모델에서 동일한 GPU로 처리된 서로 다른 입력 이미지의 실행 시간은 유사한 반면, LLM 추론 작업은 자기회귀 패턴을 가지고 있기 때문에, LLM 추론 작업은 여러 반복(iteration)으로 구성되며, 각 반복에서 하나의 출력 토큰을 생성함. 생성된 출력 토큰은 다음 반복에서 입력에 추가되어 다음 출력 토큰을 생성함. 따라서 실행 시간은 입력 길이와 출력 길이에 따라 달라지며, 출력 길이를 사전에 알 수 없음. · 기존 추론 서비스 솔루션인 Clockwork, Shepherd는 ResNet과 같은 결정론적 모델 추론을 대상으로 함. 이들은 작업을 스케줄링하기 위해 정확한 실행 시간 프로파일링에 의존하

수행기관	개발 목표	개발내용 및 범위
		지만, 실행 시간이 가변적인 대규모 언어 모델(LLM) 추론에는 효과적이지 않음. · Orca 는 LLM 추론을 위해 설계된 솔루션으로, 각 iteration의 끝에서 동적으로 새로운 작업을 추가하거나 완료된 작업을 제거하는 반복 수준 스케줄링을 도입함. vLLM은 LLM 추론 작업의 중간 상태로 인해 발생하는 메모리 단편화를 줄이기 위해 PagedAttention을 추가로 도입함. 그러나 두 솔루션 모두 추론 작업을 처리하기 위해 선입선출 (FCFS)를 사용하여 한 번 작업이 스케줄링되면 작업이 완료될 때까지 실행됨. 이러한 처리 방식은 GPU 메모리의 제한과 엄격한 지연 시간 요구사항으로 인해 현재 처리 중인 배치에 임의의 수의 새 작업을 추가할 수 없으므로, 긴 작업이 새로 들어오는 작업을 차단하는 HOL blocking 문제가 발생하는 문제가 있음. 이로 인해 입력 및 출력 길이가 긴 대규모 LLM 추론 작업은 실행 시간이 길어 짧은 작업의 처리를 차단하는 문제를 유발함. · 아래의 그래프는 실제 데이터셋을 기반으로 HOL 문제를 보임. 이상적으로는 작업 간의 입력 길이와 출력 길이가 모두 동일하다면, 추론 부하가 한계 용량에 도달하더라도(부하 ≈ 1) 대기 지연이 거의 발생하지 않음. 이는 아래 그림의 첫 두 열이 시사하는바. 그러나 ShareGPT 와 Alpaca와 같은 LLM 데이터셋은 실제 작업 부하가 크게 편향되어 있음을 보여줌. 출력 길이의 롱테일 분포로 인해 긴 대기 지연이 발생함. 특히, ShareGPT의 실제 데이터셋의 경우 총 지연 시간의 최대 90%가 대기 지연임을 보여줌. 이러한 상황에서는 실행 시간을 최적화하는 것만으로는 충분하지 않음. 실행 시간은 전체 지연 시간의 작은 부분에만 기여하기 때문. 대신, 전체 지연 시간의 주요 원인인 대기 지연을 최적화해야 함.  그림 446 LLM 추론에서 Head-of-line blocking 문제 (출처 : FastServe 2023 논문) · 위 문제를 해결하고자 LLM을 위한 분산 추론 시스템인 FastServe가 제안되었음. FastServe는 LLM 추론의 자기 회귀 패턴과 반복 수준 스케줄링을 활용하여 각 출력 토큰의 생성 작업마다 GPU 선점을 가능하게 함. 구체적으로, 한 스케줄된 작업이 출력 토큰을 생성하는 것을 완료하면 FastServe는 이 작업을 계속할지 아니면 대기열의 다른 작업으로 선점할지 결정하도록 함. 이를 통해 FastServe는 선점 스케줄링을 사용하여 HOL 차단 문제를 제거하고 지연 시간을 최소화함. · FastServe의 핵심: 스킵-조인 다단계 피드백 큐(Multi-Level Feedback Queue, MLFQ) 스케줄러. MLFQ 의 특징은 다음과 같음. (1) 정보에 무지한 환경에서 지연 시간 최소화 (2) 작업은 높은 우선순위 큐에서 시작하며, 시간 초과 시 낮은 우선순위 큐로 가능. · LLM 추론의 특성 : (1) 반(半) 정보 무지적: 출력 길이는 사전에 알 수 없으나 입력 길이는 확인 가능 (2) 입력 길이가 첫 출력 토큰생성 시간을 결정하며, 이후 토큰의 실행 시간보다 큼 (3) 긴 입력/짧은 출력의 경우, 첫 출력 토큰이 전체 작업 시간에 지배적. 이 세 가지 LLM 추론의 특성을 활용해서 FastServe에서는 고전적인 MLFQ를 스킵-조인 방식으로 확장함. 각 작업은 첫 출력 토큰의 실행 시간과 큐의 시간 할당량을 비교해 적절한 큐로 배치. 높은 우선순위 큐를 건너뛰어 강등을 줄임. · 아래 그림은 FastServe의 아키텍쳐를 보임. 추론 작업들은 작업 풀에 제출됨. 스케줄러는 작업 프로파일러에서 얻은 정보를 활용하여 초기 작업 우선순위를 결정한 후, 작업을 스킵-조인 MLFQ에 배치하여 HOL Blocking 문제를 완화함.

수행기관	개발 목표	개발내용 및 범위
		· 실행 단계에서 스케줄러는 스킵-조인 MLFQ 내의 우선순위에 따라 작업을 선택하여 사전에 정의된 최대 배치 크기를 구성한 뒤, 배치를 분산 실행 엔진에 전달하여 한 번의 반복(iteration)을 수행함. 분산 실행 엔진은 분산 키-값 캐시와 협력하여 해당 작업과 관련된 키-값 텐서에 접근하고 업데이트함. 제한된 GPU 메모리 용량 문제를 해결하기 위해 키-값 캐시 관리자는 키-값 텐서를 GPU 메모리와 호스트 메모리 간에 교체함. · OPT-175B와 같은 초대형 모델을 지원하기 위해 FastServe는 텐서 병렬성과 파이프라인 병렬성을 가능하게 하는 분산 추론을 수행함.  그림 447 FastServe 아키텍쳐 · 아래 그림에서 보이듯, FastServe는 작업이 도착할 때 정확한 프로파일링을 활용하여 초기화 단계 시간(t_{init})을 예측하고, ❶ t_{init} 이상인 가장 높은 우선순위 큐(q_i)로 작업을 스킵-조인(skip-join)시킨다는 점에서 기존 MLFQ와 차별화됨. 작업이 할당된 시간(quanta)을 모두 소비하고도 완료되지 않을 경우, 스케줄러는 현재 우선순위와 다음 반복 시간에 따라 ❷ 작업의 우선순위를 강등(demote)함  그림 448 Starvation 방지를 위한 Skip-Join MLFQ · 아래 그림은 OPT 모델을 서빙할 때 LLM 추론 SOTA 도구들과 FastServe 의 성능을 비교한 그래프임. FastServe 시스템이 Job arrival rate가 증가해도 추론 지연 시간이 가장 적게 증가하는 결과를 보임.  그림 449 OPT 모델을 서빙할 때 평균 latency 비교 (출처 : FastServe 2023 논문) · 분산 추론에서의 key-value cache 관리 : key-value (kv) 캐시가 GPU 메모리의 큰 부분

수행기관	개발 목표	개발내용 및 범위
		을 차지하는 점을 고려할 때, FastServe의 kv 캐시는 여러 GPU에 걸쳐 분할되어 관리됨. LLM 추론에서 각 kv 텐서는 동일한 LLM 단계에서 사용됨. 따라서 FastServe는 텐서 병렬 처리가 요구하는 대로 kv 텐서를 분할하고, 각 키-값 텐서를 해당 GPU에 할당하여 각 GPU에서의 계산이 해당 GPU의 로컬 kv 텐서만 사용하도록 함. · LLM 추론의 각 단계가 동시에 서로 다른 작업을 처리하기 때문에, 각 단계는 독립적으로 서로 다른 kv 텐서를 업로드하거나 오프로드할 수 있음. 불필요한 제어를 줄이기 위해, 이전 단계로부터 전달된 중간 결과를 처리하기 전에 현재 단계는 이전 단계와 동일한 오프로드 또는 업로드 작업을 수행함. 중간 결과의 전송과 kv 캐시 스왑은 병렬로 수행되므로 kv 캐시 스왑의 오버헤드가 더욱 줄어듦. · 아래 그림에서 볼 수 있듯이, 중간 결과가 다음 단계로 전달될 때, 다음 단계는 스왑 명령을 수신하여 필요한 경우 kv 캐시를 동시에 스왑할 수 있음. kv 캐시 스왑 메커니즘은 첫 번째 단계의 오프로드 또는 업로드 여부만 결정하면 됨. 텐서 병렬 처리를 사용하여 첫 번째 단계를 여러 청크로 분할하는 경우, 중앙화된 kv 캐시 스왑 관리자는 첫 번째 단계의 모든 청크에 동일 작업의 kv 텐서를 오프로드하거나 업로드하도록 지시함.  그림 450 추론 중간 결과 전송 중 key-value cache 오프로딩을 오버래핑하는 과정 - (출처 : Ma, Ruilong, et al. "Poster: PipeLLM: Pipeline LLM Inference on Heterogeneous Devices with Sequence Slicing." Proceedings of the ACM SIGCOMM 2023 Conference. 2023.) · 위 논문은 추론 장치의 컴퓨팅 성능에 따라 모델을 분배하고, 입력 시퀀스를 토큰 차원으로 분할하여 계층의 병렬 실행을 가능하게 하는 PipeLLM 프레임워크를 제안함. PipeLLM은 이기종 장치를 활용한 LLM 추론 가속 가능성을 보여주며, 소규모 기업의 하드웨어 환경에서 LLM 배포를 위한 솔루션을 제공함. · 아래 그림은 PipeLLM 아키텍처에서의 추론 워크로드가 실행되는 과정을 보임.  그림 451 PipeLLM 아키텍처에서 LLM 추론 워크플로우 · LLM 병렬추론의 성능향상을 위해서는 아래의 장벽들이 해결되어야 함. (1) 희소 요청

수행기관	개발 목표	개발내용 및 범위
		(Sparse Requests)을 위한 병렬 처리 : 기존 연구들은 일반적으로 집계된 배치 데이터를 마이크로 배치로 나누어 병렬 파이프라인을 구현함. 그러나 이는 단일 데이터 인스턴스와 긴 시퀀스로 구성될 수 있는 희소 요청을 자주 처리하는 소규모 기업의 추론 환경에서는 비효율적임. (2) 제한된 전송 (Limited Transmissions) : 통신은 LLM 훈련과 추론의 주요 성능 병목 중 하나임. 연산자 수준 병렬 처리(operator-level parallelism)는 동기화를 위해 필요한 통신 오버헤드로 인해 성능 제약을 받음. NV-Link와 같은 고속 전송 기술을 도입하여 이를 해결하려는 시도는 소규모 기업에 재정적으로 부담이 됨. (3) 이기종 장치에 적합한 분배 : 가용한 모든 컴퓨팅 자원을 활용하려면 이기종 장치의 통합이 필수적임. 작업 일정은 장치 간에 균등하게 분배하기보다는 이기종 장치의 성능 차이와 서로 다른 전송 조건을 고려해서 분배해야 함. · PipeLLM에서는 위와 같은 문제를 해결하기 위해 LLM의 자회귀적 특성을 활용하여, 입력 시퀀스를 토큰 차원에서 슬라이싱하고 파이프라인화 함. 이 과정에서 중간 결과만을 전송하므로 통신 비용을 최소화함. 또한, 컴퓨팅 성능에 기반한 LLM 분배를 통해 하드웨어 불균형을 완화하며, 자원을 효율적으로 활용하도록 보장하고 사용자가 하드웨어 세부사항에 구애받지 않도록 함. · PipeLLM은 컴퓨팅 리소스를 효과적으로 통합하여 사용자가 하드웨어 차이를 고려하지 않고도 이기종 장치에서 토큰 수준 파이프라인으로 LLM 추론을 가속화 함. · 실험 결과, PipeLLM이 연산 리소스를 활용하고 시퀀스의 최적 슬라이싱을 하는데 효과적인 기술임이 확인됨. 아래 그림의 (a)는 시퀀스의 균등한 슬라이싱과 함께 LLM을 여러 장치에 균등하게 분배한 결과를 보여줌. RTX3090은 P100에 비해 실행 시간이 매우 짧은데, 이는 LLM 작업의 분배 시에 장치의 기능을 충분히 활용하지 못해 유휴 시간이 발생하기 때문. 게다가 균일한 슬라이싱 시퀀스는 후속 하위 시퀀스의 실행 시간을 길게 만들어 파이프라인의 효율성에 병목현상을 일으킴. 아래 그림의 (b)는 PipeLLM이 하위 시퀀스들을 실행하게 스케줄링하여 추론 지연 시간을 크게 줄인다는 것을 보여줌.  그림 452 PipelineLLM 이 없는 경우와 있는 경우의 LLM 추론 성능 (출처 : PipeLLM 2023 논문) - 색상이 있는 블록은 하위 시퀀스의 실행 시간, 블록 사이 간격은 통신 시간을 나타냄. - LLM 추론 성능향상을 위한 워크로드 부하 예측 기술 · 아래 그림은 추론컴퓨팅의 이질성과 작업 부하 극복을 위해 설계된 고속 LLM 추론을 위한 GPU-NPU 통합 활용 기술의 개념도임. GPU-NPU 클러스터는 GPU 노드들과 2종 이상의 NPU 노드들로 구성됨. 이때, 추론 워크로드 중 프롬프트 계산 워크로드는 compute-bound 특성을 가지므로 높은 주파수를 제공하는 GPU 서버에서 실행되어야지 전체 추론 성능을 높일 수 있음. 토큰생성 워크로드는 memory-bound 특성을 가지므로

수행기관	개발 목표	개발내용 및 범위
		상대적으로 낮은 주파수로 데이터를 처리하는 NPU 서버들에서 실행되어야지 추론 성능을 저하시키지 않으면서도 추론에 필요한 전력사용량을 절감하는데 이점이 있음. · 아래 그림의 GPU-NPU 클러스터 관리자의 성능-에너지 프로파일링 모듈은 추론 연산 시 발생하는 전력사용량과 성능을 비교하여 compute-bound 작업은 GPU로 배치하고, memory-bound 작업은 NPU로 배치하는 역할을 수행함. · 아래 그림의 Throughput 모니터링 모듈은 다수의 추론 작업들이 잘못된 스케쥴링 정책으로 인해 GPU 또는 NPU 장치가 선점되어서 스케쥴링 지연이 발생하는 문제를 해결하는 역할을 수행함. · 아래 그림의 워크로드 부하 예측 모듈은 사용자들의 추론 요청 유형 (예, 코딩 요청, 대화 요청, 요청 길이 등) 이 동적으로 변화해서 워크로드의 부하가 발생할 수 있는데 이러한 부하의 패턴을 미리 분석 및 예측해서 대응함으로써 추론 지연이 증가하는 문제를 미리 예측하고 대응하는 역할을 수행함. · 아래 그림의 자원 재분배 오버헤드 예측 모듈은 동적으로 변화하는 추론 요청으로 인해 활용하는 GPU 개수, NPU 개수가 달라질 때 모델을 새로운 장치에 로딩하고 추론 작업을 준비하는데 부하가 발행하여 추론 지연이 증가하는 문제를 미리 예측하고 대응하는 역할을 수행함. · GPU와 NPU 장치들은 각 제조사의 집합통신 라이브러리 (Collective Communication Library, CCL)를 사용해서 통신을 수행함. 서로 다른 제조사의 GPU와 NPU 기기들은 이종가속기 통합 CCL을 이용해서 서로 통신을 수행하며 LLM 분산 추론을 수행할 수 있음.  그림 453 추론컴퓨팅의 이질성과 작업 부하 극복을 위해 설계된 고속 LLM 추론을 위한 GPU-NPU 통합 활용 기술 **• NPU 기반 AI 분산/병렬컴퓨팅 구조 연구** - Tenstorrent NPU 기반 AI 분산/병렬추론 기술 분석 · Tenstorrent는 RISC-V 기반 NPU 칩을 개발하는 팹리스 업체임. 오픈소스 RISC-V 아키텍처를 기반으로 AI와 머신러닝(ML) 솔루션 개발 속도를 가속화하고 있으며, 삼성전자, 현대자동차, LG전자 등을 포함한 국내 대기업들의 투자를 받았음. Tenstorrent에서 개발한 PCIe 카드 칩인 Grayskull 및 Wormhole은 유연하며 확장 가능한 프로세서임. Wormhole 프로세서는 Tensix 코어로 구성되며, 각 코어는 로컬 스크래치 패드인 SRAM, 다섯 개의 베이비 RISC-V 코어, 다양한 데이터 포맷을 지원하는 매트릭스 및 벡터 엔진으로 구성됨. 칩 내부는 Noc를 활용하며, ethernet 프로토콜을 사용하여 워크스테이션과 서버를 위한 multi-chip mesh로의 네트워크를 형성함. 여러 개의 Wormhole들이 multi-chip mesh 형태로 구성된 TT-LoudBox 및 TT-QuietBox 시스템은 AI 모델을 실행, 테스트, 개발할 수 있음. · Tenstorrent 하드웨어 구조

수행기관	개발 목표	개발내용 및 범위

. Wormhole n300s 칩은 PCIe 가속기 카드로, 128개의 Tensix 코어, 192MB SRAM, 24GB GDDR5, ethernet, PCIe, DRAM 인터페이스로 구성됨. 이 칩은 고성능 계산을 위한 전용 코어인 Tensix 코어를 활용하여, 데이터를 빠르게 이동하고 계산을 병렬로 수행할 수 있도록 설계되었음. Wormhole n300s는 FP8 성능 기준으로 NVIDIA H100 PCIe 대비 약 6.5배 떨어지지만, 가격은 NVIDIA H100 PCIe 대비 최대 약 32배 저렴함. 또한, 에너지 효율이 뛰어나며, 특히 AI 및 ML 작업에 최적화된 성능을 보임. Tensix 코어는 Tenstorrent의 처리 유닛으로, 5개의 베이비 RISC-V 코어로 구성됨. 각 Tensix 코어는 매트릭스 및 벡터 엔진, 1.5MB의 SRAM, 두 개의 라우터를 포함하고 있으며, 각 라우터는 별도의 온칩 네트워크(NoC)와 연결됨. 이 코어는 두 개의 데이터 이동 코어와 세 개의 계산 코어를 포함함. 두 개의 데이터 이동 코어는 데이터를 다른 Tensix 코어나 DRAM으로/으로부터 데이터를 이동시키기 위해 두 개의 라우터를 제어함. 세 개의 계산 코어는 행렬연산을 수행하는 행렬 엔진과 top-k, sort, gelu, exp, sqrt와 같은 특수 함수를 연산하는 매트릭스 및 벡터 엔진을 제어함.

표 34 NVIDIA H100 PCIe와 Tenstorrent n300s의 스펙 비교

	NVIDIA H100 PCIe	Tenstorrent n300s
Price ($)	45,000	**1,399**
FP8 (TFLOPs)	**3,026**	466
FP16 (TFLOPs)	**1,513**	131
Power (W)	350	**300**
SRAM Size (MB)	80	**192**
SRAM BW (TB/s)	6	**49**

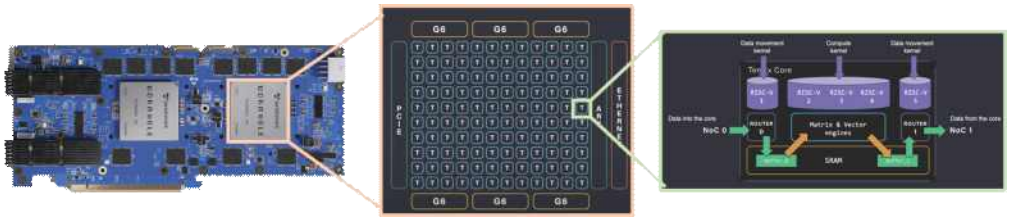


그림 454 Tenstorrent n300s의 하드웨어 구조

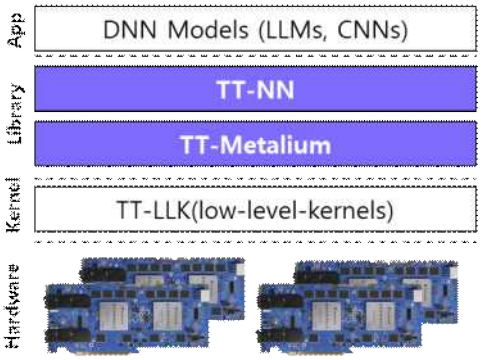
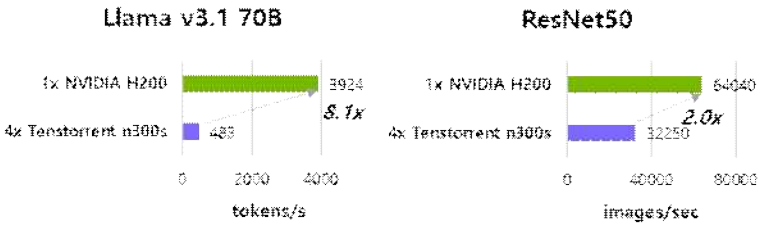
. Wormhole n300s에서 사용하는 NoC는 Tensix 코어 간의 빠르고 효율적인 데이터 이동을 지원함. NoC는 데이터 이동을 최적화하여 계산 코어와 데이터 이동 코어가 동시에 작업할 수 있도록 함. 각 Tensix 코어는 데이터 이동 코어를 통해 다른 코어와 연결되며, 이 연결은 데이터의 병렬 처리를 가능하게 함.

. Wormhole n300s는 FP8, FP16, BF16, FP32, BFP2, BFP4, BFP8, INT8, INT32, UINT8, TF32, VFP32, VTF19와 같이 다양한 데이터 포맷을 지원함. 이 시스템에서 사용되는 데이터는 주로 BF16 형식으로, 부동소수점 계산을 효율적으로 처리하기 위한 형식임. Wormhole n300s의 매트릭스 및 벡터 엔진은 BF16 형식과 같은 낮은 정밀도의 계산을 처리할 수 있으며, 이는 계산 성능을 극대화하면서도 에너지 소비를 최소화함.

· Tenstorrent 소프트웨어 스택

. TT-Buda는 Tenstorrent의 고수준 AI 프레임워크로, 머신러닝 및 인공지능 모델을 지원함. 기존 여러 개의 서로 다른 프레임워크 (PyTorch, TensorFlow 등)로부터 모델을 쉽게 포팅할 수 있으며, Python 기반의 PyBuda 라이브러리를 활용하여 TT-Buda 기능을 활용할 수 있음.

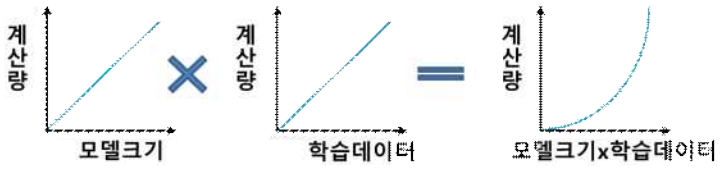
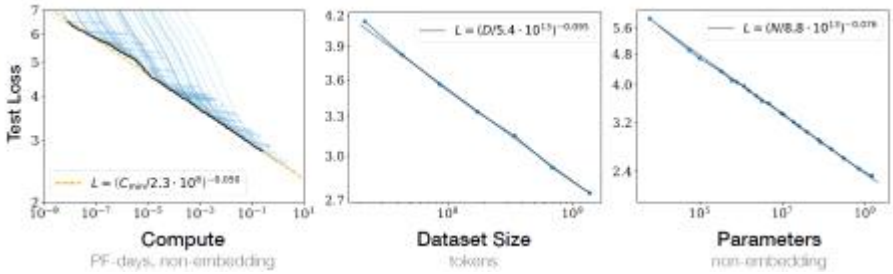
. TT-Metalium은 Tenstorrent의 C++ 기반 저수준 SDK로, 하드웨어에 직접 접근할 수 있는 API를 제공함. 이를 통해 프로그래머는 Tensix 코어 내 베이비 RISC-V 코어를 활용하여 데이터 이동, 매트릭스 및 벡터 엔진 구동, 순환 버퍼 (CB) 처리 등 다양한

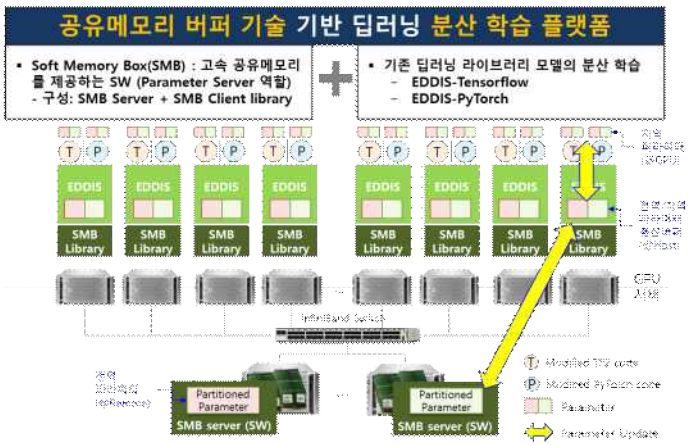
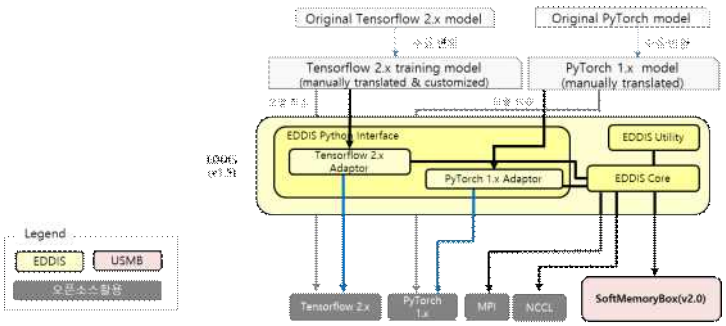
수행기관	개발 목표	개발내용 및 범위
		저수준 작업을 수행할 수 있음. TT-Metal은 Wormhole n300s를 최적화하는데 필수적인 도구로, 개발자는 이 SDK를 사용해 맞춤형 커널을 작성하고 하드웨어 성능을 최적화함. TT-Metal의 주요 특징은 베이비 RISC-V 코어 간의 통신을 위한 FIFO 기반의 순환 버퍼(CB)를 제공하여 데이터 이동과 계산을 병렬로 처리할 수 있게 함. . TT-NN은 TT-Metalium에 대한 인터페이스 역할을 하는 Python 라이브러리로, 여러 장치를 활용할 수 있는 환경에서 CCL, data parallel, tensor parallel, data parallel 등의 API를 제공하여 뉴럴 네트워크 모델을 동작시킬 수 있음. Python 기반이기 때문에 PyTorch에 익숙한 사용자가 직관적으로 사용할 수 있으며, 뉴럴 네트워크 모델은 LLM, CNN 등 DNN 모델을 동작시킬 수 있음. Wormhole n300s 4대 기준에서 LLM은 Falcon 7B (Data Parallel = 8), Falcon 40B (Tensor Parallel = 8), Llama v3.1 70B (Tensor Parallel = 8), Mixtral 7Bx8 (Tensor Parallel = 8), CNN은 ResNet-50 (224x224) (Data Parallel = 8)을 데모할 수 있는 코드를 Tenstorrent에서 제공함. Llama v3.1 70B 기준 Wormhole n300s 4대는 NVIDIA H200 1대 대비 8.1배 성능이 떨어지고, ResNet-50 (224x224) 기준 2.0배 떨어져 아직 성능 측면에서 많이 떨어짐.  그림 455 Tenstorrent 소프트웨어 스택  그림 456 NVIDIA H200 1대와 Tenstorrent n300s 4대 환경에서의 Llama v3.1 70B 및 ResNet50의 데모 성능 비교 · TT-Metal과 TT-NN을 활용한 LLM (출처: https://github.com/tenstorrent/tt-metal/blob/main/tech_reports/LLMs/llms.md) . TT-Metal과 TT-NN을 활용하여 Tenstorrent 하드웨어인 Wormhole 및 Grayskull에서 LLM을 구현하고, 최적화하는 방법을 설명함. 모듈은 임베딩, RoPE, 정규화, 어텐션으로 구성됨. 임베딩은 입력 데이터를 처리하는 초기 단계로, 텍스트 또는 토큰을 수치 벡터로 변환함. RoPE는 Q와 K 텐서에 위치 정보를 적용하는 방법으로, 트랜스포머 모델의 성능을 향상시킴. 정규화는 레이어 정규화, 분산 레이어 정규화가 있으며, 메모리 및 계산 자원을 효율적으로 사용하기 위한 방법임. 타일 기반 데이터 배치는 Row Major와 타일 크기를 기반으로 가중치를 정렬해 최적화함. 어텐션은 TT-NN에서 어텐션 연산은 커스텀 커널로 구현됨. · TT-Metal과 TT-NN을 활용한 FlashAttention

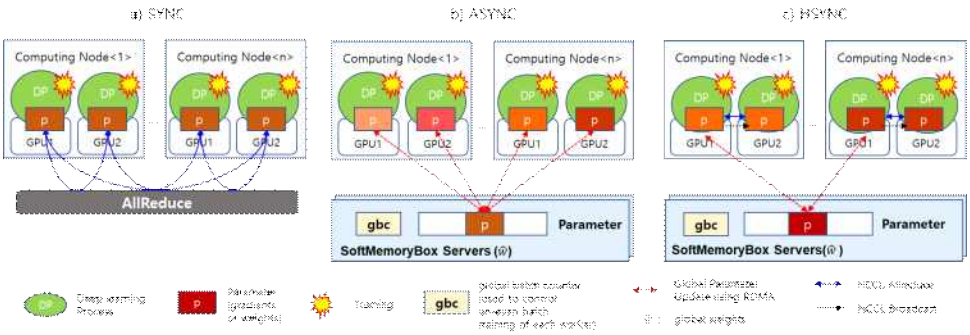
수행기관	개발 목표	개발내용 및 범위
		(출처:https://github.com/tenstorrent/tt-metal/blob/main/tech_reports/FlashAttention/FlashAttention.md) . FlashAttention 알고리즘은 GPU에서 Scaled Dot Product Attention 연산의 속도를 크게 개선함. Tenstorrent 또한 Wormhole 가속기를 대상으로 TT-Metal 소프트웨어 스택에서 FlashAttention-2 알고리즘과 FlashAttention-3의 커널 최적화 기술을 적용하여 Scaled Dot Product Attention 연산의 속도를 개선하였음. FlashAttention-2 알고리즘은 KV 데이터를 청크 단위로 온라인 Softmax 처리하며, Q를 여러 프로세서에 병렬화함. FlashAttention-3 커널 최적화는 데이터 이동 파이프라이닝 및 비동기 실행을 수행함. 이러한 구현을 통해 Wormhole의 기존 기준치를 20배 초과하는 속도를 달성하였음. FlashAttention 구현은 GitHub에서 공개되며, Llama3-70B 및 Mixtral-8x7B 같은 모델에서 사용됨. . GPU 프로그래밍 모델은 Torch Compile을 통한 자동 연산 병합과 FlashAttention 같은 수동 병합에 의존해 최적 성능을 달성함. 병합(fusion)은 커널의 산술 복잡성을 높여 글로벌 메모리 사용량을 줄이는 데 기여함. GPU는 글로벌 메모리를 통해 데이터 전달이 이루어지므로 병합이 필수적임. GPU와 비교했을 때, TT-Metal 프로그래밍 모델은 더 유연한 방식으로 설계됨. 먼저 커널의 출력을 글로벌 메모리(DRAM) 또는 L1 SRAM에 할당 가능함. 연산 그래프에서 L1 샤딩을 활용해 데이터 이동을 줄이고, DRAM 접근을 최소화함. FlashAttention 알고리즘은 기존 구현보다 더 큰 알고리즘적 이점을 제공하므로, 커스텀 병합 연산으로 구현해야 하는데, TT-Metal은 이를 가능하게 해주는 프로그래밍 환경을 제공함. TT-Metal은 2D 그리드의 Tensix 코어에서 연산(프로그램)을 병렬로 실행할 수 있음. 프로그램은 읽기(Read), 연산(Compute), 쓰기(Write) 커널로 구성되어, 각 커널은 RISC에 매핑돼 비동기로 실행됨. GPU의 블록-스레드 모델과 달리, TT-Metal은 특정 코어 그리드에 연산을 매핑하는 구조임. · TT-Metal과 TT-NN을 활용한 ViT (출처:https://github.com/tenstorrent/tt-metal/blob/main/tech_reports/ViT-TTNN/vit.md) . Vision Transformer (ViT)는 트랜스포머 모델의 셀프 어텐션 메커니즘을 비전 데이터 처리에 적용한 모델임. 이 모델은 이미지를 작은 패치로 분할하고, 이를 시퀀스로 변환해 처리함. Tenstorrent의 TT-NN 라이브러리는 이 ViT 모델을 최적화하여 구현하였으며, 비전 작업에서 트랜스포머 아키텍처의 효율성을 극대화하도록 설계되었음. . TT-NN에서는 기존의 ViT 모델과 비교해 성능을 개선하기 위해 다양한 최적화 기술을 도입함. 주요 기법은 연산의 샤딩, 행렬 곱 샤딩 변형, 트랜스포머 연산 최적화가 있음. 연산의 샤딩은 데이터를 나누어 여러 프로세싱 코어에서 병렬로 처리하는 기술임. TT-NN은 주요 연산에서 샤딩을 적용하여 데이터 이동을 최소화하고, 계산 성능을 극대화함. 이를 통해 Tensix 코어 간의 데이터 이동을 줄이고 연속적인 연산에서 효율성을 높였음. 행렬 곱(Matmul) 샤딩 변형은 세 가지 기술을 사용함. BMM(Batch Matmul) Reuse는 멀티-헤드 셀프 어텐션 모듈에서 사용되며, 입력 데이터 (in0, in1)와 출력 데이터를 높이 (height) 기준으로 샤딩함. 각 코어는 한 이미지 배치의 단일 헤드에 대한 행렬 곱을 수행함. Mcast (2D 멀티캐스트)는 블록 샤딩과 다중 전송(multi-cast)을 결합하여 QKV 생성 및 피드포워드 네트워크에서 데이터를 효율적으로 처리함. 샤딩된 데이터는 코어 그리드의 가로(y축)와 세로(x축) 방향으로 전송되어 병렬 처리를 극대화함. Mcast(1D 멀티캐스트)는 높이를 기준으로 샤딩된 데이터에서 입력(in0)은 높이(height) 샤딩, 입력(in1)은 데이터 블록이 분산된 방식으로 처리함. 이 기술은 ViT 모델에서 사용되지는 않지만, 다른 응용에서 활용될 수 있음. 트랜스포머 연산 최적화는 Q, K, V 연산을 하나의 큰 연산으로 통합하여 Tensix 코어의 계산 성능을 최대로 활용하고, 데이터 포맷 변환 및 병합을 통해 모델가중치를 사전에 처리함. 또한, GeLU 활성화 함수를 선형 연산과 결합하여 연산을 줄이고 속도를 향상함. · Tenstorrent 집합통신 (출처:https://github.com/tenstorrent/tt-metal/blob/main/tech_reports/EthernetMultichip/CclDeveloperGuide.md) . TT-NN에서 지원하는 집합통신(CCL) 연산은 여러 소프트웨어 API 계층 위에 구현됨.

수행기관	개발 목표	개발내용 및 범위
		해당 계층은 단일 및 멀티 칩 API를 모두 포함함. 이는 프로그래밍 부담을 줄이고, 테스트 및 최적화의 범위를 줄이며, 재사용성을 높이기 위해, CCL 연산은 고수준 프로그래밍 API, 특히 Erisc Data Mover(EDM)를 재사용함. 개발자는 원하는 추상화 수준에서 자유롭게 프로그래밍할 수 있음. . CCL 스택은 다음과 같은 구조를 가짐. - Ethernet Subsystem and Links : 가장 하위 레벨에 링크 및 PHY 포함 - Ethernet Transaction Command Queues I: 이 API를 통해 이더넷 트랜잭션을 시작하고 메시지 모드(예: raw 모드와 패킷 모드)를 변경할 수 있음 - Ethernet Data Flow API: 사용자가 이더넷 링크를 통해 데이터를 쓰는 작업을 시작할 수 있도록 함. 이러한 API는 NoC 비동기 API와 유사하며, 명령이 명령 버퍼에 기록되고 이후의 알 수 없는 시점에 전송 - Erisc Data Mover(EDM): 멀티칩 작업에 필요한 일반적인 기능(시작/종료 동기화, 흐름 제어, 가상 채널, 다중 버퍼링 등)을 캡슐화 - Multichip Operation Workload : 최상위 레벨에 구현됨 . TT-NN 계층은 멀티칩 환경에서 고수준의 Collective 연산(예: All-Gather)을 손쉽게 실행할 수 있도록 돕는 오케스트레이션 API를 제공함. 또한, 이 API는 사용자가 멀티칩 텐서가 분산된 메쉬 구조의 특정 방향(행 또는 열)에 따라 독립적으로 연산을 수행하도록 명령할 수 있는 추상화를 제공함. . CCL 구성요소는 아래와 같음. - ERISC Data Mover (EDM): 이더넷 링크를 통해 양방향으로 데이터를 이동시키는 멀티채널 데이터 이동 엔진 - Worker 커널 유틸리티: 워크로드 데이터를 패킷화하고 디패킷화 지원 - 호스트 데이터 구조 : 클러스터 구성(topology)을 정의하는 구조체 - 호스트 빌더 : 매개변수 전달과 커널 실행 과정을 단순화 . ERISC Data Mover(EDM)는 이더넷 링크를 통해 데이터를 전송하는 데 사용되는 재사용 가능한 데이터 이동 구성요소임. 이는 이더넷 링크 활용을 제공하는 동시에 멀티칩 Collective 연산 구현에서 발생하는 설계 문제를 처리하도록 설계되었음. EDM은 멀티채널, 양방향 데이터 전송을 지원하며, 데이터를 신호화하고 버퍼 가용성을 알리는 통신 프로토콜을 제공함. EDM은 ERISC 코어에서 실행되는 공통적이고 재사용 가능한 소프트웨어 구성 요소로 구현되었으며, 이를 통해 모든 CCL 연산에 대해 최적화 및 기능 개선이 적용될 수 있음. . EDM 통신 방식은 초기화 및 송수신 준비 과정과 실제 통신 과정으로 구분됨. 초기화를 위해 양쪽(송신과 수신)이 준비되었는지 확인하기 위한 핸드셰이크 과정을 진행함. 송신 측은 보통 이더넷 채널을 가진 쪽으로 정해지며, 두 측 모두 송신자일 경우 ID가 더 작은 채널이 송신자로 선택됨. 송신자는 더미 패킷을 보내고, 수신자는 이를 기다렸다가 응답(acknowledge)을 보냄. 응답을 받은 송신자는 정상적으로 작업이 수행되며, 핸드셰이크 주소와 채널 상태 정보는 지속적으로 유지되므로, 이전 작업에 영향을 주지 않음. . 송신자 워커와 EDM 인터페이스 : EDM은 송신자 워커에게 버퍼가 준비되었다고 세마포어를 통해 알리며, EDM은 알림 전 세마포어 값을 초기화해 경쟁 조건(race condition)을 방지함. 그 후, 송신자 워커는 데이터를 버퍼에 쓰고, 세마포어를 통해 EDM에서 알림. 워커도 마찬가지로 알림 전 세마포어를 초기화해야 경쟁 조건을 방지할 수 있음. 이러한 과정은 모든 통신이 종료될 때까지 반복됨. . 수신자 워커와 EDM 인터페이스 : EDM은 송신자로부터 메시지를 도착할 때까지 대기함. 메시지가 도착하면 수신 워커에게 버퍼가 준비되었다고 알림. 수신 워커는 데이터를 읽고, 완료되었다는 신호를 EDM에게 전달함. 마찬가지로 알림 전 세마포어 값을 초기화하여 경쟁 조건을 방지함. . 호스트에서 CCL 디바이스 구성요소(예: EDM)를 작업자와 연결하거나 초기화하는 작업을 단순화하기 위한 구성요소로는 EDM Builder, Tensor Slicer (Scheduler), Topology Config 등이 있음. EDM Builder는 EDM을 더 큰 연산 또는 작업에 통합하

수행기관	개발 목표	개발내용 및 범위
		는 호스트 코드 설정을 단순화하는 도구임. 주요 역할로는 워커와 EDM 커널 간의 인자 설정을 간소화하고, EDM 기능 추가 시 하위 호환성을 보장함. 또한, 버퍼와 메모리 관리 등을 자동화하여 메모리를 효율적으로 관리함. Tensor Slicer는 CCL 작업 구축 인프라에 새로 추가된 구성요소로, 작업량에 대한 설명을 기반으로 텐서 슬라이스 형태를 생성하는 역할을 포함함. Topology Config는 라인 또는 링과 같은 구조 정보를 표준화하여 다양한 작업에서 재사용할 수 있는 공유 데이터 구조임. 이를 통해 효율적이고 일관된 작업 구성이 가능함. . CCL 라이브러리는 다양한 멀티칩 연산(Multi-Chip Operations)을 제공함. 현재는 All-Gather와 Reduce-Scatter 등을 지원하며 향후 더 많은 연산을 추가할 예정임. All-Gather는 각 장치의 로컬 데이터를 모아 전체 데이터를 병합함. Reduce-Scatter는 데이터를 분산(Scatter)한 후 결합(Reduce) 연산을 수행함. . CCL은 MeshDevice 단위로 수행될 수 있음. MeshDevice란 2D 네트워크 형태로 연결된 물리적 장치를 추상화한 가상 장치임. Tenstorrent에서 제공하는 제품에 따라 각기 다른 네트워크 구성을 가짐. N300의 경우, 2개의 장치를 포함하며, 각각 200 Gbps의 이더넷 링크로 연결됨. T3000은 N300 카드 4개로 이루어진 2x4 네트워크를 가지며, Galaxy는 N300 카드 16개로 구성되어 8x4 네트워크 구성으로 이루어져 있음. 그림 457 서버 구성에 따른 병렬화 방식 구조(예시) . 분산 추론을 위한 병렬화 과정은 MeshDevice 단위에서 사용자의 설계 방향에 따라 다양하게 구성할 수 있음. 현재 가능한 병렬화 방식은 데이터 병렬, 텐서 병렬, 하이브리드 병렬 등이 있으며, 하이브리드는 데이터 병렬과 텐서 병렬을 결합하여 성능을 극대화하는 방법임. 예를 들어, LLAMA-3.1 70B 모델을 하이브리드 방식으로 실행하여 아래 표와 같은 시스템 확장성을 이루었음. 표 35 시스템별 병렬화(데이터, 텐서) 성능 변화 System | Batch Size | tok/s/u | tok/s T3000 | 32 | 15.1 | 483.2 Galaxy | 128 | 14.3 | 1835.5 . Tensortorrent에서 지원하는 CCL은 단일 노드 기준에서만 분산 추론이 가능하며 PyTorch와 같은 오픈 프레임워크 상에서의 호환성을 목표로 설계되었으나 현재까지는 지원하는 API의 범위에 한계가 있고 이기종 가속기와의 접목 측면에서 고려해야 하는 사항이 많음. 이를 해결하기 위해서 추가적인 구현 및 설계가 필요함. **• 데이터 중심 AI 분산 학습 SW 구조 연구**

수행기관	개발 목표	개발내용 및 범위
		- **연구 배경 및 목표** · **배경 및 필요성**  그림 458 모델 크기 및 학습 데이터의 따른 딥러닝 계산량 변화 - 고성능 딥러닝 학습을 위해 요구되는 파라미터 크기와 컴퓨팅 연산량이 점차 증가하는 추세를 보임 - 모델 크기와 학습 데이터량이 증가할수록 모델의 정확도는 향상되나, 딥러닝 계산량은 지수 증가함 - (PS + α > GPU Memory) 단일 GPU로 학습이 불가능 -> 모델 병렬 학습 (PS : 파라미터 크기, α : batch size, optimizer 등 기타 메모리 할당 요소) - (PS + α < GPU Memory) 학습 데이터 증가 -> 데이터 병렬 학습 - 딥러닝 모델의 고속학습을 위해서는 고성능 컴퓨팅 시스템(Multi-node, Multi-GPU)을 이용한 분산 학습 필요 - 모델의 파라미터 수, 훈련 데이터의 양, 그리고 계산 리소스 증가할수록 성능이 향상됨 - 큰 모델일수록 적은 데이터로 높은 성능을 달성할 수 있음 - 컴퓨팅 성능이 10억 배 증가하는 상황에서 최적의 컴퓨팅 효율을 위해 대부분의 증가분은 모델 크기를 키우는 100만 배, 데이터 증가는 1000 배 정도임  그림 459 신경망 언어 모델에서의 확장성과 성능 변화 (출처 : Scaling Laws for Neural Language Models, OpenAI,arXiv2020) - 분산 학습의 경우, 통신 병목과 이종화 가속기 문제가 지속적으로 다루어짐. - 분산 학습은 대규모 통신이 필요하여 통신 병목이 발생함. 이를 해결하는 고속 병렬 학습 기술이 필요 - 분산 학습 통신량 : (PS x NP x 2) / step [PS : 파라미터 크기, NP : 프로세스 수] - ResNet50 (PS = 25.6M(100MB), batch size = 64, step time == 200ms), NP = 16 - Bandwidth 요구량/sec = 3.2GB / 200ms = 128 Gbps - 딥러닝 학습에 사용되는 GPU의 경우 지속적으로 개선된 다양한 성능의 제품 출시로 이종가속기 파편화가 심화됨 - 기존 오픈소스 분산 학습 기술 등은 대부분 동종 GPU에 최적화되어 있음 - 이기종 분산 학습에 최적화된 기술 및 소프트웨어를 개발하는 것이 향후 대규모 딥러닝을 위해 필수적임 · **연구 목표** - 본 연구는 이기종 분산 학습에 대한 통신 병목과 이종화 문제를 해결하기 위한 기술을 설계하는 것을 목표로 함 - 분산 학습 수행 간 통신량 감소를 통한 통신 병목현상 해소 방안 연구

수행기관	개발 목표	개발내용 및 범위
		- (SoftMemoryBox) : TCP/IP의 통신 오버헤드 제거 및 데이터 통신을 다중 서버로 분산 - 이기종 가속기 학습에서의 속도 향상 및 성능 개선을 위한 하이브리드 딥러닝 파라미터 업데이트 기술 연구 - (EDDIS) : 설계된 기술의 사용성을 높이기 위해 소프트웨어 라이브러리 형태로 구현 **- 연구 내용** **· EDDIS (Efficient Distributed Deep learning lIbrary based on SoftMemoryBox)** **(1) EDDIS 딥러닝 분산 학습 플랫폼**  그림 460 EDDIS 딥러닝 분산 학습 플랫폼 구조도 - EDDIS는 SoftMemoryBox의 고속 공유 메모리 SW를 활용하여 다중 서버와 클라이언트 분산 학습을 최적화하고, 파라미터 업데이트 방식을 최적화하여 이기종한 환경에서의 학습 성능을 개선하는 플랫폼임 - SMB는 다중 파라미터 서버로, SMB 서버와 클라이언트 구조를 가짐(파라미터 서버 역할) - 사용성 향상을 위해 TensorFlow와 PyTorch 등 기존 딥러닝 라이브러리 모델의 분산 학습을 지원함 - 각 노드는 지역 파라미터를 가지고 학습을 수행하고, 학습한 파라미터 및 그레디언트를 전역 파라미터 형태로 통신하여 전체 노드의 학습한 결과를 업데이트하는 방식으로 수행 - SoftMemoryBox는 파라미터를 취합, 분배 등의 역할을 가지고, 실제 파라미터 연산과 업데이트는 EDDIS 소프트웨어상에서 이루어짐 **(2) EDDIS 구조 및 HW/SW 요구사항**  그림 461 EDDIS SW 구조도 - EDDIS 소프트웨어는 크게 Adapter, Core, SMB로 구성되어 있음 - Adapter : 분산 학습을 위해 원본 TensorFlow 또는 PyTorch 코드를 수정할 수 있는 API 제공 - EDDIS Core : SoftMemoryBox와 함께 글로벌 파라미터의 교환과 업데이트를 관리하며,

수행기관	개발 목표	개발내용 및 범위
		각 프로세스의 계산 능력에 맞춰 학습 부하 조정 - SoftMemoryBox(SMB) : 분산된 프로세스 간 파라미터 교환을 지원하고, 여러 SMB 서버와 RDMA를 사용하여 InfiniBand 네트워크에서 통신 병목 최소화 - EDDIS는 분산 파라미터 공유 버퍼를 제공하는 USMB 기반으로 동작 - USMB는 파라미터 서버(모델 정보는 가지지 않음) 역할 수행 - 기존 TensorFlow/PyTorch 모델은 매뉴얼에 따라 수동 변환 필요 - 데이터 병렬 학습 기능을 수행 (동기, 비동기, 하이브리드 파라미터 업데이트 방식 지원) - 이종 GPU 클러스터에 특화된 분산 학습 기술로써 이종 GPU 클러스터 환경에서 동기식 대비 고속학습, 이종 GPU 특화된 학습 진도 제어 등을 수행 - 분산 프로세스의 딥러닝과 전역 파라미터 업데이트(통신) 병렬화 - 연산/통신 overwrapping, 비동기 전역 파라미터 업데이트에는 Elastic Averaging SGD 기법 적용 - MPI를 이용한 멀티 노드 분산 학습 실행 및 관리 기술 보유 - HW 요구사항 : Mellanox InfiniBand 네트워크 및 NVIDIA GPU 서버 - SW 요구사항 : 코드 일부 변환 필요 - 제한사항 : FP32(+AMP)만을 지원, 파라미터 동결 방식 미구현, 단일 Optimizer 사용 **(3) EDDIS 훈련 모드** - EDDIS는 파라미터 업데이트 방식에 따른 세 가지 훈련 모드를 지원함 - SYNC 모드 : 기존 프레임워크와 동일한 구조인 synchronous 모드로, 각 노드의 훈련 파라미터를 All-Reduce 방식으로 동기화함 - ASYNC 모드 : Asynchronous 모드로, 각각의 파라미터 결과를 SoftMemoryBox에 읽고 쓰며 비동기적으로 파라미터를 업데이트함 - HSYNC 모드 : Synchronous 모드와 asynchronous 모드를 결합한 방식으로, 서버 내 통신에서는 동기식을 따르고 서버 간 통신은 비동기식 업데이트를 수행함 - HSYNC 모드는 노드 간 통신량과 학습 성능에서 모두 이점을 얻을 수 있음  그림 462 EDDIS 훈련모드 (SYNC, ASYNC, HSYNC) - 단일 프로세스상에서의 딥러닝의 경우, Foward/Bacward 학습을 수행하여 연산된 그레디언트로 가중치를 업데이트하는 구조임 - 분산 프로세스에서부터 각 노드에서 연산된 그레디언트를 평균화하는 과정이 필요 - 동기식 모드는 intra/inter 노드 대상으로 평균화(All-Reduce)한 그레디언트로 가중치를 도출하는 구조임 - 비동기식 모드는 로컬에서 가중치를 연산한 후, 글로벌 가중치를 SMB 서버를 통해 업데이트함 - 하이브리드 모드는 그레디언트 평균화를 노드 내 클러스터(subgroup)에서 수행하고, 가중치 업데이트는 각 서버 중 하나의 노드씩 대표하여 수행하는 구조임

수행기관	개발 목표	개발내용 및 범위

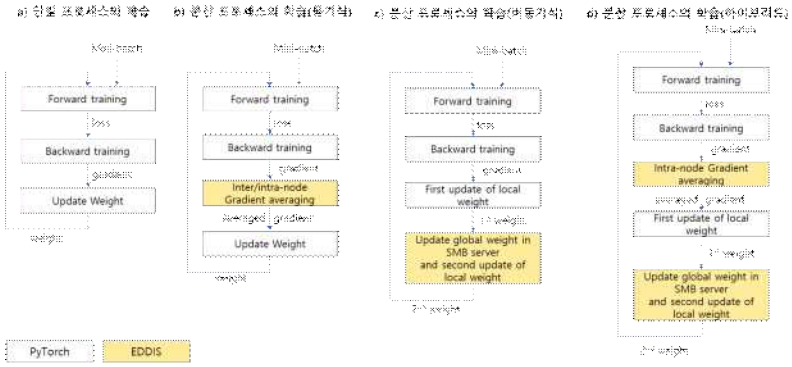


그림 463 훈련 모드별 PyTorch/EDDIS에서의 학습 단계 비교

(4) EDDIS 가중치 업데이트 알고리즘

- 분산 학습은 로컬과 글로벌 가중치 업데이트 방식에 따라 성능 변화가 큼
- SSGD 알고리즘은 SYNC 훈련 모드에서 사용되는 업데이트 알고리즘이며 나머지 알고리즘은 ASYNC와 HSYNC 훈련 모드에서 사용함
- 각 수식에서 파란색은 각 프로세스의 파라미터 업데이트 알고리즘을 나타내며, 빨간색은 글로벌 가중치에 대한 식을 나타냄
- EDDIS는 Elastic Avariging 기법을 사용하여 가중치를 업데이트하며 여기서 α는 moving rate로 로컬과 글로벌 가중치 차이를 반영하는 비율을 뜻함
- HEASGD는 각 노드의 그레디언트를 누적한 값을 전체 프로세스 수로 나누어 SSGD와 유사하게 동작하지만, 분산 프로세스 그레디언트가 반영된 파라미터 업데이트 방식 사용함

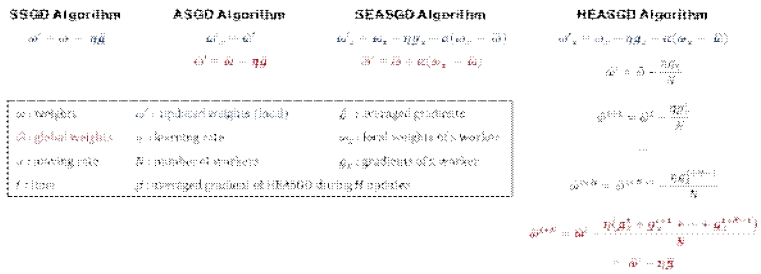


그림 464 가중치 업데이트 알고리즘 수식

(4) ASYNC/HSYNC 훈련 모드에서 Full parallel 가중치 업데이트를 위한 interaction

- EDDIS는 SoftMemoryBox를 사용해 full parallel uneven-batch 학습을 지원함
- 연산과 통신을 완전 병렬로 수행하여 글로벌 가중치를 업데이트하며, 이기종 GPU 환경에서도 각 노드가 서로 다른 배치 수로 학습을 진행할 수 있음
- SoftMemoryBox 서버는 Global Batch Counter(GBC)를 통해 InfiniBand RDMA atomic 연산으로 균일하지 않은 batch를 제어함
- SEASGD와 HEASGD 알고리즘에서는 로컬 학습과 글로벌 가중치 업데이트를 각기 다른 스레드가 관리하여 연산과 통신을 병렬화함
- HSYNC 모드에서는 같은 서브 그룹 노드 간에 대표 워커가 업데이트한 weight를 broadcast 하여 노드들이 다음 mini-batch 학습을 준비하며 이 방식은 서버 간 GPU가 이기종일 때 최적임

(4) EDDIS Optimizer

- EddisOptimizer는 제공되는 기본 Optimizer를 학습 모드에 맞게 확장하여 반환하는 wrapper 함수임
- SYNC 모드에서는 EddisOptimizer가 그레디언트 연산을 오버라이드하여 모든 분산 프로세스 간에 평균을 내는 All-Reduce 연산 수행

수행기관	개발 목표	개발내용 및 범위

- ASYNC 모드에서는 로컬과 글로벌 가중치 차이를 반영한 추가 업데이트가 적용되고, 글로벌 가중치를 갱신함
- HSYNC 모드에서는 서브 그룹 내 분산 프로세스만 All-Reduce 연산을 수행하여 그레디언트를 평균화함
- 분산 훈련이 아닌 경우에는 EddisOptimizer가 기존 Optimizer 인스턴스를 그대로 반환

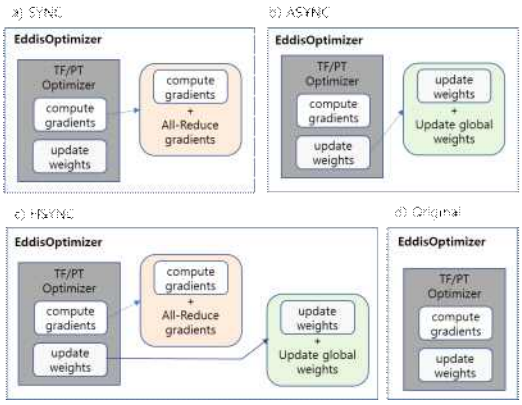


그림 465 EddisOptimizer 구조도

· 이기종 분산 딥러닝 학습 성능 평가 및 분석

(1) 실험 구성 및 환경

- 학습 모델 & 데이터셋
  . ResNet50 (Parameter : 102MB) 모델 : TensorFlow, PyTorch
  . 이미지넷(1000종, 148.5GB) 데이터셋 : 1,281,167 학습 이미지, 50,000 테스트 이미지
  . Yolov5-Medium (Parameter : 84.8MB) 모델 : PyTorch
  . Yolov5m Coco (80종, 25.8GB) 데이터셋 : 118,287 학습 이미지, 5000 검증 이미지, 20288 테스트 이미지
- 분산 학습 자원 : 16 노드 (총 64 이기종 GPU), 순차적으로 실행

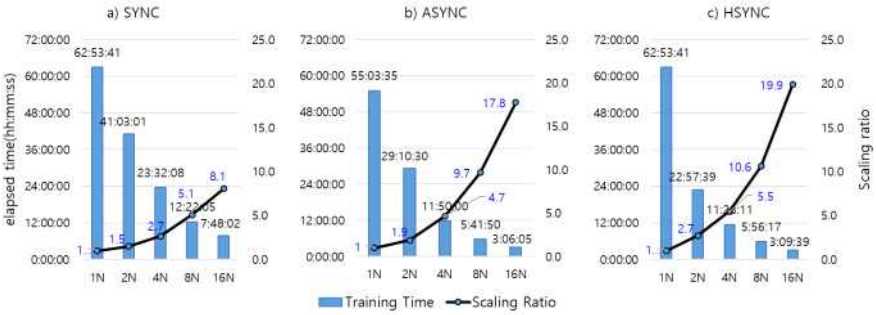
표 36 서버 CPU, Memory, Network 환경

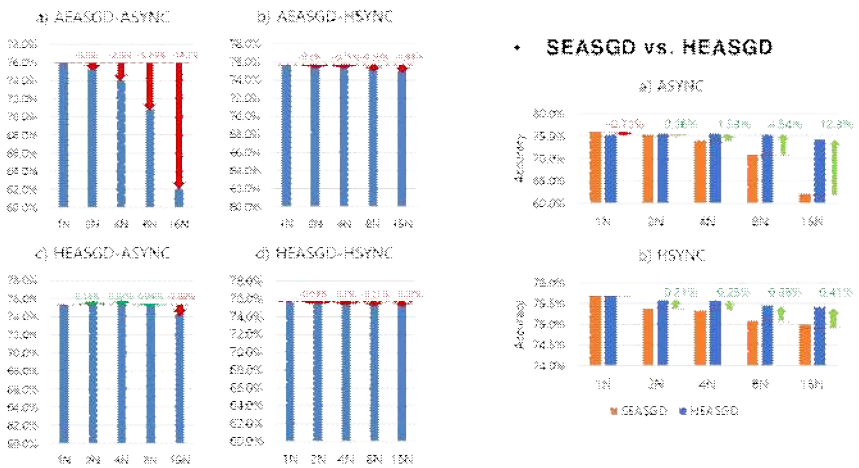
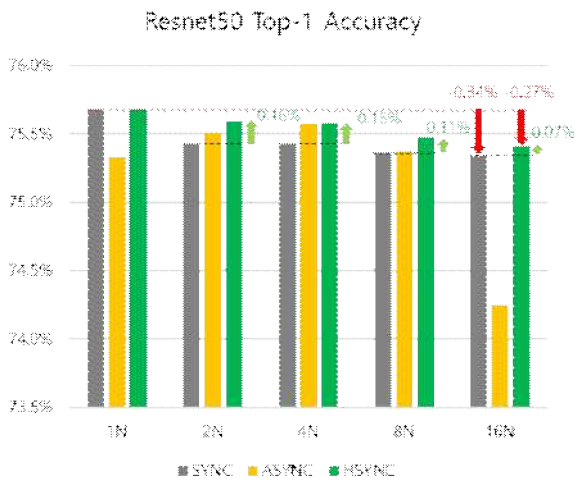
Server (#)	CPU	Memory	Network
4028GR-TRT2 (9)	2 Intel Xeon CPUs (E5-2690 v4, 14 cores, 2.3 GHz)	128 GB DDR4-2400 MHz/ECC	HDR InfiniBand HCA (200 Gbps) Mellanox QM8700 InfiniBand switch
4124GS-TNR (4)	2 AMD EPYCTM CPUs (7343, 16 cores, 3.2 GHz)	256 GB DDR4-3200 MHz/ECC	
7048GR-TR Supermictro GPU server(2)	2 Intel Xeon CPU (ES-2620 v4, 8 cores, 2.1 GHz)	128 GB DDR4-2400 MHz/ECC	
ASUS ESC 4000A	1 AMD EPYC CPU (7313P, 16 cores, 3.0GHz)	128 GB DDR4-3200 MHz/ECC	

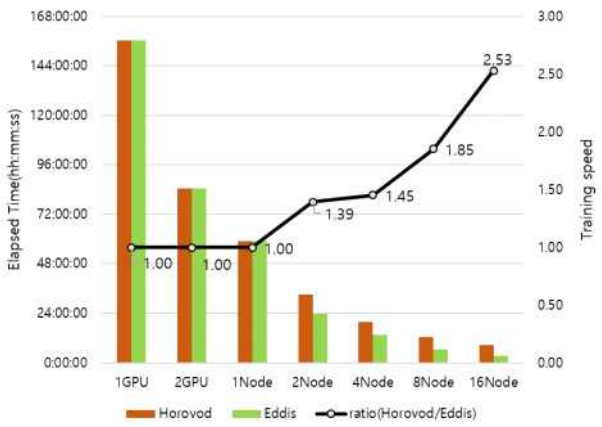
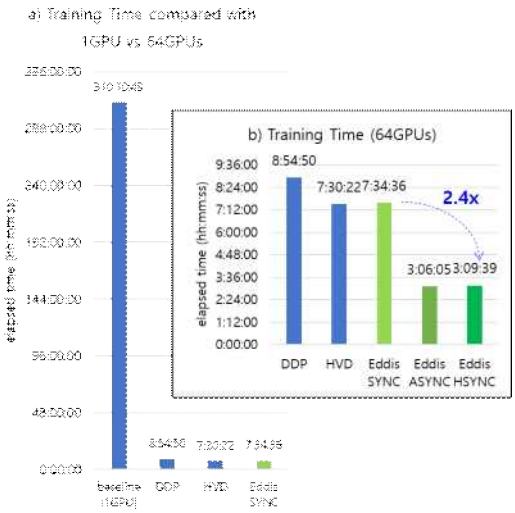
표 37 서버 GPU 환경

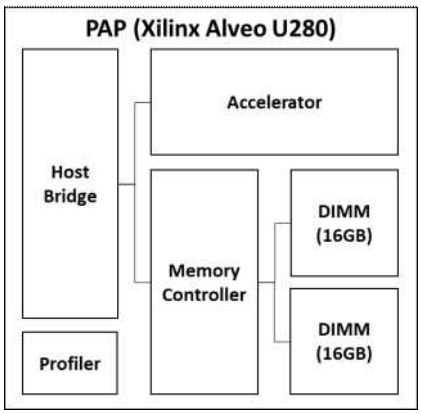
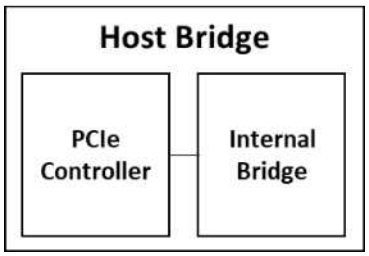
Server (#)	GPU	VRAM (MiB)	Server(#)	GPU	VRAM (MiB)
1	Titan XP	12196	9	RTX 3090	24268
2	Titan RTX	24220	10	RTX 3090	24268
3	Titan RTX	24220	11	RTX 3090	24268
4	RTX A5000	24256	12	A100	40536
5	Titan V	12066	13	A100	80994
6	P100	16280	14	RTX 3090	24268
7	RTX 2080	11019	15	RTX 3090	24268
8	RTX 2080	11019	16	Titan XP	12196

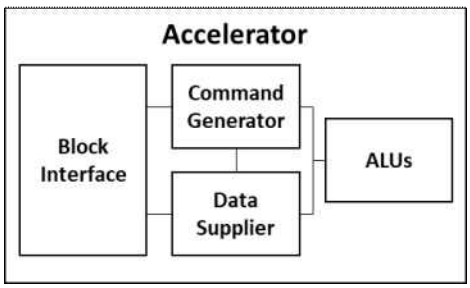
- 성능 분석을 위해 아래와 같은 비교 실험을 수행함

수행기관	개발 목표	개발내용 및 범위
		- SYNC vs. ASYNC vs. HSYNC - SEASGD vs. HEASGD - DDP vs. Horovod vs. EDDIS **(2) EDDIS 훈련 모드별 성능 분석** - 16개의 이기종 GPU 노드를 사용하여 ResNet50과 Yolo5m 모델 학습 수행 그림 466 EDDIS (TensorFlow) ResNet50 학습 소요시간  그림 467 EDDIS (PyTorch) Yolov5m 학습 소요시간 - ResNet50 기준 동기식 모드에서는 1개의 노드에서 이미지넷 학습이 59시간이 소요되었으며 16개 노드에서는 7시간으로 단축되었음. 속도 향상은 8.39배, 스케일링 효율은 약 52%로 상대적으로 낮은 성능을 보임 - 비동기식 모드는 16개 노드에서 3시간 13분 만에 완료되어 속도 향상은 20.1배, 스케일링 효율은 100%를 초과하였음 - 하이브리드 모드에서는 16개 노드에서 3시간 22분의 학습 시간을 기록하여 속도 향상 17.5배, 스케일링 효율은 109.3%를 기록하였음 - Yolov5m 기준 동기식 모드에서는 Yolov5m Coco 데이터셋 학습 시간이 1개의 노드에서 62시간 53분 소요되었으며 16개 노드에서는 7시간 48분으로 단축되었음. 속도 향상은 8.06배, GPU 이기종성으로 인한 스케일링 효율은 약 50.3%로 측정되었음 - 비동기식 모드는 16개 노드에 3시간 6분으로 단축되어 속도 향상은 17.8배, 스케일링 효율은 111%를 기록하였음 - 하이브리드 모드에서는 16개 노드에서 3시간 9분의 학습 시간을 기록하여 속도 향상 19.9배, 스케일링 효율은 124.5%를 기록. 동기식 모드보다 2배 이상의 빠른 성능을 보임 - 학습 시간 기준으로 비동기식, 하이브리드 모드에서 2배 이상의 성능향상을 검증하였음 **(3) EDDIS 파라미터 업데이트 방식 알고리즘 성능 분석** - ResNet50 모델에 대한 파라미터 및 훈련 모드별 학습 성능을 분석하였음

수행기관	개발 목표	개발내용 및 범위
		 그림 468 ASYNC, HSYNC 모드에서의 파라미터 알고리즘별 학습 성능 - SEASGD 알고리즘을 사용하였을 때, 노드 수가 증가할수록 성능이 크게 저하됨 - SEASGD 알고리즘을 비동기 학습 모드에서 사용할 경우에서 성능 저하가 가장 크게 나타났으며, 이를 해결하기 위해 도입된 하이브리드 모드를 도입함 - 하이브리드 모드에서는 학습 노드를 서브 그룹으로 나누어 전역 가중치를 비동기적으로 업데이트하는 노드 수가 줄어 성능 저하를 크게 개선할 수 있음 - 추가적인 성능향상을 위해 HEASGD 알고리즘을 개발하여 비동기식과 하이브리드 모드에 적용한 결과, ASYNC 모드에서는 SEASGD보다 12.3% 성능 개선을 보였으며 HSYNC는 0.41% 향상되었음  그림 469 학습 모드 별 HEASGD 알고리즘을 적용 결과 - SYNC, ASYNC, HSYNC 세 가지 훈련 모드에서 ResNet50의 Top-1 정확도를 비교함 - SYNC 모드에서는 SSGD 알고리즘을 사용하였으며, ASYNC와 HSYNC 모드에서는 HEASGD 알고리즘 적용 - 결과적으로 학습 노드가 증가할수록 성능이 저하되는 경향성을 보임 - SYNC 모드에서 노드 수가 1개에서 16개로 늘었을 때 0.34%의 성능 저하가 나타났음 - ASYNC 모드는 2, 4, 8 노드에서 SYNC 모드보다 더 나은 성능을 보였지만, 16 노드에서는 성능이 더 낮아짐 - 반면 HSYNC 모드는 노드 수가 증가하더라도 SYNC 모드보다 더 나은 성능을 유지함

수행기관	개발 목표	개발내용 및 범위
		(3) EDDIS, Horovod, DDP 간 성능 비교분석 - 분산 학습 플랫폼별 ImageNet 데이터셋을 사용하여 ResNet50 모델을 학습하였음  그림 470 EDDIS vs. Horovod (ResNet50) 학습 시간 - EDDIS(HSYNC) 모델은 baseline 모델에 비해 46.36배 더 빠르게 훈련을 완료했으며, 확장성 효율성은 72.43%를 기록하였음 - 또한, EDDIS는 Horovod보다 2.53배 더 빠른 성능을 보였음 - 최종 훈련 정확도는 baseline이 76.63%, EDDIS가 75.42%, Horovod가 73.4%였고, EDDIS는 Horovod보다 2% 더 나은 성능을 보임  그림 471 EDDIS vs. DDP vs. Horovod (Yolov5m) 학습 시간 - YOLOv5m 모델을 Coco 데이터셋에 대해 플랫폼별로 학습하였음 - EDDIS는 DDP보다 1시간 20분 빠른 훈련 시간을 기록했으며, **HSYNC**모드는 DDP와 Horovod보다 각각 2.8배, 2.3배 더 빠른 성능을 보임 - 훈련 성능(mAP)에서는 baseline이 0.441, DDP가 0.429, Horovod가 0.426, EDDIS가 0.431로 비슷한 성능을 나타냄 **· 연구 결과 요약** - 본 연구에서는 분산 딥러닝 학습에서 발생하는 통신 병목 문제를 해결하고, 이기종 GPU

수행기관	개발 목표	개발내용 및 범위
		를 효율적으로 활용하여 학습 속도를 크게 향상시킨 EDDIS를 제안함 - EDDIS는 SYNC, ASYNC, HSYNC세 가지 훈련 모드를 제공하여 최적의 모드를 선택할 수 있게 함 - EDDIS는 노드 확장성이 뛰어나며, SMB 메모리 서버 수를 늘려 통신 용량을 확장함 - 연산과 통신을 동시에 처리하여 동기화 시간을 크게 줄이고, SYNC, ASYNC, HSYNC모드에서 각각 8.2배, 19배, 18.7배 더 빠른 학습을 제공함 - HSYNC모드는 DDP와 Horovod보다 각각 2.8배, 2.3배 더 빠름 - 기존 SEASGD 알고리즘의 한계를 해결하기 위해 HEASGD 알고리즘을 도입하여 성능을 개선하였음 **• PCIE 가속 장치간 고성능 집합통신 구조 연구** - PAP (PCIe Accelerator PoC device) 하드웨어 구조 · PAP는 PCIe 가속 장치 하드웨어로써 PCIe add-in 카드 형태로 구현되어 호스트 CPU와 PCIe PHY/Link 계층 규격에 호환하는 신호체계로 연결됨. PAP 하드웨어는 다음과 같이 구성됨.  그림 472 PAP 하드웨어 구조 . Host Bridge는 호스트와의 PCIe를 통한 연결을 제공하며, 내부 블록들의 접근을 위한 주소 및 데이터 전달함. Host Bridge는 PCIe Controller와 Internal Bridge로 구성됨. PCIe Controller는 FPGA 벤더에서 제공하는 PCIe Core IP를 통해 구현되어, 인터페이스 브리지로써 인터페이스 간 변환 (PCIe <-> AXI) 기능, PCIe BAR (Base Address Register) 영역 제공 및 BAR 접근을 위한 address translation 기능, 인터럽트 (ex. user interrupt, MSI) 제어 기능을 제공함. Internal Bridge는 FPGA 벤더에서 제공하는 AXI Interconnect와 AXI Smartconnect IP를 통해 구현되어, PAP 내부 블록 간 연결 기능 제공, 호스트 및 Accelerator의 DIMM에 대한 접근을 제어함.  그림 473 Host Bridge 구조 . Memory Controller는 DIMM 관리 기능 및 PAP 내부 블록들의 DIMM에 대한 접근 제어 기능을 제공하는 블록임. Memory Controller는 FPGA 벤더에서 제공하는 DDR4 SDRAM Controller IP를 통해 구현되어, 호스트 또는 Accelerator로부터 전달된

수행기관	개발 목표	개발내용 및 범위
		DIMM read/write 동작 수행, DIMM refresh control 및 ECC Encoding/Decoding 기능 수행, 시스템 부팅 단계에서의 DIMM training 및 memory interleaving 기능을 제공함. . DIMM은 PAP 내부에 탑재된 DIMM 형태의 DDR4 DRAM 메모리임. . Accelerator는 FPGA의 PL (Programmable Logic) 상에서 구현되는 하드웨어 가속 기능을 제공하는 블록임. Accelerator는 Block Interface, Command Generator, Data Supplier, ALUs로 구성됨. Block Interface는 AXI 프로토콜을 통한 Host Bridge의 Internal Bridge와의 인터페이스를 제공하고, 전달받은 AXI traffic을 디코딩하여 Accelerator 내부 블록에게 전달함. Command Generator는 Block interface로부터 전달받은 명령어 패킷을 디코딩하여 Function, Data Type, Source Address, Result Address, # of Elements의 파라미터를 추출함. Function은 Accelerator에서 처리하는 operation의 type으로 SUM, MIN, MAX를 지원함. Data Type은 Accelerator에서 처리하는 데이터 type으로 integer, float을 지원함. Source Address는 Accelerator가 처리할 데이터가 저장된 DIMM의 주소임. Result Address는 Accelerator가 처리 완료한 데이터가 저장된 DIMM의 주소임. # of Elements는 parallel하게 처리하는 data vector의 수로 ALU 개수 또는 MPI rank 수와 동일하거나 작은 값이어야 함. 해당 파라미터를 기반으로 Accelraotr 내부 블록들을 제어하는 신호를 생성하고, 호스트가 Accelerator로 할당한 작업이 완료되었을 때, 호스트가 이를 인지할 수 있도록 호스트 인터럽트를 생성함. Data Supplier는 Block Interface를 통해 DIMM으로부터 전달받은 연산에 필요한 데이터를 ALU에 제공하는 기능과 ALU에서 연산이 완료된 후, 연산 결과를 Block Interface를 통해 DIMM에 저장하는 기능을 수행함. ALUs는 Command Generator 및 Data Supplier로부터 전달받은 명령 및 데이터를 바탕으로 연산을 수행함.  그림 474 Accelerator 구조 . Profiler는 PAP 하드웨어에 대한 성능 정보 모니터링 기능을 제공하는 블록임. Event를 기록하는 Event Counter Register를 통해 성능 정보 모니터링을 수행함. Event Counter Register는 Host_Rd_Req, Host_Wr_Req, Host_ACC_Req가 있음. Host_Rd_Req는 Host Bridge에서 발생한 read request의 수, Host_Wr_Req는 Host Bridge에서 발생한 write request의 수, Host_ACC_Req는 Accelerator에서 발생한 request의 수임. 모든 Event Counter Register는 PAP 하드웨어의 CSR 주소 영역에 위치하여 호스트에서 접근이 가능함. - 인터페이스 · PAP는 PCIe 프로토콜을 통해 단일 호스트가 PAP의 메모리에 접근하고, 이에 대한 읽기/쓰기 작업 수행이 가능하도록 설계되었음. 호스트를 포함하여 각 하드웨어 블록 간 인터페이스는 다음과 같이 구성됨.

수행기관	개발 목표	개발내용 및 범위

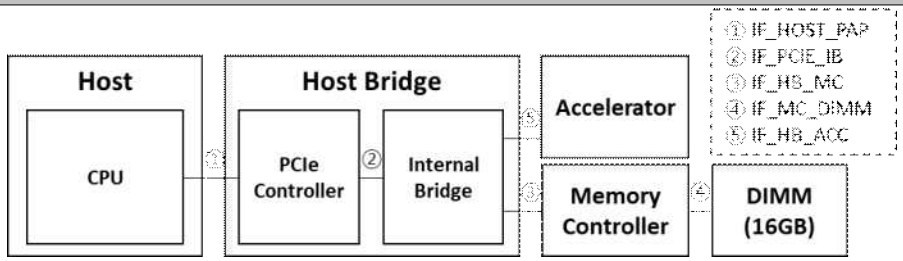


그림 475 PAP의 블록 간 인터페이스

. IF_HOST_PAP는 호스트와 PAP 간 인터페이스로 PCIe 3.0 프로토콜을 통해 이루어지며, 해당 인터페이스는 레인 당 8.0GT/s의 대역폭을 지원함. 호스트는 PAP의 PCIe Controller 내부에 위치한 endpoint 블록과 연결됨.

표 38 IF_HOST_PAP 신호 목록

Signal Name	Description
sys_clk	DDR 클럭 및 내부 시스템 클럭, 100MHz
sys_clk_gt	PCIe 참조 클럭, 100Mhz
sys_rst_n	PCIe 에지 커넥터 재설정 신호
pci_exp_rxp [15:0]	PCIe RX 인터페이스 신호, 16 lane
pci_exp_rxn [15:0]	PCIe RX 인터페이스 신호, 16 lane
pci_exp_txp [15:0]	PCIe TX 인터페이스 신호, 16 lane
pci_exp_txn [15:0]	PCIe TX 인터페이스 신호, 16 lane

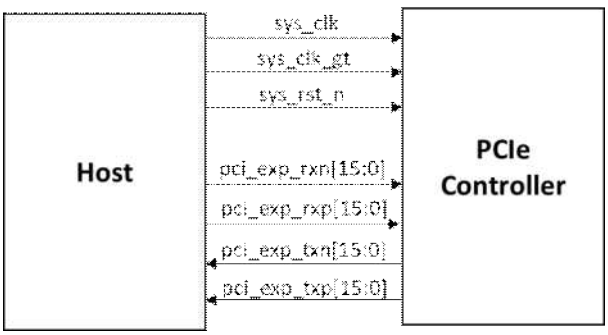


그림 476 IF_HOST_PAP 구성도

. IF_PCIE_IB는 Host Bridge 내 PCIe Controller와 Internal Bridge 간 인터페이스임. IF_PCIE_IB는 AXI4-MM 및 AXI4-Lite 프로토콜을 통해 이루어짐.

표 39 IF_PCIE_IB 신호 목록

Signal Name	Description
AXI4-MM	AXI4-MM Interface 신호들로 구성. Address 32bit, Data 512 bit 크기를 가진 고대역 버스
AXI4-Lite	AXI4-Lite Interface 신호들로 구성. Address 32bit, Data 32 bit 크기를 가지며 Master/Slave 1:1 연결
ACLK	AXI Interface 클럭은 250 MHz이며, PCIe Controller -> Internal Bridge로 제공

수행기관	개발 목표	개발내용 및 범위

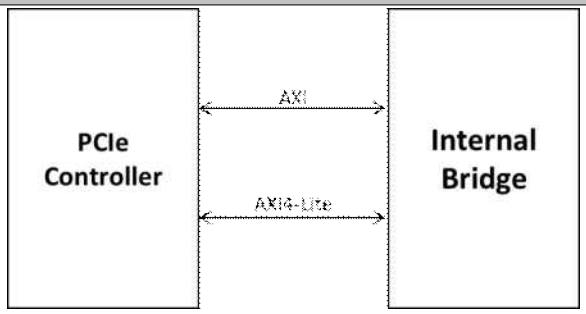


그림 477 IF_PCIE_IB 구성도

. IF_HB_MC는 Host Bridge와 Memory Controller 간 인터페이스임. IF_HB_MC는 데이터 전송 등 사용자 인터페이스를 위한 AXI4 인터페이스 및 ECC 제어 레지스터 접근 등을 위한 AXI4-Lite 인터페이스를 통해 이루어짐. AXI-MM 및 AXI-Lite 프로토콜은 IF_PCIE_IB를 참조하도록 함.

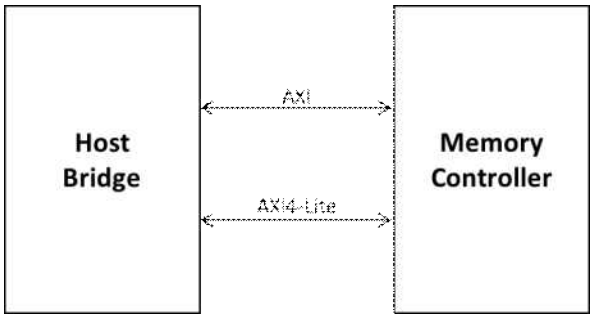


그림 478 IF_HB_MC 구성도

. IF_MC_DIMM은 Memory Controller와 DIMM 간 인터페이스임. 메모리인 DIMM은 16GB 용량의 288핀 DDR4 RDIMM으로, 최대 240MT/s의 데이터 속도로 작동 가능한 싱글 랭크 DIMM임. DIMM의 I/O는 Memory Controller의 물리 계층(physical layer)를 통해 mapping 되며, Memory Controller의 물리 계층에는 DDR4 메모리 인터페이스가 있음. 해당 물리 계층에는 FPGA 내부의 하드 블록과 DDR4 SDRAM에 인터페이스 하는 하드 블록의 최적 타이밍을 보장하는데 필요한 소프트 블록 보정 로직이 포함됨.

표 40 IF_MC_DIMM 신호 목록

Signal Name	I/O[Size]	Description
c0_ddr4_act_n	output	Register output for activate output
c0_ddr4_adr	output [16:0]	Register address output
c0_ddr4_ba	output [1:0]	Register bank select output
c0_ddr4_bg	output [1:0]	Register bank group select output
c0_ddr4_ck_c	output [0:0]	Register clock output (positive line of differential pair)
c0_ddr4_ck_t	output [0:0]	Register clock output (negative line of differential pair)
c0_ddr4_cke	output [0:0]	Register clock enable line output
c0_ddr4_cs_n	output [0:0]	DIMM rank select line output
c0_ddr4_dq	inout [71:0]	DIMM memory data bus(64b) + DIMM ECC check bits(8b)
c0_ddr4_dqs_c	inout [17:0]	Data buffer data strobes (positive line of differential pair)
c0_ddr4_dqs_t	inout [17:0]	Data buffer data strobes (negative line of

		differential pair)
c0_ddr4_odt	output [0:0]	Register on-die termination control line output
c0_ddr4_parity	output [0:0]	Register parity output
c0_ddr4_reset_n	output [0:0]	Reset output

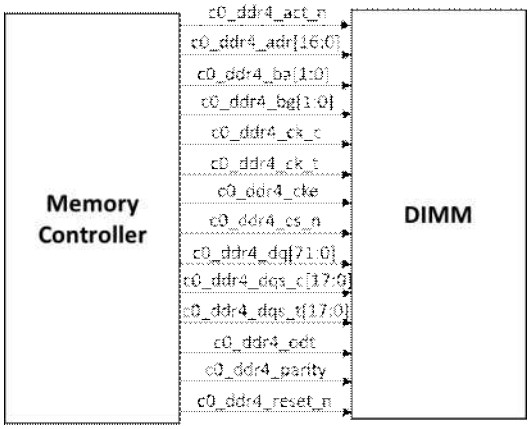


그림 479 IF_MC_DIMM 구성도

. IF_HB_ACC는 Host Bridge와 Accelerator 간 인터페이스임. IF_HB_ACC는 Accelerator 내부 연산 유닛에서 사용할 데이터 전송을 위한 AXI4 인터페이스 및 사용자 레벨에서의 Accelerator 제어 레지스터 접근을 위한 AXI4-Lite 인터페이스로 구성됨. AXI-MM 및 AXI-Lite 프로토콜은 IF_PCIE_IB를 참조하도록 함.

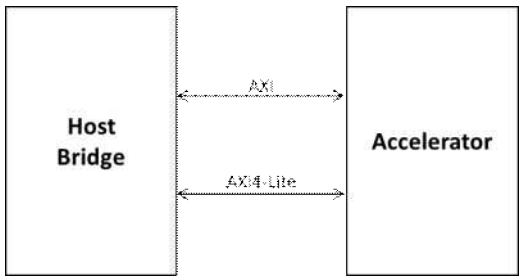
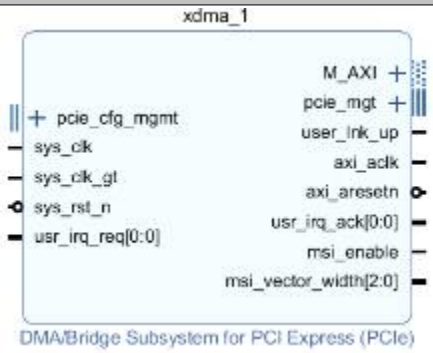
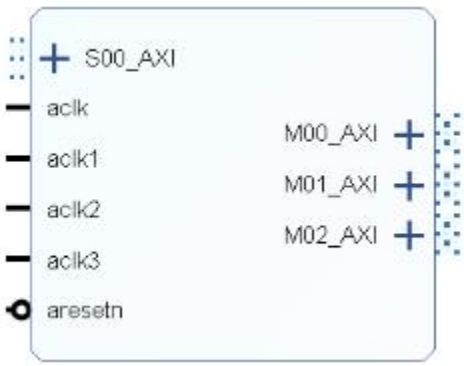


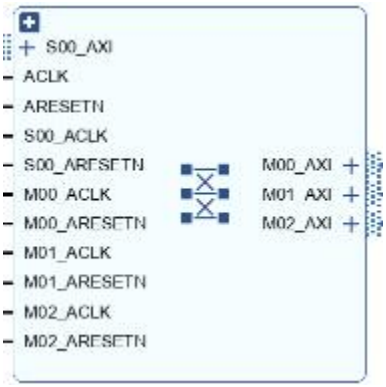
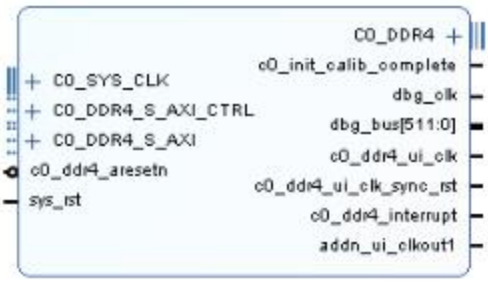
그림 480 IF_HB_ACC 구성도

- FPGA IP 구조 및 기능

· Xilinx 사의 Alveo U280 FPGA 카드를 통해 PCIe 기반 메모리 확장기능 및 집합통신 가속 기능을 제공하는 PAP 하드웨어를 설계하였음. PAP 하드웨어 개발을 위해 사용된 Xilinx Vivado 2021.1에서 제공하는 FPGA IP의 기능 및 설정 방법 및 등에 대해 설명함. 또한, Verilog HDL을 통해 자체적으로 설계한 IP들에 대해서도 설명함.

. DMA/Bridge Subsystem for PCI Express IP (이하 PCIe Core IP)는 DMA를 통한 호스트와 PAP 간 연결을 지원하는 IP임. PCIe Core IP는 PCIe 프로토콜을 통해 호스트와 인터페이스 되고, AXI4-MM을 통해 AXI SmartConnect IP와 인터페이스 되며, AXI4-Lite를 통해 AXI Interconnect IP와 인터페이스 됨. IP는 PCIe Core 및 PCIe-AXI bridge 기능을 가진 "DMA/Bridge Subsystem for PCI Express" IP (IP version 4.1)를 사용함. 주요 설정 탭은 Board, Basic, PCIe ID, PCIe: BARs, PCIe: MISC, AXI: MISC로 구성됨. Board는 IP 인터페이스에 대해 설정함. Basic은 PCIe IP의 동작 모드, 디바이스 형상, 클럭 등에 대해 설정함. PCIe ID는 Vendor ID, Device ID, class code 등에 대해 설정함. PCIe: BARs는 PCIe BAR 레지스터의 type, size, value 등에 대해 설정함. PCIe: MISC는 PCIe IP의 interrupt (User, MSI 등) 및 link 등에 대해 설정함. AXI: MISC는 AXI 관련 기타 사항에 대해 설정함.

수행기관	개발 목표	개발내용 및 범위
		 그림 481 DMA/Bridge Subsystem for PCI express IP . AXI SmartConnect IP는 AXI-MM을 통한 PAP 내부 블록 간 연결을 위해 필요한 IP임. AXI SmartConnect IP는 AXI4 MM 인터페이스를 통해 PCIe Core IP, Memory Controller IP, 그리고 Accelerator IP와 인터페이스 됨. IP는 IP Catalog에서 "AXI SmartConnect" IP (IP version 1.0)를 사용함. 주요 설정은 Standard Properties에서 AXI Master/Slave 인터페이스의 수 및 clock/reset 관련 설정, Advanced Properties에서 clock speed 및 timing parameter 등을 설정할 수 있음. 본 설계에서는 PCIe Core IP 또는 Accelerator IP로부터 2:1 MUX를 통해 연결되는 1개의 Slave 인터페이스와 각각 Memory Controller IP#1, Memory Controller IP #2, 그리고 Accelerator IP로 연결되는 3개의 Master 인터페이스로 IP를 구성하였음. 또한, AXI SmartConnect IP를 위해 PCIe Core IP 및 Accelerator IP 중 AXI-MM 버스를 선택하여 인가하기 위한 AXI4 2:1 MUX IP, AXI4 Timing을 맞추기 위한 AXI4_Retransmit IP를 연결하였음.  그림 482 AXI Smart Connect IP . AXI Interconnect IP는 AXI4-Lite를 통한 호스트의 Accelerator 블록에 대한 제어 신호 전송을 위해 사용되는 IP임. AXI SmartConnect IP는 AXI4 MM 인터페이스를 통해 PCIe Controller IP, Memory Controller IP, 그리고 Accelerator IP와 인터페이스 됨. IP는 IP Catalog에서 "AXI Interconnect" IP (IP version 2.1)를 사용함. 본 설계에서는 PCIe Core IP와 연결되는 1개의 Slave 인터페이스와 각각 Memory Controller IP #1, Memory Controller IP #2, 그리고 Accelerator IP로 연결되는 3개의 Master 인터페이스로 IP를 구성하였음. 각 인터페이스는 32bits의 width를 가짐.

수행기관	개발 목표	개발내용 및 범위
		 그림 483 AXI Interconnect IP . DDR4 SDRAM Controller IP (이하 Memory Controller IP)는 PAP 내부와 DIMM을 연결을 위해 필요한 IP임. Memory Controller IP는 AXI4-MM 인터페이스를 통해 AXI SmartConnect IP와 인터페이스 되며, AXI4-Lite 인터페이스를 통해 AXI Interconnect IP와 인터페이스 되고, DDR4 인터페이스를 통해 DIMM 블록과 인터페이스 됨. IP는 IP Catalog에서 "DDR4 SDRAM (MIG) IP" (IP version 2.2)를 사용함. 주요 설정 탭은 Board, Basic, AXI Options, Advanced Clocking, Advanced Options으로 구성됨. Board는 IP 인터페이스에 대해 설정함. Basic은 device mode, 클럭, 메모리 옵션 등에 대해 설정함. AXI Options는 AXI 인터페이스에 대해 설정함. Advanced Clocking은 클럭 관련 advanced 한 값에 대해 설정함. Advanced options은 debug, ECC, refresh 관련 advanced 한 값에 대해 설정함.  그림 484 DDR4 SDRAM Controller IP . Accelerator IP는 PAP의 서브 블록인 Accelerator의 기능을 수행하는 IP임. Accelerator IP는 AXI Slave/Master 모듈과 집합연산 가속 모듈로 구성됨. AXI-MM Slave 모듈은 Accelerator IP의 동작에 필요한 설정 파라미터를 호스트로부터 전달받아, 이를 내부 레지스터에 기록함. 본 설계에서 AXI-MM의 data width는 512bits임. AXI-MM Master 모듈은 AXI-MM Slave모듈과 AXI-Lite Slave 모듈로부터 설정 파라미터와 동작 수행 명령을 받아 동작을 개시함. 우선 AXI-MM Master Read 모듈은 DIMM으로부터 연산에 필요한 데이터를 연산 모듈인 Collective Operation Accelerator 모듈에 전달함. 이후 연산이 완료된 후 AXI-MM Master Write 모듈은 연산 모듈로부터 결과를 받아 이를 DIMM에 저장함. 또한, AXI-MM Master 모듈은 PAP-copy를 통해 DIMM에 저장된 결과 데이터를 Accelerator로 가져오며, 가져온 데이터를 AXI-MM slave 블록에서 설정된 주소 번지를 참조하여 DIMM에 복사함. 집합연산 가속 모듈은 집합연산 중 SUM/MIN/MAX를 계산하는 블록들로 구성됨. 호스트로부터 AXI-MM Master를 통해 입력되는 집합연산을 위한 데이터는 각 프로세스 별 16개의 데이터가 동시에 입력되어 한 번에 입력되는 데이터 비트는 4096bits(8 x 16 x 32bits)임. 입력 필터(Input Filter) 블록에서는 유효한 프로세스의 개수를 호스트로부터 전달받아, 유효한 프로세스의 데이터만을 masking하여 계산 블록으로 전달함. 계산 블

수행기관	개발 목표	개발내용 및 범위

록에서 지정된 연산이 완료된 후, 3:1 MUX를 통해 연산 종류 별 결과값이 저장됨. 결과값을 저장하는 FIFO 버퍼를 통해 연산의 결과값은 AXI-MM Master 블록으로 전달됨.

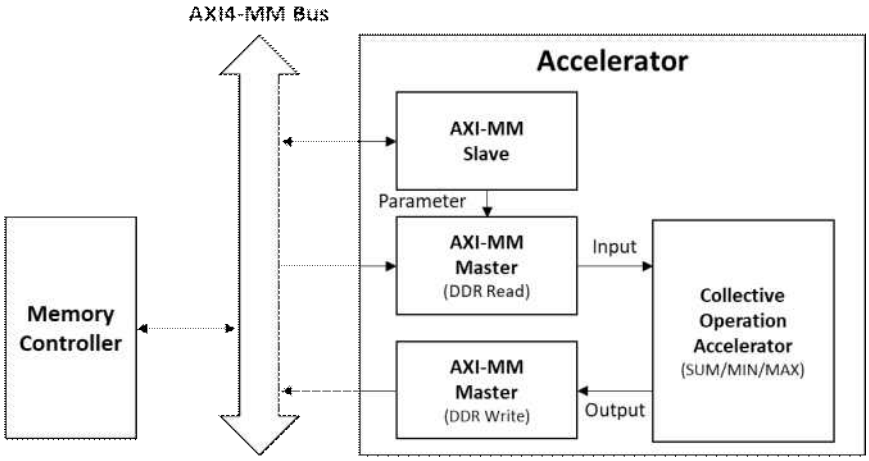


그림 485 Accelerator IP의 블록도

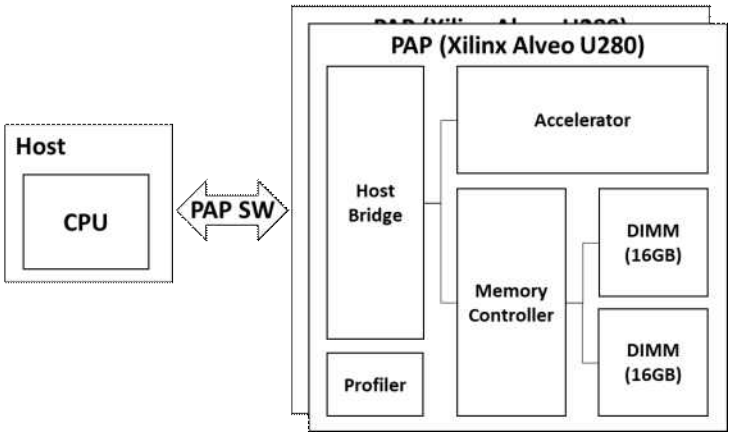
- PAP 소프트웨어 스택

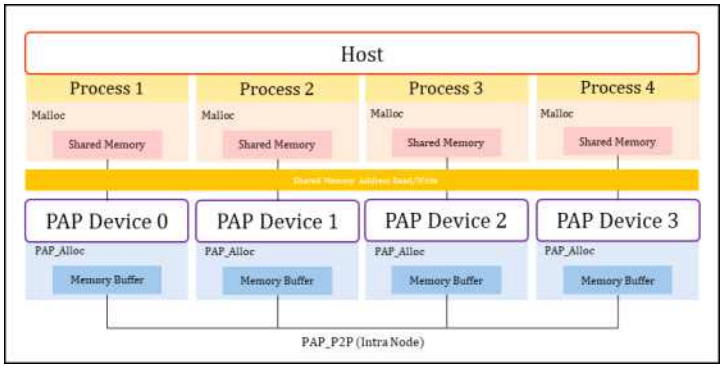
· PAP 소프트웨어는 PAP 드라이버와 PAP 라이브러리로 구성되며, PAP 내 메모리와 가속기를 활용하여 MPI 집합통신 성능을 향상시키기 위해 개발되었음. PAP 소프트웨어는 x86 서버에 Xilinx 사의 Alveo U280 FPGA 카드 2개를 탑재된 환경에서 수행되었음.

표 41 PAP 라이브러리 개발 환경 및 개발 도구

		Value
Host Server 개발 환경	CPU	Intel® Xeon® CPU E5-2620 v4 @ 2.10GHz
	Main Memory	128GB
	Operating System	Ubuntu 20.04.6 LTS
	Libraries	MVAPICH2 version 2.3.7 (for MPI)
FPGA 개발 환경	FPGA Card	AMD Xilinx® Alveo™ U280 Data Center accelerator cards - Memory: 32 GB DRAM, 8GB Host BridgeM - Logic Resources: 1,079,000 LUT 등

그림 486 Xilinx 사의 Alveo U280 FPGA 2대가 탑재된 x86 서버

수행기관	개발 목표	개발내용 및 범위
		· 호스트 CPU에서는 PAP 소프트웨어인 PAP 드라이버와 PAP 라이브러리를 이용하여 PAP와 통신을 수행함. 커널 레벨에서 동작하는 PAP 드라이버는 PAP 하드웨어에 맞추어 기존 XDMA 드라이버를 수정하였고, 장치 정보 접근, 장치 메모리 할당 및 해제, 여러 장치 간 P2P (Peer-to-Peer) 동작, P2P 내 레지스터 접근 기능을 추가 구현하였음. 사용자 영역에서 동작하는 PAP 라이브러리는 시스템 콜을 통해 PAP 드라이버를 호출하여 장치, 메모리, Accelerator 동작을 수행함.  그림 487 호스트-PAP 간 인터페이스 · PAP 하드웨어는 호스트 서버에 PCIe 인터페이스로 연결됨. 커널 영역에서는 수정된 XDMA 드라이버인 PAP 드라이버가 동작하며, cdev_sgdma.h에 추가된 ioctl 함수가 정의되어 있음. 사용자 영역에서는 PAP 라이브러리가 동작하며, 해당 라이브러리는 C언어 함수 형태로 제공됨. 라이브러리의 원형은 mex_api.h 헤더파일에 정의되어 있음. 어플리케이션은 C를 기반으로 한 MPI 라이브러리와 PAP 라이브러리를 활용하여 데이터 집약적인 특성을 갖는 MPI 응용이 동작함. 그림 488 PAP 소프트웨어 스택 . PAP 드라이버는 커널 레벨에서 동작하는 기존 XDMA 드라이버를 PAP 하드웨어에 맞추어 수정하였음. PAP 하드웨어에서 필요하지 않은 기존 XDMA 드라이버 파일이나 코드를 제거하고, 복잡성을 줄이기 위해 1개의 장치만 열어서 동작하게끔 코드를 수정하였음. 또한, ioctl 시스템 콜을 기반하여 장치 정보 접근, 장치 메모리 할당 및 해제, P2P (Peer-to-Peer), 레지스터 접근 기능을 추가 구현하였음. . PAP 라이브러리는 사용자 영역에서 동작하며, 시스템 콜을 통해 PAP 드라이버를 호출하여 하드웨어의 장치, 메모리, 가속기 동작을 수행함. 장치 관련 라이브러리는 PAP_SetDevice, PAP_GetDeviceCnt, PAP_GetDevicePCIBusId, PAP_GetDeviceBAR가 있고, open, ioctl 시스템 콜을 통해 PAP 드라이버를 호출하여 호스트가 여러 개의 PAP에 대한 각 정보를 읽어올 때 사용됨. 메모리 관련 라이브러리는 PAP_SetMem, PAP_Alloc, PAP_Free, PAP_Write, PAP_Read, PAP_P2P가 있고, iotctl 시스템 콜을

수행기관	개발 목표	개발내용 및 범위
		통해 PAP 드라이버를 호출하여 호스트가 장치 내 메모리 접근할 때 사용할 수 있게 함. 가속기 관련 라이브러리는 PAP_Reduce가 있고, ioctl 시스템 콜을 통해 PAP 드라이버를 호출하여 집합통신 가속 기능을 수행함. PAP 라이브러리 함수 모두 리턴 타입은 PAP_RETURN으로 함수 동작에 대한 결과를 반환함. PAP_RETURN의 종류는 papSuccess, papErrorInvalidDevice, papErrorInvalidDeviceFunction, papErrorInvalidValue, papErrorUnknown으로 구성됨. papSuccess는 값이 0이고, 함수가 성공적으로 실행될 때 반환됨. papErrorInvalidDevice는 값이 1이고, 요청된 PAP가 유효하지 않거나 존재하지 않을 때 반환됨. papErrorInvalidDeviceFunction은 값이 2이고, 호출한 장치 함수가 현재 장치에서 실행할 수 없는 경우 반환됨. papErrorInvalidValue는 값이 3이고, 함수에 전달된 인수 중 하나가 유효하지 않을 때 반환됨. mexErrorUnknown는 값이 999이고, 어떤 문제가 발생했는지 알 수 없을 때 반환됨. - PAP 집합통신 라이브러리 설계 . PAP 간 통신 기능의 확장성을 위해 PAP API 등을 활용한 집합통신 라이브러리 PCCL(PAP Collective Communication Library)를 설계하였음. 해당 라이브러리는 시스템 메모리와 장치 메모리 간 복사와 장치 간 메모리 복사 등을 사용하여 다중 프로세스 상에서의 메모리 통신이 이루어짐. PCCL의 메모리 구조도는 아래와 같음.  그림 489 PCCL 메모리 아키텍처 . 메모리 할당으로는 호스트와 PAP 디바이스로 이루어지며 집합통신은 PAP 디바이스 간 통신(P2P)로 동작함. 호스트에서는 공유 메모리 형태로 할당하여 디바이스 장치 정보 및 상태 정보를 각 프로세스에서 참조함. 호스트 메모리는 모든 프로세스에서 다른 프로세스 정보를 참조하여 집합통신에 필요한 메모리 전달 및 상태를 모니터링함. 저장되는 구조체는 아래 표와 같음. 공유 메모리는 이중 리스트 형태로 저장함. 첫번째 리스트는 랭크의 순서대로 내부 리스트를 생성함. 이로써 특정 랭크의 정보를 확인하는데 사용됨. 내부 리스트에는 recv_offset, send_offset, status, PCIBusId등을 기록함. Recv_offset과 send_offset은 PAP 장치 버퍼로 사용되는 송수신 데이터가 포함 된 메모리 오프셋을 가지며 이는 프로세스 수행 간 변경하지 않음. Status는 프로세스 간 synchronize한 수행을 위해 특정 step이 진행되었는지 전체 프로세스가 참조할 수 있도록 함. 이는 상태 정보에 따라 값이 바뀔 수 있음. PCIBusId는 장치 메모리를 직접 접근하기 위해 필요한 PCIe Bus ID정보르 저장함. 특정 랭크 및 프로세스에 사용되는 장치의 P2P 통신으로 사용함.

수행기관	개발 목표	개발내용 및 범위

표 42 시스템 공유 메모리 구조체

리스트 1	리스트 2	설명
rank	recv_offset	장치의 수신 데이터 오프셋 저장
	send_offset	장치의 송신 데이터 오프셋 저장
	status	장치의 상태 정보 저장
	PCIBusId	장치에 해당하는 PCIe 버스 ID 저장

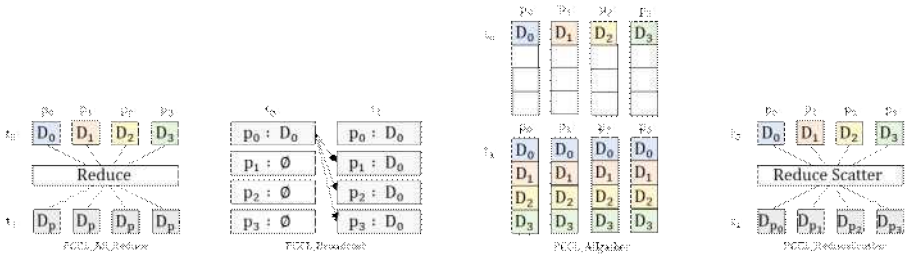


그림 490 PCCL API 통신 방식

. PCCL 라이브러리는 사용자 영역에서 동작하며, PAP API 라이브러리를 활용한 집합 통신 프로토콜을 수행함. PAP_Reduce 연산이 포함된 API로는 pccl_All_Reduce, pccl_ReduceScatter 등이 있고, P2P 통신 위주의 API는 pccl_Broadcast, pccl_Allgather가 있음. pccl_All_Reduce는 모든 프로세스에 대한 Reduce를 수행하고 그 결과를 모든 수신 버퍼에 Write하는 방식임. Pccl_ReduceScatter는 Reduce와 동일한 연산을 수행하지만, 결과는 프로세스 번호에 따라 동일한 블록으로 분산되고, 각 랭크 인덱스에 따라 데이터 chunk를 수신함. Pccl_Broadcast는 지정한 랭크에 있는 버퍼를 모든 랭크에 복사하는 방식이며, pccl_Allgather는 각각의 모든 프로세스(랭크)에 값을 집계하는 과정임. 모든 프로세스 수가 N이라 하고, 버퍼의 크기가 K라 할 때, pccl_Allgather API 이후에는 K*N 크기의 버퍼를 가짐.

. PCCL All-Reduce는 Data 크기(길이)를 Chunk 단위로 나누어 All-Reduce (P2P + Reduce) 수행함. N을 Rank(장치) 수라 하였을 때, All-Reduce가 모두 수행되기 위한 step 수는 2(N-1)임.

표 43 PCCL All-Reduce Arguments

이름	타입	설명
myrank	Int	프로세스 Rank
nprocs	Int	총 프로세스 수
*send_data	Int	각 프로세스가 All-Reduce 수행 전 가지고 있는 데이터
*recv_data	Int	P2P 통신으로 전달받을 데이터 공간
*dev_tdata	Int	All-Reduce 수행 결과가 저장될 데이터 공간
send_offset	uint64_t	Send data 를 장치에 write 할 오프셋 주소
recv_offset	uint64_t	Receive data 를 장치에 write 할 오프셋 주소
dev_bar	uint64_t	Next rank 장치의 bar 오프셋 주소
data_size	Int	데이터 크기(개수)
dtype	uint8_t	데이터 타입(float : 0, int : 1)
mode	uint8_t	Reduce 연산 모드 (sum : 1, max : 2, min : 4)

수행기관	개발 목표	개발내용 및 범위

. PCCL Broadcast는 Root Rank를 기준으로 나머지 프로세스에 대해 P2P를 수행함. N을 장치 수라 하였을 때, Broadcast가 모두 수행되기 위한 step 수는 N-1임.

표 44 PCCL Broadcast Arguments

이름	타입	설명
myrank	int	프로세스 Rank
root_rank	int	데이터를 Broadcast할 Rank
nprocs	int	총 프로세스 수
*data	int	Broadcast 할 데이터
send_offset	uint64_t	Data 를 장치에 write 할 오프셋 주소
recv_offset	uint64_t	Receive data 를 장치에 write 할 오프셋 주소
dev_bar	uint64_t	Next rank 장치의 bar 오프셋 주소
data_size	int	데이터 크기(개수)
dtype	uint8_t	데이터 타입(float : 0, int : 1)

. PCCL Allgather는 모든 프로세스 값을 취합하는 과정임. N을 Rank(장치) 수라 하였을 때, Allgather가 모두 수행되기 위한 step 수는 N-1임.

표 45 PCCL Allgather Arguments

이름	타입	설명
myrank	int	프로세스 Rank
nprocs	int	총 프로세스 수
*send_data	int	다른 프로세스로 전송할 데이터 (DATA_SIZE)
*recv_data	int	다른 프로세스로부터 수신할 데이터(DATA_SIZE * nprocs)
send_offset	uint64_t	Send data 를 장치에 write 할 오프셋 주소
recv_offset	uint64_t	Receive data 를 장치에 write 할 오프셋 주소
dev_bar	uint64_t	Next rank 장치의 bar 오프셋 주소
data_size	int	데이터 크기(개수)
dtype	uint8_t	데이터 타입(float : 0, int : 1)

. PCCL ReduceScatter는 Rank에 해당되는 index에 Reduce 결과 값을 갖도록함. N을 장치 수라 하였을 때, ReduceScatter가 모두 수행되기 위한 step 수는 N임.

수행기관	개발 목표	개발내용 및 범위

표 46 PCCL ReduceScatter Arguments

이름	타입	설명
myrank	int	프로세스 Rank
nprocs	int	총 프로세스 수
*send_data	int	각 프로세스가 Reduce 수행 전 가지고 있는 데이터
*recv_data	int	P2P 통신으로 전달받을 데이터 공간
*dev_tdata	Int	ReduceScatter 수행 결과가 저장될 데이터 공간
send_offset	uint64_t	Send data 를 장치에 write 할 오프셋 주소
recv_offset	uint64_t	Receive data 를 장치에 write 할 오프셋 주소
dev_offset	uint64_t	ReduceScatter 결과가 장치에 저장될 오프셋 주소
dev_bar	uint64_t	Next rank 장치의 bar 오프셋 주소
data_size	Int	데이터 크기(개수)
dtype	uint8_t	데이터 타입(float : 0, int : 1)
mode	Int	Reduce 연산 모드 (sum : 1, max : 2, min : 4)

. PCCL 라이브러리 설계를 통해 다중 장치에 대한 집합통신 설계를 수행하였음. 연구 결과를 통해 이기종 가속기 기반 집합통신 연구의 필요조건과 설계 과정을 구체화하였음. 이를 통해 향후 연구 방향으로 PCCL의 성능 최적화 및 다중노드 기반 집합통신 설계 등을 수행하고 이기종 간 집합통신 방안과 설계 가능성을 제시함.

• 분리 자원 노드 추상화 구조 기술 연구

- 연구 개요 및 필요성

· 지난 수십년간 고성능컴퓨팅 시스템은 고성능을 낮은 비용으로 구축하기 위해 스케일-아웃 구조를 채택하여 왔으나, 도메인 엔지니어들은 분산/병렬 프로그래밍의 어려움을 토로하여 왔음

· 최근 스케일-아웃 시스템의 기술환경 변화가 감지됨. 고속인터커넥트(e.g. 800GbE), CXL(v3.0표준화), DPU 등의 기술 성숙으로 물리 노드간 자원의 공유가 없던 구조에서 물리 노드간 다양한 물리 자원(메모리, 스토리지, 가속기 등)을 공유할 수 있는 기술적 토대가 마련되기 시작함.

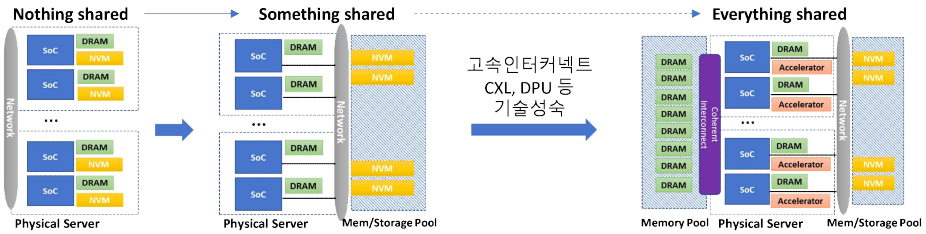


그림 491 최근 scale-out 시스템의 기술 환경 변화

· 대규모 컴퓨팅 자원을 기존 프로그램의 수정없이 스케일링하여 더 빠르게 더 큰 문제를 풀기 위해서는 다중 노드의 분리 자원(CPU, GPU, 메모리, 스토리지 등)들을 수평적으로 통합하여 고성능, 모듈러, 프로그램 용이, 확장성 우수한 고성능 가상 대형 시스템 구조 기술의 개발 필요

수행기관	개발 목표	개발내용 및 범위

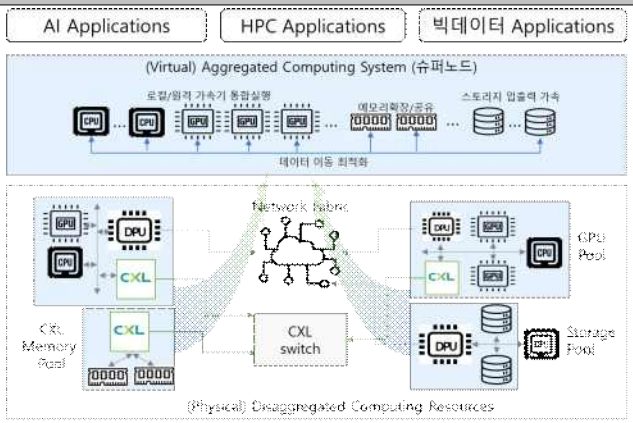


그림 492 초고성능컴퓨팅 슈퍼노드 제공을 위한 패브릭 컴퓨팅 아키텍처 개념도

- 연구 목표

· 본 연구는 이종 노드의 분리 자원들을 가상으로 통합하여 하나의 시스템 이미지를 구성하고, 단일 시스템 OS를 통해 도메인 전문가들에게 초고성능컴퓨팅을 제공하는 기술 개발을 목표로 함

· OpenMP로 작성된 Legacy HPC 응용 프로그램을 별도의 코드 수정 없이 더 빠르게 실행 가능하며, 더 큰 공유 메모리를 이용해서 더 큰 문제를 풀 수 있는 슈퍼 스케일링 가능한 기술 개발을 목표로 함

- 연구 내용

· 분리자원 노드 추상화 관련 기술 분석

(1) GiantVM 기술

- 기술개요

. GiantVM은 여러 물리적 기계로부터 자원을 집적하고 게스트 OS에 대한 통일된 하드웨어 추상화를 제공하는 다대일(many-to-one) 가상화를 제공하는 오픈소스 분산 하이퍼바이저임

. Linux와 같은 수정되지 않은 일반 게스트 OS가 실행될 수 있는 컴퓨팅, 메모리, 스토리지, I/O 리소스를 통합하기 위한 최첨단 타입-II 하이퍼바이저인 QEMU-KVM을 기반으로 사용하기 쉬운 SSI(Single-system Image) 플랫폼을 구축

- 타입-II 하이퍼바이저는 호스트 OS에서 RDMA, GPU 및 기타 이기종 디바이스를 포함한 HW 드라이버를 지원하고, 최신 클라우드 컴퓨팅 인프라에 배포 편의성을 가짐. 호스트 OS에서 기능을 구현하는 방식보다 덜 복잡하고 오류 발생이 적음

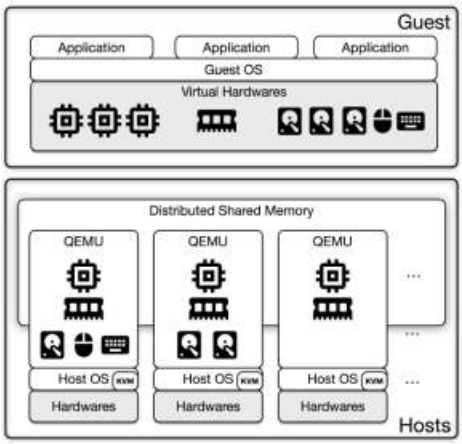
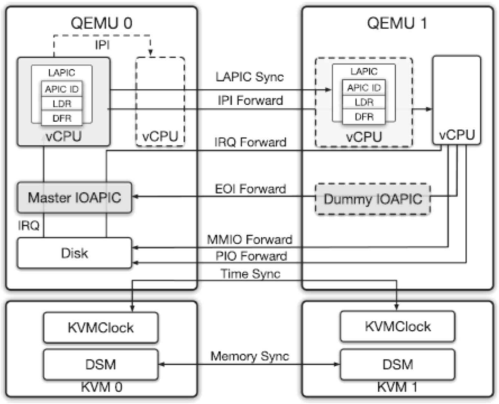
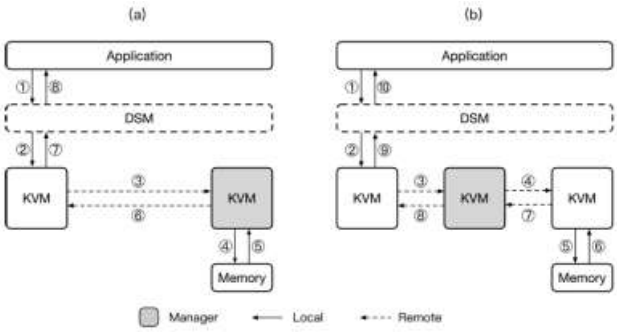
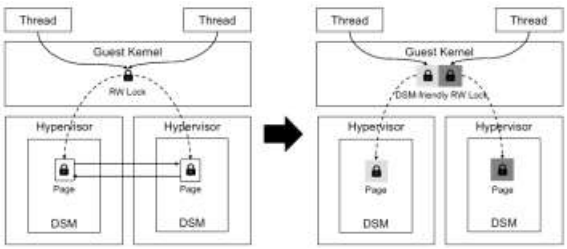
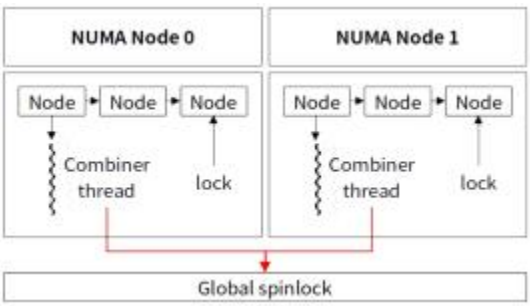
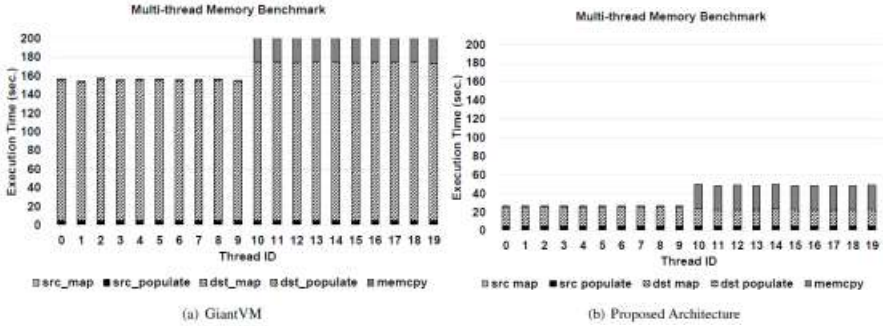
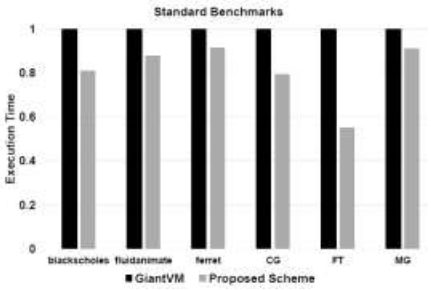
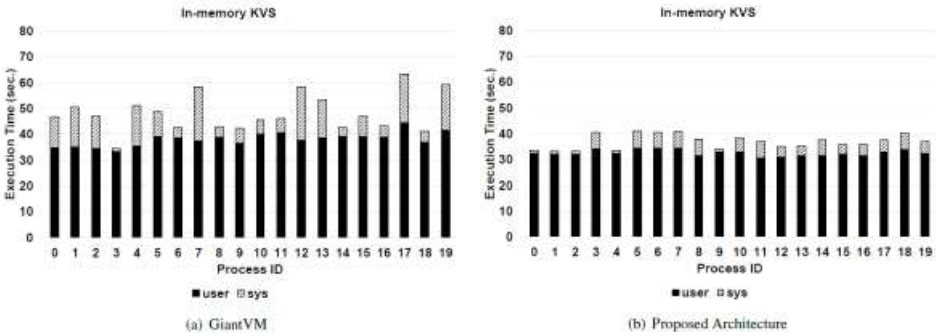


그림 493 Architecture of GiantVM

수행기관	개발 목표	개발내용 및 범위
		- 기술구현 . QEMU 내에서 원격 vCPU와 I/O 가상화를 구현하였고, KVM 내에서 메모리 가상화를 구현하였음 . 하이퍼바이저 계층에서 SSI를 구현하여 통합된 가상화 ISA 인터페이스를 효과적으로 제공하고, x86 플랫폼을 지원하는 모든 OS와 해당 OS용으로 구축된 애플리케이션을 지원 . IPI 포워딩, 인터럽트 포워딩, I/O 포워딩, DSM 관련 최적화 기술 등 여러 머신에서 CPU, 메모리, I/O 장치를 공유하는 데 따르는 어려움을 극복하기 위한 다양한 최적화 기술을 적용 . 각 node 별 자원(CPU, memory)은 NUMA node 형태로 보여짐 . 네트워크 전송을 가속화하기 위해서 Infiniband 기반의 RDMA를 사용함 . CPU 가상화를 위해서 하이퍼바이저가 local과 remote CPU를 구분함. local vCPU는 평소와 같이 취급하고 remote vCPU는 쓰레드와 data structure 초기화만 수행함. IPI(Inter Processor Interrupt) 포워딩 메카니즘을 통해서, IPI가 remote vCPU를 대상으로 하면 remote 하이퍼바이저로 전송하여 remote vCPU의 APIC에 주입  그림 494 Inside of GiantVM . 분산 공유 메모리(DSM)를 활용하여 메모리 가상화를 달성함. 메모리 가상화는 IVY DSM을 기반으로 함.  그림 495 GiantVM에서 DSM의 동작도 . I/O 가상화를 위해서 device로부터 생성된 interrupt를 hypervisor가 intercept하여 target vCPU로 forward하고, vCPU가 I/O request 생성하면 hypervisor가 intercept 하여 master node로 보냄 - 성능 개선을 위한 최적화 기술 . DSM-aware scheduler (DaS)를 통해서 빈번하게 페이지를 공유하는 쓰레드를 동일한 NUMA 노드에 할당함. DSM matrix를 참조하여 thread clique를 생성하고 각 clique를 NUMA 노드에 스케줄링. 노드 간 페이지 공유 최소화로 GiantVM의 네트워크 오버헤드 감소시킴

수행기관	개발 목표	개발내용 및 범위
		. Load-aware scheduler (LaS)를 통해서 LC(latency-critical) task의 CPU 요구사항이 낮은 경우 BE(Best-effort) task를 LC task와 동일한 노드에 배치. LC task의 CPU 사용량이 폭증하면 BE task를 부하가 적은 노드에 스케줄링하여 LC task의 QoS 보장 (2) MemOS 기술 - 기술 개요 . 물리적으로 분산된 자원을 단일 시스템으로 통합하는 확장 가능한 단일 OS NUMA 추상화 아키텍처를 제안함. 기존의 스케일 업 및 스케일 아웃 시스템이 가진 높은 비용과 성능 최적화 문제를 해결하기 위한 기술 개발. . 기술 구현은 GiantVM의 extension으로 구현하고, GiantVM의 성능을 개선하기 위해서 hypervisor level의 최적화와 Guest OS level의 최적화 기법을 적용함 - 구조 . 다수의 물리적 노드를 단일 가상 머신(VM)으로 통합하는 하이퍼바이저 기반 시스템 설계를 제안 . CPU 및 메모리와 같은 분산된 하드웨어 자원을 NUMA 노드로 추상화하여, 기존 운영 체제(OS)와 호환성을 유지 - Hypervisor level 최적화 . 메모리 관리 및 vCPU 할당을 NUMA 환경에 최적화 . clear_page()와 RDMA(원격 직접 메모리 접근) 통신을 개선하여 메모리 연산 성능 향상 - Guest OS 최적화 . DSM(분산 공유 메모리)에 적합한 독자적인 동기화 (DSM-friendly RW lock) 메커니즘 개발  그림 496 DSM-friendly RW Synchronization . 기존 spinlock을 대체하는 NUMA-aware CC-lock을 통해 성능 저하를 방지  그림 497 NUMA-aware CC-lock . 커널 데이터의 정렬(alignment)을 최적화하여 불필요한 메모리 접근 비용 감소 - 성능평가

수행기관	개발 목표	개발내용 및 범위
		 그림 498 Memory-intensive Benchmark Results . 메모리 집약적 작업에서 기존 기술(GiantVM) 대비 최대 4.74배의 성능 향상  그림 499 Performance with Standard Benchmarks . 표준 병렬 벤치마크(PARSEC, NAS)에서 평균 19% 성능 향상  그림 500 In-memory KVS Benchmark Results . 멀티스레드 및 인메모리(in-memory) key-value store 워크로드에서 성능 개선 확인 **· 가상화 환경에서 NUMA Linux scalability 기술 분석** (1) 가상화 환경에서 리눅스 scalability 분석 - 4.1.16 커널은 30코어에서 성능 저하 - 4.9.26 커널은 45-60코어에서 성능 저하 - 성능 저하의 원인 분석 . 가상화 환경에서 spinlock 동작의 문제 * LHP(lock holder preemption), LWP(lock waiter preemption), SSA(sleepy spinlock anomaly) 문제 * 잦은 VM halt exit 문제

수행기관	개발 목표	개발내용 및 범위

그림 501 가상화 환경에서 리눅스 scalability 분석

- 4.1.16 커널의 성능 저하 해결을 위한 Opportunistic spinlock 알고리즘 분석
- 4.9.26 커널의 성능 저하 해결을 위한 qspin lock 알고리즘 분석

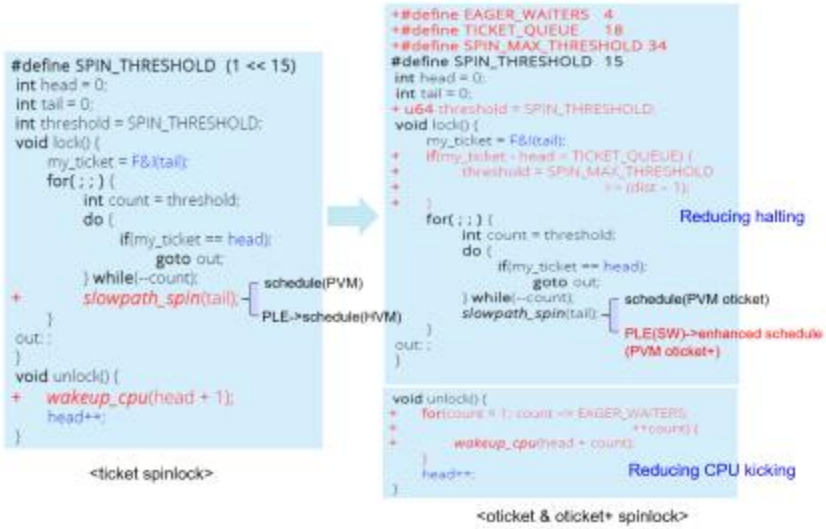


그림 502 Opportunistic spinlock 알고리즘

(2) VM halt exit 문제 분석

- Crosvm@ChromeOS, 개선방법인 KVM halt polling 최적화 기법 제시
- TidalScale, 머신러닝으로 문제 해결

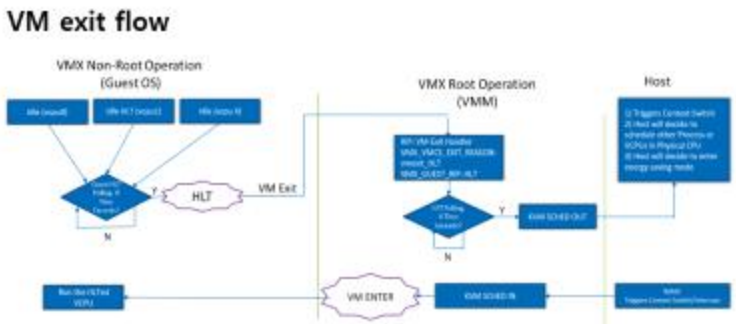
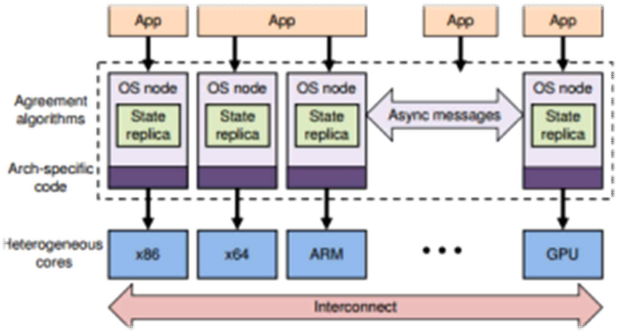
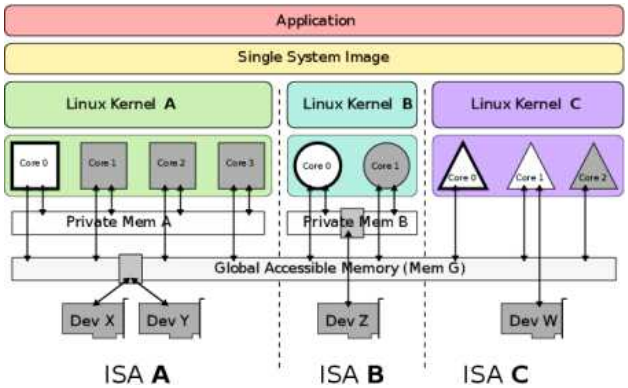
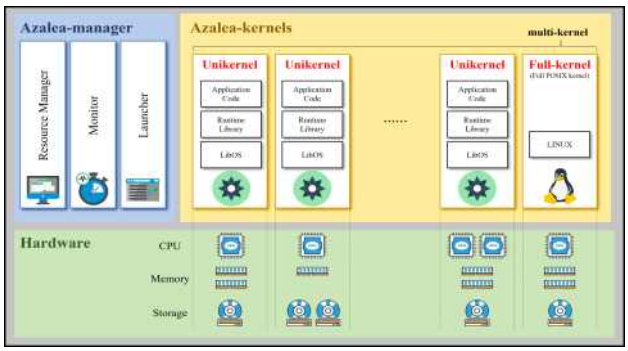


그림 503 VM exit Flow

· **OS functionality(kernel service) 분산 기술 분석**

- 멀티 코어 및 다중 코어 하드웨어 플랫폼에서 운영 체제, 특히 Linux의 확장성을 향상
- 커널 인스턴스는 단일 이미지 운영 체제의 추상화를 유지하도록 조정하여 레거시 Linux 응용 프로세스를 투명하게 실행
  . E.g, Multikernel, Replicated-kernel , Azelea-kernels, etc.

수행기관	개발 목표	개발내용 및 범위
		- Multikernel . OS functionality를 메시지 전달 기반 통신을 통해 분산된 시스템의 프로세스로 요청  그림 504 Multikernel - Replicated-kernel . 복수의 커널 이미지 인스턴스 상에 구축 . 커널 인스턴스는 이기종 커널 인스턴스를 통해 (부분적으로) 복제되는 공통 운영 체제 상태를 유지하기 위해 커널 간 직접 통신 . 여러 Linux 커널 인스턴스가 동일한 하드웨어에서 실행(ISA당 하나의 커널, NUMA 노드당 하나의 커널 또는 코어 전용 메모리가 있는 두 매핑의 조합)  그림 505 Replicated-kernel - Azalea-kernels . OS-Noise를 최소화하여 애플리케이션 수행을 최적화 할 수 있는 환경을 제공 . Kernel을 libOS로 생성(single address space) . Multi-instance 환경 지원  그림 506 Azalea-kernels · **기술개발전략 수립 - 개발 방안 비교 분석**

수행기관	개발 목표	개발내용 및 범위

표 47 분리자원 노드 추상화 개발 방안 비교 분석

개발 방안	분리자원 통합 기술(기능)			분리자원 분산성극복 기술(성능)	접근법	구현 레벨	Status
	CPU 통합 (C+C=2C)	메모리 통합 (M+M = 2M)	I/O 통합	2048-core scalability 지원 가능성			
1안) GiantVM extension	제공	일부제공 (DSM, M+M=M)	일부 제공	DSM성능 심각, ~10% 예상	가상화 기반	Hypervisor, Guest OS	연구시제품
2안) NUMA kernel extension	Todo	Todo	Tod o	50% 예상	OS-level SSI	Kernel-level	N/A
3안) Kernel service offloading	Todo	Todo(DSM for shared memory application)	일부 제공	70% 예상	OS-level SSI	Kernel-level	연구시제품
4안) GiantVM ext. /w CXL	제공	일부제공 (DSM, M+M=M)	일부 제공	50% 예상	가상화 기반	Hypervisor, Guest OS	연구시제품

· **CXL 메모리 기반의 분리자원 노드 추상화 시스템 구조 설계**

- 1단계

. CXL (emulated) memory 기반, Host OS는 하나만 실행, cache coherence 보장

. Guest에 CXL memory만 할당하도록 GiantVM 수정

. GuestOS가 수행되는 각 VM vCPU는 한쪽 socket core로 Pinning 수행

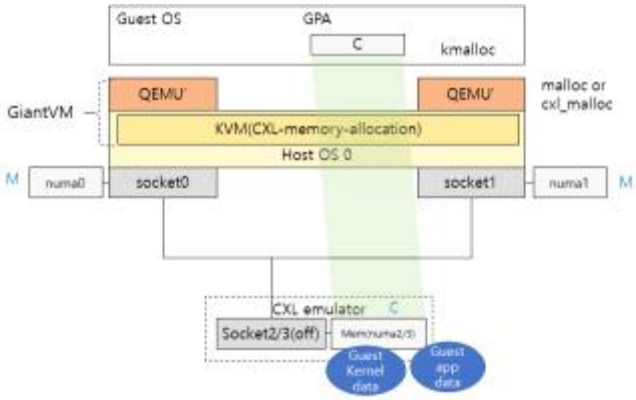


그림 507 1 단계 시스템 구조도

- 2단계

. 1단계 구성 + 성능 개선을 위해 NUMA memory를 각 VM을 위한 Cache로 사용하는 연구 수행

* 각 VM의 hot data를 로컬 NUMA memory로 옮겨서 서비스

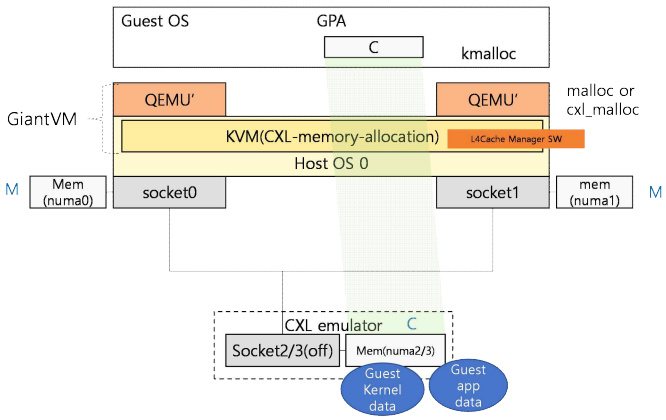
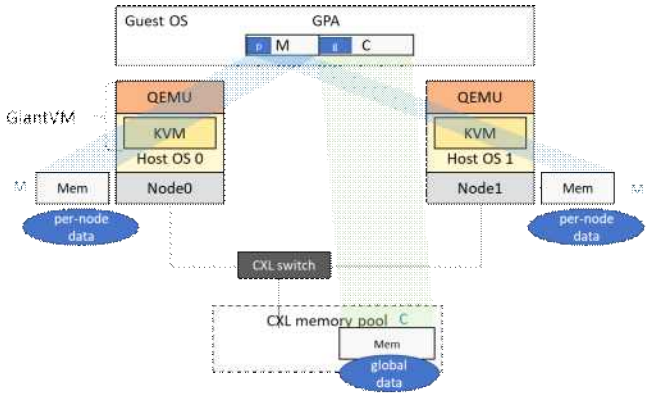
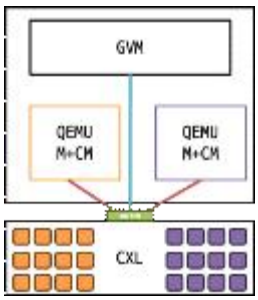


그림 508 2 단계 시스템 구조도

- 최종 단계

. CXL memory 기반, Host OS는 노드마다 실행, cache coherence 보장

수행기관	개발 목표	개발내용 및 범위
		* Environment: 2 Server, CXL pooled memory . Guest에서 local memory와 CXL memory 모두를 할당할 수 있도록 GiantVM을 수정 * node0와 node1간 데이터 migration 필요 . GuestOS가 수행되는 각 VM vCPU는 한쪽 socket core로 pinning 수행  그림 509 최종 단계 시스템 구조도
충북 대학교 (위탁1)	CXL 기반 계층 메모리 플랫폼에서 커널 데이터 배치 연구	**• NUMA 시스템 기반 CXL 메모리 에뮬레이션 기반 계층 메모리에 OS 커널 데이터 최적 배치 연구** **- 연구 배경** · 현대 서버 컴퓨팅은 다량의 서버 자원을 통합하여 제공하며, 이를 위해 MPI, DSM, SSI 같은 기술을 활용함 . GiantVM은 DSM 커널과 QEMU를 결합하여 여러 서버를 통합된 환경으로 제공하지만, OS가 자원을 사용할 때 성능 저하가 발생할 수 있음. 특히, 태스크가 다른 서버 메모리에 접근할 경우 지연시간이 증가하며, 이는 성능 저하를 가중 시킴 · 최근 CXL 기술은 고속 비휘발성 메모리를 활용한 메모리 풀링과 메모리 접근 가속화를 가능하게 하며, SSD보다 약 10배 빠른 속도를 제공함. 본 연구에서는 CXL의 높은 대역폭과 빠른 접근 속도를 활용하여 자원 통합 환경에서의 공유 데이터 접근 성능 저하를 해결하고자 함 · 리눅스 커널의 NUMA 페이지 마이그레이션 정책을 기반으로 CXL 메모리 풀을 공유 데이터 풀로 사용하여 공유 데이터 접근 지연을 줄이고 효율적인 데이터 관리를 위한 연구가 필요함 **- 연구 목표** · NUMA 시스템 기반 CXL 메모리 에뮬레이션 기술 개발 · CXL 메모리 기반의 분산 노드 통합을 위한 OS 커널 데이터의 최적 배치 기술 개발 · 리눅스 커널의 swap 메커니즘 최적화 및 CXL 기반 환경에 적용 방안 제시 · 워크로드 실험 분석 **- 연구 내용** **· 자원 통합 기술** (1) GiantVM - GiantVM은 여러 대의 노드들의 자원을 통합하여 보여주는 자원 통합 기술임. GiantVM은 QEMU 기반 가상화이므로 여러 대의 QEMU를 하나의 QEMU로 통합하는 기술임. DSM 커널을 사용해 분산 공유 메모리를 구성함. - 실제로 GiantVM을 구성해 구동한 결과 메모리가 통합 안되는 문제가 발생되었음. DSM에서는 두 프로세서가 동일한 페이지에 접근하는 현상을 페이지 공유라고 함. 이는 true sharing와 false sharing으로 나뉨. - True sharing은 두 스레드가 동일한 변수를 접근하는 경우임. False sharing은 각기 다른 변수를 동일한 페이지에 저장해 접근하는 경우임. 이 중 true sharing 현상은 2 노드 이상에서 data copy 오버헤드를 불러옴. CXL을 이용하여 GiantVM의 오버헤드를 줄이고자 함.

수행기관	개발 목표	개발내용 및 범위
		(2) GiantVM with CXL - GiantVM를 이용하여 CXL을 emulation 하기 위해서는 CPU 없는 노드가 다른 노드에 비해 높은 latency를 지녀야 함. GiantVM은 QEMU를 통해서 가상화해야 함. CPU가 없는 QEMU는 생성할 수 없고, CPU 없는 NUMA 노드는 생성이 가능함. - 하나의 QEMU에 CXL 메모리를 붙이는 상황과 두 개의 QEMU에 CXL 메모리를 붙이는 상황을 가정할 수 있음. - 하나의 QEMU에 CXL을 부착하는 방식은 GiantVM의 메모리 통합 문제와 연결됨. GiantVM은 메모리를 통합하는 과정에 문제가 생겨 메모리가 작은 부분을 메모리 통합해 큰 메모리가 인식되지 않는 현상이 발생함. CXL을 부착한 큰 메모리가 인식되지 않으므로 해당 방법은 적용할 수 없음. - QEMU 2대에 CXL 메모리를 부착하는 방식은 각기 다른 QEMU에 CPU 없는 NUMA node를 추가하여 CXL 메모리를 구분할 수 있도록 해야 함. 하지만, 각기 다른 QEMU에 서로 다른 옵션을 주면 NUMA node 설정 관련 충돌이 발생해 CXL 노드를 구분할 수 없게 됨. - QEMU에 mempath 라는 기능을 통해 CXL 디바이스를 등록할 수 있는데 이는 CXL 하드웨어가 있어야 가능함. 가능한 시나리오에서 GiantVM으로 CXL구성이 불가능해 대체 방안인 KVM으로 CXL을 구성하였음  그림 510 GiantVM과 CXL 구성 (3) KVM with CXL - KVM을 통해서 distance가 다른 NUMA node를 이용해서 해당 노드를 CXL 노드라 칭하고 CXL emulation을 개발함. 사용하는 machine를 단일 노드로 진행하여 개발가능성을 확인하였음. 각각의 노드들을 노드1 노드2로 칭하여 다중 노드에서 공유 데이터를 식별하여 CXL 노드로 적재할 수 있도록 NUMA node를 구성하였음 **· 리눅스 커널의 swap 메커니즘 최적화 (HitShield 기술)** (1) Multi-Generational LRU - Multi-Generational LRU (MGLRU)는 Linux kernel 5.17부터 고안되어 6.1에 정식으로 병합되어 사용 중인 swap 메커니즘의 LRU 알고리즘임. - MGLRU는 기존 LRU/2가 이전 메모리 참조 기록들을 굉장히 제한적으로 사용했고, 이로 인해 페이지가 정확한 리스트에 있는지를 제대로 판단하지 않는 상황을 지적하며 연구된 방식임 - 기존의 LRU/2의 list 2종(active list, inactive list) 대신 세대(generation)이라는 개념을 사용하며, 가장 최근에 참조된 folio(page wrapper 구조체)는 가장 어린 세대로 구성 후 참조된지 오래될수록 세대가 밀리는 방식으로 folio list를 관리함. - MGLRU는 Reverse Mapping(RMAP) 방식을 사용하지 않고, PTE 직접 스캔을 사용하여 swap 메커니즘의 오버헤드를 크게 경감시킨 것이 특징임 - 그러나, MGLRU는 가장 오래된 세대에서 folio를 강등시킴에 있어 folio의 상태를 확인하지 않는다는 단점이 있어, 이를 해결하는 것을 목표로 HitShield 기법을 고안하였음

수행기관	개발 목표	개발내용 및 범위

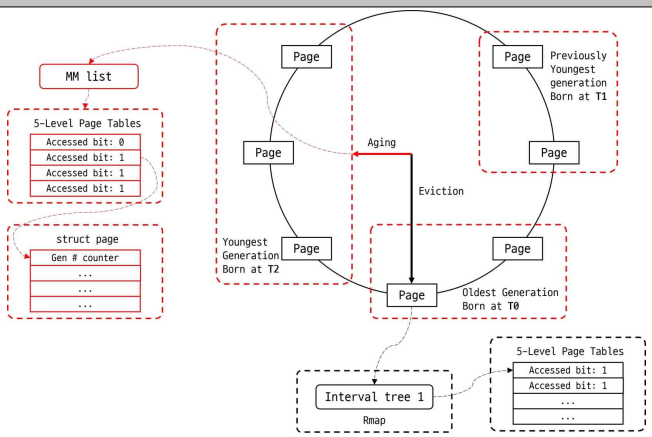


그림 511 Linux 운영체제에서의 MGLRU 구성

(2) Folio Update Count 구현

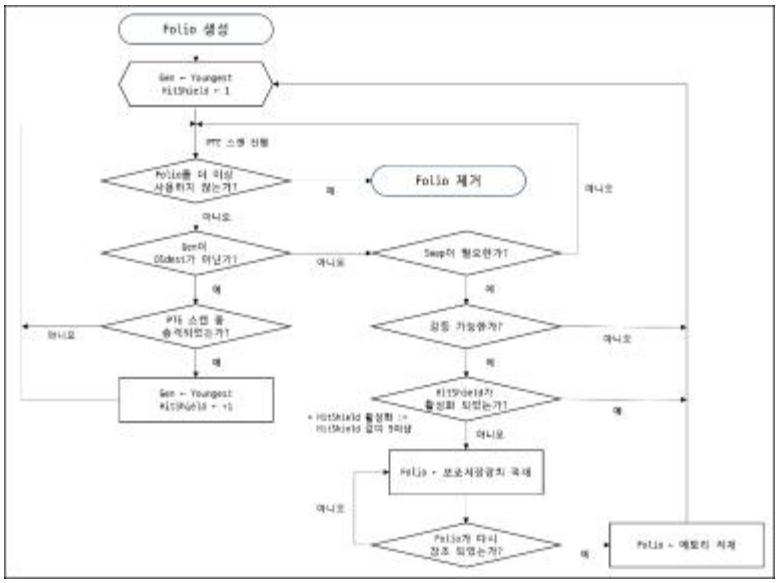
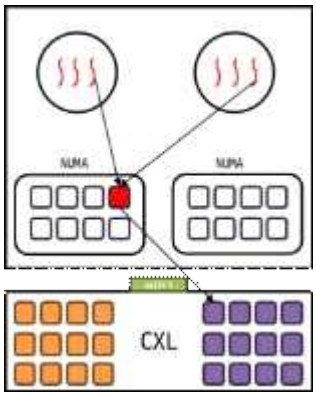


그림 512 HitShield의 동작 순서도

- HitShield는 일반적으로 MGLRU의 내부 구성에 Page Table Entry (PTE) 스캔 시 승격된 folio들에 대해 count를 기록하여 동작하는 방식으로 구성되어 있음.
- 해당 기법은 자주 참조되지 않는 folio는 지역성이 떨어졌을 것이라는 가정을 기반으로 함.
- 이는 일반적으로 동작하는 MGLRU의 generation 기록을 통한 승격 및 강등을 방해하지 않으며, 오직 강등에 사용될 folio들을 evict 시키는 것에만 관여하여 MGLRU의 안정적인 동작을 보장할 수 있음
- Count를 어떤식으로 구현하냐는 이 메커니즘의 동작에서 중요한 역할을 수행함. 정확한 장소에서 초기화 및 기록을 수행하지 않는다면 역으로 swap 횟수를 증가시킬 수 있는 가능성을 내포함. 예를 들어, folio의 count 초기화 지점을 folio 생성 시기에서 수행할 경우, folio가 다양한 이유로 lru 리스트 안에서 움직일 때 count가 정확한 횟수를 기록하지 않을 가능성이 있음.

수행기관	개발 목표	개발내용 및 범위
		- 이러한 사유를 바탕으로 count를 folio가 생성되는 시점이 아닌 lru 리스트 안에서 움직일 때 초기화한다면, 초기 생성된 folio도 무조건 lru 리스트에 적재되는 과정이 포함되어 움직이므로 모든 과정에 대해 적합하게 동작함을 보장할 수 있음 (3) Eviction으로부터의 folio 보호 - 일반적으로 folio를 evict 시킴은 folio를 메모리에서 보조 기억장치로 강등시키기 위한 전조 단계이고, 이때 evict 대상에서 folio를 제외시키고 최대 generation으로 folio를 승격시킴으로써 folio를 보호할 수 있는 방식으로 동작함. - folio를 어떻게 evict 대상에서 제외할 것인지를 설계해야 함. 이 방식은 unevictable flag의 동작을 유사하게 모사하여 구현하였음. Evict 대상을 선정하는 함수인 sort_folio에서 folio의 evict를 방지하고, generation을 youngest generation으로 격상하는 방식으로 보호하였음. - 해당 방식을 이용하면 HitShield가 활성화된 folio는 generation의 range의 2배 만큼의 생존을 보장하여 HitShield가 활성화되지 않은, 즉 자주 참조되지 않은 folio 만이 evict 대상으로 선정됨을 확보할 수 있음 · **TACO (Task-Aware CXL Optimization)** - Linux kernel 6.8.0을 수정해서 구현한 CXL emulation 환경에서 공유되는 데이터를 식별해 CXL 노드로 마이그레이션을 수행하는 기능을 포함한 커스텀 커널인 Task-Aware CXL Optimization(TACO)에 대해서 설명함 (1) NUMA Auto Balancing - NUMA Auto Balancing(Auto NUMA)은 Linux kernel 3.8부터 정식으로 병합된 메커니즘으로, NUMA 환경에서 태스크와 메모리 간의 접근 거리를 최소화하기 위해 자동으로 태스크 및 폴리오를 마이그레이션하는 방식으로 진행됨. 이는 태스크가 사용하는 CPU의 노드와 다른 노드의 메모리를 접근하는 과정에서 많은 오버헤드가 발생하기 때문에 해당 태스크 혹은 폴리오를 가까운 노드의 메모리로 옮겨 성능 저하 요인을 줄이는 것을 목표로 함. - Auto NUMA는 태스크가 생성될 때 마다 NUMA scan을 통해 임의로 page fault를 유발함. kernel에서는 page fault handler를 이용하여 NUMA scan을 통해 발생한 page fault를 식별하면서 해당 page의 NUMA 노드 위치와 page에 마지막으로 접근한 태스크를 기록함. 이러한 정보들을 NUMA hinting fault라고 함. Auto NUMA는 NUMA hinting fault의 통계를 기반으로 folio 혹은 태스크의 마이그레이션을 결정함. - Auto NUMA에서는 NUMA hinting fault를 2가지 범주를 통해 구분함. 먼저 폴리오의 메모리 위치와 태스크의 위치를 기준으로 하는 local fault와 remote fault임. 이는 태스크와 폴리오의 위치가 다를 경우 마이그레이션의 대상으로 선정할 확률이 높다는 것을 기반으로 구분함. - 다음으로 private fault와 shared fault 임. 이는 같은 태스크가 한 폴리오에 지속적으로 접근하는 것과 서로 다른 태스크가 한 폴리오에 번갈아 가면서 접근하는지를 구분하는데, Auto NUMA에선 shared fault가 발생한 폴리오는 마이그레이션을 진행하지 않음. - 본 연구에서는 Auto NUMA의 shared fault 식별 방식 및 폴리오 마이그레이션의 과정을 활용하여 연구 목표인 커널 공유 데이터의 식별 및 CXL 노드로의 적재 방식을 개발하였음 (2) 공유 데이터 식별 및 CXL 노드로의 이동 - 커널 공유 데이터를 식별하기 위한 구현 접근 방식으로 1) 기존 Auto NUMA의 shared fault가 발생하면 해당 폴리오를 CXL 노드로 마이그레이션 하는 방식과 2) 폴리오 struct에 추가적인 멤버 변수를 추가하는 방식, 3) folio struct의 기존 변수를 활용하면서 자체적인 shared fault를 검출하는 방식 총 세 가지 방식을 시도했음 1) 기존 Auto NUMA의 Shared fault를 통해 식별 후 마이그레이션을 진행하는 방식

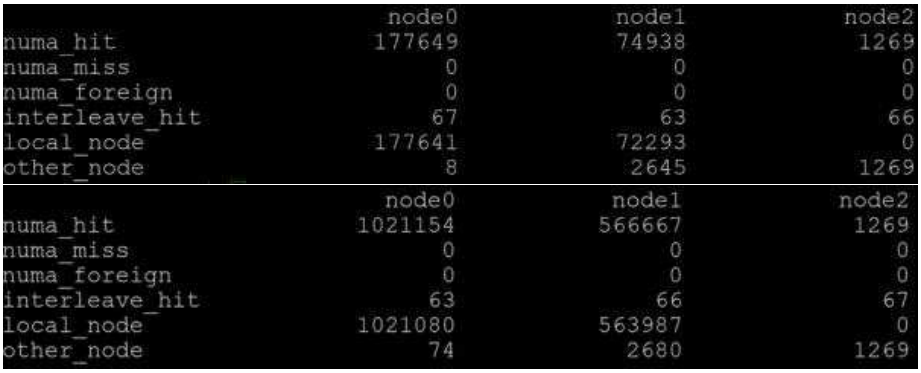
수행기관	개발 목표	개발내용 및 범위
		· 전체 shared fault 중 remote fault에 해당하는 폴리오만 마이그레이션을 진행해야 함. . 서로 다른 태스크와 태스크가 접근하려는 폴리오가 모두 같은 노드에 존재하는 경우는 local fault이면서 shared fault인데, 이 경우엔 서로 다른 태스크가 같은 노드에 존재하고, 폴리오 또한 같은 노드에 존재하기에 공유 데이터를 사용하기 가장 좋은 위치에 있으므로 CXL 노드로의 마이그레이션을 할 필요가 없음. · 하지만 이 방식에서는 실제 워크로드에서 remote-shared fault의 발생이 매우 미비하게 일어난다는 점과 Auto NUMA에서 shared fault가 난 페이지는 마이그레이션을 하지 않게 설계 되어있어 Auto NUMA의 대부분을 수정해야 한다는 점이 발생함. · 또한 CXL 노드로의 이동 이후에 Auto NUMA에서 해당 노드에 위치한 폴리오들은 적절하지 않은 위치에 있다고 판단해 다시 이전 노드로 옮기는 작업을 진행한다는 문제가 발생했음. 2) folio struct에 새로운 멤버변수를 추가하는 방식 · 이 방식은 폴리오를 강제로 CXL 노드로 옮겼을 경우 Auto NUMA에 의해 다시 이전 노드로 돌아가는 것을 방지하기 위해 Auto NUMA를 비활성화 한 상태에서 진행했음. · folio struct에 마지막으로 접근한 태스크의 노드 위치를 담는 변수와 접근 횟수를 담는 변수를 추가해 마지막으로 접근한 노드가 몇 번을 접근했는지 기록하고, 이 내용을 기반으로 각 노드에서 CXL 노드로, 혹은 CXL 노드에서 각 노드로의 마이그레이션 여부를 결정함. · 이 방식은 확실한 공유 데이터를 식별할 수 있으나 folio struct가 더 무거워지고 folio의 접근 마다 각 변수를 업데이트하는 연산이 추가된다는 점에서 전체적인 성능이 떨어지는 문제가 발생함. · 또한 각 접근 횟수를 담는 변수의 일관성을 유지하기 위한 오버헤드 또한 발생하며 Auto NUMA의 비활성화에 따른 폴리오 마이그레이션의 구현에 어려움이 있음  그림 513 TACO 동작 과정 3) folio struct의 _last_cpupid 멤버변수를 활용 · 이 식별 방식을 기반으로 한 커널 구현의 필요 요구사항은 Auto NUMA의 동작 주기와 동일하게 작업을 해야 한다는 점과 Auto NUMA로 인해 CXL 노드에서 다시 기존 노드로의 재배치를 피해야 한다는 점이 있음 · 결과적으로 Auto NUMA의 동작 과정에서 folio의 현재 노드 번호와 태스크의 노드 번호, _last_cpupid 멤버변수, 태스크의 현재 프로세스 번호 총 네 개의 정보를 조합 및 활용해 shared fault를 자체적으로 구분했음 · TACO는 그림 513와 같이 공유 데이터를 구분하고 CXL 노드로의 마이그레이션을 진행함. · 이는 식별된 폴리오를 Auto NUMA 상에서의 마이그레이션 목표 노드를 CXL 노드로 고정하고, 이후 CXL 노드에서는 할당이 해제가 되지 않는 이상 기존 노드로의

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>

재배치를 하지 않게 하는 기능을 포함함

(3) TACO의 검증

- TACO의 공유 데이터 식별 및 CXL 노드로의 마이그레이션을 정확하게 수행하는지에 대한 여부를 확인함
- CXL emulation 환경을 구성하고, 해당 환경에서 Linux kernel 6.8과 TACO를 비교했음
- 각 노드에 대한 폴리오의 이동을 확인하기 위해서 numastat 도구의 각 노드 별 폴리오가 할당되고 접근한 횟수인 numa_hit의 수치 변화를 통해 확인했음
- Linux kernel 6.8의 부팅 시의 numastat 결과 화면과 특정 워크로드를 실행한 이후의 numastat 결과 화면



	node0	node1	node2
numa_hit	177649	74938	1269
numa_miss	0	0	0
numa_foreign	0	0	0
interleave_hit	67	63	66
local_node	177641	72293	0
other_node	8	2645	1269

	node0	node1	node2
numa_hit	1021154	566667	1269
numa_miss	0	0	0
numa_foreign	0	0	0
interleave_hit	63	66	67
local_node	1021080	563987	0
other_node	74	2680	1269

그림 514 Linux kernel 6.8 numastat 결과 화면

표 48 numastat의 각 통계 카테고리

numa_hit	이 노드에 성공적으로 할당된 페이지수
numa_miss	의도한 노드의 메모리 부족으로 인해 이 노드에 할당된 페이지 수
numa_foreign	다른 노드에 할당된 노드에 대해 처음 사용되는 페이지 수
interleave_hit	이 노드에 성공적으로 할당된 임시 정책 페이지 수
local_node	이 노드의 프로세스에 의해 이 노드에 성공적으로 할당된 페이지 수
other_node	다른 노드의 프로세스에 의해 이 노드에 할당된 페이지 수

- CXL emulation 환경에서 CXL 노드인 2번 노드는 CPU가 없는 메모리 노드이기 때문에 Auto NUMA에서 폴리오와 태스크의 마이그레이션 우선순위가 가장 낮음
- 기존의 Linux kernel 6.8에서는 CXL 노드에 폴리오가 할당되는 것은 부팅을 제외하고 CPU가 있는 노드의 메모리에 여유 공간이 없을 때에만 할당됨

</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

	node0	node1	node2
numa_hit	176903	77076	1260
numa_miss	0	0	0
numa_foreign	0	0	0
interleave_hit	70	61	65
local_node	176835	74384	0
other_node	68	2692	1260

	node0	node1	node2
numa_hit	1041102	519466	346891
numa_miss	0	0	0
numa_foreign	0	0	0
interleave_hit	64	61	71
local_node	1040957	516893	0
other_node	145	2573	346891

그림 515 TACO numastat 결과 화면

- TACO의 부팅 시 numastat 결과 화면과 이전 기존 커널의 실험과 같은 워크로드를 실행한 이후의 numastat 결과 화면
- TACO에서는 서로 다른 태스크가 공유 데이터에 접근하는 경우 CXL 노드로의 마이그레이션하기 때문에 워크로드를 실행한 이후엔 2번 노드로의 numa_hit 값이 증가함을 볼 수 있음
- TACO가 정확하게 공유 데이터를 CXL로 옮기는지에 대한 여부를 검증하기 위해 실험에 사용한 워크로드는 공유 데이터를 여러 태스크에서 사용하도록 직접 제작했음
- 워크로드의 동작 방식은 공유 메모리를 생성한 이후 각 스레드를 서로 다른 노드의 CPU에 분배해서 할당함. 각 스레드는 공유 메모리의 같은 위치를 접근하고 읽기/쓰기 행동을 반복하며 공유 메모리에 접근함. 서로 다른 태스크가 같은 폴리오에 접근하게 되어 shared fault의 상황을 만들 수 있음

- 연구 결과

· 자원 통합 환경 구성

- KVM 가상머신에 NUMA node 3개를 구성하고 각각의 latency를 CXL latency에 맞게 조정하여 CXL 에뮬레이션 환경을 구축
- Memory Latency Checker(MLC)를 통해 latency를 측정하여 CXL 에뮬레이터로서 효용성이 있는지 검증

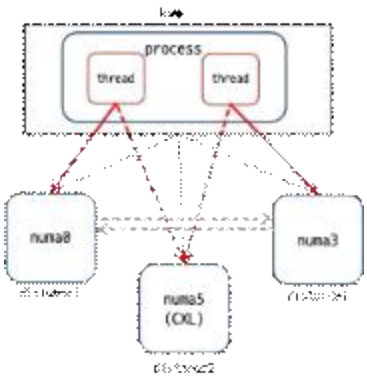


그림 516 KVM+CXL 에뮬레이터

	Numa node		
Numa node	0	1	2
0	209.3	429.1	455.4
1	304.3	313.2	451.6

그림 517 KVM CXL latency

수행기관	개발 목표	개발내용 및 범위

· **MGLRU 성능 평가**

- 기존 LRU 알고리즘인 LRU/2와 MGLRU 간의 성능을 비교하여 MGLRU의 효용성을 판단

(1) 실험환경

표 49 MGLRU 실험 환경

OS	Ubuntu 22.04 LTS
Kernel	Linux Kernel 6.8.4
CPU	Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz
Memory	DDR3 12GB(2GB * 6)
Storage	RAID 1 (HDD 1TB * 2)

- 7zip Benchmark, Yahoo Cloud Serving Benchmark(YCSB) 사용
- Docker를 통해 750M으로 DRAM 가용량을 제약하여 swap을 강제한 후 평가하였음
- 평가 결과로는 벤치마크에서 제공하는 결과 (7zip에서의 Total MIPS와 Total CPU Usage, YCSB의 runtime)와 /proc/vmstat에 기록되는 pswpin, pswpout 값을 활용

(2) 벤치마크를 통한 성능 평가

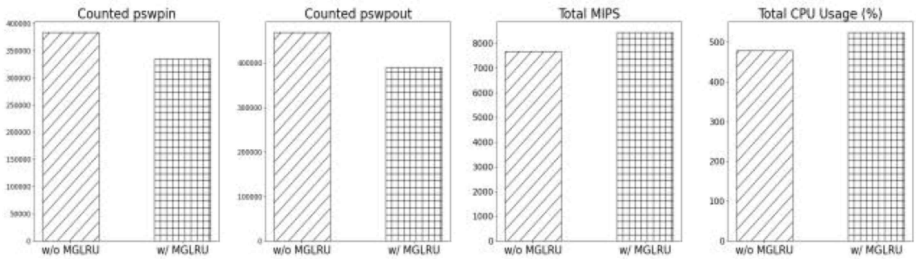


그림 518 7zip 벤치마크

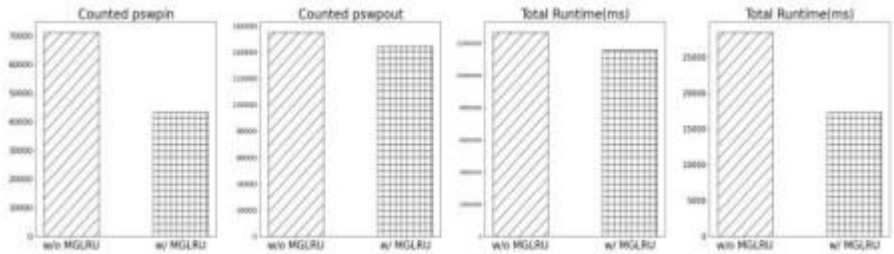


그림 519 YCSB (runtime 각 load / run 명령)

- 7zip은 pswpin과 pswpout이 각각 16.3%, 22.2% 감소하였고, Total MIPS는 10.2%, Total CPU Usage는 9.7% 증가하였음
- YCSB은 pswpin과 pswpout이 각각 38.9%, 7.0% 감소하였고, load 명령에서는 8.8%, run 명령에서는 39.3%의 성능 향상이 확인되었음

· **HitShield 성능 평가**

- Linux 6.8.0 기준으로 MGLRU와 HitShield의 성능을 비교하여 HitShield의 효용성을 판단함

(1) 실험환경

수행기관	개발 목표	개발내용 및 범위

표 50 HitShield 성능 평가 실험 환경

OS	Ubuntu 24.04 LTS
Kernel	Linux Kernel 6.8.0
CPU	AMD Ryzen 9 5950X 16-Core Processor
Memory	DDR4-3200 64GB(32GB * 2)
Storage	SSD 1TB

- HitShield의 성능을 평가하기 위해 2가지 벤치마크 7zip Benchmark, YCSB와 Linux kernel 6.8.0 소스코드를 빌드하는 실험을 통해 측정하였음
- Docker를 통한 DRAM 제약을 통해 swap을 강제한 환경에서 평가
- 평가 방식은 실행시간과 /proc/vmstat에 기록되는 pswpin, pswpout 값을 확인

(2) 벤치마크를 통한 성능 평가

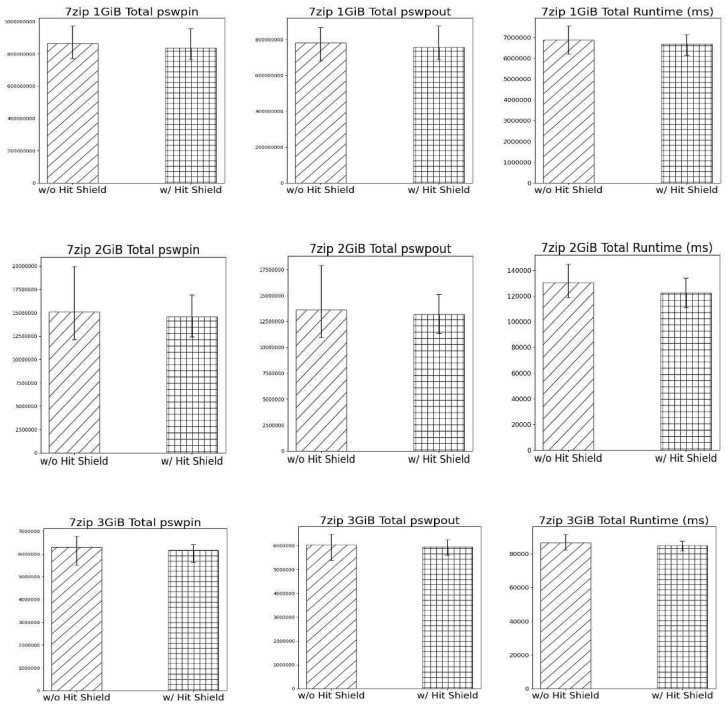
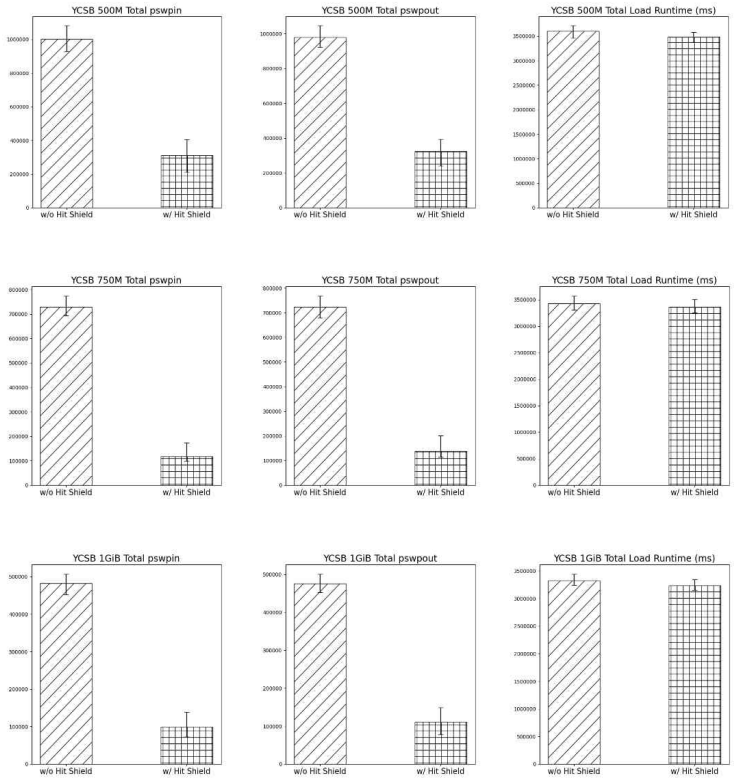
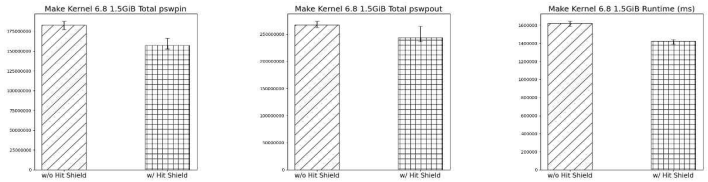


그림 520 7zip 벤치마크(각1G, 2G, 3G 제약)

수행기관	개발 목표	개발내용 및 범위
		 그림 521 YCSB (각500M, 750M, 1G 제약)  그림 522 Make Kernel 6.8.0 (1.5G 제약) - 7zip benchmark : 메모리 1G 제약시 pswpin, pswpout, runtime에 대해 각 3.18%, 3.12%, 2.9%의 향상이 있었으며, 2G 제약시 3.34%, 3.36%, 6.21%의 향상이 있었고, 3G 제약시 2.04%, 1.4%, 1.79%의 향상이 있었음 - YCSB : 메모리 500M 제약시 pswpin, pswpout, runtime에 대해 각 68.9%, 67.01%, 3.17%의 향상이 있었으며, 750M 제약시 83.75%, 80.93%, 1.7%의 향상이 있었고, 1G 제약시 79.45%, 76.65%, 2.7%의 향상이 있었음 - Make Kernel 6.8 : 메모리 1.5G의 제약시 pswpin, pswpout, runtime에 대해 각 14.06%, 8.94%, 12.1%의 향상이 있었음

수행기관	개발 목표	개발내용 및 범위

표 51 HitShield 전체 실험 결과

workload(dram)	pswpin	pswpout	runtime
7zip(1G)	3.18%	3.12%	2.9%
7zip(2G)	3.34%	3.36%	6.21%
7zip(3G)	2.04%	1.4%	1.79%
YCSB(500M)	68.9%	67.01%	3.17%
YCSB(750M)	83.75%	80.93%	1.7%
YCSB(1G)	79.45%	76.65%	2.7%
Make Kernel(1.5G)	14.06%	8.94%	12.1%

· **TACO 성능 평가**

(1) 실험환경

표 52 TACO 성능 평가 진행 실험 환경

OS	Ubuntu 24.04 LTS
Kernel	Linux Kernel 6.8.0
CPU	Intel(R) Xeon(R) CPU E7-8890 v4 @ 2.20GHz 각 노드에 24코어 씩 총 48코어 할당
Memory	DDR4-3200 각 노드에 18GB 씩 총 54GB 할당
Storage	SSD 125GB

- CXL 에뮬레이션 환경 구성을 위해 CPU 없는 NUMA노드가 포함된 VM 환경에서 진행

(2) TACO 검증 워크로드를 통한 실험 내용 및 결과

- TACO를 검증 및 평가를 위한 워크로드를 자체 제작
- 공유 메모리의 양과 스레드 수에 따른 실행시간을 측정하고 각 워크로드의 노드 별 메모리 사용량과 함께 numastat을 통한 CXL 노드의 numahit를 측정해 성능을 평가함
- 실험 결과

표 53 공유메모리 사이즈와 쓰레드 수에 따라 CXL 노드로 마이그레이션된 워크로드의 비율

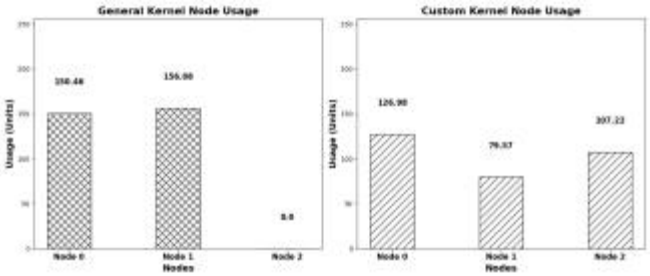
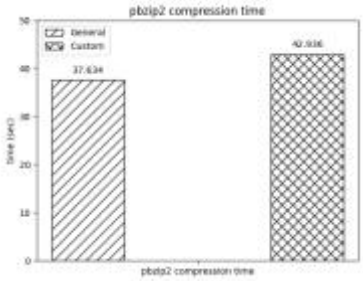
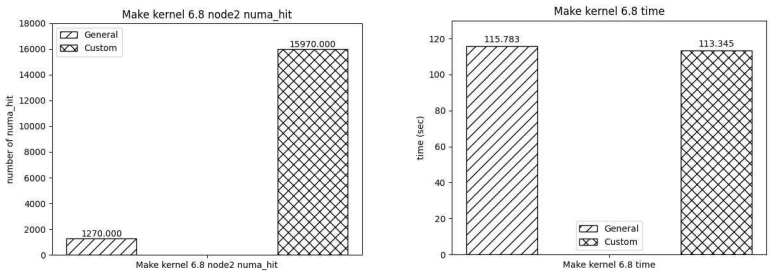
workload(shm size)	8스레드	16스레드	32스레드	64스레드
공유 데이터 이동(2G)	56%	57%	57%	69%
공유 데이터 이동(4G)	67%	65%	61%	71%
공유 데이터 이동(8G)	71%	66%	60%	84%

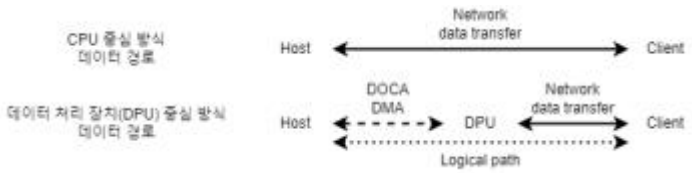
표 54 공유메모리 사이즈와 쓰레드 수에 따른 실행시간 증가

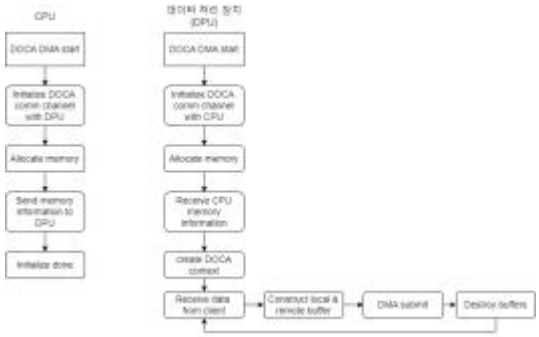
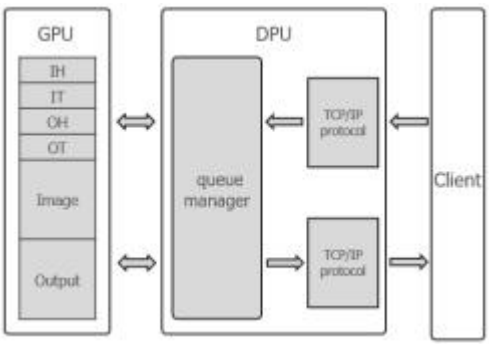
workload(shm size)	8스레드	16스레드	32스레드	64스레드
실행시간(2G)	4%	13%	14%	1%
실행시간(4G)	2%	11%	12%	1%
실행시간(8G)	1%	8%	11%	1%

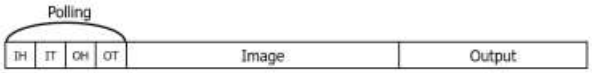
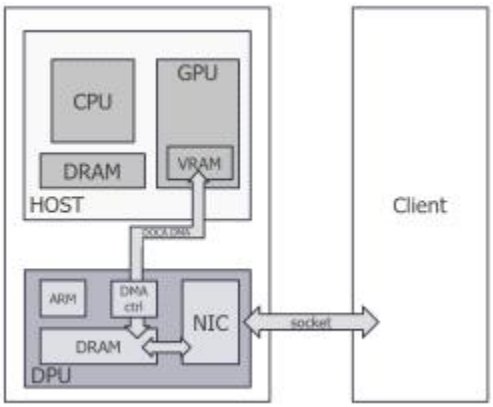
(3) 실제 워크로드를 통한 실험 내용 및 결과

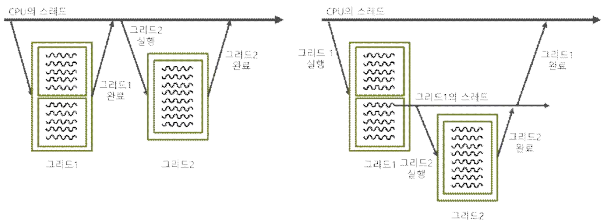
- pbzip2 실험 및 결과
  · 멀티 스레드를 활용한 병렬 압축 도구의 압축 시간 및 메모리 사용량 측정.

수행기관	개발 목표	개발내용 및 범위
		· 워크로드의 34%가 CXL 노드로 이동 · TACO에서 압축시간 14% 증가  그림 523 pbzip2 각 노드 메모리 사용량  그림 524 pbzip2 압축 시간 비교 - Make Kernel 6.8 실험 결과 · CXL 노드로의 numa_hit 증가 · 전체 빌드 시간 1% 감소 - CXL 노드로의 마이그레이션 자체는 전체 성능에 큰 영향을 주지 않지만, CXL 노드로 옮겨진 folio에 대한 접근이 많을 수록 성능이 저하됨  그림 525 Make Kernel 6.8.0 CXL 노드 numa_hit 및 빌드 시간 비교 **· 연구 결과 요약** (1) 자원 통합 & KVM CXL emulation - NUMA node의 latency 차이가 CXL의 latency 차이와 비슷한 점 이용을 이용하여 NUMA node를 활용해서 CXL emulation 환경 구성 - GiantVM에서 CXL 하드웨어 없이 재현 불가 - QEMU에서 mem-path 기능을 통해서 CXL 디바이스 등록 가능하므로 CXL 하드웨어가 있으면 실현 가능할 것으로 예상 (2) HitShield 연구 - Swap 메커니즘을 수정하여 보다 우수한 swap eviction 가능 - CXL 메모리로 페이지 in/out에 활용 가능 (3) 공유데이터 to CXL (TACO) - folio struct의 _last_cpupid 멤버변수 활용하여 구현

수행기관	개발 목표	개발내용 및 범위
		- 적절하게 공유 데이터를 구분하고 CXL로 이동을 확인 - CXL 하드웨어를 이용하여 cluster 환경에서 유용함을 확인할 필요가 있음
한국항공대학교 (위탁2)	데이터 처리 장치 기반의 AI 분산 컴퓨팅 SW 구조 선행 연구	**• 고속의 데이터 처리 장치를 이용 변혁적인 성능 개선이 가능한 AI 분산 컴퓨팅 SW의 선행 연구** **- 연구 배경 및 목표** **· 배경 및 필요성** - 데이터센터의 수요가 증가하며 효율적인 데이터 처리 구조가 요구됨 - CPU만으로 증가하는 데이터 처리 요구를 감당하기 어렵고, 기존의 CPU 중심 구조는 가속기 및 NIC 성능 향상 속도를 따라가지 못해 비효율적임 - 데이터 처리 장치를 활용해 CPU의 데이터를 오프로드함으로써 CPU 병목현상을 완화하고 효율성을 높이는 방안이 필요함 - 기존 구조는 네트워크 데이터를 CPU가 처리 후 가속기로 전달하지만, 이 방식은 경로가 길어져 비효율적임 - 데이터 처리 장치와 가속기 간 직접 통신을 구현해 CPU 개입을 최소화하고 데이터 경로를 단축해야 함 - CPU 의존성이 남아 있으면 가속기 성능을 제한할 수 있으므로, 가속기 메모리에 있는 데이터를 CPU로 복사하지 않고 직접 추론 연산을 수행하는 구조가 필요함 - 이러한 접근은 CPU 부하 감소와 데이터 처리 효율성을 동시에 달성할 수 있음 **· 연구 목표** - 데이터 처리 장치를 사용한 네트워크 오프로딩 및 가속기와 직접통신 기술 개발 - 직접통신 기술과 연계된 저 지연 추론 서비스 SW 개발 - 성능 최적화를 위한 통합 AI 추론 기반 테스트 SW 개발 **- 연구 내용** **· 데이터 처리 장치 중심 네트워크 데이터 처리 구조** **(1) 네트워크 오프로딩** - 기존의 CPU 중심 네트워크 데이터 처리 구조에서는 CPU가 데이터 평면(Data Plane)과 제어 평면(Control Plane) 모두를 처리하며, 모든 네트워크 부하를 CPU가 전담함 - 이 방식은 CPU에 과도한 부하를 초래하고, 데이터 경로에 불필요한 처리가 추가되면서 자원 낭비와 비효율적인 처리로 이어지며 특히, CPU가 데이터를 처리하는 데 필요한 시간이 많아질수록, 네트워크 처리 성능이 저하되고, 서비스 지연 시간이 증가하는 문제 발생 - CPU의 개입을 최소화하면서도 네트워크 처리 성능을 극대화하려면 CPU와 데이터 처리 장치의 역할을 명확히 분리해야 함  그림 526 CPU 중심의 네트워크 데이터 처리 구조와 데이터 처리 장치 중심의 네트워크 데이터 처리 구조 비교 - 본 연구는 CPU의 데이터 평면과 제어 평면을 데이터 처리 장치(DPU)로 완전히 오프로드하는 네트워크 오프로딩 기술을 제안함 - 위 그림과 같이 기존의 CPU 중심 네트워크 처리 방식과 달리, DPU가 네트워크 작업을 처리하고 호스트에는 필요한 데이터만 전달하여 처리 성능을 향상함 - RDMA를 활용한 기존 방식은 네트워크 인터페이스 컨트롤러(NIC)의 부하를 증가시키는 단점이 있음

수행기관	개발 목표	개발내용 및 범위
		- 이를 해결하기 위해, 본 연구는 DOCA DMA 기술을 사용하여 DPU의 DMA 컨트롤러로 데이터를 메모리 간 복사하고 CPU 개입을 최소화함 - CPU는 초기 설정에서만 개입하며, 데이터 전송 및 복사는 DPU가 직접 처리함 - DPU는 설정된 메모리 정보와 DOCA Context를 활용해 네트워크 데이터를 수신하고, 로컬 버퍼에서 호스트 메모리로 데이터를 복사함 - 이러한 방식은 네트워크 데이터 처리에 대한 CPU 부하를 줄이고 데이터 전송 효율을 높임 - 초기 설계에서는 데이터 수신마다 DOCA Context를 생성하여 초기화 오버헤드로 처리 성능이 크게 저하됨 - 초기 데이터 처리 장치 중심 네트워크 구조는 CPU 중심 구조 대비 약 1/50의 처리량을 보여 실제 적용에 어려움이 있었음 - 최적화된 구조에서는 그림과 같이 DOCA Context를 한 번만 생성하고 데이터 수신 시 이를 재사용해 초기화 오버헤드를 제거함  그림 527 DOCA DMA 동작 흐름도 - 네트워크 오프로딩 구조에서는 소켓 통신의 엔드포인트를 데이터 처리 장치와 호스트 메모리 간 연결로 설정, CPU는 초기 설정만 담당하고 이후 데이터 전송에는 개입하지 않음 - 최적화된 구조는 CPU 부하를 최소화하고 데이터 전송 효율을 극대화하여 실시간 데이터 처리와 같은 고성능 컴퓨팅 환경에 적합한 성능을 제공함 표 55 버퍼 사이즈 2MB 네트워크 처리 구조 성능 비교 CPU 중심 구조 처리량: 10661.073 Mbps 초기 데이터 처리 장치 중심 구조 처리량: 204.788 Mbps 개선된 데이터 처리 장치 중심 구조 처리량: 4947.564 Mbps - 결과적으로 표와 같이 네트워크 처리 효율성이 향상되고 서비스 지연 시간이 단축됨  그림 528 데이터 처리 장치 중심의 네트워크 데이터 처리 구조도 - 본 연구는 데이터 처리 장치 중심의 네트워크 오프로딩 기술을 통해 CPU 개입을 최소

수행기관	개발 목표	개발내용 및 범위
		화하면서 높은 성능을 달성하는 구조를 제안함 - 그림과 같은 구조는 데이터센터와 고성능 컴퓨팅 환경에서 네트워크 부하를 효과적으로 분산시키고, 서비스 지연 시간을 줄일 가능성을 보여줌 **(2) 데이터 처리 자료구조 정의** - 네트워크 처리 속도와 호스트 작업 속도의 불일치로 인해 데이터가 누적될 가능성이 있어 적합한 자료구조가 필요함 - 스택(LIFO)은 순차적 데이터 처리 시스템에 적합하지 않으며, 큐(FIFO)는 효율적 구현이 어려움 - 링 버퍼는 선입선출 방식의 순환구조로, 메모리 이동 없이 데이터를 추가할 수 있어 지속적인 데이터 처리 환경에 적합함 - 본 연구에서는 링 버퍼를 네트워크 데이터 처리 자료구조로 채택하고, 가속기 메모리에 위치시켜 로컬 메모리 접근만 필요하게 함 - 이를 통해 로컬 메모리 복사 오버헤드를 줄이고 데이터 처리 속도와 효율성을 크게 향상시킴  그림 529 네트워크 데이터 처리 자료구조 메모리 맵 - 링 버퍼 전체를 폴링하는 기존 방식은 비효율적이며 작업 지연을 초래함 - 본 연구는 헤드와 테일 포인터만 폴링하는 방식을 제안, 불필요한 메모리 접근 횟수와 범위를 줄임 - 그림과 같이 메모리 시작 16바이트를 폴링 전용 공간으로 설정해 포인터(입력 헤드/테일, 출력 헤드/테일)를 관리하며, 필요한 경우에만 데이터 영역에 접근 - 링 버퍼는 데이터 처리 장치와 가속기가 공유하는 메모리로, 동시 작업 충돌을 방지하기 위해 락(lock) 대신 쓰기 권한 구분 방식을 사용 - 데이터 처리 장치: 입력 헤드 포인터, 입력 이미지 버퍼, 출력 테일 포인터에 쓰기 작업 수행 - 가속기: 출력 헤드 포인터, 출력 결과 버퍼, 입력 테일 포인터에 쓰기 작업 수행 - 각 장치의 쓰기 작업 공간을 명확히 분리해 충돌 문제를 해결하고, 링 버퍼 사용 효율을 극대화함 **(3) 데이터 처리 장치와 가속기 간 직접통신 구조**  그림 530 데이터 처리 장치와 가속기 간 직접 통신구조를 적용한 네트워크 처리 구조도 - 가속 컴퓨팅 환경에서는 데이터 처리 장치가 호스트에게 데이터를 전송할 때 CPU 메인 메모리를 거치지 않는 최적화된 데이터 경로가 필요함 - 기존 CPU 중심 구조는 NIC, CPU, 가속기 경로를 따르지만, 데이터 처리 장치 추가로

수행기관	개발 목표	개발내용 및 범위
		데이터 경로가 길어지는 문제가 발생함 - DOCA DMA는 호스트와 데이터 처리 장치 간 메모리 전송을 지원하나, 가속기 메모리에 직접 접근하는 데 한계가 있음 - DOCA GPUNetIO 라이브러리를 활용하여 CPU 개입 없이 PCIe 장치 간 직접 메모리 접근을 구현 - DPDK GPU, CUDA GPUDirectRDMA, 피어 메모리 기능을 활용 - 데이터 처리 장치가 가속기 메모리에 직접 접근 가능 - 연구에서는 DOCA DMA와 DOCA GPUNetIO를 결합해 데이터 처리 장치와 가속기 간 직접통신 기술을 구현하여 데이터 전송 경로를 최적화하고, 가속기 중심 데이터 처리 구조를 지원함 **· CPU 개입 최소화 추론 연산 구조** - 딥러닝 추론에서 가속기 활용은 필수적이며, 가속기는 다차원 행렬연산에 최적화된 대규모 병렬 처리를 수행 - 그러나 실제 추론 과정에서 CPU가 데이터 전송, 작업 스케줄링 및 제어에 관여하며, 이는 추론 속도 저하와 오버헤드를 초래 . 메모리전송 병목: PCIe 전송 속도가 가속기 연산 속도에 비해 느림 . 스케줄링 지연: CPU의 작업 제어 과정에서 추가 지연 발생 - 기존 구조에서는 CPU 개입을 완전히 제거하는 데 한계가 있었음. - 본 연구는 DOCA DMA를 활용해 데이터 처리 장치에서 가속기로 데이터를 전송, CPU 개입 최소화를 달성 - 더 나아가, 가속기가 추론 과정의 스케줄링을 직접 담당하는 방안도 연구함 **(1) 추론 연산 구조 관련 도구** - 엔비디아 CUDA 라이브러리는 가속기의 병렬 연산(CUDA 커널)을 효율적으로 호출 및 관리 가능 - 하지만 성능 병목 현상 존재: . 데이터 전송 지연: CPU와 가속기 간 데이터 전송(H2D 및 D2H cudaMemcpy)이 연산보다 더 오래 걸릴 수 있음 . 커널 런치 오버헤드: 연산 요청과 실제 실행 간의 지연으로 성능 저하 발생 - 문제 해결을 위해 CUDA Dynamic Parallelism과 CUDA Graph 기능을 제공, 데이터 전송과 연산 스케줄링 효율성을 높이고 오버헤드 감소에 기여함 **(가) CUDA Dynamic Parallelism**  그림 531 CUDA Dynamic Parallelism 적용 전(왼쪽), 후(오른쪽)의 가속기 작업 흐름 - CPU와 가속기 간 연산 수행을 위해 스레드 수, 블록 구성, 그리드 등의 실행 정보가 전달됨 - CUDA Dynamic Parallelism은 가속기의 스레드가 자체적으로 커널을 호출하여 CPU 개입 없이 작업을 진행하게 함 - 이를 통해 CPU-가속기 간 빈번한 통신을 줄이고, 연산 효율성을 높일 수 있음 - CUDA Dynamic Parallelism 적용 전: CPU는 매번 가속기 연산을 호출해야 함 - CUDA Dynamic Parallelism 적용 후: 가속기 스레드가 다른 작업을 직접 실행,

수행기관	개발 목표	개발내용 및 범위

CPU 개입 없이 연산 병렬성 극대화

(나) CUDA Graph

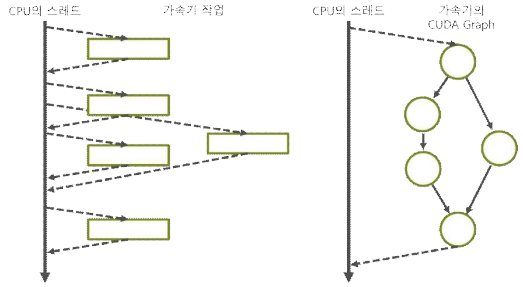


그림 532 CUDA graph 적용 전(왼쪽),
후(오른쪽)의 가속기 작업 중 CPU 개입

- CUDA Graph는 반복적인 가속기 작업에서 CPU와 가속기 간 오버헤드를 줄여 성능을 향상시키기 위해 도입된 기능임
- 커널 호출 및 메모리 전송 작업을 하나의 그래프로 묶어 미리 정의한 후, 이를 한 번만 제출하고 여러 번 실행할 수 있게 함
- 그래프 생성 방법:
  . Graph API: 사용자가 직접 노드 간 의존성을 고려하여 복잡한 그래프 구조 설계
  . Stream Capture: 기존 CUDA Stream에서 실행되는 작업을 캡처하여 그래프 생성, 프로그램 구조 변경 없이 그래프 전환
- CUDA Graph 적용 전: CPU가 각 작업 완료를 감지하고 반복적으로 개입해야 함
- CUDA Graph 적용 후: 그래프의 시작과 종료 시점에만 CPU가 개입, CPU 개입 최소화, 작업 효율성 증가
- CUDA Graph는 기존 작업의 순서를 유지하며 그래프로 정의 가능하여, CUDA Dynamic Parallelism보다 더 유연하고 복잡한 작업에 적합함

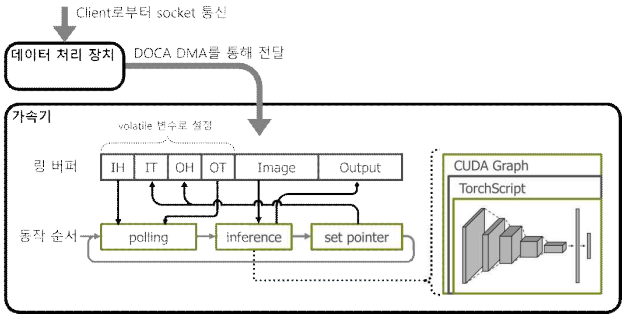
(2) 추론 연산 소프트웨어

- 딥러닝 프레임워크인 Pytorch와 TensorFlow는 Python 환경에서 딥러닝 모델을 개발하고 활용할 수 있으나 Python 외의 언어로 개발된 시스템에서는 활용이 어려운 단점이 있음
- TorchScript는 Pytorch에서 제공하는 기능으로, Python 환경에서 개발된 모델을 직렬화하여 C++, Java 등 다른 언어로도 사용할 수 있게 함
- 이를 통해 호환성을 높이고, CUDA Graph와 Stream Capture 방식을 활용하여 CPU 개입을 최소화할 수 있음
- DOCA 프레임워크에서 Python 의존성을 제거하고, TorchScript 모델을 활용하여 Python 외 환경에서도 기존 모델을 원활히 사용할 수 있도록 하며 변환된 모델에 CUDA Graph를 적용하여 CPU 개입을 최소화함

(3) CPU 개입 최소화 추론 성능 비교

표 56 세 가지 상황에서의 추론 소요시간, 가속기 사용량 비교

Metric	Pytorch	TorchScript	TorchScript + CUDA Graph
언어	Python	C++	C++
추론 소요 시간(ms)	8.1899	6.6153	3.4077
가속기 사용량(%)	58	63	100

수행기관	개발 목표	개발내용 및 범위
		- ResNet152 모델에 대해 TorchScript만 사용했을 때 추론 소요시간이 19.23% 단축, 가속기 사용량은 8.62% 증가 - TorchScript + CUDA Graph를 함께 사용했을 때, 추론 소요시간이 58.39% 단축, 가속기 사용량은 100%에 도달하며 72.41% 증가 - 결론: CUDA Graph를 사용하여 CPU 개입을 줄이고, 가속기 활용 효율을 높여 성능 향상을 가져옴 **(4) 직접통신 적용 CPU 개입 최소화 추론 연산 구조** - 일반적인 추론 연산 구조에서는 CPU가 데이터를 가속기에 송신하고, 추론 연산을 실행시킨 후 결과를 수신하는 방식으로, CPU의 스케줄링에 의존 - DOCA DMA를 활용한 데이터 처리 장치와 가속기의 직접 통신 구조에서는 그림과 같이 CPU의 개입 없이 가속기에 데이터를 전달하고, 가속기가 직접 데이터 수신 여부를 판단하여 추론을 진행함 - Volatile 변수를 사용하여 DOCA DMA에 의해 변경되는 값을 읽고, 원자적 연산 대신 입력 헤드 포인터와 출력 테일 포인터를 폴링하여 데이터를 처리함 - TorchScript 모델을 사용하여 데이터를 처리하고, 입력 텐서를 TorchScript 모델에 전달하여 출력을 출력 버퍼에 저장 - 이미지넷 데이터셋을 사용하여 ResNet152 모델에 대한 추론 연산을 수행하며, 데이터 크기는 3×224×224로 리사이징하여 약 0.574MB로 설정됨  그림 533. CPU 개입 최소화 추론 연산구조 전체 동작도 - 모델은 TorchScript로 직렬화 및 최적화 후, CUDA Graph를 적용하여 추론 성능을 향상시킴 **- 연구 결과** **· 데이터 처리 장치 중심 네트워크 데이터 처리 구조 성능 실험 결과** **(1) 실험개요** - 데이터 처리 장치 중심 구조에서 CPU 메모리 접근과 가속기 메모리 접근의 성능 비교 - 블루필드(2, 3) 서버 2대 사용 : 클라이언트 노드(분리된 호스트 모드)와 서비스 노드(데이터 처리 장치 모드) - 비교 조건 : CPU 메모리 전송 구조와 가속기 메모리 전송 구조 성능 비교 - 데이터 처리 장치는 프로세서 개입 없이 데이터 전송 수행 - 클라이언트는 인피니밴드 통신이 가능한 네트워크 인터페이스로 대체 가능

수행기관	개발 목표	개발내용 및 범위

표 57 실험 서버 환경

	BlueField-2 서버	BlueField-3 서버
OS	Ubuntu 20.04.6 LTS	Ubuntu 22.04.4 LTS
CPU	Intel® Xeon® CPU E5-2620 v4	AMD EPYC 9554 64-core Processor
Memory	128 GB	512 GB
Network	BlueFiled-2	BlueField-3
GPU	A100 40GB	

(2) 블루필드와 GPU 간 직접통신 처리량

- 블루필드를 사용하여 DPU <-> GPU 1GB 데이터 전송 시간 측정
- 1GB 랜덤 데이터를 DPU 메모리에 할당한 후 전송
- 그림과 같이 DPU와 GPU 간에 DOCA DMA를 사용하여 데이터 송수신
- 첫 번째 데이터를 수신하고 마지막 데이터를 수신하기까지의 시간 측정

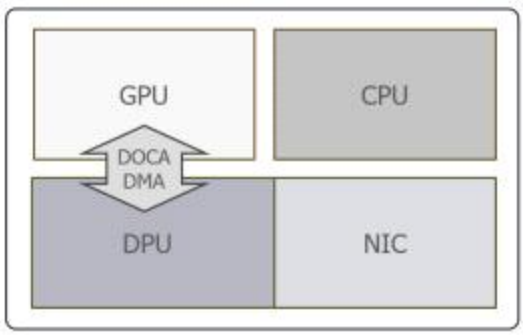


그림 534 블루필드-GPU 직접통신 실험 구조도

표 58 블루필드-GPU 직접통신 실험 결과

버퍼 크기	서버	GPU->DPU (Gbps)	DPU -> GPU (Gbps)
2MB	BlueField-2	14.86	17.79
	BlueField-3	31.63	36.43
이미지넷	BlueField-2	12.95	14.25
	BlueField-3	25.55	30.56

(3) End-to-End 처리량

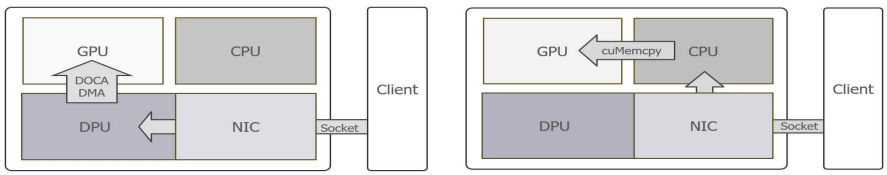


그림 535 데이터 처리 장치 중심 네트워크 오프로딩 구조(왼쪽), CPU 중심 네트워크 처리 구조(오른쪽)

- 클라이언트 -> GPU 1GB 데이터 전송 시간 측정
- 1GB 랜덤 데이터를 클라이언트 메모리에 할당한 후 경로에 따른 전송(GPU 메모리)
- 첫 번째 데이터를 수신하고 마지막 데이터를 수신하기까지의 시간 측정

수행기관	개발 목표	개발내용 및 범위

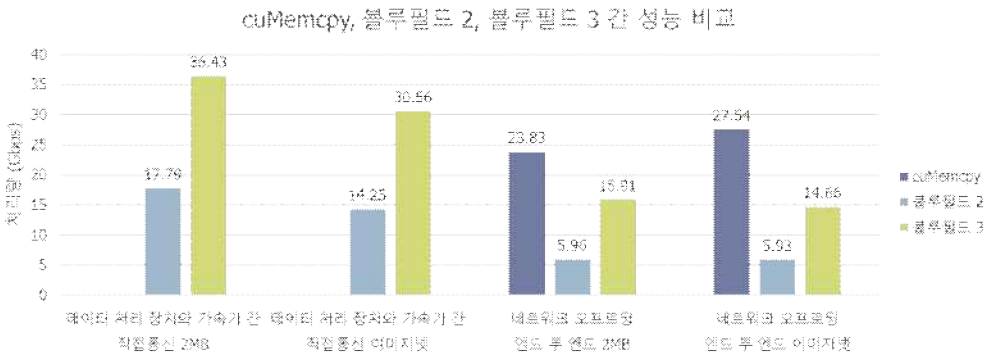


그림 536 직접통신 및 End-to-End 성능 측정 결과

- 블루필드-2의 경우 클라이언트 노드가 서비스 노드의 데이터 처리 장치로 전송할 때 네트워크 대역폭이 5.5 Gbps로 측정
- 네트워크 대역폭이 제한되는 요소로는 데이터 처리 장치의 프로세서와 메모리 성능의 한계 존재
- 블루필드-3는 블루필드-2에 비해 코어 개수, L1 및 L3 캐시 크기, 메모리 클럭 속도 등에서 모두 뛰어난 성능을 보임
- 실험 결과, 블루필드-3는 블루필드-2 대비 처리량이 약 2배 증가
- End-to-End의 경우, CPU 중심 구조 대비 DPU 중심 구조는 절반 수준임

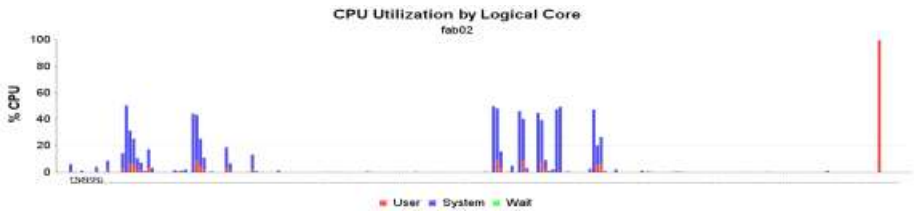


그림 537 CPU 중심의 네크워크 처리구조 코어 사용량

그림 538 DPU 중심의 네트워크 처리구조 코어 사용량

- CPU 코어의 수가 220개로 프로세서의 성능이 데이터 처리 장치 대비 현저하게 높음
- 본 연구는 다수의 프로세서가 사용되고 있는 환경에서 가용 CPU 자원을 최소로 사용하는 것을 목표이므로 CPU 코어 수를 제한하여 실험 수행
- Grup을 통해 단일 코어를 사용하도록 제한한 결과, CPU 중심의 네트워크 처리 구조는 20.37 Gbps 처리량, 평균 98.99%의 코어 사용량을 보인 반면, 네트워크 오프로딩 구조(블루필드-3)는 16.01 Gbps 처리량, 평균 1.19%의 코어 사용량을 보임

(3) Inference 실험

- 본 연구의 목표인 네트워크 구조에 따른 추론 성능 비교 실험 수행
- 1 클라이언트가 3000장의 이미지를 반복해서 전송
- DPU는 polling thread 1개 네트워크 처리 thread 1개 실행

수행기관	개발 목표	개발내용 및 범위

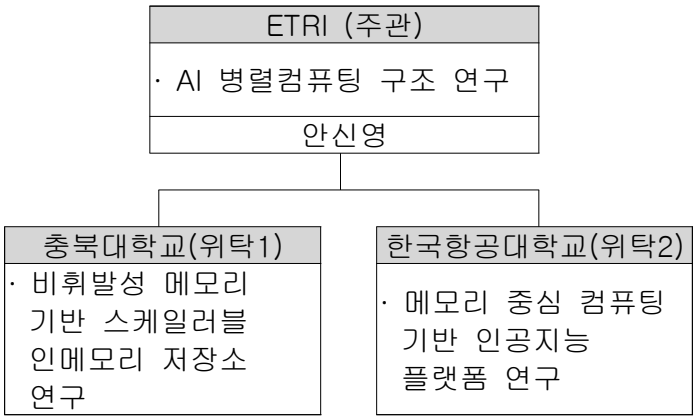
표 59 CPU 및 DPU(블루필드-3) 중심의 네트워크 구조의 추론 성능

네크워크 구조	CPU Core 수	100장 추론 시간 (평균)	GPU 사용률
CPU 중심 네트워크 처리 구조	All	0.35	97%
	1	0.66	52%
DPU 중심의 네트워크 오프로딩 구조	All	0.34	99%
	1	0.34	99%

- 추론 성능의 경우 전체 CPU 코어를 사용하더라도 DPU 중심의 네트워크 오프로딩 구조에서 추론시간 및 GPU 사용률 측면에서 높은 성능을 보임
- CPU 코어 수를 제한한 경우에는 기존 대비 성능 격차가 크게 나타남

(4) 연구 결과 요약

- CPU 개입 없이 네트워크 처리가 가능한 DPU 중심의 네트워크 오프로딩 구조 개발
- DPU와 GPU 간 직접통신 기술을 통해 DPU에서 처리된 데이터가 CPU 개입 없이 GPU로 전달
- 추론 프로세스에서의 CPU 개입을 줄여 추론에서 발생하는 CPU 병목을 최소화



(6) (3단계) 1차년도(2025년도) 개발 내용 및 범위

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td>ETRI
(주관)</td><td>이종 GPU 기반 AI 병렬 학습 연구</td><td>

(1) AI 모델 학습 환경 변화 분석

● 딥러닝 모델의 크기가 기하급수적으로 증가함에 따라 단일 GPU로는 학습 수행이 불가한 수준에 도달

- 2017년에 제안된 초기(2017년) 트랜스포머 모델은 약 6,500만 개의 파라미터 규모였으나, 이후 모델 크기가 증가하며 GPT-3는 1,750억 개 파라미터에 이르는 거대 모델로 발전함
- 최근에는 Llama 3 405B, Gemini 1.5, GPT-4 계열 등 수백억~수천억 파라미터급 모델이 일반적이며, 일부 초거대 모델은 1조(Trillion) 파라미터 이상까지 확대됨

● 대규모 언어 모델(Large Language Model, 이하 LLM)은 모델 규모(매개변수 수)가 증가할수록 추론 능력, 문맥 이해, 일반화 등의 정확성과 정교함이 향상되는 것으로 알려짐

- 모델의 크기가 커질수록 고급 추론, 다국어 처리, 코드 생성 등의 복잡한 작업을 수행할 수 있으며, 이에 대한 성능도 높아지지만 계산 비용, GPU 자원, 지연시간 등이 크게 증가함

표 60 모델 사이즈 구간 별 주요 특징

<table>
<tr><th>모델 규모</th><th>활용 범위(대표적 작업)</th><th>성능</th><th>한계(trade-off)</th></tr>
<tr><td>1-3B</td><td>간단한 자연어 처리,
임베딩, 모바일 추론</td><td>빠른 처리, 제한된
추론 능력</td><td>얕은 문맥 이해</td></tr>
<tr><td>7-13B</td><td>일반 대화, 요약,
질의 응답</td><td>균형 잡힌 성능</td><td>중간 수준의 연산
비용</td></tr>
<tr><td>30-70B</td><td>고급 추론, 다국어 처리,
코드 생성</td><td>높은 정확도</td><td>엔터프라이즈급
GPU 필요</td></tr>
<tr><td>100B+</td><td>멀티 모달,
연구 수준 모델</td><td>최고 성능</td><td>매우 높은 비용과
지연시간</td></tr>
</table>

출처 : LABEL YOUR DATA, LLM Model Size: Comparison Chart & Performance Guide (2025)

● 매년 대규모 병렬 연산에 강점을 갖는 GPU의 새로운 세대 제품이 출시됨에 따라, 분산학습 인프라는 서로 다른 세대의 성능과 특성이 상이한 GPU가 혼재되는 환경이 증가하고 있으며, 이에 기투자된 GPU 자원의 활용을 극대화할 필요성이 증가하고 있음

● 초거대화된 LLM을 위한 학습에서 3D 분산학습은 선택이 아닌 필수적인 학습 방식으로 자리를 잡음

- 3D 분산학습은 매우 큰 모델이나 대규모 데이터 셋을 효율적으로 학습하기 위해 데이터 병렬화, 파이프라인 병렬화, 텐서 병렬화의 세 가지 주요 병렬화 기법을 조합하여 사용하는 접근 방식임

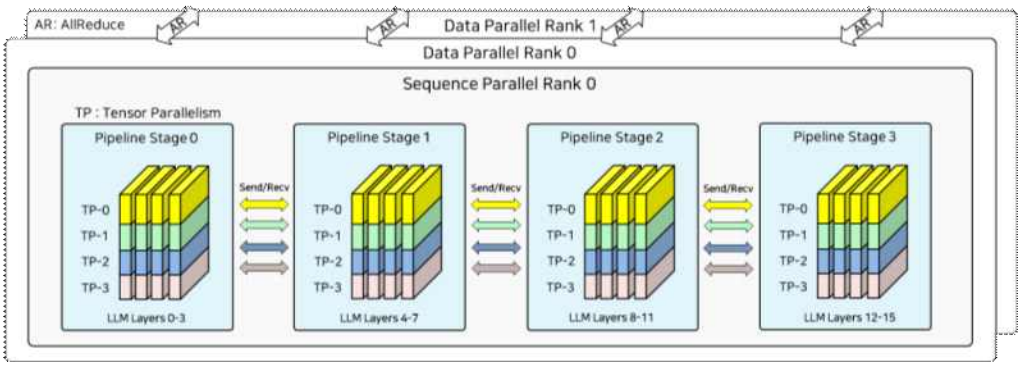


그림 539 대규모 모델 3D 분산학습(3-Distributed Training) 전략

</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		- 데이터 병렬화는 전체 데이터셋을 여러 노드로 나누어 병렬 학습을 수행함으로써 학습 속도를 높일 수 있으나, 기본적인 데이터 병렬화 방식에서는 모든 노드가 동일한 모델 전체를 복제하기 때문에 모델 크기가 단일 GPU 메모리를 초과할 경우 학습이 불가함 - 이를 해결하기 위해 제안된 텐서 병렬화와 파이프라인 병렬화 방식은 모델을 여러 GPU에 분할하여 학습하지만, 이 과정에서 GPU 간 통신 병목이 발생하고, 순차 처리로 인한 GPU 유휴 시간(bubble 현상)이 불가피함 - 데이터 병렬화가 가지는 한계점을 극복하는 ZeRO(Zero Redundancy Optimizer)는 파라미터, 그래디언트, 옵티마이저 상태를 GPU 간에 분산 저장함으로써 메모리 중복을 제거하여 기존 데이터 병렬화의 메모리 제약을 완화하고, 대규모 모델 학습의 확장성을 높임 - 이러한 ZeRO의 설계 개념은 PyTorch의 FSDP(Fully Sharded Data Parallel)로 발전하여 파라미터 샤딩 기반의 효율적인 분산학습을 더욱 손쉽게 구현할 수 있게 됨 - 하지만, FSDP는 균일한 GPU 성능과 초고속 네트워크 환경(예 : 2Tb/s급)을 전제로 하기에, 이기종 GPU 클러스터나 일반적인 InfiniBand 네트워크 환경에서는 동일한 효율을 보장하기 어려움 - 구글, Meta 등과 같은 데이터센터의 규모가 아닌 일반 연구기관과 사업체의 경우에는 FSDP를 대상으로 대규모 언어 모델을 학습하기 위한 자원 구성이 어려움 ● 거대 모델에 대한 3D 분산학습에서 모델 구조와 데이터 특성뿐 아니라 이종 GPU 클러스터 환경을 고려한 병렬화 전략의 조합이 성능과 효율성 향상의 핵심 요소임 (2) EDDIS(Efficient Distributed Deep learning lIbrary based on SoftMemoryBox) 분석 ● 이종 GPU 클러스터 환경에서도 효율적인 분산학습을 수행할 수 있는 프레임워크로 개발됨 ● 기존 분산학습 프레임워크와 달리, 모든 학습 프로세스가 동일한 속도로 파라미터를 교환해야 하는 완전 동기식 구조의 제약에서 벗어나, 비동기 및 부분 동기식 학습 방식을 지원함으로써 GPU 성능이 상이한 환경에서도 느린 노드로 인한 병목 현상을 완화함 ● 핵심 구성 요소인 SoftMemoryBox(SMB) 시스템은 기존 ShmCaffe 및 TensorFlow 기반 SoftMemoryBox의 기술을 계승하여 확장한 것으로 노드 간 비동기 통신 기반의 효율적인 메모리 공유 및 파라미터 갱신을 제공 ● 현재 EDDIS 구조에서는 DDP(Distributed Data Parallel)와 동일하게 모델 전체를 GPU 메모리에 적재하는 방식이었기 때문에 거대 모델을 학습할 수 없으므로, EDDIS가 LLM 모델을 학습할 수 있도록 확장 필요 ● EDDIS 활용 학습에서는 각 서버에 EDDIS와 SMB 환경을 구축한 후 MPI runner를 통해 학습을 수행하는 방식을 사용하나, 서버별 상태가 상이한 환경에서 이를 일관되게 구성하는 데 어려움이 있으므로 EDDIS의 손쉬운 사용을 위한 환경 구축 수단의 제공이 필요함 (3) 대규모 모델 학습 지원을 위한 학습 모드 개발 ● EDDIS 학습 모드 - 기존 EDDIS는 동기식(SYNC), 비동기식(ASYNC), 부분 동기식(HSYNC)의 3가지 학습 모드를 지원하였음 - EDDIS의 모델 크기 제약을 해소할 수 있게 하려고 EDDIS에 FSDP 알고리즘을 통합하여 새로운 학습 모드인 FSYNC 모드 추가 개발 - 각 학습 모드의 동기화 수행 방법 • SYNC 모드 : 기존 분산 학습 프레임워크와 동일한 구조인 완전 동기식(synchronous) 방식으로, 각 노드의 학습 파라미터를 All-Reduce 방식으로 동기화함 • ASYNC 모드 : 비동기식(Asynchronous) 방식으로, 각각의 파라미터 결과를 SoftMemoryBox에 읽고 쓰며 비동기적으로 파라미터를 업데이트함 • HSYNC 모드 : 동기식과 비동기식을 결합한 하이브리드 방식으로, 서버 내 통신에서는 동기식을 따르고 서버 간 통신은 비동기식 업데이트를 수행함 • FSYNC 모드 : 그레디언트 평균화를 노드 내 클러스터에서 수행, 가중치 업데이트는 샤드별로 비동기식으로 수행함

수행기관	개발 목표	개발내용 및 범위

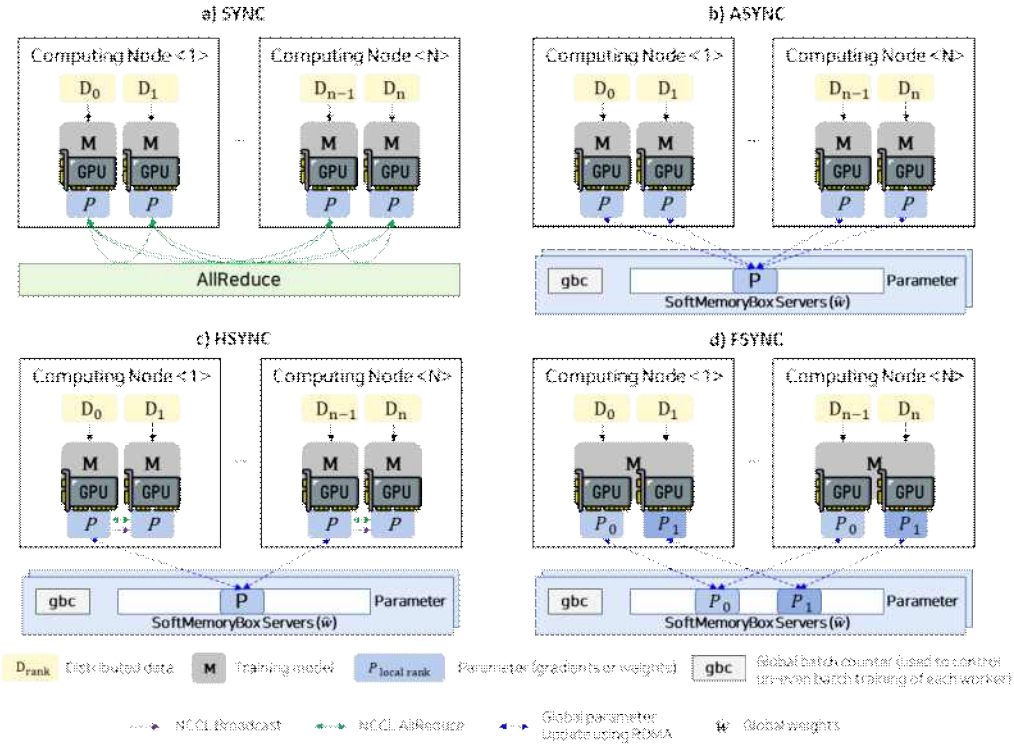


그림 540 EDDIS의 학습 모드 : SYNC, ASYNC, HSYNC, FSYNC

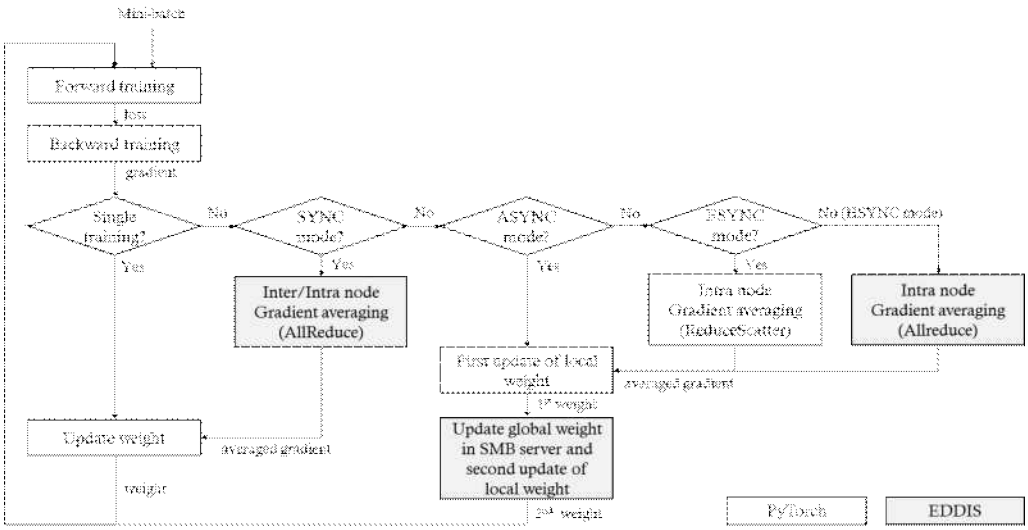


그림 541 학습 모드별 Pytorch/EDDIS에서의 흐름

- EDDIS FSYNC 모드 개발
  - DDP와 FSDP의 한계점
    - DDP는 각 프로세스마다 전체 모델을 메모리에 할당해야 하므로, 단일 GPU의 vRAM 사이즈가 담을 수 있는 모델만 학습 가능 (RTX 2080 : 11GB, LLM 학습 불가)
    - FSDP는 인트라노드(Intra-Node)에서 모델 샤딩을 통한 훈련 지원으로 모델 전체를 로드하지 않고 학습이 가능하나, 집합통신(all-gather, reduce-scatter 등) 호출이 잦아 인터노드(Inter-Node) 환경에서 통신 병목현상 발생하여 노드 확장성 측면에서 부담이 큼

수행기관	개발 목표	개발내용 및 범위

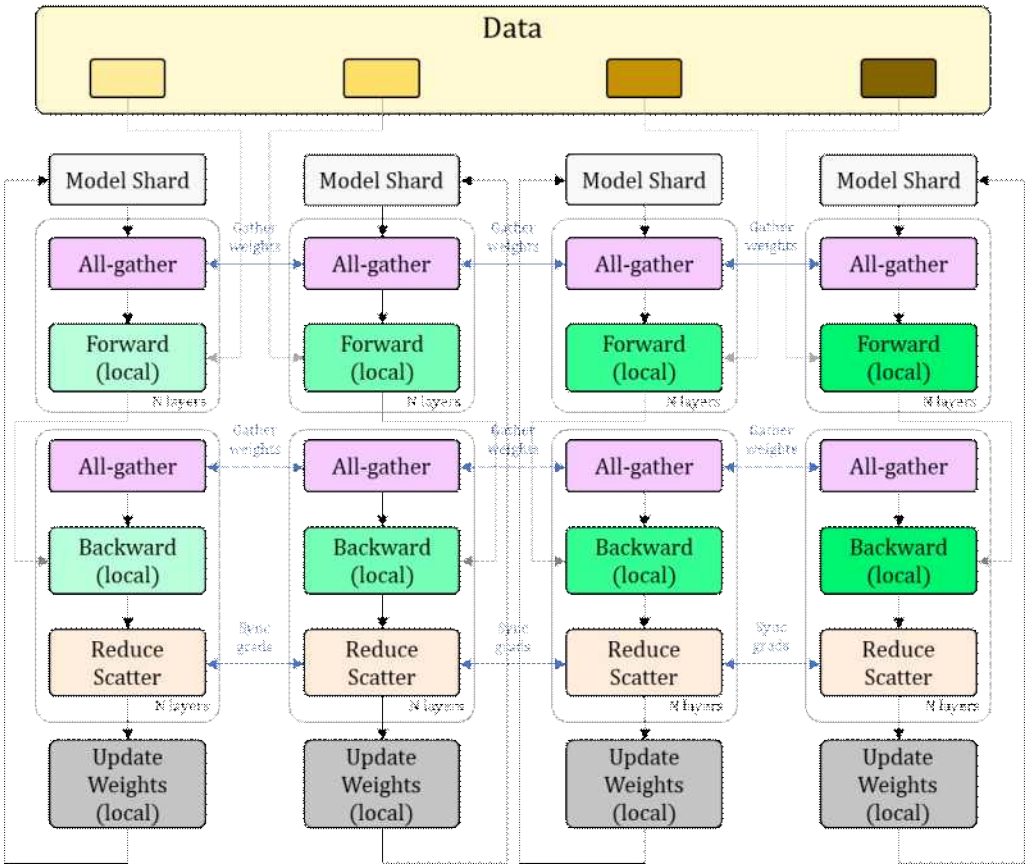


그림 542 FSDP 분산학습 구조

- FSDP와 EDDIS 비동기식 학습 모드를 결합한 FSYNC 모드
  - Intra Node 단위에서 집합통신 호출, Inter Node 간에는 SMB를 활용
  - 노드 확장성과 대규모 모델 지원 가능
- 샤딩된 모델로 학습된 Weight 대상으로 Local Update 후 SMB 버퍼 단위 업데이트 수행

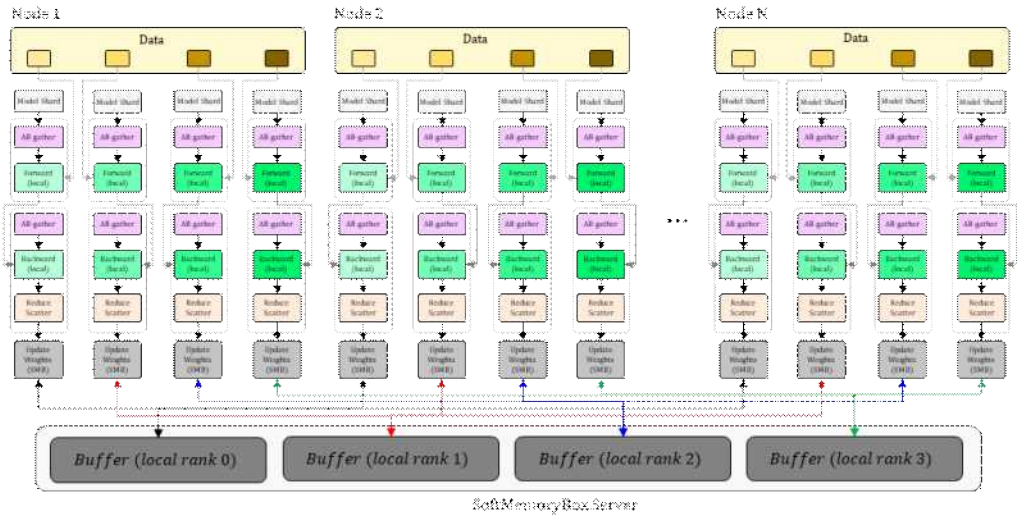
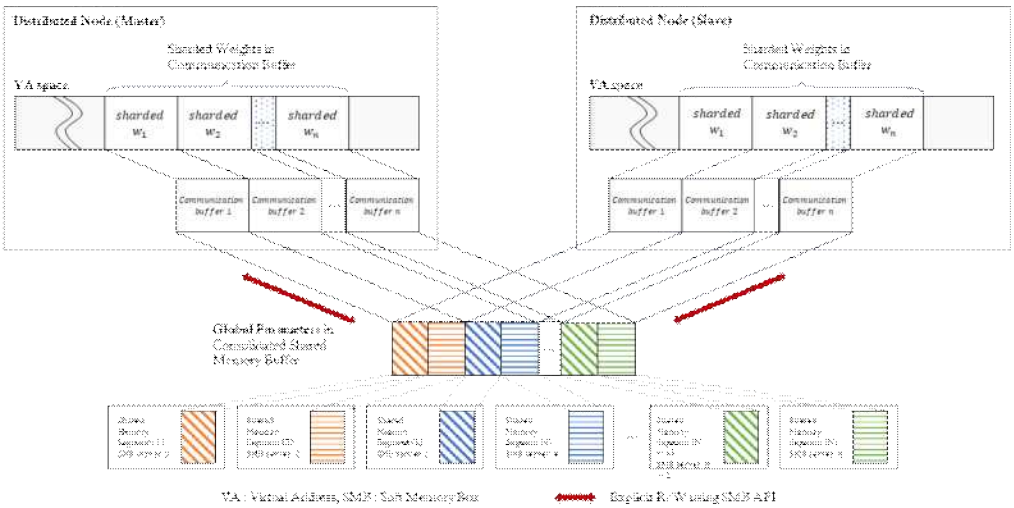
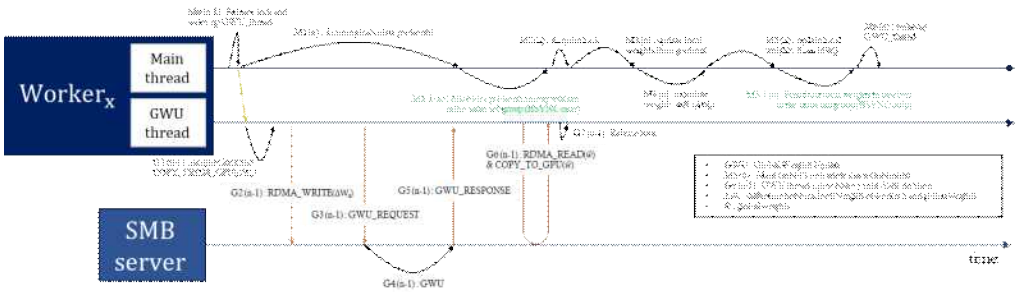
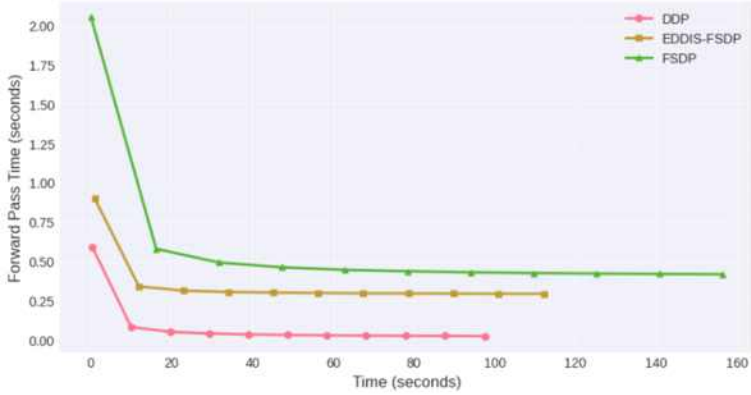


그림 543 EDDIS FSYNC 모드 분산 학습 구조

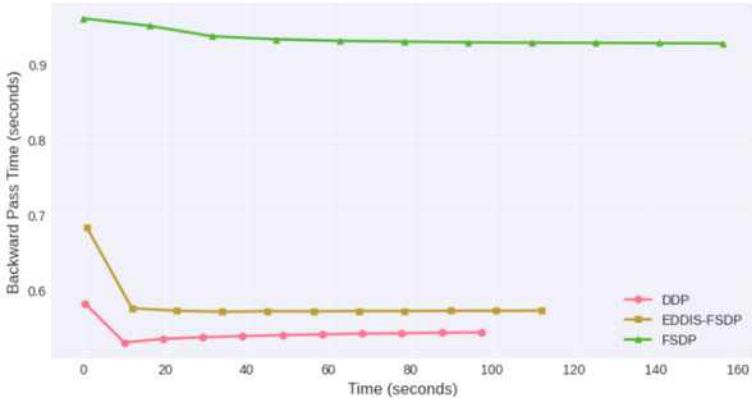
- SMB는 Local Rank 개수만큼 Buffer를 할당하여 해당되는 Sharding Weight를 비동기식으로 관리

수행기관	개발 목표	개발내용 및 범위
		- SMB로부터 업데이트된 Weight로 다음 batch 수행 - SMB의 경우 다중 노드로 설정하여 통신 경합 방지  그림 544 FSYNC 모드에서 SMB로의 데이터 분산 - FSYNC 학습 모드에서의 가중치 업데이트를 위한 interaction • 완전한 가중치 업데이트 병렬화를 위해 각 worker는 Main thread와 Global Weight Update(GWU) thread를 구분하여 실행 • GWU thread와 SMB 간 InfiniBand RDMA atomic 연산으로 균일하지 않은 batch를 제어 • 로컬 학습과 글로벌 가중치 업데이트를 각기 다른 thread가 관리함으로써 연산과 통신을 병렬적으로 수행하도록 함  그림 545 Full Parallel Weight update 구조 ● FSYNC 학습 모드 성능 실험 - 실험 1 : Step별 소요시간 비교 (DDP, FSDP, EDDIS) • 2개의 노드에 각각 A100 4개와 H100 4개를 장착하고, Llama3.2 1B Base 100 step 기준 성능 분석 • 그림 546를 참고하면 분산병렬처리 형태에 따른 소요시간은 DDP, EDDIS, FSDP 순으로 나타남 • Forward pass에서는 각 step별로 분산병렬처리 형태별 소요시간 격차가 적으나, Backward pass에서 FSDP의 지연시간이 크게 나타남 • DDP에서는 gradient만 all-reduce하는 과정이지만, FSDP는 weight shard가 다르므로 gradient도 shard로 reduce-scatter가 레이어마다 수행되어야 함 • 반면, EDDIS에 적용한 FSDP의 경우, reduce-scatter 통신이 intra-node에 한정하기 때문에 상대적으로 소요시간이 적게 나타남

수행기관	개발 목표	개발내용 및 범위



(가) Forward Pass Time Over Training



(나) Backward Pass Time Over Training

그림 546 Llama 3.2 1B 모델 Forward/Backward Pass 소요시간

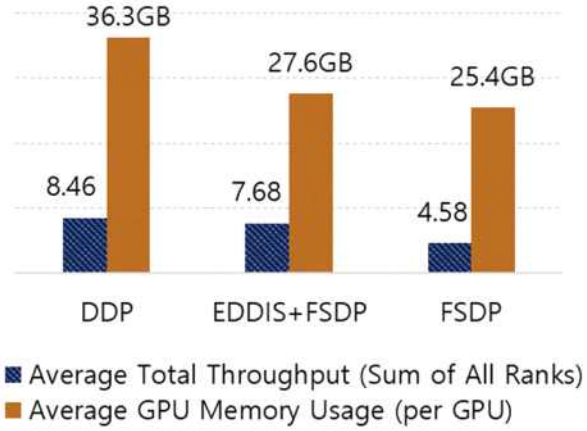


그림 547 각 DP 형태별 평균 처리량 및 GPU 메모리 사용량

- 그림 547은 DDP, EDDIS, FSDP 상에서의 처리량과 GPU 메모리 사용량을 나타냄
- 처리량 관점에서 DDP와 EDDIS는 큰 격차를 보이지 않지만, FSDP는 DDP 대비 40.36% 감소한 4.58을 보임
- 메모리 사용량 관점에서도 EDDIS와 FSDP 간에는 적은 격차를 보이지만 DDP에서는 EDDIS 대비 31.52% 많은 메모리를 사용

- 실험 2 : Bert 모델 1 epoch 소요시간 비교 (FSDP, EDDIS)
  - 실험 구성 : 단일 노드와 다중 노드 상에서의 학습 속도 비교 (A100, H100 사용)
  - 단일 노드에서의 FSDP 결과 기준으로 A100과 H100의 성능은 약 2배 차이

수행기관	개발 목표	개발내용 및 범위

- 2개의 노드 사용 시 프로세스별로 1 epoch 당 할당되는 Step 수는 절반으로 줄어듦
- 확장성 측면에서 노드 수가 증가할수록 학습 속도가 증가하는 것이 일반적이나 집합 통신 등의 병목현상으로 학습 속도가 현저하게 줄어들 수 있음
- 일반 FSDP의 경우, step 수가 줄어듦에도 불구하고 A100 단일 노드 학습 대비 3.24배 느린 학습 속도를 보인 반면, EDDIS를 적용하였을 때 2배 더 빠른 학습 속도를 보임(즉, 기존 FSDP 대비 EDDIS FSYNC는 6.22배 빠른 학습을 수행함)

표 61 Bert FSDP, EDDIS FSYNC 학습 소요시간

Experiment	GPU Config	Epoch	Steps	Sharding Model Size	Elapsed time	ratio
FSDP (1 Node)	A100 x 4	1	112896	12.8 GiB	7h 15m	1.00×
FSDP (1 Node)	H100 x 4	1	112896	12.8 GiB	4h 05m	0.56×
FSDP (2 Node)	(A100, H100) x 4	1	56448	6.4 GiB	23h 31m	3.24×
EDDIS FSYNC	(A100, H100) x 4	1	56448	6.4 GiB	3h 47m	0.52×

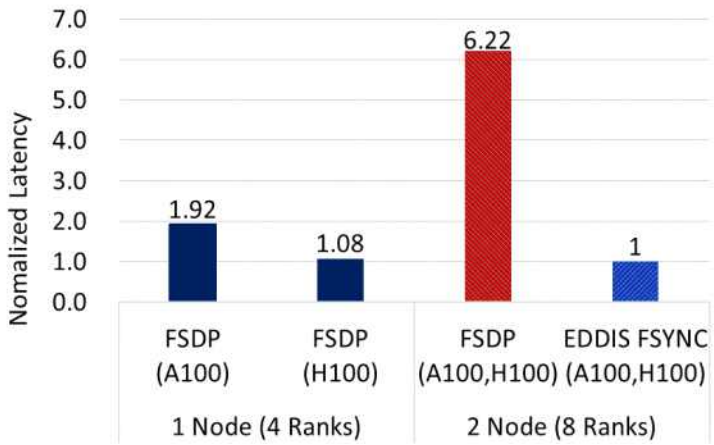


그림 548 각 DP 형태별 평균 처리량 및 GPU 메모리 사용량

- EDDIS FSYNC 모드는 FSDP에 비동기 파라미터 업데이트 메커니즘을 적용하여 기존 FSDP가 가진 통신 병목 문제를 완화함으로써, 이기종 GPU 간의 계산 성능 차이를 보완하면서도 높은 확장성과 효율성을 달성함
- EDDIS는 이기종 GPU 자원을 효과적으로 통합하여 활용할 수 있는 분산학습 프레임워크로서, 기존 동기식 학습 구조의 제약을 극복하고 InfiniBand 기반의 일반적인 클러스터 환경에서도 대규모 모델 학습을 안정적이고 효율적으로 수행함

(4) EDDIS를 위한 K8S 기반 DLOps 개발

- EDDIS 사용자가 사용하고자 하는 모델 설정(Model Config)과 데이터셋을 제출하면 쿠버네티스 Launcher가 기존 PyTorch코드와 EDDIS Wrapping 코드를 구성하여 분산학습을 수행할 수 있도록 편의성을 제공하는 것을 목표로 K8S 기반 DLOps를 개발함

수행기관	개발 목표	개발내용 및 범위

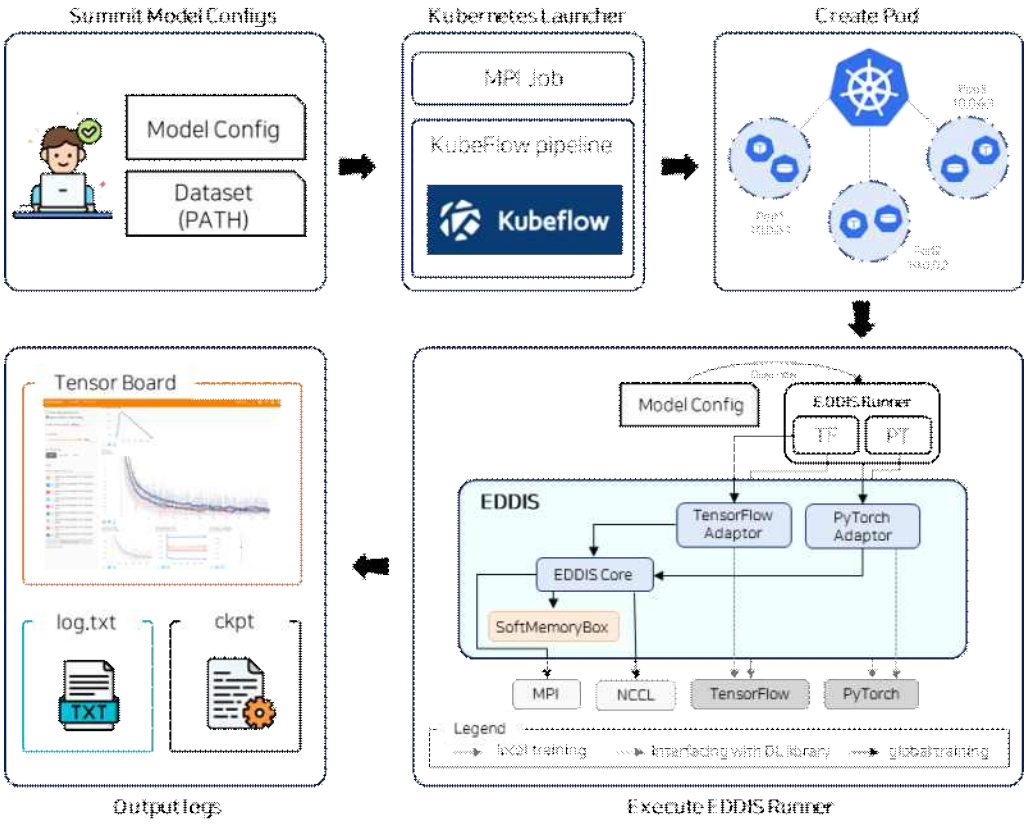


그림 549 EDDIS를 위한 K8S 기반 DLOps 구조

● 모델 설정 제출(Summit Model Configs)

- EDDIS를 통한 모델 학습 설정을 쉽게 표현하기 위한 모듈화된 구조를 제공

```
python3 eddis_run.py --config bert_config.py \
        --override \
        num_cpu_threads=16 \
        output_dir=/data/bert_eddis \
        epochs=3 \
        preprocessed_data_dir=/data/preprocessed_wikipedia_en
```

```
.
├── configs
│   ├── base_config.py
│   ├── bert_config.py
│   ├── __init__.py
│   └── llama_config.py
├── eddis_run.py
```

그림 550 EDDIS 실행 방식 (Bert 모델 예시)

- EDDIS를 통한 학습 스크립트(eddis_run.py)를 실행을 위해 각 모델에 맞는 설정을 인자로 전달
  - --config : 모델 학습을 위한 정보를 포함한 설정 파일(예, bert_config,py)의 경로
  - --override : 해당 설정에서 오버라이드할 항목들(예, num_cpu_threads=16)
- 설정 파일은 base_config.py에 정의된 클래스를 상속받아 모델, 데이터셋, 옵티마이저와 관련된 내용을 정의함(예, bert_config.py)
  - base_config.py : 학습 수행에 필요한 공통 설정을 BaseConfig 클래스로 추상화하여 일관된 구조를 제공
  - training, eddis, hardware, logging, checkpointing 등의 핵심 파라미터 설정을 제공하며, 필요시 모델별 특화 파라미터를 추가할 수 있음
- 설정 파일을 기반으로 EddisRunner 클래스가 생성됨

● 쿠버네티스 런처(Kubernetes Launcher)

- 쿠버네티스 상에서 머신러닝 워크플로우를 쉽게 구축하고 배포, 관리할 수 있도록 하는

수행기관	개발 목표	개발내용 및 범위

Kubeflow를 활용하여, 컨테이너 기반의 확장 가능하고 이식성 높은 ML 스택을 구성

- Kubeflow의 핵심 구성요소인 Kubeflow Pipelines와 Trainer/MPI Operator를 활용하여 분산학습 작업을 효율적으로 관리하며, PyTorch Distributed 및 MPI Job 등을 지원함
- Kubeflow Pipelines는 ML 워크플로를 DAG(Directed Acyclic Graph) 형태로 구성하여, 실험을 손쉽게 재현 가능하도록 지원
- 각 단계를 컨테이너로 캡슐화함으로써, 단계 간 의존성(예, 데이터 전처리 → 학습)을 자동으로 관리함

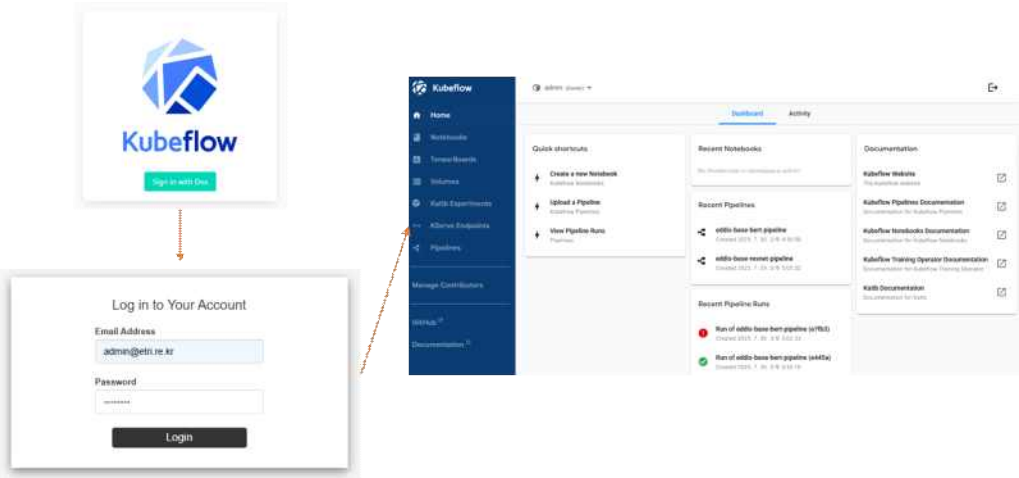


그림 551 Kubeflow 콘솔 접속 과정

- 그림 550은 EDDIS 파이프라인 생성 및 관리하는 Kubeflow Console 접속 과정으로, 사전에 구성한 Kubernetes Cluster의 Worker 노드 중 하나에 접속(예, 10.11.12.13:31000)
- EDDIS에 대한 Kubeflow Pipeline을 자동으로 생성하는 Generator(Kubeflow Pipeline Generator)를 만들어, 사용자가 요구하는 특정 설정값만 조정하여 실행시킬 수 있도록 함
  - Kubeflow Pipeline의 Python + YAML의 조합으로 Component를 선언하여야 하는 복잡한 과정을 단순화시킴
  - 그림 552과같이 Kubeflow Pipeline 생성을 위한 코드(generate_pipeline.py)를 실행하면 Kubeflow Pipeline YAML 파일이 생성되고, 생성한 YAML 파일을 Kubeflow 상에 업로드하면 실행할 수 있는 파이프라인이 생성됨

```
python3 generate_pipeline.py \
  --pipeline-name eddis-fsync-bert-pipeline \
  --kubeflow-profile admin \
  --script-path /dlss/eddis/eddis_run.py \
  --data-path /pssd \
  --nodes 1 \
  --gpus-per-node 4 \
  --launcher-node-selector "dl005" \
  --node-selector "dl005,dl006,dl007,dl008,dl009" \
  --max-pods-per-node 1 \
  --shm-size 32Gi \
  --worker-cpu 16 \
  --worker-memory 32Gi \
  --script-args "--config /dlss/eddis/configs/bert_config.py \
                 --override \
                 num_cpu_threads=16 \
                 output_dir=/data/bert_eddis \
                 epochs=1 \
                 preprocessed_data_dir=/data/preprocessed_wikipedia_en"
```

그림 552 Kubeflow Pipeline Generator 실행 예

- 파이프라인을 실행하면 일괄적으로 컨테이너(pod) 생성 및 할당한 작업을 순차적으로 수행되며, 실시간으로 실행 로그를 확인 가능

수행기관	개발 목표	개발내용 및 범위

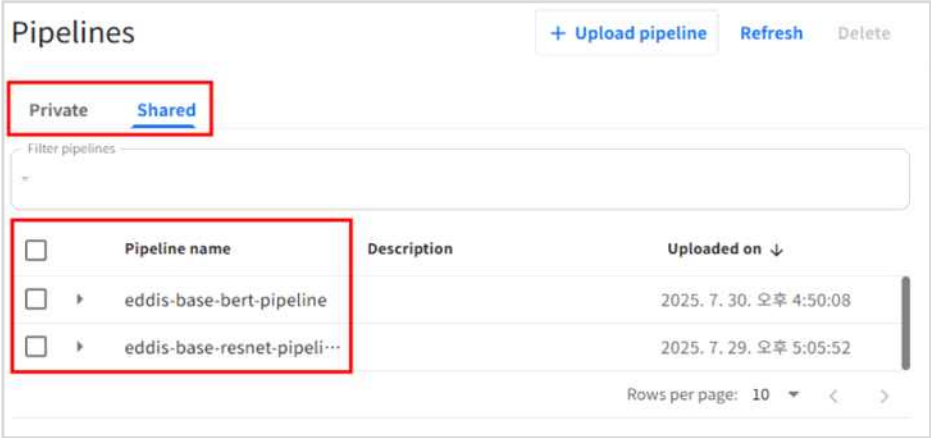


그림 553 Pipeline 생성 화면

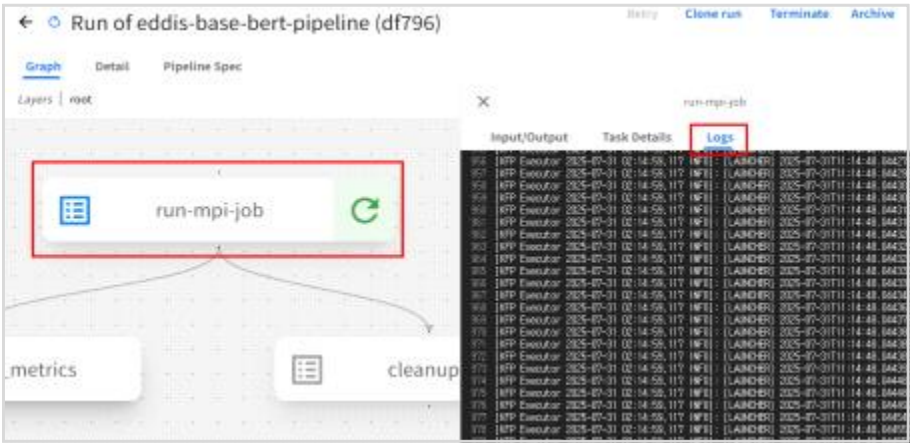
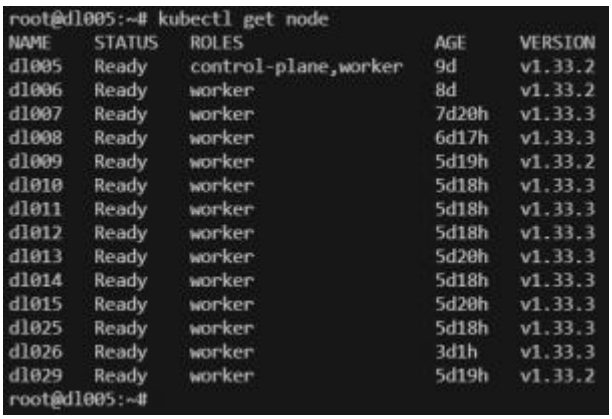


그림 554 Pipeline 실행 화면

- Pod[3] 생성(Create Pod)
  - 기존 EDDIS 실행의 경우는 mpirun 실행 명령어로 직접 노드를 지정하고 환경을 서버 레벨에서 맞추어야 하는데, 이를 베어메탈에서 컨테이너 환경으로 전환함으로써 이기종 서버 환경에서도 의존성 문제없이 일괄적으로 실행 가능(확장성 및 유연성 확보)
  - 이전 단계에서 Pipeline 작업이 전달되면서 Kubernetes 클러스터 상에 배포됨



```
root@dl005:~# kubectl get node
NAME     STATUS   ROLES                  AGE     VERSION
dl005    Ready    control-plane,worker   9d      v1.33.2
dl006    Ready    worker                 8d      v1.33.2
dl007    Ready    worker                 7d20h   v1.33.3
dl008    Ready    worker                 6d17h   v1.33.3
dl009    Ready    worker                 5d19h   v1.33.2
dl010    Ready    worker                 5d18h   v1.33.3
dl011    Ready    worker                 5d18h   v1.33.3
dl012    Ready    worker                 5d18h   v1.33.3
dl013    Ready    worker                 5d20h   v1.33.3
dl014    Ready    worker                 5d18h   v1.33.3
dl015    Ready    worker                 5d20h   v1.33.3
dl025    Ready    worker                 5d18h   v1.33.3
dl026    Ready    worker                 3d1h    v1.33.3
dl029    Ready    worker                 5d19h   v1.33.2
root@dl005:~#
```

그림 555 Kubernetes 클러스터 구성

수행기관	개발 목표	개발내용 및 범위

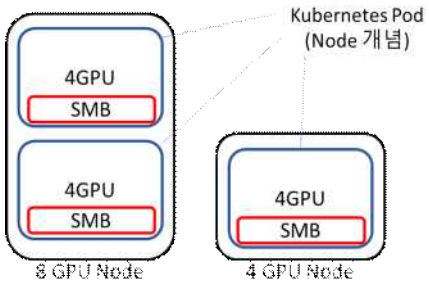


그림 556 Kubernetes Pod 생성 형태

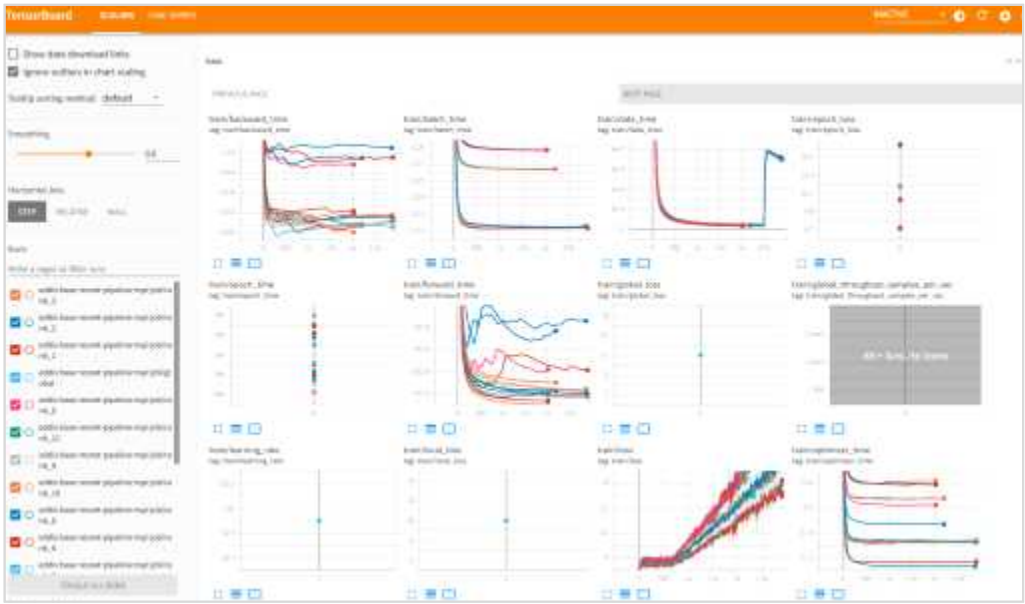
- 클러스터 상에 활성화 가능한 노드에 가용한 자원을 Pod 형태로 생성하여 노드의 개념으로 사용
  - 예를 들어, 서브그룹 단위가 4 GPU이며 서버 내 GPU 8대를 사용할 수 있는 경우, 2개의 pod를 생성하며 각 pod마다 SMB를 붙임
- 생성된 pod 상에 InfiniBand를 활용하기 위해 SR-IOV virtual function(VF) Configuration을 수행함
- 각각의 InfiniBand 인터페이스를 가상화하여 Container에 독립적으로 제공 가능

```
root@dl002:~# mstconfig -d 0000:0b:00.0 set SRIOV_EN=1

Device #1:
----------

Device type:    ConnectX6
Name:           MCX653105A-HDA_Ax
Description:    ConnectX-6 VPI adapter card; HDR IB (200Gb/s) and 200GbE;
Device:         0000:0b:00.0

Configurations:                                 Next Boot       New
         SRIOV_EN                               True(1)         True(1)
```

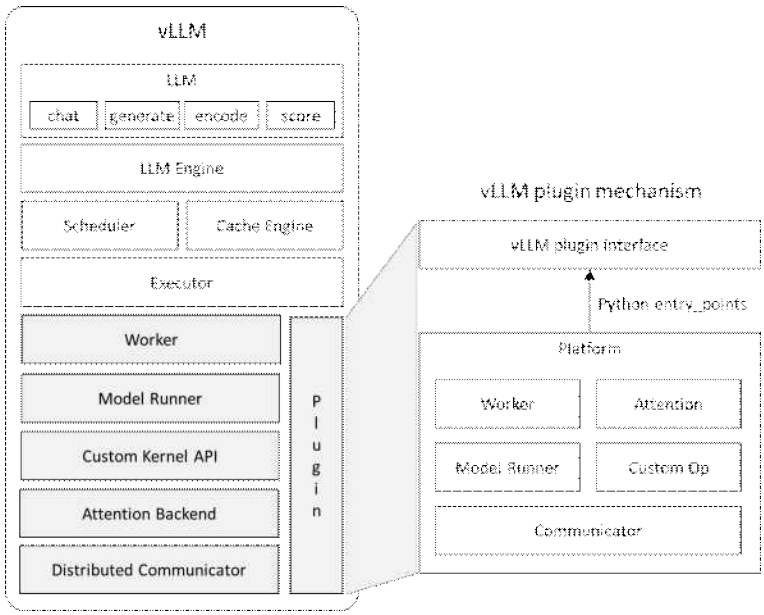
그림 557 SR-IOV 기능 활성화

- EDDIS Runner 실행
  - EDDIS Runner 구성
    - Adapter : 분산학습 프레임워크인 TensorFlow, PyTorch 등을 수정하는 API 제공
    - EDDIS Core : 이종 GPU를 활용한 분산학습을 위한 학습 부하 조정 및 SMB와의 글로벌 파라미터 업데이트를 관리해 주는 EDDIS 프레임워크 중심부
    - 제출된 모델 설정을 바탕으로 Adapter에서 EDDIS 코드 변환, 학습 모드에 따른 SMB 인프라(통신 토폴로지) 구성
  - EDDIS Runner 실행 과정
    - 인자로 받은 설정에 기반하여 train/validation Data를 불러와 EDDIS Replica Rank 별로 데이터를 나누어 훈련 data 구성
    - PyTorch nn.Module 형태의 모델을 생성하고 Optimizer, loss function, learning rate 등을 설정
    - DataGenerator를 사용해 Train Generator 구성
    - Epoch 만큼 학습을 수행하며, 주기적으로 Model Checkpoint 저장

- 결과 로그(Output logs)
  - EDDIS에서는 모델 학습 전과정에서 생성되는 각종 결과값을 기록함
  - 각 Pod별로 실행 로그(log), 학습 모델 체크포인트, 그리고 텐서보드 로그(TensorBoard log) 등을 남김
  - 텐서보드를 활용해 KubeFlow 상에서 실시간 모니터링 가능

수행기관	개발 목표	개발내용 및 범위
		 그림 558 Kubeflow Tensorboard 예
	이종 가속기 기반 AI 병렬 추론 연구	(1) 다양한 가속기 활용 관점에서 vLLM 분석 ● vLLM은 대규모 언어 모델 추론을 빠르고 효율적으로 서빙하기 위한 시스템으로, GPU 환경을 중심으로 개발됨 ● vLLM 주요 구성 요소 : 그림 559 참고 - LLM Engine : 사용자의 추론 요청을 받아 처리하는 흐름을 관리 - Scheduler : 추론 요청들을 효율적으로 묶어 배치를 구성하여 동시 처리가 최적화되도록 실행 순서를 관리 - Cache Engine : Attention 연산 중 Key, Value 텐서를 PagedAttention 기법을 사용하여 관리 - Executor : Scheduler 가 구성한 배치 정보를 기반으로 실행 계획을 수립하고, Worker에 연산을 할당하여 실행 - Worker : 각 가속기에서 실제 추론 작업을 실행 - Model Runner : Worker 내부에서 모델의 forward 연산을 수행 - Custom Kernel API : 각 하드웨어 플랫폼에 맞도록 최적화된 커널 연산을 추상화 - Attention Backend : 어텐션 연산을 실제로 수행 - Distributed Communicator : 여러 가속기나 노드 간 통신을 담당

3) Pod(파드, 포드, 팟) : 쿠버네티스가 생성하고 관리하는 가장 작은 컴퓨팅 단위로, 한 개 이상의 리눅스 컨테이너로 구성되며, 응용의 인스턴스가 실행되는 논리적 호스트
4) 파이프라인 병렬화(Pipeline Parallelism) : 모델을 여러 단계(stage)로 나누고, 각 단계는 별도의 장치(예, GPU)에 할당하여 처리
5) 텐서 병렬화(Tensor Parallelism) : 단일 레이어 내부의 텐서 연산 자체를 여러 장치(예, GPU)에 나누어 분산하여 처리
6) TTFT(Time To First Token) : 사용자가 LLM에게 입력을 주고 답변의 첫 토큰을 받을 때까지의 소요시간
7) TPOT(Time Per Output Token) : 하나의 output token을 생성하는데 소요된 시간
8) Patel, Pratyush, et al. "Splitwise: Efficient generative llm inference using phase splitting." 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA). IEEE, 2024.
9) Zhong, Yinmin, et al. "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving." 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)
10) Hu, Cunchen, et al. "Inference without interference: Disaggregate llm inference for mixed downstream workloads." arXiv preprint arXiv:2401.11181 2024.
11) Wu, Bingyang, et al. "Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism." (SOSP 24)
12) Qin, Ruoyu, et al. "Mooncake: Trading more storage for less computation—a KVCache-centric architecture for serving

수행기관	개발 목표	개발내용 및 범위
		 그림 559 vLLM 주요 구성 요소와 하드웨어 플러그인 메커니즘 ● vLLM에서는 GPU, CPU, TPU 등 다양한 하드웨어 백앤드(Backends)를 지원 - 각 백앤드는 자신을 위한 Worker, Model Runner, Attention, Communicator 등의 컴포넌트를 가짐 - 엔진 초기화 과정에서 사용 가능한 하드웨어를 파악한 후 이에 기반하여 해당 백앤드를 위한 컴포넌트(예, Worker)를 동적으로 로딩하여 실행 환경을 구성 - 백앤드 수가 증가할수록 코드 구조가 복잡해짐으로 인해 유지 보수 비용이 증가하고 확장의 어려움이 발생 ● 많은 가속기 하드웨어 밴더들은 vLLM을 자체적으로 확장하여 vLLM API 호환 소프트웨어 스택을 구성하여 LLM 추론 지원 - 확장이 vLLM 메인스트림에 통합되어 반영되지 않았기 때문에, vLLM 메인스트림이 업데이트될 때마다 해당 변경 사항을 주기적으로 통합하는 노력이 필요 - 각 가속기 하드웨어 밴더별로 지원하는 vLLM 버전이 다름 - 각 가속기에서 동작하는 vLLM 버전이 다르면 세부 연산(예, 정렬/정규화 등) 처리 방식이 다르기에 이종 가속기 통합 활용시 결과의 정확도를 보장하기 어려움 - 이종 가속기 통합 활용을 위해 각각 가속기에서 동작하는 vLLM 버전을 동일하도록 맞추는 것은 쉽지 않음 ● vLLM에서는 하드웨어 백엔드를 쉽게 확장할 수 있는 모듈화된 플러그인(Plugin) 방식 지원 : 그림 559 참고 - 파이썬 엔트리포인트(entry_points) 기반으로 플러그인 연동 - vLLM 0.9.x 버전부터 안정적으로 지원 - vLLM이 제공하는 플러그인 개발 가이드에 따라 하드웨어에 특화된 기능을 확장하여 vLLM과 연동 가능 - 여러 가속기 하드웨어(NPU) 개발 업체들에서 자사의 하드웨어와 vLLM을 연동하기 위한 플러그인을 제공 : 표 62 참고 • vLLM이 새로운 버전으로 업데이트될 때마다 해당 버전과 호환성을 유지하기 위해 플러그인도 지속적인 갱신 필요

LLM chatbot." 23rd USENIX Conference on File and Storage Technologies (FAST 25).

13) Jin, Yibo, et al. "P/D-serve: Serving disaggregated large language model at scale." arXiv preprint arXiv:2408.08147 (2024).

14) Jiang, Youhe, Ran Yan, and Binhang Yuan. "HexGen-2: Disaggregated Generative Inference of LLMs in

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>

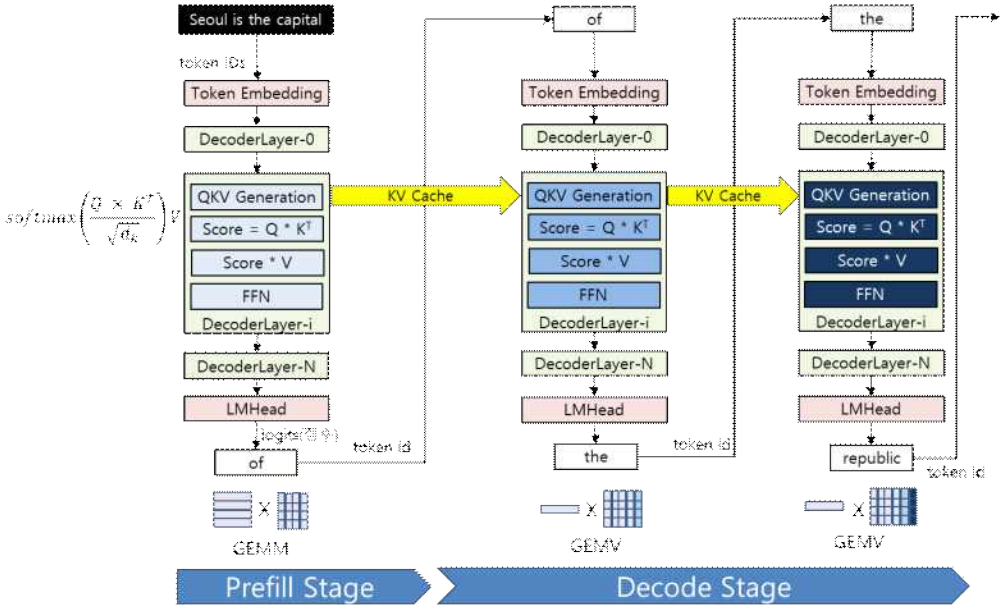
표 62 vLLM 지원 하드웨어 플랫폼들

<table>
<tr><th>지원 방식</th><th>vLLM 메인 저장소</th><th>자체 하드웨어 플러그인</th></tr>
<tr><td>하드웨어 플랫폼</td><td>• NVIDIA GPU
• AMD GPU
• Intel GPU
• Intel/AMD x86 CPU
• ARM Arch64 CPU
• Apple silicon CPU
• IBM Z(S390X)</td><td>• AWS Neuron
• Cambricon MLU
• Google TPU
• Huawei Ascend NPU
• IBM Spyre AIU
• Intel Gaudi(HPU)
• MetaX MACA GPU
• Rebellions ATOM/REBEL NPU</td></tr>
</table>

● vLLM에서는 동종 가속기 자원이 장착된 노드들로 구성된 클러스터 환경에서 동작함을 가정하고 있음
- 파이프라인 병렬화(pipeline parallelism, 이하 PP)[4]와 텐서 병렬화(tensor parallelism, 이하 TP)[5]를 위해 실행되는 노드에 장착된 가속기와 동일한 종류의 가속기 자원을 Ray Placement Group을 통해 한 번에 선점(reservation)
• 예약하는 자원 수 : 파이프라인 병렬화 크기 X 텐서 병렬화 크기
- 예약된 자원을 활용하여 Worker를 동작시켜 추론 작업 실행
- 이종 가속기 간에는 연산 방식, 메모리 구조, 가중치 변환 포멧 등이 다르기에 다른 가속기에서 수행된 결과(출력)를 활용하기 위해서는 데이터 형태 변환이 필요
● vLLM은 병렬처리를 위한 정보인 파이프라인 병렬화 크기(PP degree)와 텐서 병렬화 크기(TP degree) 등을 사용자로부터 받을 뿐 아니라 추론 능력이 다른 동일 유형의 다른 세대의 자원을 동일하게 간주하고 있음
- 모델 로딩시 PP를 위해 사용자가 환경변수(VLLM_PP_LAYER_PARTITION)에 설정한 분할 정보에 기반하여 모델의 계층(레이어)을 분할(디폴트는 균등 분할)하고, 다시 각 계층에 대해 TP를 위해 균등 분할을 적용하여 로딩
- 특성과 능력이 다른 다양한 세대의 가속기 자원이 혼재하는 환경에서 균등 분할은 능력이 다른 자원들(예, GPU의 A100와 H100)을 동일한 자원으로 간주함을 의미하며, 이로 인해 집합통신을 위한 대기나 파이프라인 버블이 발생함
● 이종 가속기들이 장착된 노드들로 구성된 클러스터 환경에서 고효율 LLM 추론을 위한 최적 실행 계획 수립과 이종 가속기간 추론 중간 결과의 빠른 공유를 위한 수단이 필요
- 개별 이종 가속기 자원의 특성과 능력을 효과적으로 통합 활용할 수 있도록 실행 계획을 수립
- PP와 TP 같은 병렬처리에 수반되는 데이터(중간 결과)를 이종 가속기간 저지연 고속 전송(예, 호스트 메모리를 거치지 않고 이종 가속기간 통신)

(2) 추론 서빙을 위한 생성형 모델의 연산 특성 분석
● 트랜스포머 기반 LLM에서 추론(토큰 생성) 과정 분석
- 입력 토큰의 임베딩을 기반으로 $Q \cdot K^T$ 연산을 수행하여 토큰 간 상관관계를 계산
- Softmax로 정규화된 Attention 가중치를 Value 행렬에 적용하여 필요한 정보만 강조
- 이러한 어텐션과 FFN으로 구성된 디코드 레이어들이 누적되면서 표현이 점차 정교해짐
- 마지막 은닉 상태(hidden states)를 출력 단어 벡터와 매핑하여 다음 토큰의 확률 분포를 생성
● 프리필(prefill) 단계는 입력 시퀀스를 한 번에 처리하는 계산 중심(compute-bound) 단계로, 대규모 행렬 곱 연산이 집중적으로 발생
● 디코드(decode) 단계는 자기회귀(autoregressive) 방식으로 토큰을 한 개씩 생성하며, 이전 토큰의 Key/Value(KV) 캐시를 지속해서 참조하는 메모리 중심(memory-bound)으로 연산보다 메모리 접근 패턴과 캐시 관리 효율이 성능의 핵심 병목으로 작용

</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		 그림 560 트랜스포머 기반 LLM에서 토큰 생성 과정 (3) 이종 가속기 활용 추론을 위한 구조 설계 ● vLLM의 하드웨어 플러그인 메커니즘을 활용하여 이종 가속기 하드웨어 지원 - 오픈소스로 공개되어 있는 텐스토렌트(Tenstorrent) NPU용 vLLM을 플러그인화 - 국내 NPU 업체(하이퍼엑셀)와 연구 협력 추진을 통해 vLLM 플러그인 개발 진행 ● 이종 가속기 자원의 개념을 확장하여 동일 유형의 자원도 세대를 구분하여 이종 가속기로 간주 ● 이종 가속기 자원을 통합 활용하여 LLM 추론을 서빙할 수 있도록, 추론 작업을 분할하고 각각 분할된 작업에 최적 이종 가속기 자원을 할당하는 컴포넌트인 플래너(Planner) 추가 - 가용 자원 활용 최적화를 통한 파이프라인 버블 최소화 ● Planner가 생성한 실행 계획에 따라 Worker를 생성하고 추론 작업을 실행하도록 vLLM(LLM Engine, Executor)을 확장 ● 이종 가속기간 저지연 고속 통신을 지원하기 위한 통합 집합 통신 라이브러리(Unified Collective Communication Library) 추가 - 호스트 메모리를 거치지 않고 이종 가속기간 통신을 위한 RDMA 통신 라이브러리 • 예, GPU와 텐스토렌트 NPU간 고속 통신을 위한 RDMA 기반 Send/Recv 라이브러리 - 고성능 Send/Recv 개발 후 이종 가속기간 CCL로 확장 ● 이종 가속기간 KV 캐시 공유를 위한 수단으로 Mooncake를 활용 ● 분리형 프리필(disaggregate prefill) 형태의 추론 서빙시 프리필 단계에서 생성한 KV 캐시를 디코드 단계로 전달을 담당하는 컴포넌트인 KV Transfer 확장 - 프리필 단계와 디코드 단계를 수행하는 가속기가 다를 수 있으므로, 이종 가속기간에서도 KV 캐시 전송이 가능하도록 확장(가속기 플러그인에 포함) - 프리필 단계와 디코드 단계의 병렬처리 구성(모델 분할)이 다를 수 있으므로 계층(layer)과 어텐션 헤드(attention head)를 고려하여 연관있는 모델 샤드를 담당하는 가속기로 KV 캐시를 전달하도록 KV 전송 모듈 확장

Heterogeneous Environment." arXiv preprint arXiv:2502.07903 (2025).

15) Hu, Cunchen, et al. "Memserve: Context caching for disaggregated llm serving with elastic memory pool." arXiv preprint arXiv:2406.17565 (2024).

16) TTFT(Time To First Token): 사용자가 요청을 전송한 시점부터 첫 번째 출력 토큰이 생성될 때까지의 시간으로, prefill 단계의

수행기관	개발 목표	개발내용 및 범위

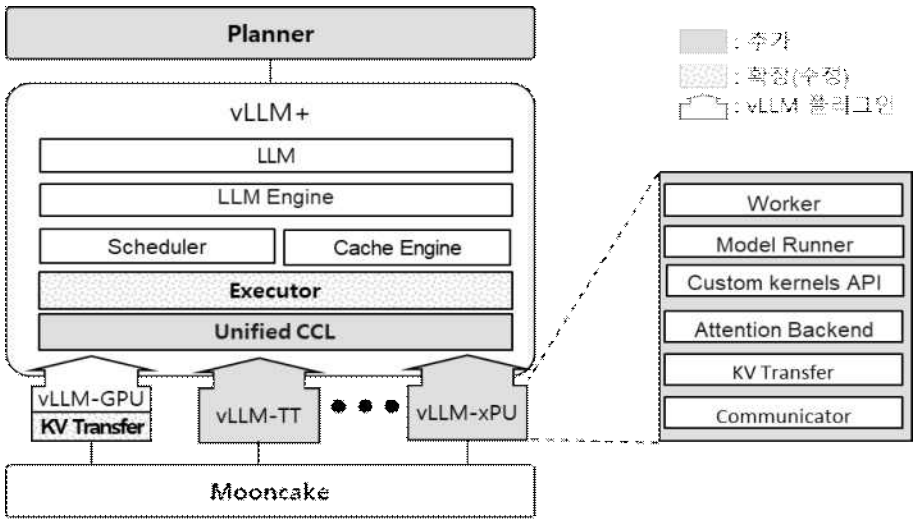


그림 561 이종 가속기의 통합 활용 추론을 위해 확장된 vLLM 기반의 구조

(4) 비균등 분할 텐서 병렬화의 가능성 확인을 위한 PoC

- 비균등 분할 텐서 병렬화
  - 모든 가속기에 동일한 크기의 모델 파라미터를 분배하는 대신, 각 가속기의 연산 성능과 메모리 용량을 고려하여 파라미터를 비균등하게 분할하여 병렬로 연산을 수행
  - 트랜스포머 구조에서 레이어 정규화(Layer Normalization)를 제외한 주요 연산 모듈(Attention, MLP 등)에 대해 열 병렬화(Column Parallelism)와 행 병렬화(Row Parallelism) 적용하여 비균등 분할 가능

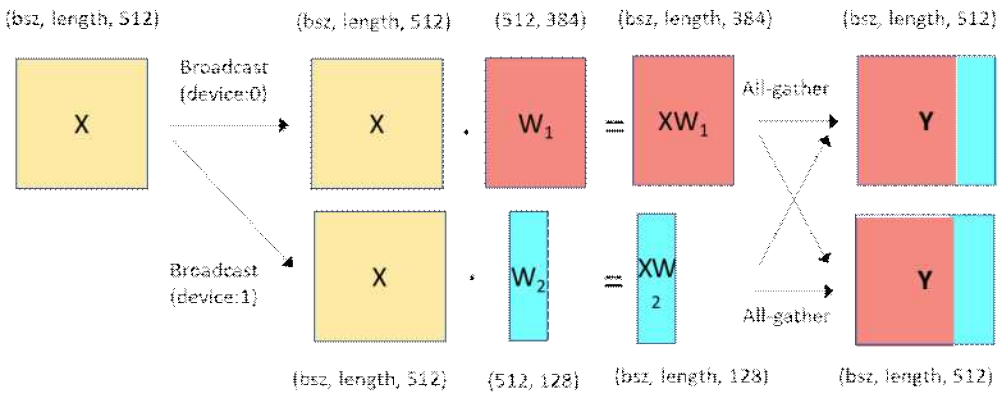
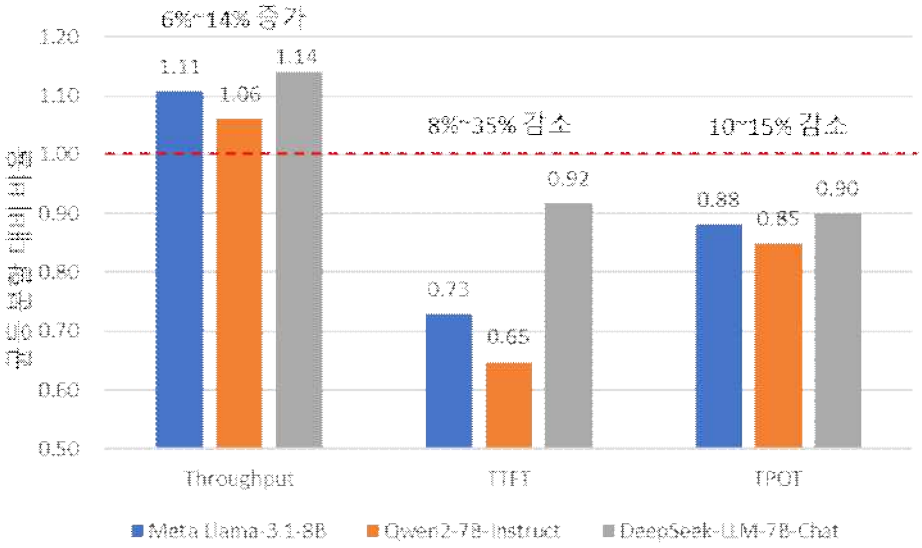


그림 562 비균등 분할 기반 열 병렬화 개념(분할 비율 3:1)

- PoC를 위한 vLLM 의 모델 로딩 부분 수정
  - 균등 분할을 전제로 설계된 vLLM 내부 병렬 처리 모듈을 비율 기반으로 확장하여 트랜스포머의 임베딩, 어텐션, 피드포워드 네트워크 구성 요소에 적용
  - 각 구성 요소에서 연산 비율에 맞춰 파라미터를 비균등하게 로드하도록 수정
- 가능성 평가를 위한 실험 구성
  - NVIDIA A100 GPU가 장착된 노드와 NVIDIA RTX 3090 GPU가 장착된 노드로 실험 환경 구성하고 각 노드에는 하나의 GPU만 보이도록 설정
  - 7B~8B 파라미터 규모를 갖는 세 가지 LLM을 대상으로 TP 크기는 2로 하여 균등 분할 대비 비균등 분할(3:1)의 상대 성능 평가
    - Llama-3.1-8B, Qwen2-7B-Instruct, DeepSeek-LLM-7B-Chat
  - 벤치마크 워크로드는 초당 3개의 요청을 발생시켜 총 100개의 요청을 생성하도록 구성하였으며 최대 동시 처리 가능한 요청 수는 16으로 제한함. 또한, 모든 요청의 토큰 길

수행기관	개발 목표	개발내용 및 범위
		이는 입력 1,024, 출력 128로 일괄 고정 ● 실험 결과 : 처리량과 응답 지연 시간 관점에서 비균등 분할의 효과 확인 - 균등 분할 대비 처리량 최대 14% 증가, TTFT[6] 최대 35% 감소, TPOT[7] 최대 15% 감소  그림 563 균등 분할 수행 대비 비균등 분할(3:1) 수행의 상대 성능 (5) 텐스토렌트(Tenstorrent) NPU용 vLLM 플러그인 vLLM-TT 개발 ● 텐스토렌트 NPU 특화 기능을 활용하여 추론이 동작되도록 확장된 텐스토렌트용 vLLM을 플러그인화 - 오픈소스로 공개되어 있는 TT-Metal v0.59.0-rc53, vLLM Tenstorrent f028da1 기반으로 개발 - vLLM이 제공하는 플러그인 개발 가이드에 따라 setup.py, __init__.py, platform.py 등을 추가 - entry_point를 통해 Tenstorrent용 백엔드가 자동으로 등록되도록 설정 • vLLM 내부에서 사용할 플러그인 확장 포인트 이름 vllm.platform_plugins으로 vllm_tt:register()를 등록 - 환경 변수 VLLM_DEVICE 설정에 따라 vLLM이 자동으로 GPU 또는 Tenstorrent NPU 경로를 선택 • vllm_tt:register()에서는 환경 변수 VLLM_DEVICE가 TT로 설정된 경우에 vllm_tt.platform.TTPlatform을 반환하여 vLLM이 TTPlatform 클래스를 임포트하게 함 - 기존 vLLM의 Platform 인터페이스를 확장하여 Tenstorrent 전용 플랫폼 구현 • TTPlatform에서는 vLLM 실행기가 호출할 워커 클래스(vllm_config.parallel_config.worker_cls)로 텐스토렌트용 워커인 vllm_tt.worker.tt_worker.TTWorker를 지정

처리 성능을 나타냄.
17) TPOT(Time Per Output Token): request별 평균 토큰 생성 시간으로, 각 request에 대해 첫 번째 토큰 이후 생성되는 토큰들의 평균 생성 시간을 의미함.
18) ITL(Inter-Token Latency): 전체 request에 대해 토큰 간 생성 시간을 측정한 지표로, 첫 번째 토큰 이후 생성되는 각 토큰 사이의 지연 시간을 의미함
19) E2EL(End-to-End Latency): 클라이언트가 요청을 전송한 시점부터 최종 응답을 수신할 때까지의 총 경과 시간으로, 사용자가 실제로 체감하는 전체 응답 성능을 나타냄
20) https://www.nvidia.com/en-us/data-center/a100/
21) https://directmacro.com/blog/post/nvidia-a100-in-2025#_toc_NVIDIA_A100_80GBs_Price_Update_and_Overview_in
22) https://tenstorrent.com/ko/hardware/blackhole

수행기관	개발 목표	개발내용 및 범위

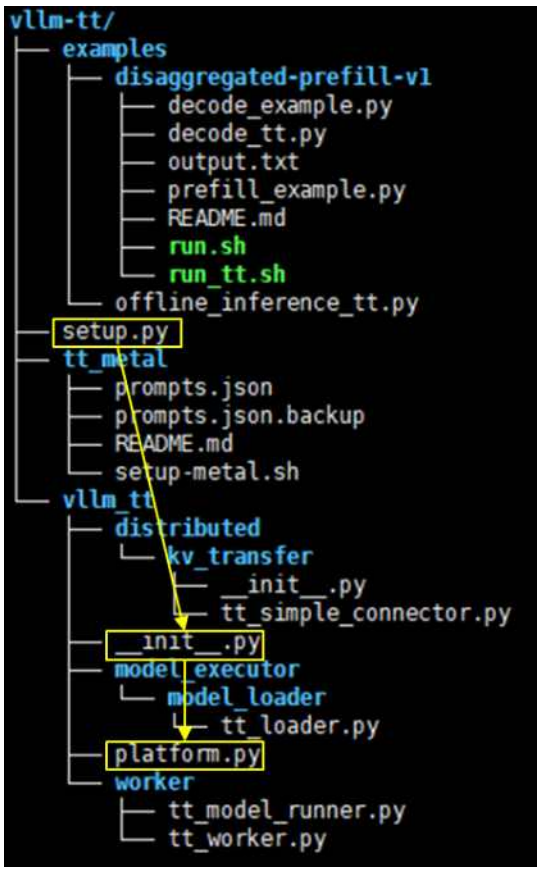


그림 564 vLLM-TT 플러그인 구성

- vLLM-TT의 주요 구성 요소
  - TTPlatform : vLLM의 Platform 인터페이스를 상속하여 텐스토렌트 장치 플랫폼 정의
  - TTWorker : vLLM의 WorkerBase를 상속하여 Tenstorrent NPU의 추론 제어 및 통신 처리를 담당
  - TTModelLoader : Tenstorrent 메모리 배치 초기화 및 모델 가중치 로딩
  - TTModelRunner: Tenstorrent NPU 상에서 실제 LLM 연산 수행
  - TTSimpleConnector : Mooncake store connector와 Mooncake connector 기반으로 GPU-NPU 간 KV 전송 기능 제공
  - TT-NN : 텐스토렌트 하드웨어에 텐서 연산을 수행할 수 있는 고수준 신경망 연산 라이브러리
  - TT-Metal : Tenstorrent 하드웨어에 직접 접근할 수 있는 저수준 SDK

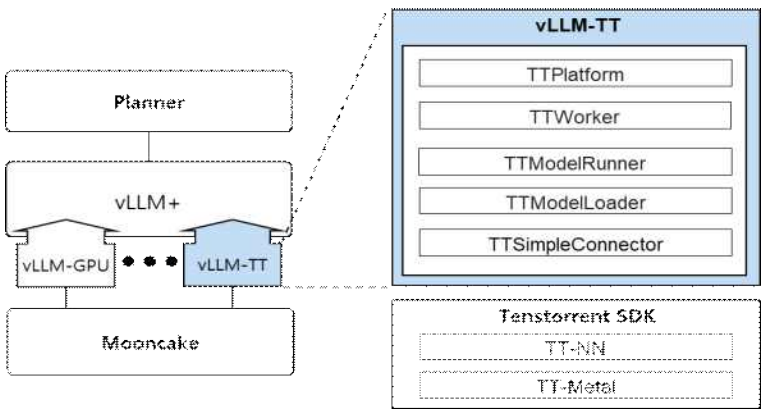


그림 565 Tenstorrent NPU용 vLLM 플러그인

수행기관	개발 목표	개발내용 및 범위

● vLLM-TT 기능 시험

- 텐스토렌트 NPU 하드웨어가 장착된 노드에 vLLM v0.10.0와 개발한 vLLM-TT 플러그인을 설치 후 텐스토렌트 NPU를 활용한 오프라인 LLM 모델 추론 기능 시험을 수행 (상세 실험 구성은 아래 표63 참고)

표 63 실험 구성

시험 케이스	텐스토렌트 NPU 하드웨어	소프트웨어 환경	시험 LLM 모델
1	n300 × 4	vLLM v0.10.0	Llama 3.2 1B, Llama 3.1 8B, Llama 3.1 70B
2	p150 × 1	vllm-tt v0.5 (vLLM Tenstorrent f028da1 기반)	Llama 3.2 1B, Llama 3.1 8B
3	p150 × 4	TT-Metal v0.59.0-rc53	Llama 3.2 1B, Llama 3.1 8B, Llama 3.1 70B

- 3가지 시험 케이스 모두에서 정상적으로 LLM 추론 요청이 서빙됨을 확인

(6) 분리형 프리필 기술 분석

● 분리형 프리필 개념

- LLM 추론 과정을 프리필(Prefill) 단계와 디코드(Decode) 단계로 분해하고 이를 서로 다른 가속기에서 분리 처리하여 자원 활용률과 추론 성능을 높이는 기술
- 다른 가속기 자원에서 분리 수행함으로써 연산 간섭과 자원 활용의 비효율성 제거
- 프리필 단계에서 생성한 KV Cache를 디코드 단계를 수행하는 가속기에 전달하여 활용

● 분리형 프리필 기본 구조

- 스케줄러(Scheduler), Prefill/Decode 인스턴스, KV Cache 전송, 배치(batch)로 구성
- 스케줄러는 사용자들의 요청을 분할하여 배치로 구성하는 Router와 각 요청에 적합한 Prefill/Decode 인스턴스 쌍을 결정하는 Selector로 구성

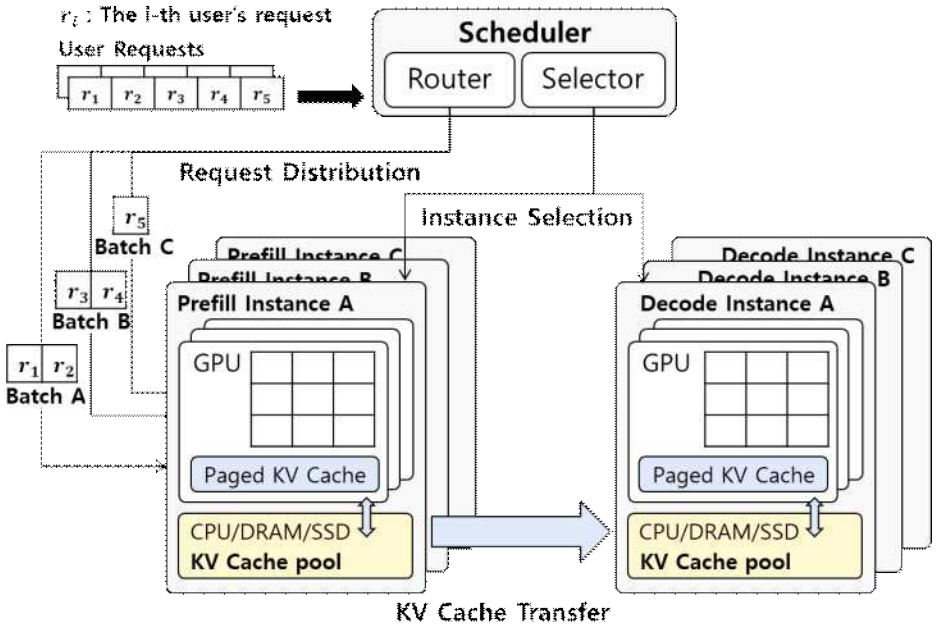
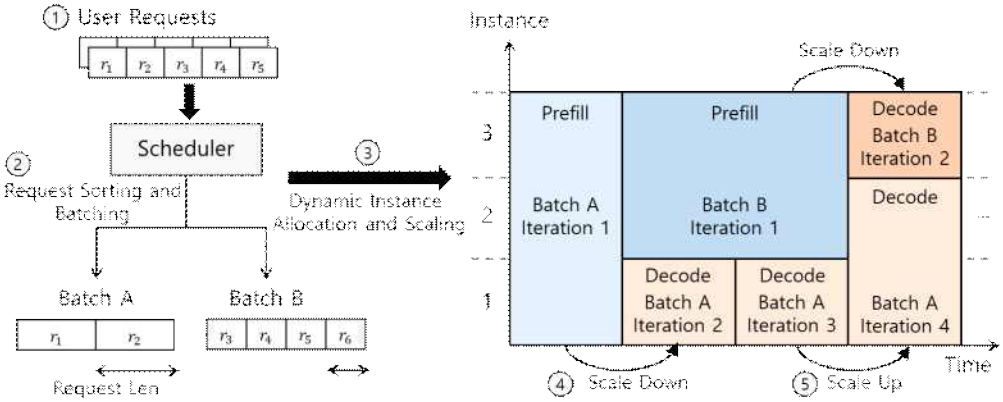
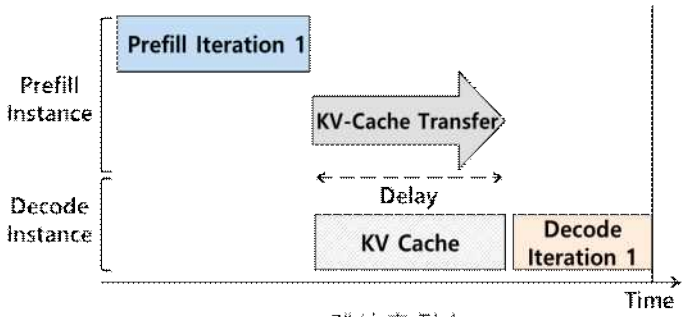
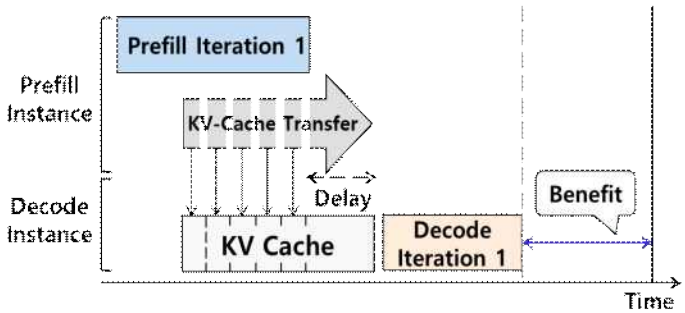
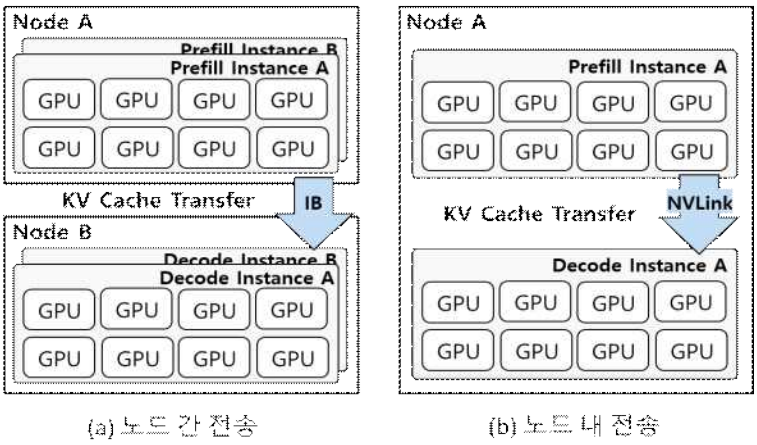
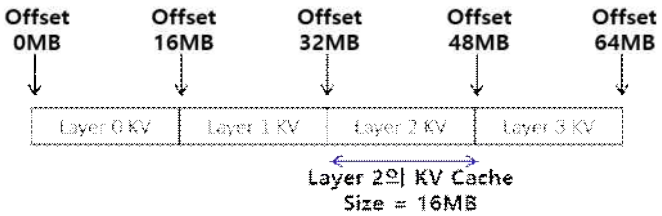


그림 566 분리형 프리필 기본 구조

수행기관	개발 목표	개발내용 및 범위
		● 고성능 LLM 추론을 수행하기 위한 분리형 프리필 환경의 4가지 요소 기술 - P/D 인스턴스 구성 기술 • 정적 인스턴스 구성 : LLM 서빙 시스템 시작 전에 사전 워크로드 프로파일링을 기반으로 Prefill, Decode 인스턴스의 비율을 미리 결정(예, Splitwise, DisServe, Mooncake, HexGen-2 등) • 동적 인스턴스 구성 : 실행 중 부하를 모니터링하여 동적으로 인스턴스 비율을 조정(예, TetriInger, Longserve, P/D-Serve 등)  (a) 사용자 요청 길이 기반 배치 구성 (b) 동적 P/D 인스턴스 할당 그림 567 요청 길이 기반 배치 구성 및 동적 P/D 인스턴스 할당 (예, Loongserve) - KV Cache 전송 기술 • Prefill 인스턴스에서 생성된 KV Cache를 Decode 인스턴스로 전송하는 지연시간이 전체 시스템의 주요 병목으로 분리형 프리필의 중요한 최적화 대상임. • KV Cache 계산과 전송의 중첩 : 모델의 각 레이어 계산이 완료되는 즉시 해당 레이어의 KV Cache를 생성하고, 이를 비동기적으로 전송하는 동시에 다음 레이어의 Prefill 계산을 병렬로 수행  (a) KV Cache 계산 후 전송  (b) KV Cache 계산과 전송 중첩 그림 568 KV 캐시 전송 방법

수행기관	개발 목표	개발내용 및 범위
		• 네트워크 성능을 고려한 전송 구조 변경 : KV Cache 전송 시 시스템의 네트워크 환경에 따라 전송 구조를 유동적으로 조정하여 전송 지연 최소화  그림 569 KV 캐시 전송 구조(예, DistServe) • 네트워크 호출 횟수를 줄이고 전송 제어 오버헤드를 제거하기 위한 일괄 전송 : Prefill 인스턴스에서 KV Cache를 전송 효율 향상을 위한 자료구조(예, contiguous buffer)로 구성하여 일괄 전송하고, Decode 인스턴스에서 이를 다시 복원  그림 570 Contiguous Buffer 구조 - KV Cache 관리 기술 • 각 인스턴스의 GPU VRAM에 KV Cache를 저장한 후, GPU 간 P2P 통신을 통해 CPU를 경유하지 않고 직접 KV Cache를 전송 : 노드 간 전송에는 RDMA를, 노드 내 전송에는 NVLink와 같은 고속 인터커넥트를 활용 • 최근 일부 시스템(예, Mooncake, Memserve)에서는 인스턴스 내부에 KV Cache 풀을 구성하여, 생성된 KV Cache를 GPU 메모리뿐만 아니라 CPU 주메모리 및 SSD와 같은 다양한 저장 매체를 Cache Store로 사용하여 저장 • Cache Store를 사용함으로써 모든 인스턴스 간에 KV Cache를 공유할 수 있으며, 다계층 메모리 및 저장 자원의 계층적 활용을 통해 메모리를 효율적으로 관리 --> KV Cache의 재사용률이 크게 향상되며, 전체 처리시간이 단축되는 효과 - 요청 스케줄링 기술 • 다수 사용자의 요청을 분할하여 배치를 구성한 후, 각 배치를 적절한 P/D 인스턴스에 할당 • TTFT 및 TPOT 지연시간에 직접 영향을 미치며 전체 시스템 성능 결정짓는 핵심 요소 중 하나 • 작업 특성을 고려하지 않고 각 인스턴스 대기열에 포함된 요청 수를 비교하여 가장 짧은 대기열을 가진 인스턴스에 할당 : 요청 간 처리시간 차이를 고려하지 않는 한계 • 이를 극복하기 위해 워크로드 특성, GPU 메모리 사용량, KV 캐시 재사용 가능성(Cache-aware) 등을 고려한 다양한 방식 존재

수행기관	개발 목표	개발내용 및 범위

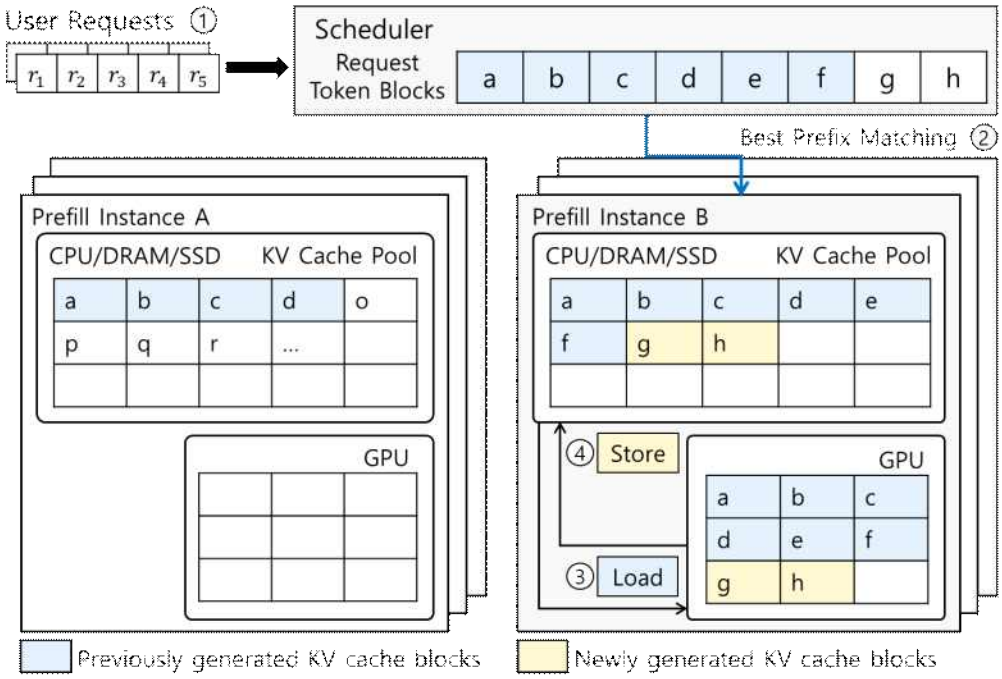


그림 571 KV Cache-aware 스케줄링 (예, Mooncake)

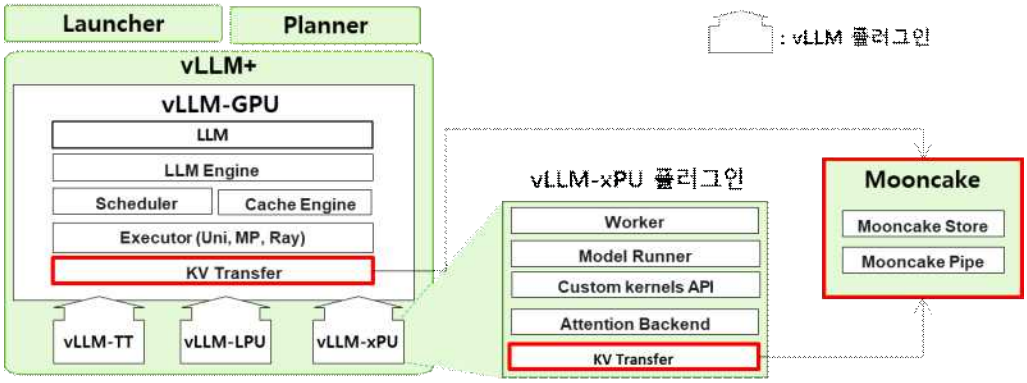
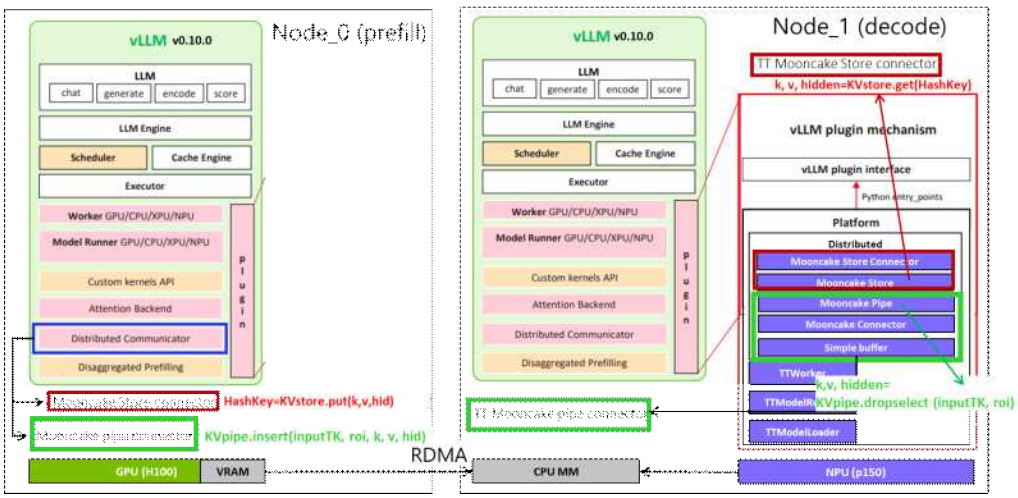
- 주요 분리형 프리필 선행 연구의 특징 비교

표 64 주요 분리형 프리필 선행 연구 특징 비교

<table>
<tr><th rowspan="2">선행 연구
(시스템)</th><th colspan="4">요소 기술</th></tr>
<tr><th>P/D Instance 구성</th><th>KV Cache 전송</th><th>KV Cache 관리</th><th>요청 스케줄링</th></tr>
<tr><td>Splitwise[8)]</td><td rowspan="2">정적</td><td>KV Cache 계산과 전송의 중첩</td><td rowspan="4">Cache Store 미사용</td><td rowspan="3">작업 특성 미 고려</td></tr>
<tr><td>DistServe[9)]</td><td>네트워크 성능을 고려한 전송 구조 변경</td></tr>
<tr><td>TetriInfer[10)]</td><td rowspan="2">동적</td><td rowspan="3">-</td></tr>
<tr><td>Loongserve[11)]</td><td>워크로드 특성 및 GPU 메모리 사용량 고려</td></tr>
<tr><td>Mooncake[12)]</td><td>정적</td><td>Cache Store 사용</td><td>Cache-aware</td></tr>
<tr><td>P/D-Serve[13)]</td><td>동적</td><td>전송 효율 향상을 위한 자료구조 (continuous buffer)</td><td rowspan="2">Cache Store 미사용</td><td>인스턴스 로컬 대기열을 제거한 스케줄링</td></tr>
<tr><td>HexGen-2[14)]</td><td>정적</td><td>-</td><td>-</td></tr>
<tr><td>MemServe[15)]</td><td>동적</td><td>전송 효율 향상을 위한 자료구조 (huge page)</td><td>Cache Store 사용</td><td>Cache-aware</td></tr>
</table>

(7) 이종 가속기 통합 추론을 위한 분리형 프리필 기술 개발

- 이종 가속기 통합 추론을 위한 분리형 프리필에서 Prefill을 수행하는 가속기(예, GPU)가 장착된 노드와 Decode를 수행하는 가속기(예, 텐스토렌트 NPU)가 장착된 노드 간 KV 캐시 전달이 필요
- vLLM+는 다중 노드에 장착되어 있는 이종 가속기 간 KV 캐시 전달 엔진으로 Mooncake를 외부 모듈로 사용

수행기관	개발 목표	개발내용 및 범위
		- KV 캐시 재활용을 위해서는 Mooncake Store를 사용하고, KV 캐시 재활용을 하지 않는 경우에는 Mooncake Pipe를 사용함 - vLLM-GPU 내의 KV Transfer 모듈과 vLLM-xPU 플러그인 내의 KV Tranfer 모듈에서 Mooncake Store와 Mooncake Pipe를 사용하기 위해 각자의 KV Transfer 모듈에 Mooncake Store 커넥터 (connector) 와 Mooncake Pipe 커넥터를 개발해야 함  그림 572 vLLM+에서 KV 캐시 전달 관련 주요 모듈 ● vLLM-GPU와 vLLM-TT 간 KV 캐시 전달을 위한 vLLM-TT의 컴포넌트 - TTMooncakePipeConnector • vLLM-GPU와 vLLM-TT 간에 Mooncake Transfer Engine을 사용해서 Pipe 형태로 KV 캐시를 직접 전달하는 기능을 담당 • vLLM-GPU의 MooncakePipeConnector에 대응하는 컴포넌트 • 그림 573의 초록색 박스 참고 - TTMooncakeStoreConnector • vLLM-GPU와 vLLM-TT 간에 Mooncake Store를 사용해서 KV 캐시를 전달하는 기능을 담당 • vLLM-GPU의 MooncakeStoreConnector에 대응하는 컴포넌트 • 그림 573의 빨간색 박스 참고  그림 573 GPU와 텐스토렌트 NPU간 KV 캐시 전달 구조 ● vLLM-GPU와 vLLM-TT 간 Mooncake 기반 KV 캐시 전달을 위한 3종 Connector 개발 - 개발 환경

23) G. Xiao et al., "Efficient streaming language models with attention sinks," arXiv:2309.17453, 2023.

수행기관	개발 목표	개발내용 및 범위

표 65 Mooncake 호환 Connector 개발 환경

항목		버전	제조사	사용방식
HW	GPU	A100, H100, A6000, RTX 6000 Ada (4종)	NVIDIA	활용
	NPU	Wormhole, Blackhole (2종)	Tenstorrent	
SW	vLLM-GPU	vLLMv. 0.10.0	UC Berkeley	활용
	vLLM-TT	vLLMTenstorrentrepo 739dcaa	Tenstorrent	수정
	TT-Metalium	v0.59.0-rc3		수정
	mooncake	v.0.3.5	Moonshot AI	활용

- Mooncake 연동 KV 캐시 전달 방식에 따라 tt-mooncake-store-rdma-connector, tt-mooncake-mm- rdma-connector, tt-mooncake-npu-rdma-connector 의 3종 컨넥터 개발
  - tt-mooncake-store-rdma-connector : mooncake-store-rdma-connector와 상호 통신하며 Mooncake Store에 저장된 KV 캐시를 InfiniBand RDMA 방식으로 주메모리에 전송

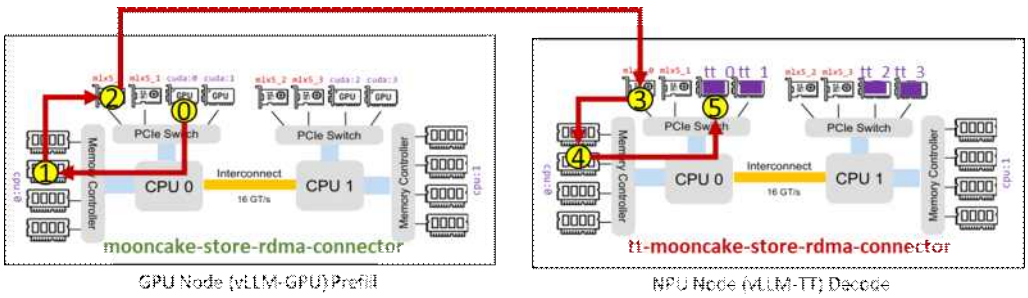


그림 574 mooncake-store-rdma-connector와 tt-mooncake-store-rdma-connector 간 KV 캐시 전달 흐름

  - tt-mooncake-mm-rdma-connector : mooncake-gpu-rdma-connector와 상호 통신하며 Mooncake Pipe를 이용하여 KV 캐시를 InfiniBand RDMA 방식으로 주메모리에 전송

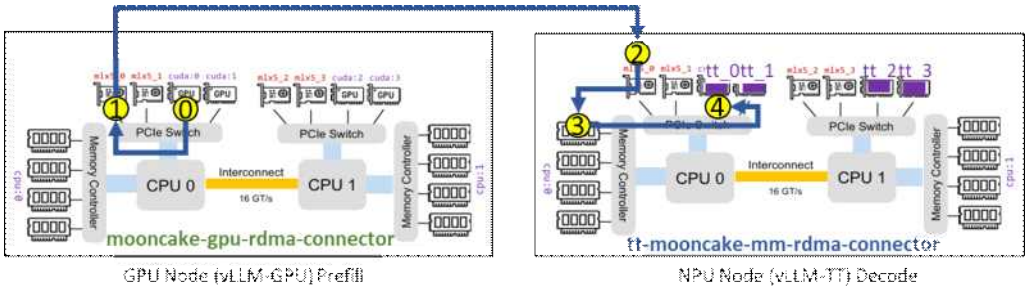


그림 575 mooncake-gpu-rdma-connector와 tt-mooncake-mm-rdma-connector 간 KV 캐시 전달 흐름

  - tt-mooncake-npu-rdma-connector : mooncake-gpu-rdma-connector와 상호 통신하며 Mooncake Pipe를 이용하여 KV 캐시를 InfiniBand RDMA 방식으로 NPU 메모리에 전송

24) J. Willette et al., "Training-free exponential context extension via cascading KV cache," arXiv:2406.17808, 2024.

수행기관	개발 목표	개발내용 및 범위

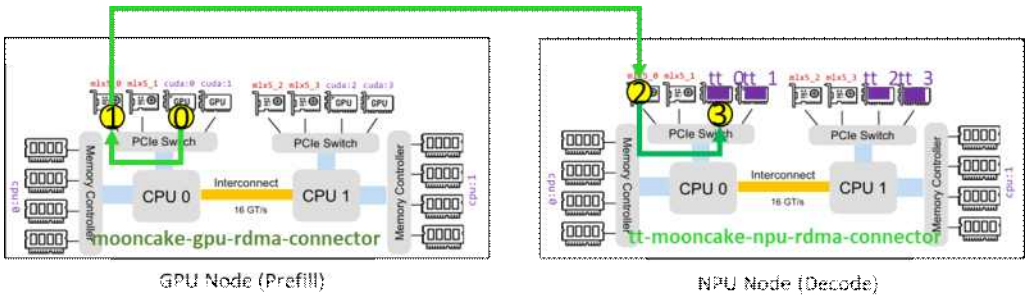


그림 576 mooncake-gpu-rdma-connector와 tt-mooncake-npu-rdma-connector 간 KV 캐시 전달 흐름

- Llama 모델을 활용하여 개발한 커넥터들의 기능 검증

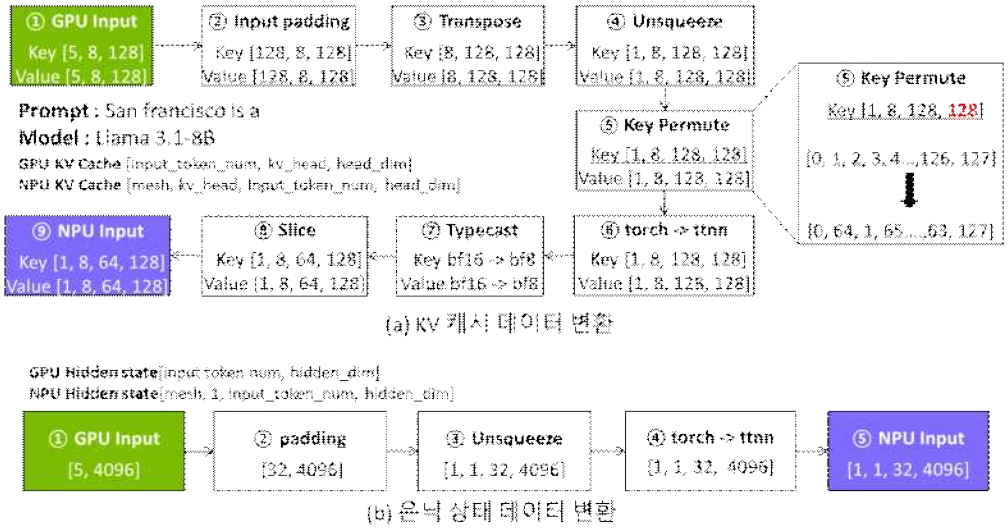
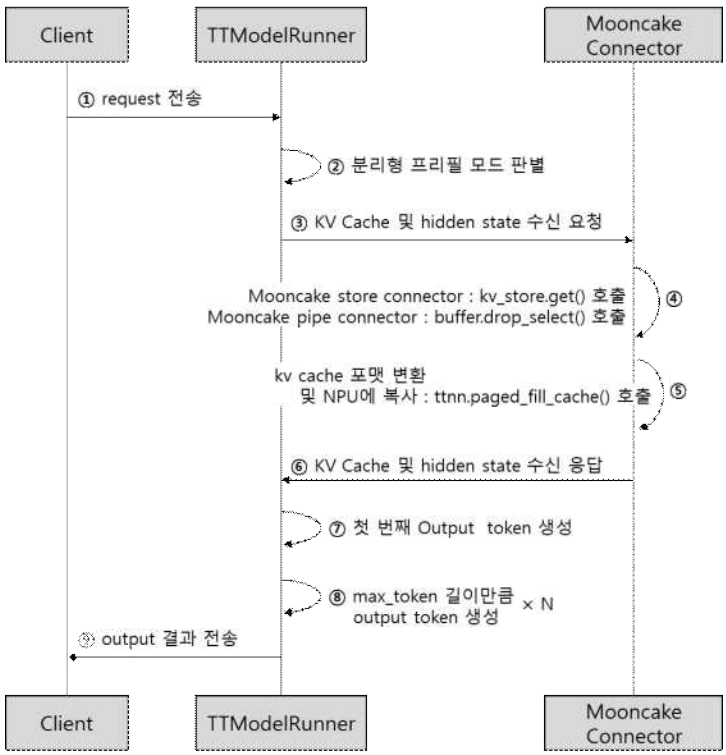
표 66 개발한 Mooncake 호환 Connector 기능 검증에 사용한 모델

모델 이름	버전	제조사
Llama 3.1 8B	NVIDIA GPU 지원 모델	HuggingFace
Llama 3.1 70B		
Meta-Llama 3.1 8B	TT-Metalium/ TT-NN 지원 모델	Meta
Meta-Llama 3.1 70B		

● 이종 가속기(GPU-NPU) 간 데이터 변환 기술 개발

- GPU, NPU 등과 같은 AI 가속기는 낮은 전력으로 높은 성능을 제공하기 위해 연산 방식, 데이터 배치 전략, 메모리 계층 구조를 각각 다르게 최적화해 왔으며, LLM 추론에 필요한 연산(attention, matmul, KV-Cache 처리 등) 또한 해당 구조를 전제로 설계됨
  - 서로 다른 가속기는 동일한 연산이라도 요구하는 텐서 차원 배치, 메모리 정렬 방식, 데이터 레이아웃이 다름
- 이종 가속기를 통합해 활용하려면 각 가속기의 연산 패턴과 메모리 구조 간 차이를 해소하기 위해 데이터 모양과 레이아웃을 해당 가속기에 최적으로 맞추어 변환해야 함
  - stride 재배치, quantization format(Q8, FP16, BF16 등), KV-Cache 구조 등
- GPU-NPU 간 KV 캐시 데이터 변환
  - 텐서 차원 구조 변환 : [input_token_num, kv_head, head_dim] 형태의 3차원 텐서로 구성된 GPU KV 캐시를 Tenstorrent NPU 아키텍처 특성에 맞춰 [mesh, kv_head, input_token_num, head_dim] 형태의 4차원 텐서로 변환
  - 정밀도 포맷 변환(quantization format) : 텐스토렌트 NPU에 있는 KV 캐시 구조에 저장하기 위해 BF16에서 BF8로 변환
- GPU-NPU 간 은닉 상태 데이터 변환
  - 텐서 차원 구조 변환 : [input_token_num, hidden_dim] 형태의 2차원 텐서로 구성된 GPU 은닉 상태를 텐스토렌트 NPU 아키텍처 특성에 맞춰 [mesh, 1, input_token_num, hidden_dim] 형태의 4차원 텐서로 변환

25) Y. Sun et al., "ShadowKV: KV cache in shadows for high-throughput long-context LLM inference," in Proc. International Conference on Machine Learning (ICML), 2025
26) W. Lee et al., "InnFineGen: Efficient generative inference of large language models with dynamic KV cache management," in Proc. 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2024
27) Y. Sheng et al., "FlexGen: High-throughput generative inference of large language models with a single GPU," in Proc. International Conference on Machine Learning (ICML), PMLR, 2023

수행기관	개발 목표	개발내용 및 범위
		 그림 577 GPU-NPU 간 데이터 변환 예 ● Mooncake기반 텐스토렌트 NPU(vLLM-TT)용 분리형 프리필 처리 로직 개발 - GPU에서 수행된 prefill 결과를 텐스토렌트 NPU에서 수신하여 decode 전용으로 동작 • Mooncake Connector의 store 및 pipe 방식을 모두 지원 - vLLM-TT에서 분리형 프리필 처리 흐름  그림 578 vLLM-TT에서 분리형 프리필 처리 흐름 - 단일 노드 환경과 다중 노드 환경에서 Mooncake Store와 Mooncake Pipe 방식 모두 정상 동작 확인 • 단일 노드 실험을 위해 하나의 노드에 RTX6000 GPU 1개와 텐스토렌트 Blackhole NPU 1개를 장착하여 단일 노드 환경 구성 • 다중 노드 실험을 위해 H100 GPU 1개가 장착된 노드와 텐스토렌트 Blackhole NPU

수행기관	개발 목표	개발내용 및 범위
		1개가 장착된 노드로 다중 노드 환경 구성 • Llama 3.1 8B 모델에 대해 분리형 프리필 형태의 추론 서빙 동작 확인 - NPU 노드가 여러 개인 다중 노드 환경에서 실험은 텐스토렌트의 멀티프로세스 지원 이슈로 진행되지 않음 (8) 분리형 프리필 병렬화 기술 개발 ● 단일 가속기 메모리 용량을 초과하는 LLM 모델 추론을 위해 vLLM에서는 텐서 병렬화와 파이프라인 병렬화를 적용하여 여러 가속기에 다중 프로세스 기반으로 분산 병렬처리 - 파이프라인 스테이지의 랭크 별로 별도의 프로세스로 실행 ● 텐스토렌트에서는 여러 가속기 칩이 이더넷 메시로(ethernet mash)로 연결되어 하나의 거대한 병렬 컴퓨팅 자원처럼 동작하도록 설계됨(즉, 단일 프로세스가 하나의 거대한 가속기로 보고 사용하도록 여러 칩을 숨기고 단일 논리 디바이스처럼 인식) ● vLLM-GPU는 멀티 프로세스를 통해 독립적으로 분할된 데이터를 전송하는 반면, vLLM-TT는 싱글 프로세스로 동작으로 인해 분리형 프리필 환경에서 데이터를 수신하고 재구성하기 위한 추가적인 로직 개발 ● GPU-NPU간 텐서 병렬화 처리 흐름 개발 - Mooncake Pipe Connector 방식에서의 텐서 병렬화 처리 흐름 • Prefill 실행을 위한 서버에서 각 Worker 프로세스가 각각 독립적인 GPU를 사용하여 할당된 연산을 수행하고, Mooncake Pipe를 통해 각 데이터(KV 캐시와 은닉 상태)를 RDMA 기반으로 Decode 실행을 위한 서버(vLLM-TT)의 메모리로 고속 전송함 • vLLM-TT에서는 분할된 KV 캐시와 은닉 상태를 수신하기 위해 Mooncake Pipe의 drop_select() API를 텐서 병렬화 수준만큼 반복 호출한 후 수신한 텐서를 병합함(은닉 상태는 모두 같은 값을 전송하므로 한 번만 호출) • 병합된 KV 캐시와 은닉 상태를 ttnn.tensor로 변환할 때, ttnn.ShardTensorToMesh(dim=x)를 활용하여 지정된 차원(dim=x)을 기준으로 텐서를 분할하여 NPU에 분산 저장함. • KV 캐시는 kv_head 차원, 은닉 상태는 hidden_dim 차원을 기준으로 분할함. - Mooncake Store Connector 방식에서의 텐서 병렬화 처리 흐름 • Prefill 실행을 위한 서버에서 각 Worker 프로세스가 각각 독립적인 GPU/VRAM을 사용하여 할당된 연산을 수행하고, Mooncake Store API인 put을 활용하여 각 Worker가 처리한 KV 캐시와 은닉 상태를 로컬 CPU 주메모리에 저장함 • Prefill 실행 서버에서 저장한 KV 캐시와 은닉 상태는 Decode 실행을 위한 서버(vLLM-TT)의 CPU 주메모리에 RDMA 기반으로 병렬 전송함 • vLLM-TT에서는 분산된 KV 캐시와 은닉 상태를 수신하기 위해 Mooncake Store의 get() API를 텐서 병렬화 수준만큼 반복 호출하여 수신하고, 수신한 텐서를 병합함(은닉 상태는 모두 같은 값을 전송하므로 한 번만 호출) • 이후는 Mooncake Pipe Connector에서와 동일

수행기관	개발 목표	개발내용 및 범위

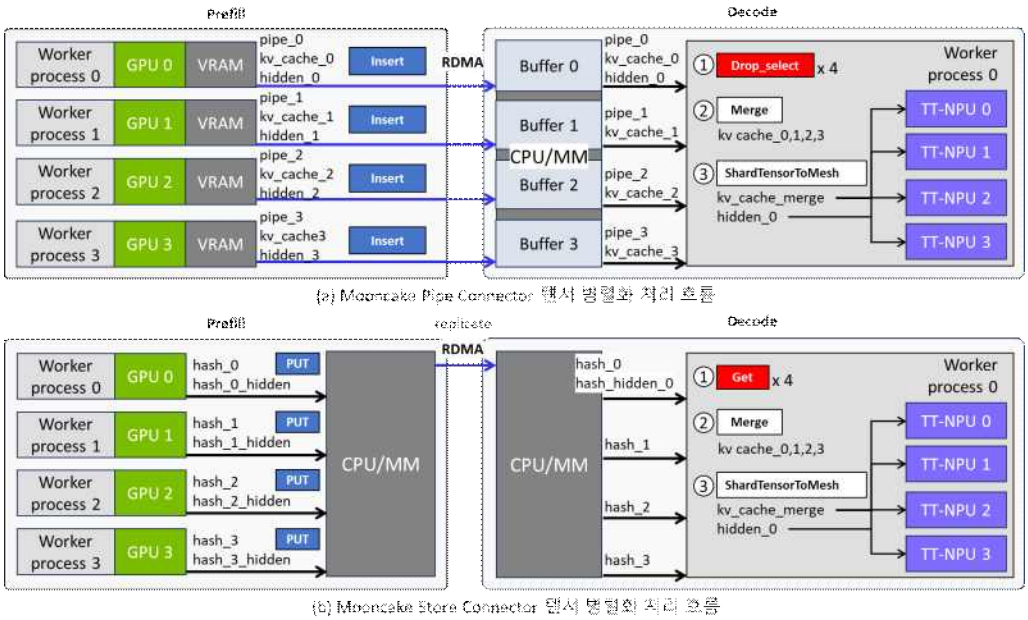


그림 579 GPU-NPU 텐서 병렬화 처리 흐름

- 다중 가속기를 활용한 환경을 구축하고 GPU-NPU간 Mooncake Store와 Mooncake Pipe 방식 모두 정상 동작 확인
  - H100 GPU 4개가 장착된 노드와 텐스토렌트 Blackhole NPU 4개가 장착된 노드 1개로 다중 노드 환경 구성
  - llama 3.1 70B 모델에 대해 분리형 프리필 형태의 추론 서빙 동작 확인

(9) 이종 가속기 통합 추론 성능 실험

- 실험 환경 구성
  - 환경 1 : 하나의 노드에 NVIDIA A100 GPU 4개가 장착된 노드 2개로 구성
  - 환경 2 : NVIDIA A100 GPU 4개가 장착된 노드 하나와 텐스토렌트 Blackhole p150 NPU 4개가 장착된 노드 하나로 구성

표 67 이종 가속기 통합 추론 실험 환경

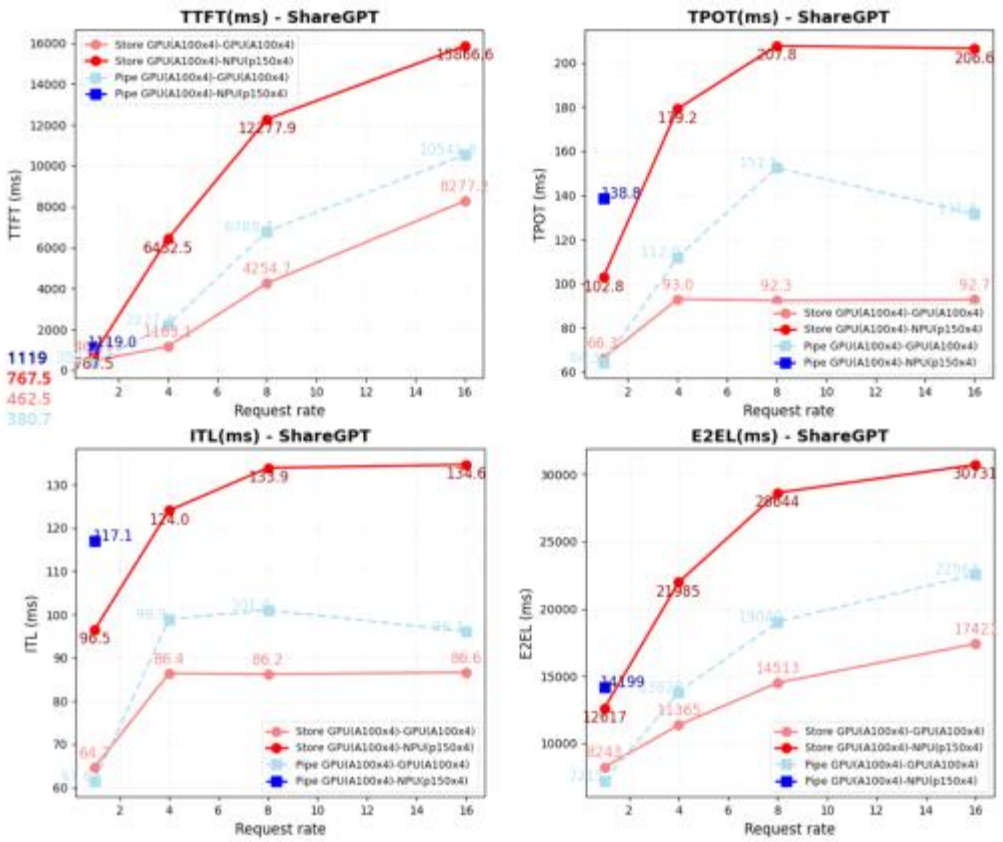
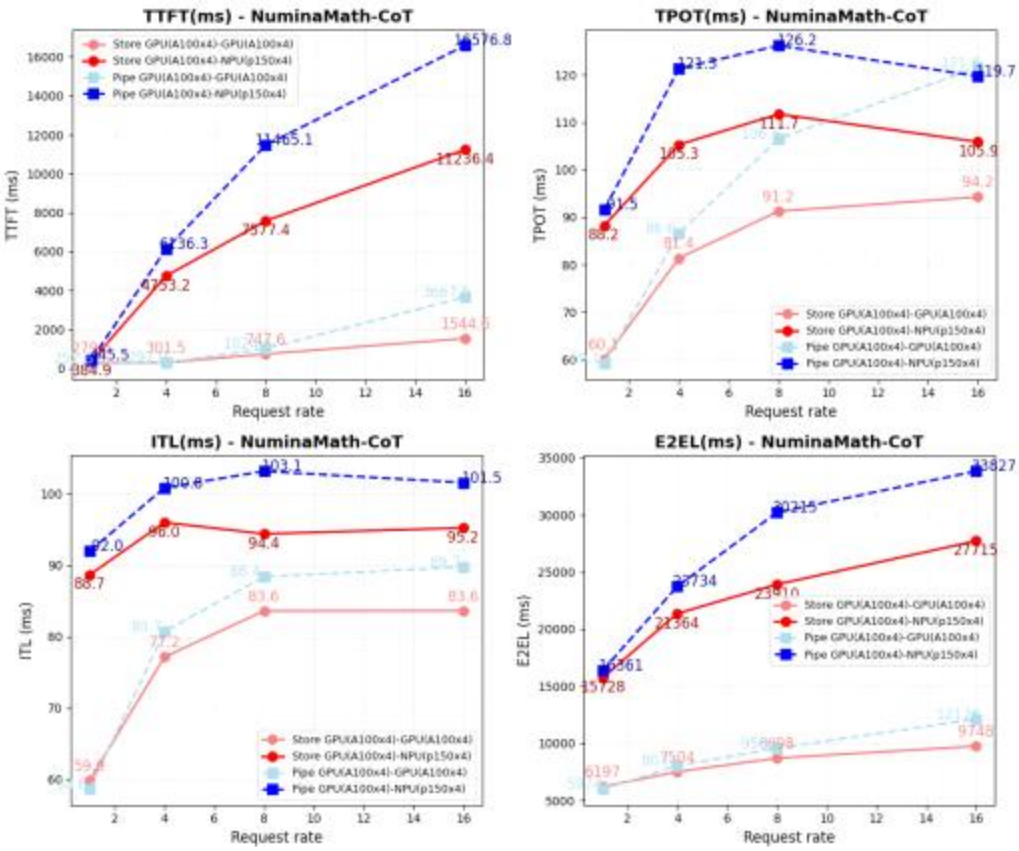
	Prefill (vLLM-GPU) 노드	Decode (vLLM-TT) 노드
환경 1	NVIDIA A100 GPU X 4개	NVIDIA A100 GPU X 4개
환경 2	NVIDIA A100 GPU X 4개	텐스토렌트 Blackhole p150 NPU X 4개
모델	Llama 3.1 70B	Meta-Llama 3.1 70B

- 워크로드
  - ShareGPT 데이터셋과 NuminaMath-CoT 데이터셋 이용
  - 각 워크로드에 대해 총 100회씩 추론 요청을 수행함

표 68 추론 실험 데이터셋

DataSet	Workload	total token (request = 100)
ShareGPT	대화형	input token(22398) > output token(13342)
NuminaMath-CoT	수학문제	input token(7387) < output token(17408)

수행기관	개발 목표	개발내용 및 범위
		● 워크로드별로 Mooncake Store 컨넥터와 Mooncake Pipe 컨넥터 각각을 이용한 Llama 3.1 70B 모델에 대한 분리형 프리필 추론 서빙 실험을 실험 환경1과 환경 2에서 수행하며 TTFT[16], TPOT[17], ITL[18], E2EL[19] 측정하여 비교 - prefill 단계(TTFT), decode 단계(ITL, TPOT), 그리고 전체 시스템(E2EL)의 성능을 종합적으로 평가 - 추론 시스템의 안정성을 평가하기 위해 초당 들어오는 요청 수인 Request rate의 부하를 1, 4, 8, 16으로 증가시키며 실험을 진행 ● GPU-NPU 환경에서 Mooncake Connector 방식별 성능 비교 - ShareGPT 데이터셋 • Store 방식이 Pipe 방식 대비 TTFT 1.46배, TPOT 1.35배, ITL 1.21배, E2EL 1.13배 빠른 성능을 보임(1 req/s기준) • 이는 Store 방식이 KV 캐시를 Mooncake Store에 저장하여 KV 캐시 재사용의 효과를 얻으므로, 모든 성능 지표에서 Pipe 방식 대비 우수한 성능을 보임 - NuminaMath-CoT 데이터셋 • req/s 가 증가할수록 Store 방식이 Pipe 대비 우수한 성능을 보임 • 특히 TTFT와 E2EL에서 KV 캐시 저장 방식의 효율성이 잘 나타남 ● 동종 가속기 구성(환경1, GPU-GPU)과 이종 가속기 구성(환경2, GPU-NPU)의 성능 비교 - ShareGPT • Store connector 방식에서 GPU-NPU는 GPU-GPU 대비 TTFT 약 2.5배, TPOT 약 2.0배, ITL 약 1.5배, E2EL 약 1.8배 성능 저하 나타남 • Pipe connector 방식은 1 req/s에서만 측정 가능하여 최적화 필요함 - NuminaMath-CoT • Pipe connector 방식에서 GPU-NPU는 GPU-GPU 대비 TTFT 약 6.6배, TPOT, ITL, E2EL은 약 1.2~2.9배 성능 저하 나타남 • Store connector 방식에서 GPU-NPU는 GPU-GPU 대비 TTFT 약 8.3배, TPOT, ITL 약 1.2배, E2EL 약 2.8배 성능 저하 나타남 - 이종 가속기 구성은 동일 가속기 대비 모든 성능 지표에서 최소 1.2배에서 TTFT의 경우, 최대 8.3배 저하되며, 특히 TTFT와 E2EL에서 성능 격차가 크게 나타남. - TTFT는 이종 가속기 간 KV 캐시 통신 지연시간과 데이터 포맷 변환 시간이 포함되므로 최적화가 필요한 것으로 보임 - 동일 가속기 구성 대비 이종 가속기 구성의 성능이 전반적으로 저하되었으며, 성능 저하 원인을 분석하여 향후 연구 방향을 아래와 같이 도출함 • TTFT에서 2.5~8.3배의 큰 성능 격차가 발생하는 주요 원인은 GPU-NPU 간 KV 캐시 전송과 데이터 포맷 변환 과정이 포함되었기 때문임 --> GPU의 3차원 텐서 [input_token_num, kv_head, head_dim]를 NPU의 4차원 텐서 [mesh, kv_head, input_token_num, head_dim]로 변환하는 과정에서 차원 추가, 패딩, 차원 재배열(Key permute) 등의 오버헤드 발생할 수 있어 최적화가 필요함. • TPOT, ITL에서 1.2~2.0배 성능 저하가 발생하는 주요 원인은 NPU 자체의 하드웨어 성능 차이임 • 프로세스 아키텍처 불일치: 멀티 프로세스(GPU) => 싱글 프로세스(NPU) 오버헤드 . vLLM-GPU는 텐서 병렬화 환경에서 멀티 프로세스 기반으로 동작하며, 각 Worker가 독립적으로 분할된 KV 캐시와 hidden state를 전송함 . 반면 vLLM-TT는 싱글 프로세스로 동작하므로, 텐서 병렬화 차원 수만큼 worker로부터 수신한 분할 데이터를 병합하고, 재분산하는 추가 처리 단계가 필요함 . 이러한 데이터 수신 및 재구성 과정에서 동기화 오버헤드가 발생할 수 있어 최적화가 필요함

수행기관	개발 목표	개발내용 및 범위
		 그림 580 ShareGPT 워크로드 실험 결과  그림 581 NuminaMath-CoT 워크로드 실험 결과

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>

● GPU A100 .vs. Tenstorrent p150 하드웨어 가격 대비 성능 비교

- p150은 A100 대비 약 1/10 가격으로, Mathematical과 ShareGPT 모두에서 decode 단계 성능 효율이 우수함
- 1~16 req/s 평균 기준 Pipe(Mathematical) 방식에서 TPOT 1.23배, ITL 1.25배, Store(ShareGPT) 방식에서 TPOT 2.02배, ITL 1.51배로 GPU-GPU와 비교적 근접한 decode 성능을 제공하여 가격 대비 실용적임
- Pipe 방식(Mathematical)에서 TTFT는 평균 6.61배, E2EL은 평균 2.92배 느리고, Store 방식(ShareGPT)에서도 TTFT 평균 2.50배, E2EL 평균 1.82배 느려 prefill과 전체 응답 시간에서 성능 격차가 존재함
- 종합적으로 p150은 워크로드 특성에 따라 A100 대비 성능 저하를 보이나, 특정 워크로드와 Connector 환경에서 decode 단계 성능은 거의 동등하며 TTFT와 E2EL에서도 2~3배 성능 지연이 발생함.
- 이러한 성능 저하에도 불구하고 10배 가격 절감 효과로 개발, 연구 환경에서 가격 대비 성능 효율을 제공할 수 있음

표 69 가속기 가격 및 스펙 비교

항목	NVIDIA A100 80GB[20)21)]	Tenstorrent p150a[22)]
가격	$9,500-$14,000 (2025년 5월)	$1,299-$1,399 (2025년 4월)
메모리	80GB HBM2e	32GB GDDR6
메모리대역폭	2,039 GB/s	512 GB/s
컴퓨트 성능 (FP16)	312 TFLOPS	194 TFLOPS
전력 소비	300~400W	300W
$/GB VRAM	$118-$175/GB	$40-$43/GB

● 동종 가속기 구성(GPU-GPU) 대비 이종 가속기 구성(GPU-NPU)의 성능 저하 원인 분석

- 동종 가속기 구성 대비 이종 가속기 구성의 성능이 전반적으로 저하됨
  - TTFT에서 2.5~8.3배, TPOT과 ITL에서 1.2~2.0배 성능 저하가 발생
- GPU-NPU 간 데이터 포맷 변환 과정 부하 : TTFT에 영향
  - GPU의 3차원 텐서 [input_token_num, kv_head, head_dim]를 NPU의 4차원 텐서 [mesh, kv_head, input_token_num, head_dim]로 변환하는 과정에서 차원 추가, 패딩, 차원 재배열(Key permute) 등의 오버헤드
- vLLM-GPU와 vLLM-TT의 텐서 병렬화를 위한 프로세스 아키텍처 불일치로 인한 오버헤드 : TTFT에 영향
  - vLLM-GPU는 텐서 병렬화 환경에서 멀티 프로세스 기반으로 동작하나 vLLM-TT는 싱글 프로세스로 동작
  - vLLM-TT에서 vLLM-GPU가 전송한 KV 캐시를 수신한 후 병합하고 재분산하는 추가 단계 수행
- GPU 대비 NPU 자체의 하드웨어 성능 차이 : TPOT, ITL에 영향

● GPU 4개가 장착된 노드와 NPU 4개가 장착된 이종 가속기 환경에서 Llama 3.1 70B에 대한 추론 서빙이 정상적으로 수행됨을 보임으로써 연차 성능 목표 달성 확인

</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

표 70 연차 성능 목표 달성 성과

항목	연차 성능 목표	연차 달성 성과
최대 지원 모델 파라미터 개수	64B	70B
병렬 추론 지원 이종 가속기 개수	8	8 (GPU 4 + 텐스토렌트 NPU 4)
통합 실행 지원 이종 가속기 종류	1	2 (GPU, NPU)

(10) 국내 NPU 개발 기업인 ㈜ 하이퍼엑셀과 연구 협력 추진

● 하이퍼엑셀 LPU 구조 분석

- LPU(LLM Processing Unit)는 LLM 추론 전용 하드웨어 아키텍처로, LLM 추론 과정에서 반복적으로 발생하는 연산 패턴(예, 어텐션, MLP)을 하드웨어 수준에서 최적화
  - 트랜스포머 연산의 데이터플로우를 하드웨어에 맞게 최적화(streamlined dataflow)하여 효율을 극대화
  - LLM 추론 과정에서 발생하는 메모리 접근 병목을 완화하기 위해 메모리 계층과 메모리 대역폭 활용을 최적화
- LLM 워크로드에 대한 추론 지연 최소화에 중점을 둠

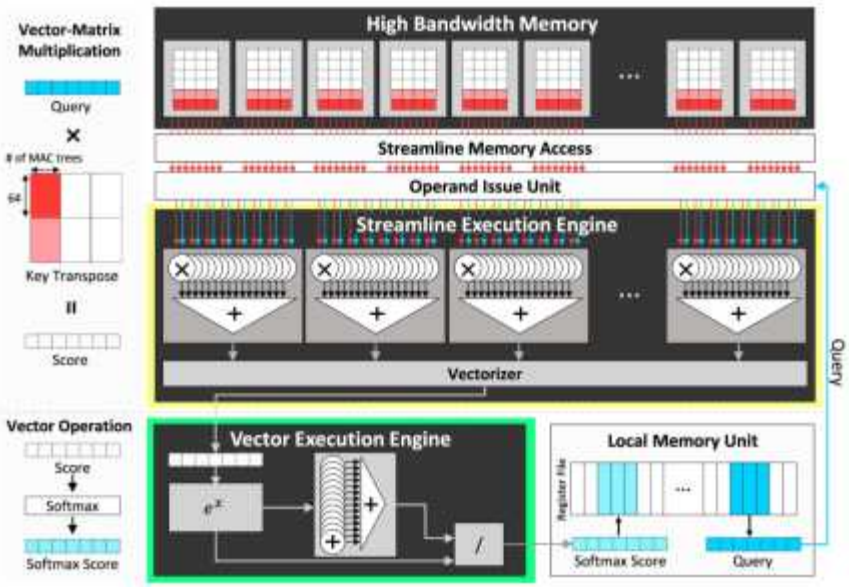


그림 582 멀티헤드 어텐션을 위한 LPU 데이터플로우

● LLM 추론을 위한 하이퍼덱스 툴체인(HyperDex Toolchain) 소프트웨어 스택 분석

- 하이퍼덱스 툴체인은 LPU 하드웨어에서 AI 워크로드를 손쉽게 배포하고 실행할 수 있도록 설계된 통합 소프트웨어 스택으로 장치 드라이버, 런타임 환경, 컴파일러 도구를 포함
  - hyperdex.tools : 허깅페이스 Transformers 모델을 LPU 하드웨어 아키텍처에 맞춰 쉽게 변환할 수 있도록 설계된 Python 기반 컴파일러 인터페이스
  - hyperdex.transformers : 허깅페이스 Transformers 인터페이스와 완전히 호환되는 API를 이용하여 LPU에서 LLM 추론을 실행하기 위한 런타임 환경을 제공
- 추론을 위해 허깅페이스로부터 모델을 다운로드하여 하이퍼엑셀 LPU에 적합하도록 모델 웨이트를 분할하고 LPU에 최적화된 추론 인스트럭션을 생성
  - 추론에 사용하는 LPU 디바이스 수에 기반하여 모델 분할
- 단일 노드에 FPGA 형태인 LPU와 GPU가 동시 장착된 환경에서 하이퍼덱스 툴체인 수준의 분리형 프리필 형태 추론 지원
- 대규모 LLM 추론 서빙에서 확장성과 성능 최적화에 다음과 같은 제약으로 한계가 존재
  - 다수의 추론 요청에 대한 배치처리 미지원

수행기관	개발 목표	개발내용 및 범위
		• 단일 프로세스가 노드 내 다중 LPU 디바이스를 활용하는 제한적 텐서 병렬화 : 다중 프로세스 기반 텐서 병렬화 미지원 • LPU 디바이스 간 파이프라인 병렬화 미지원 • 다중 노드 환경에서 다중 LPU 디바이스 활용을 위한 (집합) 통신 라이브러리 부재

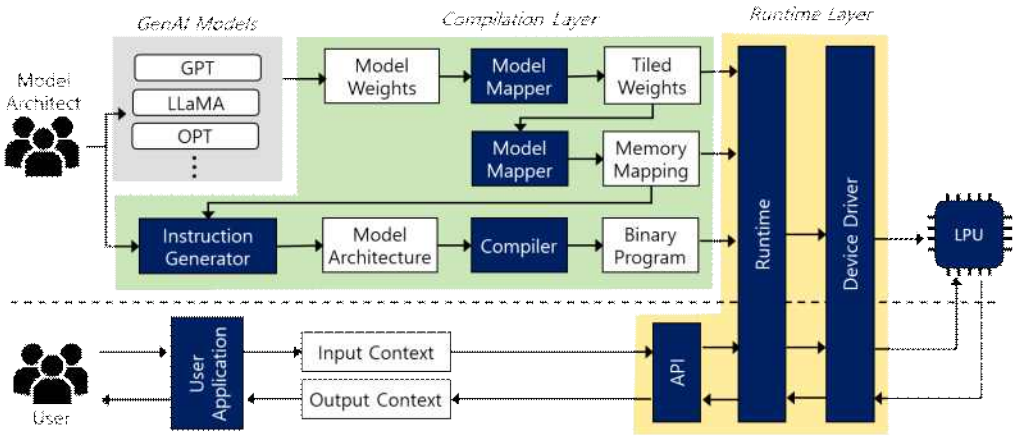
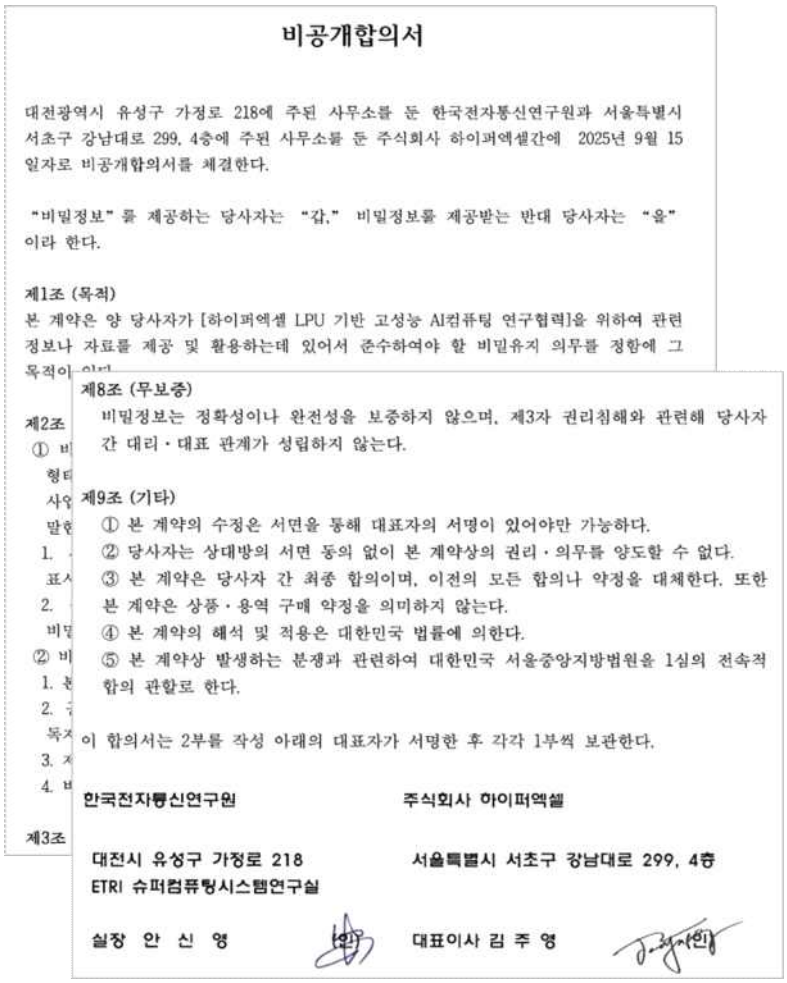


그림 583 LPU를 위한 하이퍼덱스 툴체인 소프트웨어 스택



비공개합의서

대전광역시 유성구 가정로 218에 주된 사무소를 둔 한국전자통신연구원과 서울특별시 서초구 강남대로 299, 4층에 주된 사무소를 둔 주식회사 하이퍼엑셀간에 2025년 9월 15일자로 비공개합의서를 체결한다.

"비밀정보"를 제공하는 당사자는 "갑," 비밀정보를 제공받는 반대 당사자는 "을"이라 한다.

제1조 (목적)
본 계약은 양 당사자가 [하이퍼엑셀 LPU 기반 고성능 AI컴퓨팅 연구협력]을 위하여 관련 정보나 자료를 제공 및 활용하는데 있어서 준수하여야 할 비밀유지 의무를 정함에 그 목적이

제8조 (무보증)
비밀정보는 정확성이나 완전성을 보증하지 않으며, 제3자 권리침해와 관련해 당사자 간 대리·대표 관계가 성립하지 않는다.

제9조 (기타)
① 본 계약의 수정은 서면을 통해 대표자의 서명이 있어야만 가능하다.
② 당사자는 상대방의 서면 동의 없이 본 계약상의 권리·의무를 양도할 수 없다.
③ 본 계약은 당사자 간 최종 합의이며, 이전의 모든 합의나 약정을 대체한다. 또한 본 계약은 상품·용역 구매 약정을 의미하지 않는다.
④ 본 계약의 해석 및 적용은 대한민국 법률에 의한다.
⑤ 본 계약상 발생하는 분쟁과 관련하여 대한민국 서울중앙지방법원을 1심의 전속적 합의 관할로 한다.

이 합의서는 2부를 작성 아래의 대표자가 서명한 후 각각 1부씩 보관한다.

한국전자통신연구원
대전시 유성구 가정로 218
ETRI 슈퍼컴퓨팅시스템연구실
실장 안 신 영

주식회사 하이퍼엑셀
서울특별시 서초구 강남대로 299, 4층
대표이사 김 주 영

그림 584 ㈜ 하이퍼엑셀-ETRI 간 연구협력을 위한 NDA

● ㈜ 하이퍼엑셀-ETRI간 LLM 추론 효율화를 위한 연구협력 아이템으로 하이퍼엑셀 LPU를 위한 RDMA 기반 통신 라이브러리 개발 제안

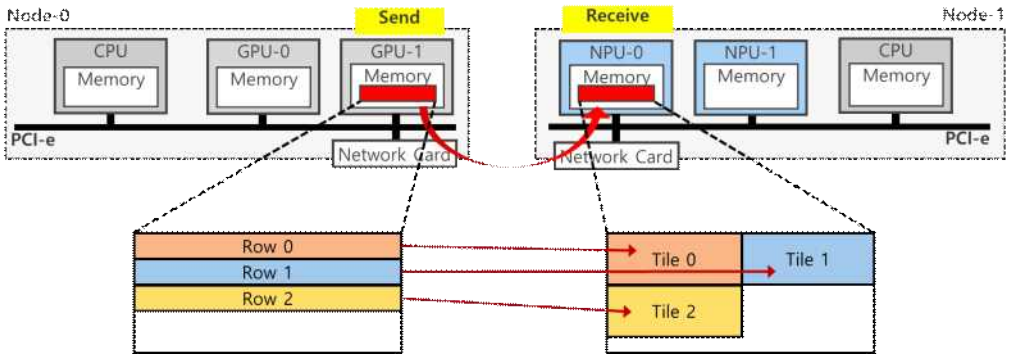
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>● vLLM 응용이 하이퍼엑셀 LPU 상에서 동작할 수 있도록 하이퍼덱스 툴체인에 기반하여 vLLM을 확장하고 있는데 이를 vLLM 플러그인화 하도록 유도
● ㈜ 하이퍼엑셀-ETRI간 고성능 AI 컴퓨팅 연구 협력을 위한 비밀유지계약(Non-Disclosure Agreement, 이하 NDA) 체결</td></tr>
<tr><td></td><td>이종 가속기 기반 추론 지원 집합통신 기술 연구</td><td>(1) 이종 가속기간 집합통신을 위한 구조 설계
● 네트워크 카드(infiniband)를 활용한 이종 가속기간 직접 메모리 통신 제공
- 인피니밴드 기반 네트워크 카드를 이용해 이종 가속기의 메모리 공간 간의 직접 데이터 전송이 가능한 RDMA(Remote Direct Memory Access) 경로를 개발하여, 호스트 메모리 경유에 따른 불필요한 데이터 복사 및 지연을 최소화
- 네트워크 카드 SW 스택과의 호환성을 확보하기 위해 가속기 측 Peer Memory Client를 개발하고, 이를 기존 가속기 소프트웨어 스택(커널 모듈 드라이버, 유저 모듈 드라이버, 고수준 Python API 등)에 연동하고 확장하여 이종 가속기 메모리의 등록 및 접근을 지원
- 네트워크 카드 소프트웨어 스택(리눅스 표준 인피니밴드 커널 모듈)을 통해 이종 가속기 메모리 영역에 대한 접근 권한을 안전하게 관리하고, 가상·물리 주소 매핑 정보를 네트워크 계층으로 전달하는 주소 체계 연동
- 표준 오픈소스 유저 레벨 RDMA 라이브러리(libibverbs)를 확장하여 이기종 피어 디바이스(Peer Device)간 메모리 공간 전송 및 수신 큐페어(Queue Pair) 관리
 

그림 585 이종 가속기간 직접 데이터 통신 개념
● 네트워크 카드의 통신 API를 이용한 이종 가속기 간 데이터 전송 API 제공
- 표준 오픈소스 유저 레벨 RDMA 라이브러리를 기반으로 이종 가속기 간 데이터 전송을 추상화한 전용 라이브러리를 개발하여, 상위 소프트웨어 계층에서 일관된 방식으로 통신 기능을 활용할 수 있도록 함
- 네트워크 카드 대역폭 이용을 최적화함으로써 전체 통신 효율을 극대화
- GPU와 NPU 등 각 이종 가속기의 메모리 구조, 접근 방식, 대역폭 및 지연 특성 등을 분석하여 디바이스별 메모리 요구 특성을 반영한 전송 전략 수립
- 이종 가속기 간 데이터 일관성과 연산 정확성을 보장하기 위해 전송을 정밀하게 제어
● 이종 가속기간 직접 메모리 통신 기반으로 워크로드 특성 및 요구에 부합하는 상위 집합 통신 제공
- LLM 추론 및 학습 워크로드의 연산 단계별 특성을 반영하여 단계별로 최적화된 집합 통신 제공
- GPU와 NPU 등 디바이스별 연산 성능, 메모리 대역폭, 통신 지연 특성을 고려하여, 동일한 집합 통신 연산이라도 디바이스 조합에 따라 차별화된 알고리즘을 적용하는 이종 인지형(heterogeneity-aware) 집합 통신 기법 제공
- 이종 디바이스 간 연결 방식(직접 RDMA 연결, 스위치 경유, 계층적 네트워크 구조 등)에 따른 네트워크 토폴로지를 분석하고, 이에 적합한 집합 통신 구조 제공</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

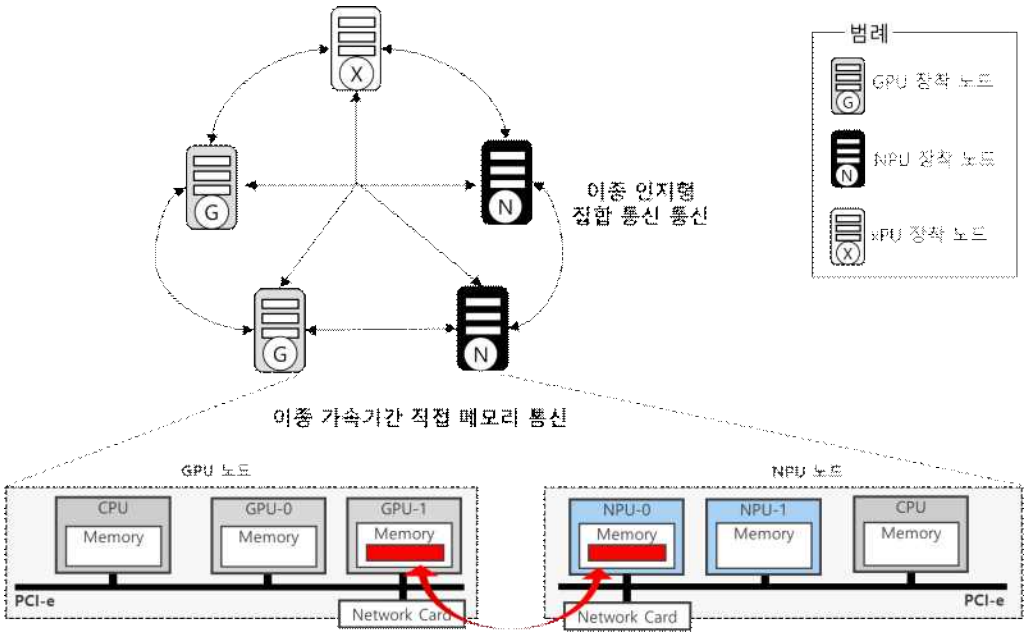


그림 586 이종 가속기간 집합 통신 구조

(2) FPGA-GPU 간 RDMA(Remote Direct Memory Access) 기술 개발

- FPGA 기반 PCIe 가속 장치 하드웨어인 PAP (PCIe Accelerator PoC Device)의 RDMA 동작을 지원하는 소프트웨어 스택 설계
  - PAP 하드웨어는 호스트 서버에 PCIe 인터페이스로 연결되며, 각각 호스트 노드는 InfiniBand HCA를 통해 다른 호스트 노드와 연결됨
  - 커널 영역에서는 수정된 XDMA 드라이버인 PAP 드라이버가 동작하고, Peer Memory Client (PMC)는 PAP 드라이버의 보조 커널 모듈로 동작함
  - InfiniBand HCA Driver는 InfiniBand HCA (NIC)를 제어하며, 데이터 전송 명령 처리 등의 핵심 기능을 수행함.
  - RDMA 라이브러리는 패킷 전송, 메모리 등록, 큐페어 관리와 같은 RDMA 통신의 핵심 기능을 수행함
  - PMC는 InfiniBand HCA Driver가 해당 RDMA 동작, 세부적으로는 PAP의 메모리 영역에 대하여 관련 동작을 처리할 수 있도록 커널 단계에서 메모리 등록 및 매핑을 지원함

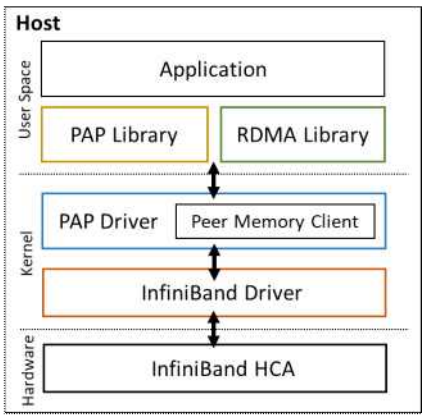


그림 587 PAP 소프트웨어 스택

- PAP Driver RDMA 지원 함수 개발
  - PAP Peer Memory Client는 PAP 장치의 RDMA 동작을 위해 PAP 장치의 메모리 영역에 대한 DMA 등록 및 해제 함수를 리눅스 표준 인피니밴드 커널 모듈 및 NIC 드라이버에 제공하는 모듈

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>

• 이를 통해 NIC 장치가 PAP 메모리를 접근하여 직접 읽고 쓰는 동작을 실행할 수 있음

- PAP 드라이버 내부에서 동작하는 함수들은 cdev_sgdma.h에 정의되어 있으며, 함수 symbol exporting을 통해 Peer Memory Client 모듈에 전달됨 : 표 71 참고

표 71 PAP 드라이버 내부 동작 함수

<table>
<tr><th>함수 이름</th><th>설명</th></tr>
<tr><td>pap_p2p_get_pages</td><td>사용자 영역 가상 주소, 메모리 영역 크기를 전달받아 PAP 장치 메모리 영역에 해당하는지 확인하고, 해당하면 해당 영역의 페이지에 대한 DMA pinning을 수행하고 메모리 매핑 정보를 Peer Memory Client 모듈에 반환</td></tr>
<tr><td>pap_p2p_put_pages</td><td>메모리 매핑 정보를 전달받아 해당 영역의 페이지에 대한 DMA pinning 상태를 해제하고 사용이 종료된 구조체들을 메모리 할당 해제</td></tr>
<tr><td>pap_p2p_dma_map_pages</td><td>pap_p2p_get_pages를 통해 전달받은 메모리 매핑 정보를 dma 주소 기반 매핑 정보로 변환하여 Peer Memory Client 모듈에 반환</td></tr>
<tr><td>pap_p2p_dma_unmap_pages</td><td>사용이 완료된 dma 매핑 정보 구조체를 메모리 할당 해제</td></tr>
</table>

- Peer Memory Client에 정의된 함수들은 PAP 드라이버 내부 동작 함수들을 호출하여 NIC 드라이버 동작에 필요한 함수들을 중계함 : 표 72 참고

표 72 PAP을 위한 Peer Memory Client 함수

<table>
<tr><th>함수 이름</th><th>설명</th></tr>
<tr><td>pap_acquire</td><td>NIC 드라이버의 acquire 함수를 hooking 하여, Peer Memory Client 동작 기본 단위인 pap context 구조체를 생성하고, 해당 구조체에 전달받은 가상주소에 대한 메모리 유효 확인 및 PAP 메모리 매핑 정보를 저장</td></tr>
<tr><td>pap_release</td><td>NIC 드라이버의 release 함수를 hooking 하여, 사용이 완료된 pap context 구조체에 대해서 생성된 sg_table 및 PAP 구조체를 메모리 할당 해제</td></tr>
<tr><td>pap_dma_map</td><td>NIC 드라이버의 dma_map 함수를 hooking 하여, PAP 메모리 매핑 정보를 dma 주소 기반 매핑 정보로 변환하고, 변한된 매핑 정보를 기반으로 RDMA 동작을 위한 sg_table을 생성</td></tr>
<tr><td>pap_dma_unmap</td><td>NIC 드라이버의 dma_unmap 함수를 hooking 하여, dma 매핑 정보를 담고 있는 구조체를 메모리 할당 해제</td></tr>
<tr><td>pap_get_pages</td><td>NIC 드라이버의 get_pages 함수를 hooking 하는데, PAP Peer Memory Client의 경우 pap_acquire에서 메모리 매핑 정보 및 dma pinning을 수행하기 때문에 별도 동작은 수행하지 않음</td></tr>
<tr><td>pap_put_pages</td><td>NIC 드라이버의 put_pages 함수를 hooking 하며, 사용이 완료된 dma 매핑 정보 구조체에 대해 메모리를 할당 해제</td></tr>
<tr><td>pap_get_page_size</td><td>NIC 드라이버의 get_page_size 함수를 hooking 하며, PAP 메모리 영역에 대해서 PAP 드라이버 내부 사용된 페이지 사이즈를 반환</td></tr>
</table>

● GPU-PAP 간 데이터 전송 성능 비교

- 기존 GPU-PAP 간 데이터 통신 대비 RDMA 직접 통신이 약 400% 향상된 성능(지연 시간)을 보임

• 기존 GPU-PAP 간 데이터 전송의 경우 각 노드의 호스트의 메모리 공간을 필수적으로 거치게 되어 추가적인 지연 및 메모리 대역폭 낭비가 발생함

• 반면 개발한 RDMA 기술을 이용할 경우, 호스트 메모리 공간 참조로 인한 오버헤드가 제거됨

</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

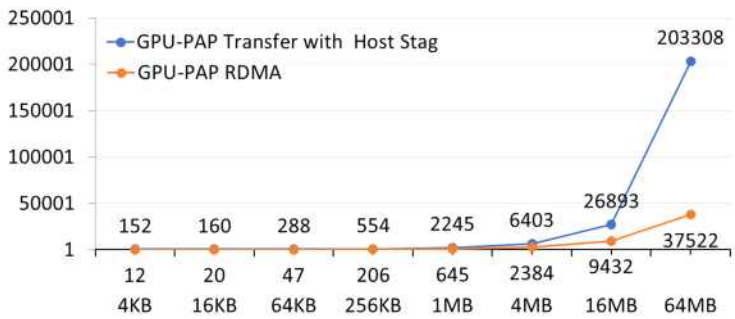


그림 588 GPU-PAP 간 데이터 전송시 호스트 경우와 RDMA 데이터 전송의 지연 시간 비교

(3) TT-GPU간 RDMA 기술 개발

- 텐스토렌트 NPU의 RDMA 동작을 지원하는 소프트웨어 스택 설계
  - Tenstorrent NPU 하드웨어는 호스트 서버에 PCIe 인터페이스로 연결되며, 각각 호스트 노드는 InfiniBand HCA를 통해 다른 호스트 노드와 연결됨
  - Tenstorrent Peer Memory Client (PMC)는 리눅스 표준 인피니밴드 커널 모듈 및 InfiniBand HCA Driver가 해당 RDMA 동작, 세부적으로는 Tenstorrent NPU의 메모리 영역에 대하여 관련 동작을 처리할 수 있도록 커널 단계에서 메모리 등록 및 매핑을 지원함

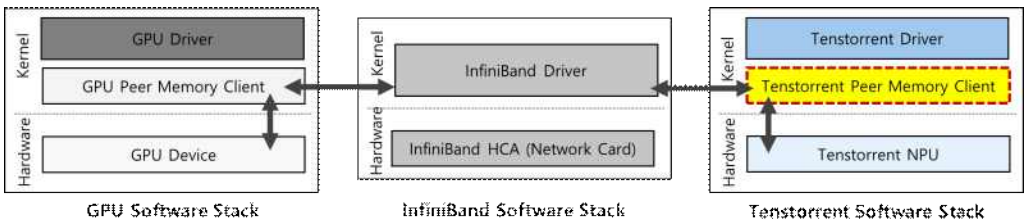


그림 589 소프트웨어 스택 관점 TT-GPU간 RDMA 통신 개념

- Tenstorrent NPU Kernel & User Module Driver RDMA 지원 함수 개발
  - Tenstorrent Peer Memory Client는 Tenstorrent NPU 장치의 RDMA 동작을 위해 NPU 장치의 메모리 영역에 대한 DMA 등록 및 해제 함수를 리눅스 표준 인피니밴드 커널 모듈 및 NIC 드라이버에 제공하는 모듈
    - 이를 통해 NIC 장치가 NPU 메모리에 접근하여 직접 읽고 쓰는 동작을 실행할 수 있음
  - NPU 드라이버 내부에서 동작하는 함수들은 symbol exporting을 통해 Peer Memory Client 모듈에 전달됨 : 표 73 참고

표 73 텐스토렌트 NPU 드라이버 내부 동작 함수

함수 이름	설명
tt_p2p_get_pages	사용자 영역 가상 주소, 메모리 영역 크기를 전달받아 NPU 장치 메모리 영역에 해당하는지 확인하고, 해당하면 해당 영역의 페이지에 대한 DMA pinning을 수행하고 메모리 매핑 정보를 Peer Memory Client 모듈에 반환
tt_p2p_put_pages	메모리 매핑 정보를 전달받아 해당 영역의 페이지에 대한 DMA pinning 상태를 해제하고 사용이 종료된 구조체들을 메모리 할당 해제
tt_p2p_dma_map_pages	tt_p2p_get_pages를 통해 전달받은 메모리 매핑 정보를 dma 주소 기반 매핑 정보로 변환하여 Peer Memory Client 모듈에 반환
tt_p2p_dma_unmap_pages	사용이 완료된 dma 매핑 정보 구조체를 메모리 할당 해제

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>

- Peer Memory Client 에 정의된 함수들은 NPU 드라이버 내부 동작 함수들을 호출하여 NIC 드라이버 동작에 필요한 함수들을 중계함 : 표 74 참고

표 74 텐스토렌트 NPU를 위한 Peer Memory Client 함수

<table>
<tr><th>함수 이름</th><th>설명</th></tr>
<tr><td>tt_acquire</td><td>NIC 드라이버의 acquire 함수를 hooking 하여, Peer Memory Client 동작 기본 단위인 tt_context 구조체를 생성하고, 해당 구조체에 전달받은 가상주소에 대한 메모리 유효 확인 및 NPU 메모리 매핑 정보를 저장</td></tr>
<tr><td>tt_release</td><td>NIC 드라이버의 release 함수를 hooking 하여, 사용이 완료된 tt_context 구조체에 대해서 생성된 sg_table 및 NPU 구조체를 메모리 할당 해제</td></tr>
<tr><td>tt_dma_map</td><td>NIC 드라이버의 dma_map 함수를 hooking 하여, NPU 메모리 매핑 정보를 dma 주소 기반 매핑 정보로 변환하고, 변한된 매핑 정보를 기반으로 RDMA 동작을 위한 sg_table을 생성</td></tr>
<tr><td>tt_dma_unmap</td><td>NIC 드라이버의 dma_unmap 함수를 hooking 하여, dma 매핑 정보를 담고 있는 구조체를 메모리 할당 해제</td></tr>
<tr><td>tt_get_pages</td><td>NIC 드라이버의 get_pages 함수를 hooking 하는데, 텐스토렌트 Peer Memory Client의 경우 pap_acquire에서 메모리 매핑 정보 및 dma pinning을 수행하기 때문에 별도 동작은 수행하지 않음</td></tr>
<tr><td>tt_put_pages</td><td>NIC 드라이버의 put_pages 함수를 hooking 하며, 사용이 완료된 dma 매핑 정보 구조체에 대해 메모리를 할당 해제</td></tr>
<tr><td>tt_get_page_size</td><td>NIC 드라이버의 get_page_size 함수를 hooking 하며, NPU 메모리 영역에 대해서 텐스토렌트 NPU 드라이버 내부 사용된 페이지 사이즈를 반환</td></tr>
</table>

● GPU-Tenstorrent NPU 간 데이터 전송 성능 비교

- 기존 GPU-NPU 간 데이터 통신 대비 RDMA 직접 통신이 약 100% 향상된 성능(지연시간)을 보임
  - 기존 GPU-NPU 간 데이터 전송의 경우 각 노드의 호스트의 메모리 공간을 필수적으로 거치게 되어 추가적인 지연 및 메모리 대역폭 낭비가 발생함
  - 반면 개발한 RDMA 기술을 이용할 경우, 호스트 메모리 공간 참조로 인한 오버헤드가 제거됨

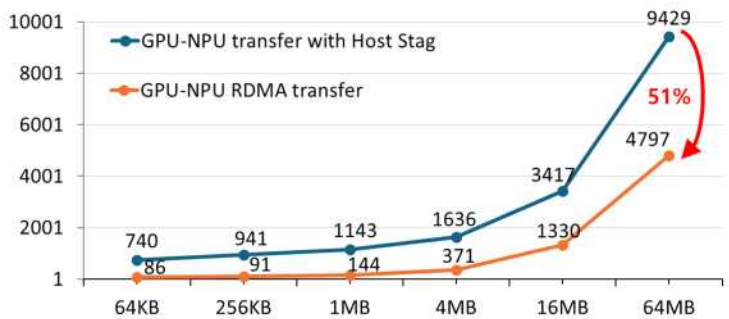


그림 590 GPU-NPU 간 데이터 전송시 호스트 경우와 RDMA 데이터 전송의 지연 시간 비교

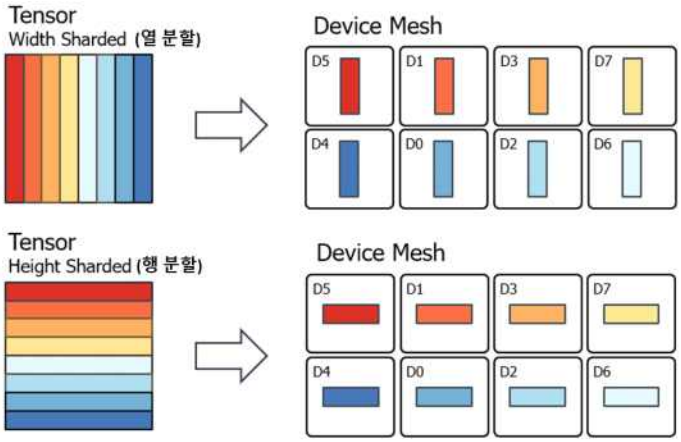
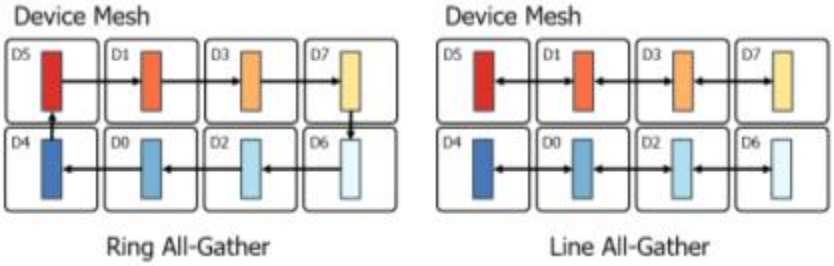
</td></tr>
<tr><td>한국항공대학교
(공동1)</td><td>NPU 기반 AI 추론컴퓨팅 자동병렬화 핵심기술 연구</td><td>

(1) 노드 레벨 NPU간 집합통신 기술 연구

● 텐서 병렬화를 위한 메쉬 내 통신 기술 분석

- 텐서 병렬화를 지원하기 위한 텐스토렌트 NPU 신경망 라이브러리(TTNN)의 집합통신 기능 분석 및 성능 평가 수행

</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위
		• TTNN의 데이터 분할 방식 : 열 분할 및 행 분할 지원  그림 591 1x8 디바이스 메쉬에 대한 텐서 열 분할과 행 분할 예  그림 592 x8 디바이스 메쉬에 대한 링 토폴로지, 선형 토폴로지 올게더 예 • NPU 8대(4카드, 8칩)를 사용하여 열 분할 및 행 분할 환경 구성하고(그림 591), 링 방식(Ring All-Gather) 및 선형 방식(Line All-Gather) 기반의 집합통신 실험 수행(그림 592) • 실험 조건: 텐서 분할 방식(열 분할, 행 분할), 통신 토폴로지(링, 선형), 데이터 크기(1Gb, 1GB, 8GB) • 실험 결과: TTNN v0.60 기준 분할 방식, 통신 토폴로지, 데이터 크기 변화와 무관하게 통신 시간이 약 0.68초로 일정하게 측정됨 • 실제 데이터 이동 시간 이외의 고정 오버헤드 존재 가능성 제기 - 실행시간 내 오버헤드 및 데이터 이동 시간 분석을 위한 프로파일링 수행 • 프로파일링 결과 전체 통신 시간 0.68초 중 대부분이 All-Gather 함수 최초 컴파일 과정에서 발생, 실제 데이터 전송 시간은 마이크로초(μs) 단위로 매우 짧게 측정 • 최초 실행 이후 동일 설정으로 반복 실행 시 컴파일 과정 생략으로 통신 시간 마이크로초 수준으로 감소

수행기관	개발 목표	개발내용 및 범위

그림 593 1x8 디바이스 메쉬 8GB 텐서 열 분할 (칩 당 1GB) 링 올 게더 1회 실행 프로파일링 결과

- 실험 및 분석 결과를 통해, TTNN 집합통신의 초기 오버헤드 특성과 실제 데이터 전송 비용의 분리 가능성을 확인
- 향후 NPU 간 병렬화 전략 수립 및 통신비용 모델링 알고리즘 설계 시, 초기 컴파일 오버헤드를 반영한 통신비용 계산 근거로 활용 가능

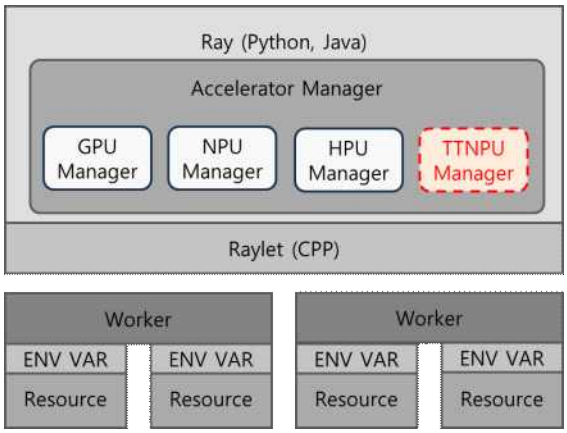
● 파이프라인 병렬화를 위한 메쉬 간 통신 기술 분석

- 파이프라인 병렬화 수행 시 스테이지 간 활성값(Activation Value) 전달이 발생하는데 활성값 이동은 동일 메쉬 내 데이터 이동이 아닌 메쉬 간 데이터 이동(Inter-Mesh Communication) 에 해당
- TTNN의 집합 통신은 메쉬 내 통신(Intra-Mesh)만 지원하여 파이프라인 병렬화 환경에서 사용 불가하므로 TTNN 소켓(Socket) 기반 통신 기능을 활용한 메쉬 간 데이터 이동 가능성 검증 필요
- 파이프라인 병렬화를 위한 TTNN 메쉬 간 송신/수신(Send/Receive) 기능 지원 여부 및 성능 평가를 수행

표 75 메쉬 간 소켓 송신/수신 실행시간

	[1,5,4096]	[1,5]	[1,32,1,5]
메모리 버퍼: DRAM 소켓 버퍼: DRAM	1.2 초	1.2 초	1.22 초
메모리 버퍼: DRAM 소켓 버퍼: L1	1.22 초	1.21 초	1.23 초
메모리 버퍼: L1 소켓 버퍼: DRAM	0.89 초	0.89 초	0.89 초
메모리 버퍼: L1 소켓 버퍼: L1	0.62 초	0.6 초	0.6 초

- Bloomz-7b1 모델을 대상으로 활성값 텐서 형태 확인: [1,5,4096], [1,5], [1,32,1,5]
- TTNN 소켓 성능 평가를 위한 비교 실험 조건: 메모리 버퍼 위치(DRAM, L1 Cache), 소켓 버퍼 위치(DRAM, L1 Cache), 전송 텐서 ([1,5,4096], [1,5], [1,32,1,5])
- 실험 결과 TTNN 소켓 통신은 TTNN 집합통신과 동일하게 데이터 크기와 무관하게 일정한 실행 시간이 소요되며, 메모리 버퍼 및 소켓 버퍼 위치(DRAM/L1 Cache)가 성능에 큰 영향을 미침을 확인
- TTNN 소켓 기반 통신은 메쉬 간 활성값 전송 기능을 제공하나, 메모리 계층 구조를

수행기관	개발 목표	개발내용 및 범위
		고려하지 않을 경우 성능 저하 발생 가능성 확인 (2) NPU기반 자동병렬화 핵심기술 연구 ● Ray 멀티프로세싱 프레임워크 활용 방안 연구 - 대규모 병렬화 시스템 구현을 위해 효율적인 자원 관리, 멀티프로세싱 스케줄링, 서버 클러스터링 기능 필요 - Ray 프레임워크는 CPU, 메모리, 가속기 자원 관리, 멀티프로세싱 스케줄링, 서버 클러스터링 기능 제공하나 현재 배포 중인 Ray는 텐스토렌트 NPU를 지원하지 않으므로, Ray 환경에서 텐스토렌트 NPU 자원을 인식하고 관리할 수 있도록 기능 확장 필요 - Ray 코드 분석 결과, Ray는 Raylet이 제공하는 가용 자원 ID를 기반으로 가속기 매니저(Accelerator Manager)가 환경변수(ENV VAR)를 이용해 가속기 자원을 관리하는 구조임을 확인 - 텐스토렌트 NPU의 저수준 프로그래밍 모델인 TT-Metal v0.60 이상 버전에서는 환경변수를 통한 가속기 격리 기능을 지원 (격리 단위: 칩 단위가 아닌 PCIe 슬롯에 인식된 카드 단위로 구성 가능) - 유휴 TensTorrent NPU 자원을 Raylet이 인식할 수 있도록 다음과 같은 기능 추가 • Ray에서 사용 가능한 가속기 관리 함수 포함한 가속기 매니저 객체(TTNPU Manager) 생성 • TT-Metal의 환경변수 기반 가속기 격리 기능 연동 • Raylet 내부에 텐스토렌트 NPU 객체 변수 등록  그림 594 Ray 가속기 자원 관리 기능 구조도 - NPU 서버 2대(1카드 장착 서버 1대, 4카드 장착 서버 1대)를 활용하여 Ray 연동 텐스토렌트 가속기 자원 인식 기능 검증 • 클러스터 상태를 확인하여 5대의 TTNPU 객체가 정상적으로 인식된 것을 확인

수행기관	개발 목표	개발내용 및 범위

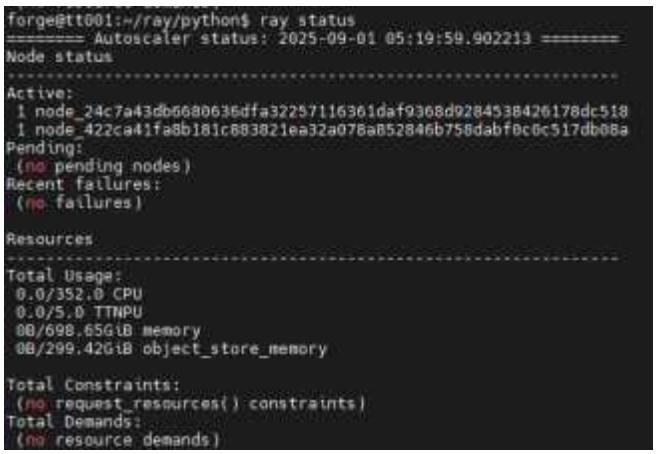


그림 595 Ray 클러스터링 상태 확인

- JAX를 사용한 자동 병렬화 기술 연구
  - 기존의 병렬화 기법은 모델 구조를 수동으로 수정해야 하므로, 사전에 병렬화가 구현된 모델만 사용 가능하고 병렬화 방식이 모델 개발자에 의해 고정적으로 지정됨
  - 허깅페이스에서 제공하는 다양한 모델에 대해 범용적으로 적용 가능한 자동 병렬화 기술의 필요성 대두됨
  - 저수준 연산 단위를 분할하고, 그래프 수준에서 병렬화를 수행할 수 있는 AI 그래프 컴파일러 기반 접근법 연구 진행
    - 텐스토렌트에서 제공하는 AI 그래프 컴파일러 중 멀티 디바이스를 지원하는 유일한 컴파일러인 TT-XLA 활용
  - XLA를 기본 컴파일러로 사용하는 JAX 프레임워크 기반으로 파이프라인 병렬화 기법 연구 진행
    - JAX는 파이썬 함수를 람다(lambda) 함수 수준의 JAXPR 표현식으로 변환하는 기능 제공
    - 허깅페이스에서 불러온 대규모 모델을 JAXPR 형태로 변환 후, 이를 분할하여 파이프라인 스테이지를 구성하는 방식으로 자동 병렬화 구조 개발
    - 거대한 네트워크를 저수준 함수로 변환하여 분할시키는 방식임으로 트랜스포머 블록 경계, 각 스테이지별 모델 파라미터, 활성값을 정확히 연결해 주지 않으면 오류가 발생하므로 경계, 파라미터, 활성값을 찾기 위한 탐색 과정 필요
    - 각 요소를 자동으로 탐색하고 연결하기 위한 경계·파라미터·활성값 탐색 알고리즘 구현

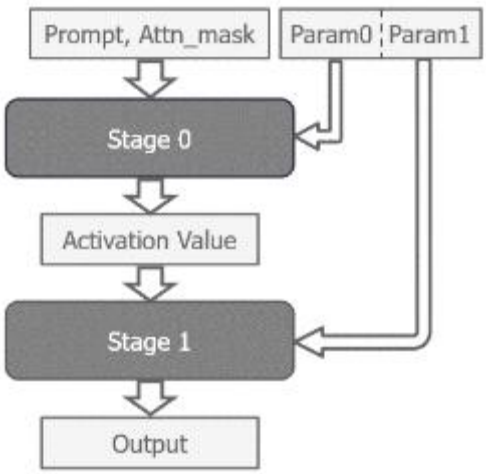


그림 596 2개의 파이프라인 스테이지 구성 예

수행기관	개발 목표	개발내용 및 범위

- 그림 595와 같이 2개의 파이프라인 스테이지를 구성한 후 파이프라인 처리 성능 향상을 위해 파이프라인 오버랩(Pipeline Overlap) 기법 적용하고, 오버랩 병렬화를 위한 Ray 멀티프로세싱 기반 실행 구조 구현
- 웜홀(Wormhole) 서버와 블랙홀(Blackhole) 서버를 Ray 클러스터링으로 연결하여 분산 실행 환경을 구성한 후 장치별 성능 차이 및 오버랩 정도(2, 4, 10회 추론 요청)에 따른 성능 비교 실험 수행 : Bloomz-7b1 모델 이용
- 장치 성능별 추론 시간 비교 결과 : 블랙홀 대비 웜홀에서 첫 토큰 생성 시간 짧았으며, 블랙홀의 경우 첫 토큰 생성 시간은 증가하나 이후 토큰 생성 시간은 감소하여 결과적으로 총 토큰 생성 시간은 단축되는 경향을 확인함

표 76 장치 성능 차이에 의한 추론 시간 평가

	서버 1대 사용 (웜홀)	서버 1대 사용 (블랙홀)
첫 토큰 생성 시간	68.47초	73.81초
이후 토큰 생성 시간 (중위값)	17초	16초
총 토큰 생성 시간 (30 토큰)	575.79초	554.79초

- 오버랩 성능 측정 결과 : 4 요청 오버랩까지는 명확한 성능 향상 확인, 이후 요청 증가 시 추가 향상은 미미함

표 77 웜홀, 블랙홀 서버 클러스터 구성 및 오버랩 적용 추론 시간 평가

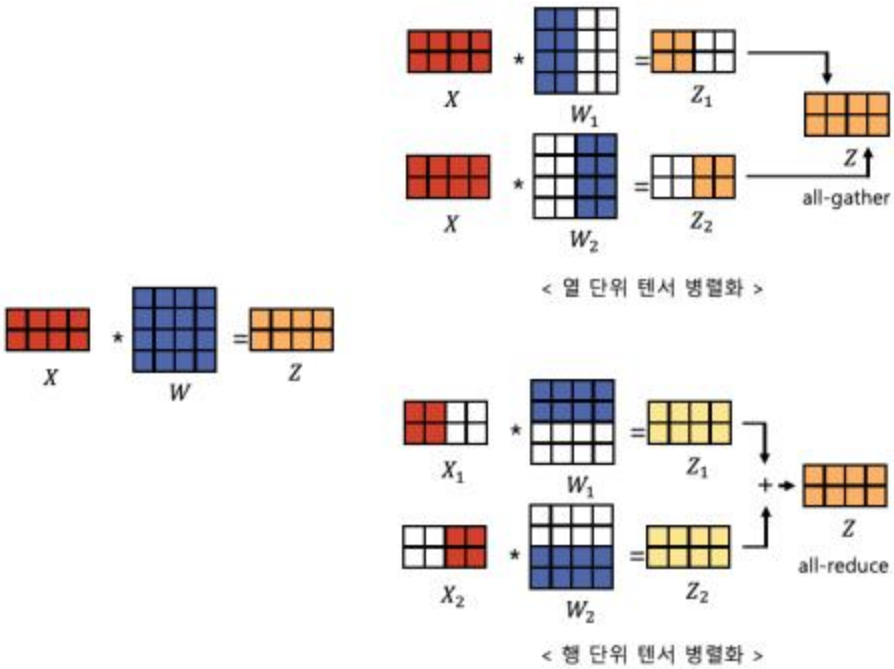
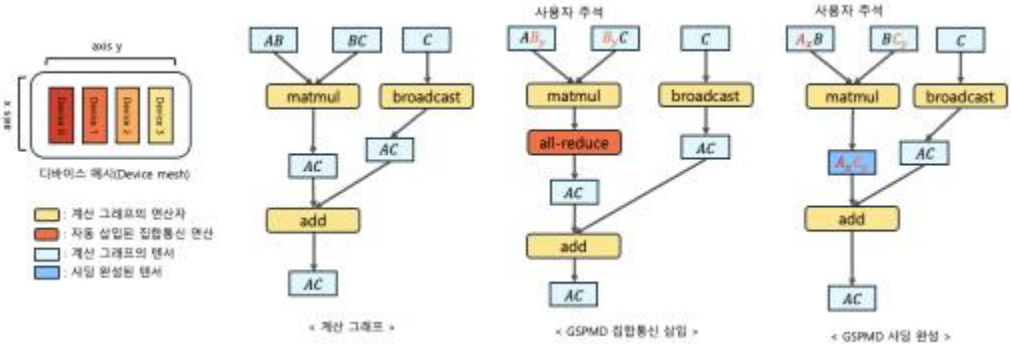
	서버 2대 사용 1 요청	서버 2대 사용 2 요청 오버랩	서버 2대 사용 4 요청 오버랩	서버 2대 사용 10 요청 오버랩
첫 토큰 생성 시간	113.27초	121.1초	120.55초	140초
이후 토큰 생성 시간 (중위값)	7초 (7초/token)	9.8초 (4.9초/token)	14.2초 (3.55초/token)	36초 (3.6초/token)
총 토큰 생성 시간 (30 토큰)	301.22초	394.11초	x	x

- 현재 OPT-125m, Bloomz-7b1, Mistral-7B 모델을 사용한 병렬화 기법 검증 완료

표 78 JAX 기반 파이프라인 병렬화 검증 모델

모델	OPT-125m	Bloomz-7b1	Mistral-7B
서버구성	단일서버	서버 2대	서버 2대
NPU 개수	1	2	2
파라미터 분리	선택	필수	필수
스테이지 분리	선택	필수	필수
스테이지 개수	2	4	4
오버랩 적용	x	o	x

- 오버랩 병렬화는 일정 수준(4요청)까지 효율적 파이프라인 활용을 통한 성능 향상 제공
- 본 실험을 통해 이기종 NPU 환경에서의 오버랩 한계점 및 최적 요청 수 탐색 기준 확인

수행기관	개발 목표	개발내용 및 범위
		- 텐서 병렬화 방식 및 XLA의 텐서병렬화 도구인 GSMPD, Shardy에 대한 분석 진행 • LLM 학습 및 추론 시 텐서 연산을 효율적으로 분할하여 병렬 처리하기 위한 텐서 병렬화 기술의 이해 및 최적화 필요 • XLA 기반의 텐서 병렬화 도구 GSPMD, Shardy는 그래프 수준에서 자동 샤딩(Sharding)과 집합통신 연산 삽입 가능 • 텐서 병렬화 방식은 열 단위 또는 행 단위로 텐서를 분할하여 병렬 실행할 수 있는데, 열 단위 분할은 결과 결합을 위해 All-Gather 연산이 필요하며 행 단위 분할은 부분합 집계를 위해 All-Reduce 연산이 필요함  그림 597 열 단위와 행 단위 텐서 병렬화 방식 예 • GSPMD/Shardy를 사용하여 사용자 주석 기반의 텐서 분할을 수행함 : 중간 텐서의 샤딩 자동 완성, 필요한 집합통신(Collective Communication) 연산 자동 삽입  그림 598 GSPD 기능 예 • 텐스토렌트 NPU 환경에서 JAX 및 GSPMD/Shardy의 지원 범위 확인을 위해 메가트론(Megatron) 방식 텐서 병렬화 성능 평가 진행 : Llama 3.2 1B에 대해 디바이스(칩) 수(1/2/4/8), TT-XLA 버전 (0.4.0, 0.5.0)을 변경하며 평가 • TT-XLA 0.4.0 기준 성능 측정 결과 : 디바이스 수 증가에 따라 추론 시간이 오히려 증가하는 경향이 보였는데, 이는 메모리 버퍼가 0번 디바이스에서만 생성되는 구조로 인해 데이터 이동 및 경합 오버헤드 발생이 원인으로 추정 • TT-XLA 0.5.0 기준 성능 측정 결과 : 0.4.0 대비 첫 토큰 이후 토큰 생성 시간이 약 8

수행기관	개발 목표	개발내용 및 범위

배 향상되었고, 디바이스 수가 4개 이상일 경우 프로그램 동작 정지(데드락 유사 현상) 발생함.

- TT-XLA 0.5.0 시험의 멀티 디바이스 환경에서 비정상적인 결과를 보인 것은 집합 통신 단계에서 문제가 발생한 것으로 추정

표 79 Llama 3.2 1B 모델 단일/멀티 디바이스 성능 측정

		단일	1x2	1x4	1x8
TT-XLA 0.4.0	첫 토큰 생성 시간	58.3s	66.22s	64.03s	67.21s
	이후 토큰 생성 시간	7.94s	9.98s	10.53s	11.28s
TT-XLA 0.5.0	첫 토큰 생성 시간	64.67s	72.04s	X	X
	이후 토큰 생성 시간	1.02s	2.62s	X	X

- 향후 TT-XLA의 멀티 디바이스 버퍼 관리 구조 개선 및 집합통신 안정화가 필요함

● PyTorch를 사용한 파이프라인 병렬화 기법 연구

- LLM 추론 최적화를 위해 개발된 vLLM 프레임워크와의 연동을 고려한 병렬화 기법 설계 필요
- vLLM은 PyTorch 기반으로 구현되어 있어, 이전 단계에서 사용한 JAX 기반 파이프라인 병렬화 기법과의 호환성에 한계 존재하므로, vLLM과 통합 가능한 파이토치 기반 파이프라인 병렬화 구조 설계 및 검증 연구 필요
- vLLM 프레임워크와 통합된 환경 구축을 목표로 파이토치 기반 파이프라인 병렬화 기법 설계 진행
- JAX 기반 방식과 파이토치 기반 방식 구조적 차이 분석
  - JAX: 모델을 저수준 함수 단위(JAXPR)로 변환하여 함수 분할 및 파이프라인 구성
  - 파이토치: 모델이 모듈화되어 있으며, 모든 레이어를 하나의 객체로 구성 가능

표 80 JAX, 파이토치 병렬화 비교

	JAX	파이토치
스테이지 분할	JAXPR 기반의 저수준 함수 분할	모듈 사용 방식
분할을 위한 그래프 탐색	전체 순회 필요	불필요
vLLM 호환	불가능	가능
허깅페이스 호환	허깅페이스 5부터 지원x, 일부 모델 사용 가능, 허깅페이스 전처리 함수 사용	대부분 모델 사용 가능, 전처리 함수 별도 구현 필요
장점	XLA 컴파일러 친화적	대부분 모델 사용 가능
단점	허깅페이스 업그레이드 불가, 그래프 탐색으로 인한 시간 소모	XLA 컴파일러가 메인이 아님, 전처리 함수 별도 구현

- 모듈 기반(Module-based) 파이프라인 분할 방식을 적용
  - 각 레이어 단위를 파이프라인 스테이지로 정의

<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>
• 스테이지 간 연결을 모듈 단위로 관리함으로써 병렬 처리 구성 단순화
- JAX 방식의 경우, 전체 계산 과정이 JAXPR로 변환되어 허깅페이스의 입력 전처리 함수까지 포함된 형태로 처리 가능했음
- 반면, 파이토치의 모듈 기반 접근 방식은 모델 모듈만 추출하여 분할하므로 허깅페이스에서 제공하는 입력 전처리 과정은 자동으로 포함되지 않아서, 허깅페이스 모델 입력을 그대로 사용할 수 없으며 모델에 맞는 별도 전처리 함수 구현이 필요함

● 분산 전략 탐색 알고리즘 연구
- 대규모 모델 병렬·분산 처리는 가속기 성능, 모델 구조, 분할 방식, 통신 성능 등 다양한 요소에 의해 처리 효율이 결정되는데 요소 간 상호작용이 복잡하므로 특정 환경에서 최적의 분산 전략을 찾는 것이 핵심 과제
- 고려해야 할 요소가 많고 조합의 폭이 매우 넓어 수작업 탐색은 불가능하며 자동화된 탐색 알고리즘 필요하여 메모리 사용량, 처리 성능, 탐색 공간의 규모를 종합적으로 고려한 분산 전략 탐색 알고리즘 연구 진행
- 메모리 사용량 관련 핵심 관찰
• 가속기 한 대가 처리할 수 있는 최대 작업량은 가속기 메모리 크기로 결정됨
• 기기 성능이 뛰어나더라도 메모리 한계를 초과하는 작업은 수행 불가
• 최적 전략 탐색 시 메모리 제약이 가장 우선적으로 고려되어야 함
- 가속기 메모리에 적재 가능한 작업량 판단을 위해 작업 그래프(컴퓨테이션 그래프) 크기 계산 방식 연구
• 그래프 크기 측정 방식에 따라 편차 존재 → 최적 방식 비교 평가
• 기기 성능이 뛰어나더라도 메모리 한계를 초과하는 작업은 수행 불가
• CPU 메모리 ↔ 가속기 메모리 간의 메모리 스왑 가능성까지 고려한 전체 그래프 크기 기준으로 분석

표 81 bloomz-7b1 2분할 시 스테이지0의 그래프 크기
<table>
<tr><th></th><th>전체 그래프 크기</th><th>한번에 실행되는 서브그래프 크기</th><th>동시에 사용되는 최대 메모리 사용량</th></tr>
<tr><td>bloomz-7b1 stage0</td><td>30.18GB</td><td>15.15GB</td><td>11.15GB</td></tr>
</table>

- 기존의 병렬화 전략 탐색 알고리즘들은 학습 프레임워크를 위해 개발되어 고려해야하는 요소가 많아 탐색 범위가 매우 넓음
- 본 연구는 추론 목적이므로 탐색 범위를 최소화할 수 있는 특성 존재
- 추론 환경 최적화를 위해 다음과 같은 탐색 요소 축소 기법 적용
• 기기 성능은 다양한 스펙을 세부적으로 고려하지 않고, 실제 작업 실행 시간을 단일 지표로 사용하여 통합
• 하드웨어 정보는 메모리 크기, 가속기 제조사 수준으로 단순화
- 탐색 범위를 불필요하게 넓히는 요인을 제거하고, 탈락 기준을 최소화하여 실제 탐색 가능한 전략의 수(유효 전략)를 최대한 유지하는 방향으로 알고리즘 설계
- 메모리 제약 기반 전략 분류, 런타임 기반 성능 평가 등을 결합하여 대규모 모델 추론을 위한 경량화된 자동 전략 탐색 알고리즘 구축
</td></tr>
<tr><td>고려
대학교
(위탁1)</td><td>GPU/NPU 통합
생성형 모델 병렬
추론 연구</td><td>
(1) GPU+NPU 자원기반 수 B급 생성형 언어 모델의 병렬 추론 기술 개발
● LLM 추론 서빙시 파이프라인 병렬화를 통한 텐서 병렬화의 한계 극복 필요
- LLM 추론시 다중 디바이스(GPU/NPU)를 활용한 병렬 처리는 필수
- 많은 상용 서버 환경은 대역폭이 제한된 PCIe 인터페이스를 사용하고 있어 기존의 텐서 병렬화 기법만으로는 성능 확장 불가
• 텐서 병렬화는 거대한 가중치 행렬과 연산을 여러 장치에 분할하여 처리함으로써 토
</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

큰당 지연 시간을 낮추는 방식으로, 연산 과정에서 동기화를 위해 장치 간 빈번한 통신(All-Reduce)을 필요로 함

- NVLink와 같은 고대역폭 인터커넥트가 지원되는 환경에서는 이러한 통신 오버헤드를 숨길 수 있지만, 많은 일반적인 데이터센터나 클라우드 인스턴스는 여전히 대역폭이 제한적인 PCIe 인터페이스에 의존
- 이러한 환경에서 텐서 병렬화의 막대한 통신량은 심각한 성능 병목을 야기하여, 결과적으로 온라인 대규모 언어 모델 서빙에 부적합하다는 한계를 가짐

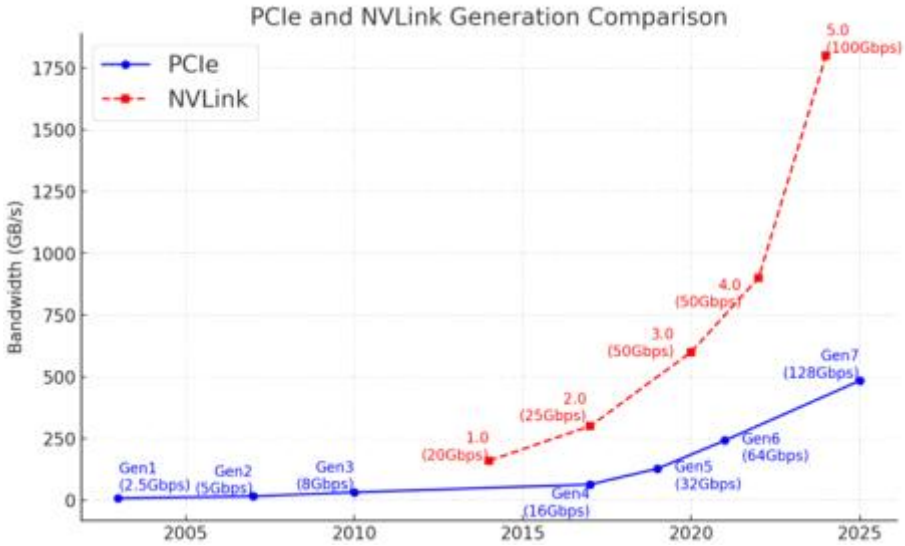


그림 599 PCIe 버전에 따른 대역폭 변화

- 파이프라인 병렬화는 모델을 깊이 방향으로 분할하여 장치 간 통신량을 획기적으로 줄일 수 있어 비용 효율적인 대안이 될 수 있음
- 온라인 서빙 환경은 요청 도착 시간이 불규칙하고, 입력 프롬프트의 길이가 다양하며, 프리필과 디코드 단계가 혼재되어 있어 워크로드의 불균형이 심화됨
  - 긴 입력(Long Prompt)과 짧은 입력(Short Prompt)이 섞여 들어올 경우, 긴 프롬프트의 연산이 끝날 때까지 짧은 프롬프트를 처리한 단계는 다음 데이터를 받지 못해 대기
  - 만약 디코드 요청이 거대한 프리필 요청 뒤에 배치될 경우, 앞선 프리필 연산이 완료될 때까지 디코드 단계의 GPU들은 멈춰 서게 됨(Stall)
- 이러한 워크로드 불균형으로 인해 발생하는 파이프라인 버블(Pipeline Bubble)은 파이프라인의 효율을 급격히 떨어뜨리는 주원인이 됨(처리량 저하, 지연시간 증가)
- 저대역폭 환경에서 파이프라인 병렬화를 효율적으로 활용할 수 있도록, 파이프라인 버블을 최소화 필요

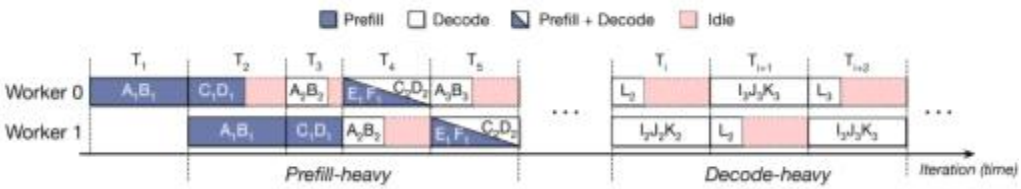


그림 600 LLM 추론 과정 중 발생하는 파이프라인 버블

● 청크 프리필(Chunked prefill) 기법을 적용하여 파이프라인 버블 감소

- 긴 입력 시퀀스를 단일 연산으로 처리하는 대신, 고정된 크기의 청크(Chunk)로 나누어 파이프라인에 주입
- 거대한 연산 덩어리가 여러 개의 조각으로 나뉘어 처리되므로, 그림 601의 (b)와 같이 파이프라인의 빈 공간에 청크들이 배치되어 유휴 시간을 줄이는 효과를 기대할 수 있음

수행기관	개발 목표	개발내용 및 범위
		- 하지만 여전히 고정된 기본 청크 크기(Default chunk size)가 일정 수준의 크기를 갖고 있어 파이프라인의 미세한 빈틈까지 완벽하게 메우지는 못하며, 이로 인해 파이프라인 버블이 여전히 잔존함

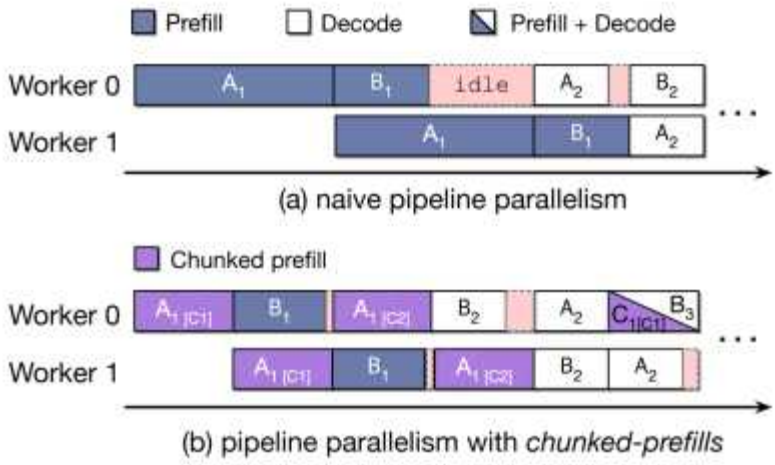
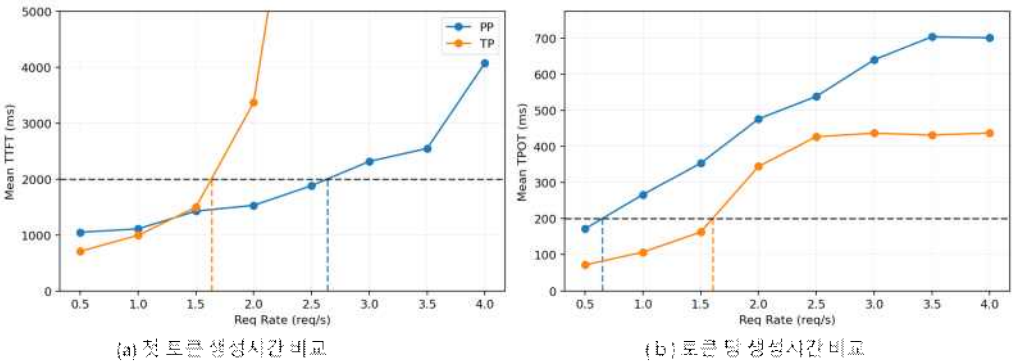


그림 601 청크 프리필을 통한 파이프라인 버블 감소

● 텐서 병렬화 대비 파이프라인 병렬화 기초 성능 분석

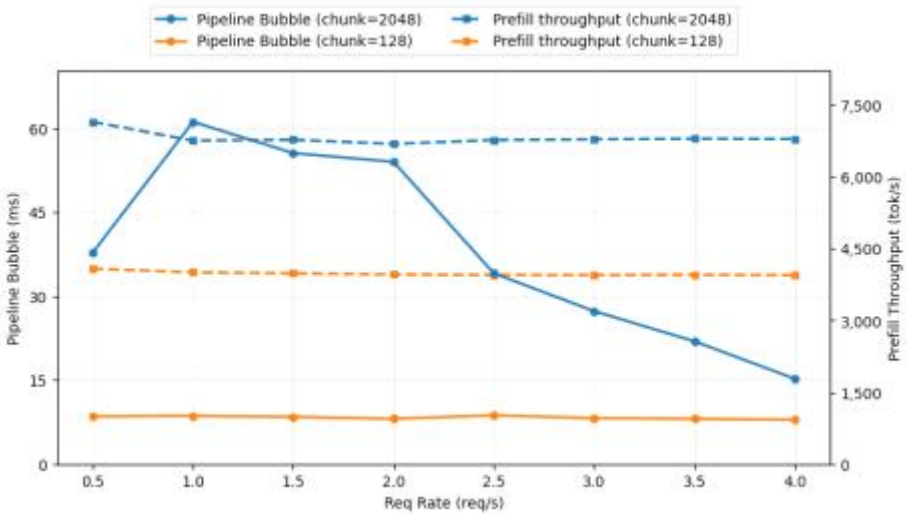
- 본 연구의 전제인 PCIe 환경에서의 파이프라인 병렬화의 유효성과 한계를 명확히 하기 위해 기존의 텐서 병렬화와 성능을 비교함
- 첫 토큰 생성 시간 지표에서는 파이프라인 병렬화가 텐서 병렬화보다 더 우수한(낮은) 지연 시간을 기록
  - 이는 PP가 점대점(Point-to-Point) 통신만을 수행하여 매 연산마다 동기화(All-Reduce)가 필요한 TP에 비해 통신 오버헤드가 적기 때문임
- 토큰 당 생성 시간 지표에서는 텐서 병렬화가 더 낮은 지연 시간을 보이며 우위를 점함
  - PP의 경우 청크 프리필(청크 사이즈 2048)을 적용했음에도 불구하고, 디코드 단계의 요청들이 프리필 청크가 처리되는 동안 대기해야 하는 간섭 현상이 여전히 존재하기 때문임
  - 이로 인해 디코드 연산의 지연 시간이 누적되어 토큰 당 생성 시간(Time Per Output Token) 성능이 저하되는 결과를 보임

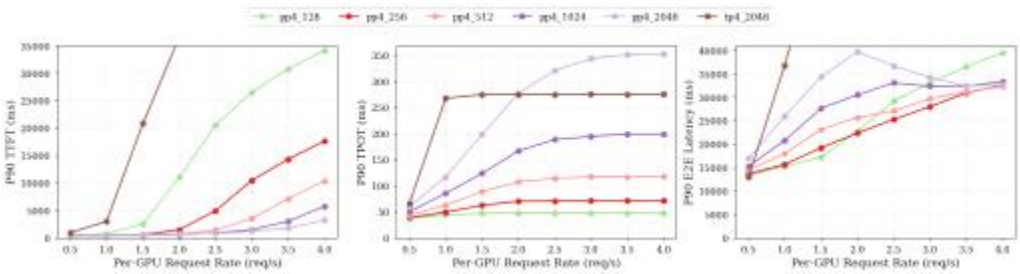


(a) 첫 토큰 생성시간 비교 (b) 토큰 당 생성시간 비교

그림 602 파이프라인 병렬화와 텐서 병렬화의 성능 비교

- 결과적으로 두 지표의 제약 조건을 모두 만족해야 하는 유효 처리량(goodput)은 현재 텐서 병렬화(TP)가 더 높게 측정됨.
  - 즉, PP가 첫 토큰 생성 시간에서는 이득을 보았으나, 불안정한 토큰 당 생성 시간 성능이 전체 시스템의 병목으로 작용하여 유효 처리량 하락을 유발한 것임.
- 이는 고정된 크기의 청크(2048)를 사용하는 것만으로는 파이프라인 버블과 단계 간 간섭 문제를 완전히 해결하기 어려우며, 토큰 당 생성 시간 성능을 개선하여 유효 처리량

수행기관	개발 목표	개발내용 및 범위
		을 극대화하기 위해서는 보다 정교한 최적화가 필요함을 시사함 ● 청크 크기에 따른 LLM 추론 성능 분석 및 실험 - 그림 603는 NVIDIA A6000 GPU 4개 환경에서 Qwen-2.5 32B 모델을 사용하여, 청크 크기와 요청 빈도(Request Rate)에 따른 파이프라인 버블 및 프리필 처리량(Prefill Throughput)의 변화를 측정한 결과임  그림 603 청크 크기에 따른 토큰 처리량 및 파이프라인 버블 - 요청 빈도가 낮은 구간에서는 청크 크기에 따른 파이프라인 버블의 격차가 뚜렷하게 확인되는데, 그래프의 주황색 실선(Chunk=128)은 버블이 약 10ms 수준으로 억제된 반면, 파란색 실선(Chunk=2048)은 60ms 이상의 거대한 버블이 발생함 • 이는 청크 크기가 클수록 한 단계에서 연산하는 시간이 길어져 다음 단계의 대기 시간이 늘어나는데, 부하가 적은 상황에서는 이 빈 시간을 채워줄 다른 대기 요청이 없기 때문임 • 즉, 큰 청크로 인해 발생한 공백이 그대로 시스템의 유휴 시간(Idle time)으로 이어져 비효율을 초래하는 것이므로, 이 구간에서는 청크 크기를 줄여 다음 단계로 작업을 빠르게 넘겨줌으로써 버블을 최소화하는 것이 중요함 - 요청 빈도가 높아질수록(High Request Rate) 큰 청크(2048)의 파이프라인 버블이 급격히 감소하여 작은 청크(128)와의 격차가 줄어드는 경향을 보임. 요청이 4.0 req/s에 도달했을 때 두 설정 간의 버블 차이는 미미해짐 • 이는 요청이 밀려드는 고부하 상황에서는 파이프라인의 빈 슬롯이 자연스럽게 다른 대기 요청들로 채워지기 때문에, 큰 청크를 사용하더라도 추가적인 버블이 크게 발생하지 않음을 의미함 - 처리량(Throughput) 측면에서는 점선으로 표시된 것과 같이, 요청 빈도와 관계없이 큰 청크(2048)가 작은 청크(128)보다 항상 월등히 높은 성능을 기록했는데, 이는 큰 청크는 GPU의 연산 병렬성을 극대화하고 메모리 접근 효율을 높여주기 때문임 - 요청 빈도가 낮을 때는 작은 청크를 사용하여 파이프라인 버블을 최소화하는 것이 유리하고, 반대로 요청 빈도가 높을 때는 버블의 영향이 상쇄되므로 큰 청크를 사용하여 처리량을 극대화하는 것이 효율적임 - 이는 고정된 청크 크기 대신, 실시간 부하량(Request Rate)에 따라 청크 크기를 동적으로 조절하는 전략이 전체 시스템의 유효 처리량(Goodput)을 최적화하는 핵심 열쇠임을 시사 ● 가변 청크 처리를 통한 파이프라인 최적화 기법의 효용성 검증 - 실험 환경 • NVIDIA A100 GPU 4개가 장착된 서버

수행기관	개발 목표	개발내용 및 범위
		• 서버 내 GPU들은 PCIe 인터페이스를 통해 연결 : 대역폭이 제한된 일반적인 상용 서버 환경 • 실험 모델로는 Qwen-2.5 14B 모델을 사용 • 워크로드 : Azure Conversation Trace 데이터셋을 기반으로 요청 패턴을 생성 - 실험 설정 • 비교 대상 : 텐서 병렬화 4-Way (TP4)와 파이프라인 병렬화 4-Stage (PP4) • 파이프라인 병렬화의 경우 청크 크기(Chunk Size)를 128, 256, 512, 1024, 2048 토큰으로 다양하게 변화시키며 성능 변화를 측정 • 평가 지표 : 지연 시간, 서비스 수준 목표 달성률, 그리고 유효 처리량(Goodput) - 실험 1 : 청크 크기에 따른 지연 시간 특성 분석 • 청크 크기는 첫 토큰 생성 시간(TTFT)와 토큰 당 생성 시간 (TPOT)에 상반된 영향을 미치는 트레이드오프 관계임이 확인됨 • 청크 크기가 작아질수록(예: 128, 256) TTFT는 증가하는 경향을 보였는데, 이는 작은 청크로 인해 작업의 분할 횟수가 늘어나면서 스케줄링 오버헤드가 발생하고, 메모리 대역폭 효율이 저하되어 초기 응답 속도가 느려지기 때문임 • 청크 크기가 작아질수록 TPOT는 감소(성능 향상)하는 결과를 보였는데, 이는 거대한 프리필 연산이 잘게 쪼개지면서 디코드 단계의 연산이 프리필 작업에 의해 장시간 대기(Stall)하는 현상이 완화되었기 때문임 • 반대로 청크 크기가 커질수록(예: 2048) 처리량 효율로 인해 TTFT는 개선되지만, 디코드 간섭 심화로 TPOT는 악화되는 양상을 보임  그림 604 요청 빈도와 청크 크기에 따른 지연 시간 비교 - 실험 2 : 서비스 수준 목표(SLO) 달성률 및 유효 처리량(Goodput) 평가 • 요청 빈도가 낮은 구간에서는 모든 청크 크기가 달성률 100%를 기록하여 큰 차이가 없어 보이지만 실제 전체 응답 시간(End To End Latency) 측면에서는 128 토큰과 같은 작은 청크 크기가 훨씬 더 빠른 응답 속도를 제공함 • 즉, 부하가 낮을 때는 처리량 한계치보다는 지연 시간이 중요한데, 이때는 작은 청크가 유리하지만, 2048과 같은 큰 청크는 부하가 낮음에도 불필요하게 높은 지연 시간을 유발하여 비효율적임

수행기관	개발 목표	개발내용 및 범위

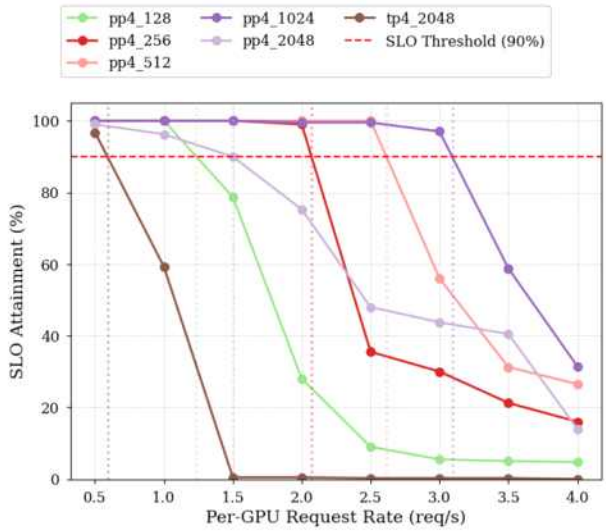


그림 605 요청 빈도 증가에 따른 청크 크기별 서비스 수준 목표(SLO) 달성률 추이

- 그림 606는 P90 지연 시간이 서비스 수준 목표 기준을 만족하는 최대 지점에서 측정된 유효 처리량 (Goodput)을 보여주는데, 기존의 기본 청크 크기인 2048 토큰이 항상 최적의 성능을 보장하지 않음을 확인함
- 유효 처리량 관점에서는 청크 크기 1024 토큰 설정이 기본값(2048)이나 더 작은 크기(128)보다 더 높은 최대 처리량을 기록함. 이는 1024 크기가 첫 토큰 생성 시간의 오버헤드와 토큰 당 생성 시간의 간섭 사이에서 가장 적절한 균형을 유지했기 때문임
- 결론적으로, 단일 고정 청크(예: 2048 또는 1024)만으로는 모든 부하 상황에서 최적의 성능을 낼 수 없음을 확인함
- 따라서 시스템의 부하 상태와 서비스 수준 목표 만족 여부에 따라, 낮은 부하에서는 작은 청크를 선택하여 응답성을 극대화하고, 높은 부하에서는 1024 등 최적의 청크를 선택하여 유효 처리량을 방어하는 동적 전략이 필수적임을 실험적으로 입증함

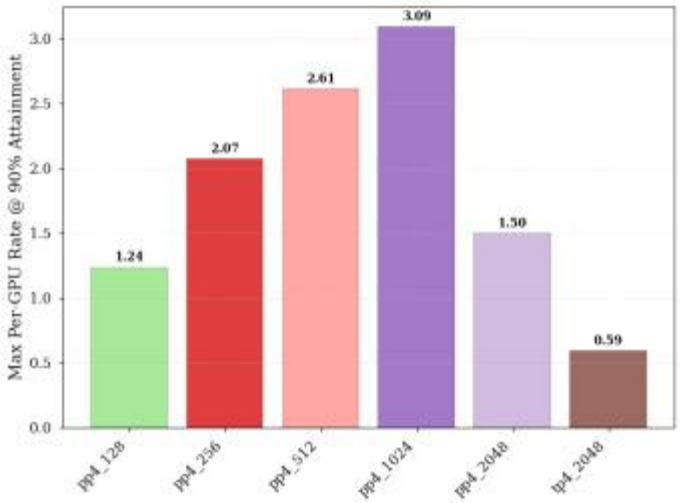


그림 606 청크 크기별 최대 유효 처리량(Goodput)

(2) 텐스토렌트 NPU 내부 구조의 최적 활용 지원 고성능 커널 라이브러리 개발

- 현재 텐스토렌트 NPU 기반 LLM 추론에서 처리량 향상의 한계 존재
  - 텐스토렌트 사는 대규모 LLM 추론에 최적화된 데이터플로우 아키텍처 기반의 NPU를 제공하고 있으나, 현재 지원되는 병렬화 방식은 텐서 병렬화와 데이터 병렬화에 한정되어 있음
  - 하지만, 텐서 병렬화로 실행할 때 디바이스 수 증가에 따라 처리량이 이상적 처리량(선

수행기관	개발 목표	개발내용 및 범위

형 스케일링)에 도달하지 못함

- 디바이스 수를 늘려도 처리량 증가율이 점진적으로 감소하는 경향이 나타나는데, 이는 텐서 병렬 실행 구조에서의 병렬화 비효율을 의미함

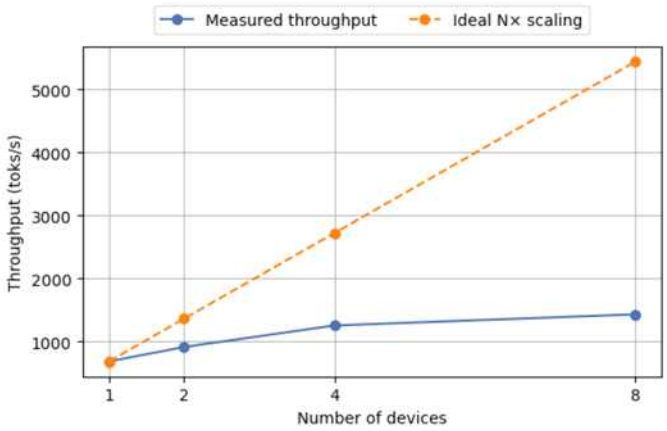


그림 607 Llama 3.1 8B 모델의 텐서 병렬 실행시 디바이스 수 증가에 따른 평균 처리량 변화

- 기존 TP 기반 실행 모델을 정밀하게 프로파일링하여 병목 지점을 분석하고, 이를 개선하기 위한 파이프라인 병렬화 기법을 적용함으로써 처리량을 향상하는 기술을 개발

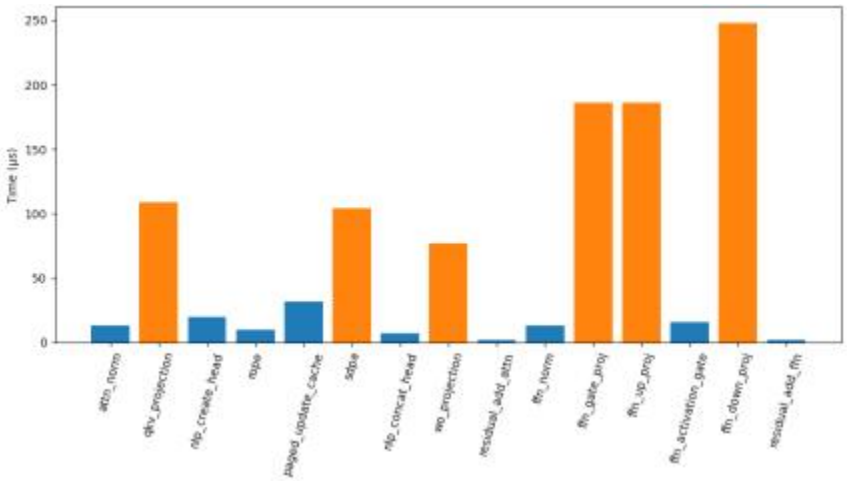
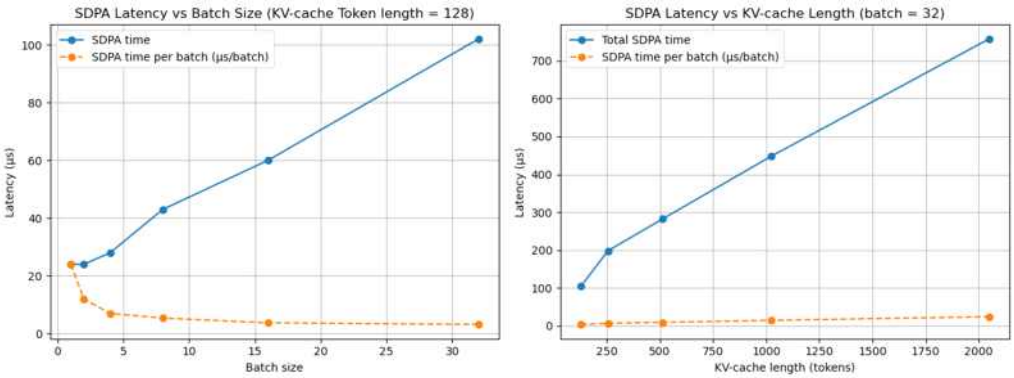
● 단일 텐스토렌트 NPU에서 LLM 디코드 단계 프로파일링

- 텐스토렌트 Trace 모드 실행시간을 기준으로 성능 측정
  - 텐스토렌트 소프트웨어는 초기 실행 시 발생하는 커널 생성 및 데이터 준비 오버헤드를 줄이기 위해 Trace 모드를 제공함
  - Trace 모드는 dummy 데이터를 이용해 레이어를 한 차례 실행하고, 이 실행 경로를 캡처하여 디바이스 L1에 저장한 뒤 동일 형태의 요청에서 이를 재사용함
  - Trace 모드 실행 지연은 컴파일 모드 대비 약 230배 감소함

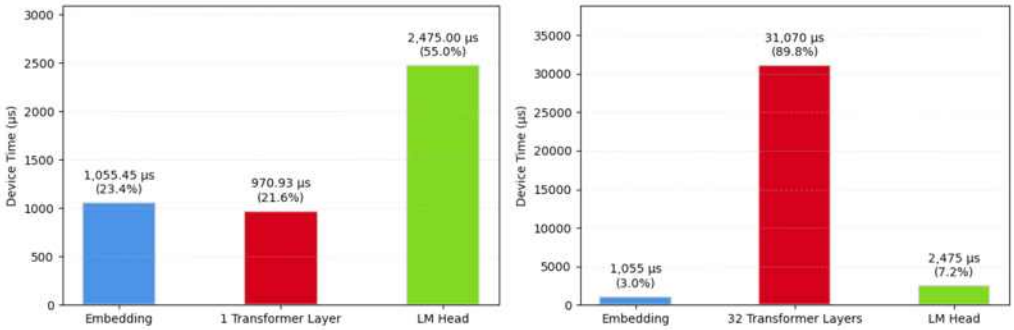
표 82 각 실행 단계별 소요 시간

실행 모드	소요 시간 (ms)	비고
Compile	8792.4	최초 1회 컴파일
Trace Capture	53.2	실행 경로 캡처
Trace Execution	37.9	반복 실행 평균

- 단일 디바이스에서 디코드 스텝의 각 연산 별 소요 시간 측정
  - Llama 3.1-8B 모델을 대상으로 32 batches, 128 tokens KV-cache 조건에서 단일 NPU 디코드 레이어의 연산별 소요 시간을 측정함
  - 그림 608에서 각 연산은 실행 순서대로 x축에 배치했으며, 50μs 이상 소요된 연산은 주황색 그래프로 강조함
  - QKV projection, SDPA(Scaled Dot Product Attention), FFN(Feed-Forward Network)과 같은 주요 행렬 연산이 상대적으로 긴 시간을 차지했으며, 나머지 연산들은 대부분 수 μs 수준으로 나타남

수행기관	개발 목표	개발내용 및 범위
		 그림 608 단일 디바이스에서 디코드 스텝의 각 연산 별 소요 시간 - 동시에 처리하는 배치 크기와 KV-cache의 토큰 길이에 따른 SDPA 연산 실행 시간 비교 • 배치 크기를 1, 2, 4, 8, 16, 32로 증가시킬 때 SDPA 실행시간은 배치 크기에 비례해 선형적으로 증가함. • 배치당 평균 지연 시간은 배치 크기가 커질수록 감소하는 경향을 보였는데, 이는 SDPA가 토큰 단위 순차 실행 구조임에도 커널 호출·스케줄링의 고정 오버헤드가 여러 시퀀스에 분산되어 전체 효율이 향상되기 때문임 • KV-cache 길이가 128, 256, 512, 1K, 2K로 증가함에 따라 SDPA 실행시간도 이에 비례해 선형적으로 증가했는데, 이는 SDPA는 현재 토큰을 모든 이전 토큰과 비교하는 구조이므로 KV-cache 토큰 수가 증가할수록 연산량이 증가하는 특성이 반영된 결과임  그림 609 배치 크기와 KV-cache의 토큰 길이 변화에 따른 SDPA 연산 실행 시간 - 단일 디바이스에서 Llama 3.1-8B 모델이 한 토큰을 생성하는 데 걸리는 시간을 Embedding, Transformer Layer, LM Head의 3단계로 나누어 측정 • Embedding은 1055.45µs, 1개의 Transformer Layer는 970.93µs, LM Head는 2475.00µs가 소요됨 • 총 32개의 Transformer Layer를 실행하는 데 걸리는 시간은 31,070µs로 나타남

수행기관	개발 목표	개발내용 및 범위



(가) 1 Transformer Layer (나) 32 Transformer Layer

그림 610 Llama 3.1-8B 모델이 한 토큰을 생성하는 데 걸리는 시간 분석

● 텐스토렌트 다중 NPU 환경에서 텐서 병렬화를 통한 LLM 디코드 단계 프로파일링

- 디바이스 수 증가에 비례해 처리량이 선형적으로 증가하지 못하는 원인을 확인하기 위해 TP 실행시 CCL 호출 횟수와 소요 시간 측정
  - Llama 3.1-8B 디코드 한 스텝에서 트랜스포머 레이어당 4회의 CCL 호출이 발생하여 총 128회이며, LM Head 직전 1회의 추가 호출을 포함해 전체 호출 수는 129회임

표 83 디코드 1스텝 기준 트랜스포머 레이어 내부 CCL 연산 구조 및 총 호출 수

구분	내용	발생 위치 / 설명	호출 수 (횟수)
레이어 구성	모델 레이어 수	Llama 3.1 8B 기준	32
레이어 당 CCL 패턴	1) all_gather	PreAttention Norm	4 / layer
	2) reduce_scatter	Attention	
	3) all_gather	PreFFN Norm	
	4) reduce_scatter	FFN	
레이어 구간 합계	총 CLL 호출 수	32 layer x 4회	128
추가 호출	레이어 외 CCL	preLM-Head	1
총합	디코드 1스텝 기준 전체 CCL 호출 수	-	129

- 그림 611은 TP 수준을 2, 4, 8로 증가시키면 디코드 한 스텝에서 CCL이 차지하는 비율이 약 13.8%, 27.4%, 39.0%로 증가함을 보여줌 (단, TP=8은 AllGatherMatmul fused 커널의 통신 시간 포함)
- 디바이스 수가 증가하면 단일 트랜스포머 레이어의 연산 시간은 감소하지만, CCL 통신 비중 증가로 인해 전체 디코드 지연은 디바이스 수에 비례해 선형적으로 감소하지 않음

수행기관	개발 목표	개발내용 및 범위

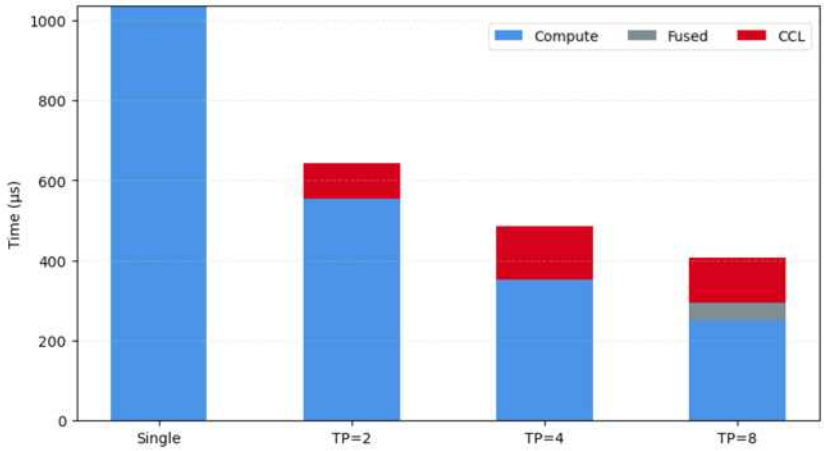


그림 611 TP에서 디바이스 수에 따른 CCL 통신과 연산 소요 시간

- 디코드 한 스텝에는 총 129회의 CCL 호출이 포함되며, 토큰 단위 순차 실행 구조에서 통신 오버헤드가 누적되어 확장성이 제한되므로, 텐서 병렬화 기법의 구조적 한계를 보완하기 위해 CCL 호출 수를 줄이는 파이프라인 병렬화 모델을 제시함

● PCIe 통신을 통한 텐스토렌트 NPU 활용 파이프라인 병렬화 구현
- 모델을 구성하는 트랜스포머 레이어를 디바이스 수에 맞추어 각 파이프라인 단계에 균등하게 분배
- 입력 토큰을 처리하는 Embedding은 첫 파이프라인 단계에서, 출력 토큰을 생성하는 LM Head는 마지막 파이프라인 단계에서 실행하게 함
- PCIe 통신을 이용하여 각 파이프라인 단계간 은닉 상태(hidden state) 전달
- 디바이스 유휴 시간을 줄이기 위해 입력 묶음을 미니배치(mini-batch)로 정의
- 디바이스가 동시에 처리하는 단위는 마이크로배치(micro-batch)로 구분했고, 마이크로배치 크기는 타일 구조에 맞춰 32로 고정함

● 파이프라인 병렬화의 성능 특성 분석 및 텐서 병렬화 대비 성능 비교
- 2개의 디바이스를 이용한 파이프라인 병렬화(PP=2)에서 미니배치 크기에 따른 처리량 변화 분석
  - 미니배치가 384일 때 처리량이 1002 toks/s으로 가장 높았으며, 그보다 커지면 감소함
  - 미니배치 증가로 디바이스 활용도는 높아지지만, KV-cache가 차지하는 메모리 규모가 커지면서 메모리 접근 지연과 관리 오버헤드가 증가하기 때문에 감소함

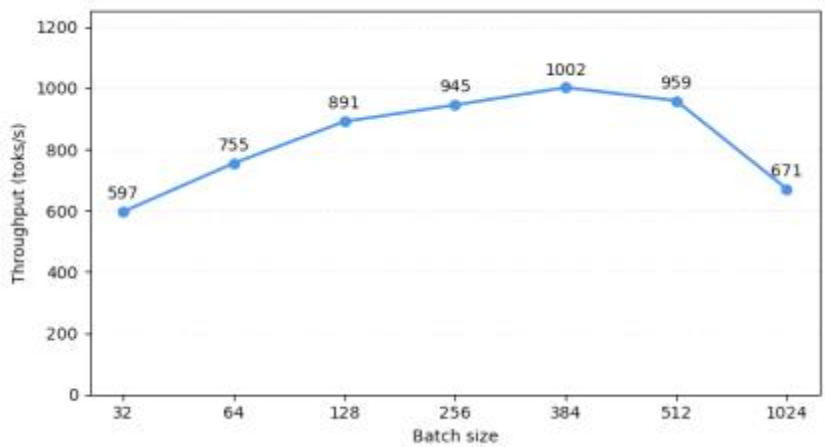
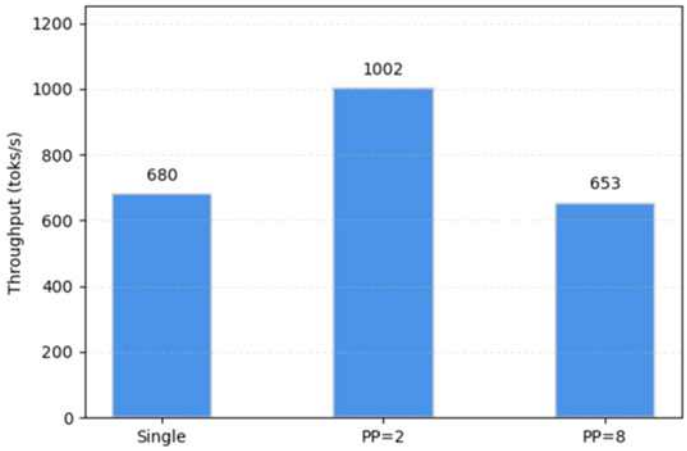
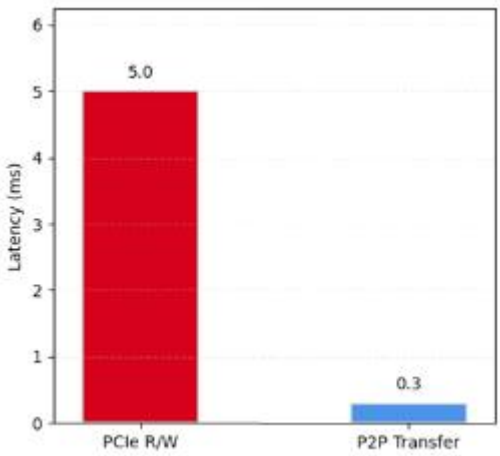
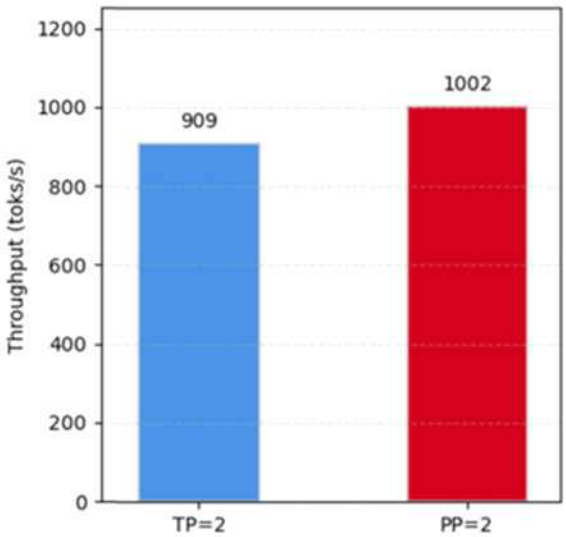
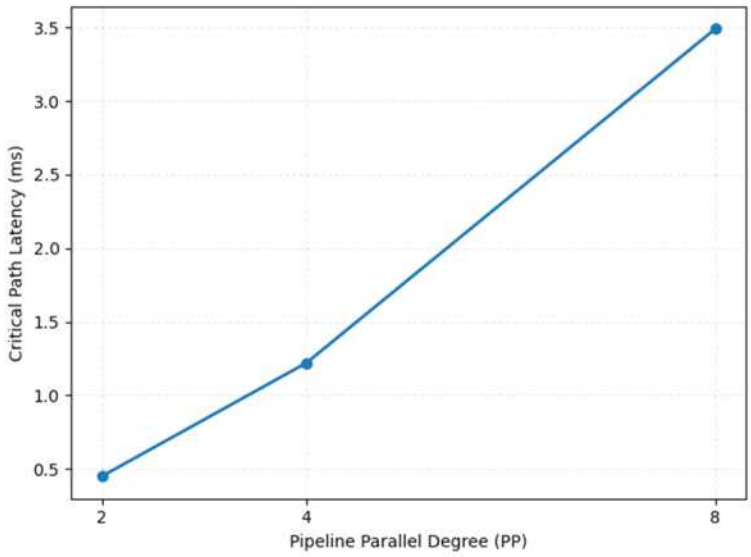


그림 612 PP=2에서 미니배치 크기에 따른 처리량 변화

수행기관	개발 목표	개발내용 및 범위
		- 파이프라인 병렬화에서 디바이스 수에 따른 처리량 변화 분석 • 디바이스 2개 구성에서는 처리량이 1002 toks/s로 증가했으나, 디바이스 8개 구성에서는 오히려 653 toks/s로 감소(그림 613)  그림 613 파이프라인 병렬화에서 디바이스 수에 따른 처리량 변화 • 8개 디바이스에서 처리량이 감소하는 이유를 규명하기 위해 PP의 hidden state 전달 경로를 분석함 • PP는 레이어를 비동기 처리하므로 디바이스 간 hidden state가 PCIe를 경유해 호스트 메모리에 저장된 뒤 다음 디바이스로 전달되는데, 이러한 PCIe 기반 데이터 전달이 디바이스 수 증가 시 병목으로 작용하여 전체 처리량을 제한함 • PCIe와 P2P 통신의 소요 시간을 비교하면, PP에서 hidden state 전달을 위해 사용하는 PCIe 통신은 약 5ms 수준의 지연이 발생했고, TP에서 사용하는 Ethernet 기반 P2P 통신은 약 0.3ms 지연을 보임(그림 614) • 이러한 통신 지연 차이는 디바이스 수가 증가할수록 파이프라인 전체 지연을 증가시키는 요인으로 작용함  그림 614 PCIe와 P2P 통신 소요 시간 - 2개의 디바이스를 이용하는 텐서 병렬화와 파이프라인 병렬화의 처리량 비교 • 디바이스 2개 구성에서 파이프라인 병렬화(1002 toks/s)는 텐서 병렬화(909 toks/s) 대비 약 10% 높은 처리량을 보임

수행기관	개발 목표	개발내용 및 범위
		 그림 615 디바이스 2개를 이용하는 경우의 병렬화 방법에 따른 처리량 비교 ● Ethernet-Core 사이의 Point-to-Point(P2P) 통신 실험 - 파이프라인 단계 수가 많아짐에 따라 PCIe 지연 시간 증가로 처리량이 디바이스 수 증가에 비례하여 증가하지 않음 • 디바이스 수가 증가하면 트랜스포머 레이어 연산 시간은 감소하지만, 레이어 경계에서 hidden state를 전달하기 위한 PCIe 통신 지연이 상대적으로 커지며, Device→Host DRAM→Device 경로가 중첩되면서 경합에 따른 추가 지연이 발생함 • bfloat16 텐서 전송 실험 결과, PCIe 통신에 동시에 참여하는 디바이스 수가 늘어날수록 단일 전송 시간이 증가하여 PCIe 경로가 병렬화되지 못함을 확인함 • PP=2 실행 시 입력 로딩, hidden state 전달, 결과 반환 모두 PCIe를 경유하기 때문에 실행 시점에 따라 통신 지연의 변동폭이 크게 나타남  그림 616 파이프라인 병렬화 수준에 따른 PCIe 통신 지연 시간 변화

수행기관	개발 목표	개발내용 및 범위

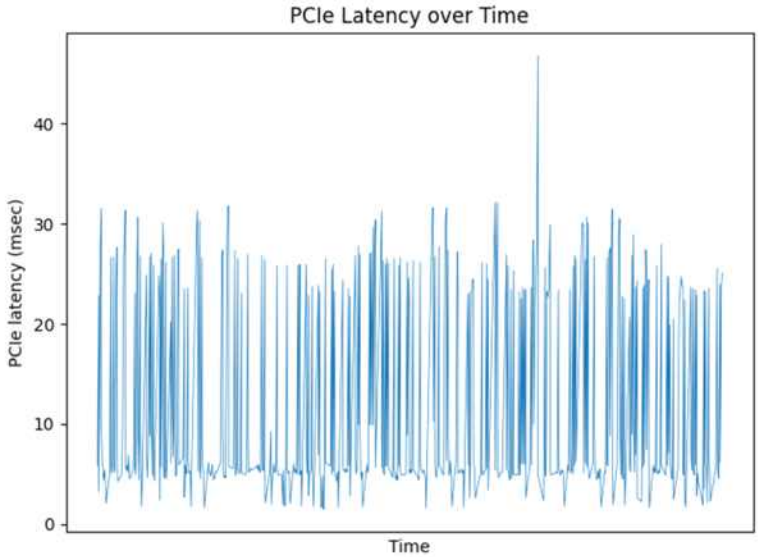


그림 617 파이프라인 병렬화(PP=2)에서 은닉 상태 전송시간

- PP 스테이지 수가 증가할수록 연산 시간은 감소하지만 PCIe 통신 시간이 점차 증가하며, PP=8에서는 디바이스 당 총 소요 시간이 크게 증가함
  - 파이프라인 병렬화 수준이 과도하게 늘어나면 PCIe 통신이 주요 병목으로 작용함(즉, PCIe 기반 파이프라인 병렬화가 PCIe 대역폭에 의해 제한됨을 의미)

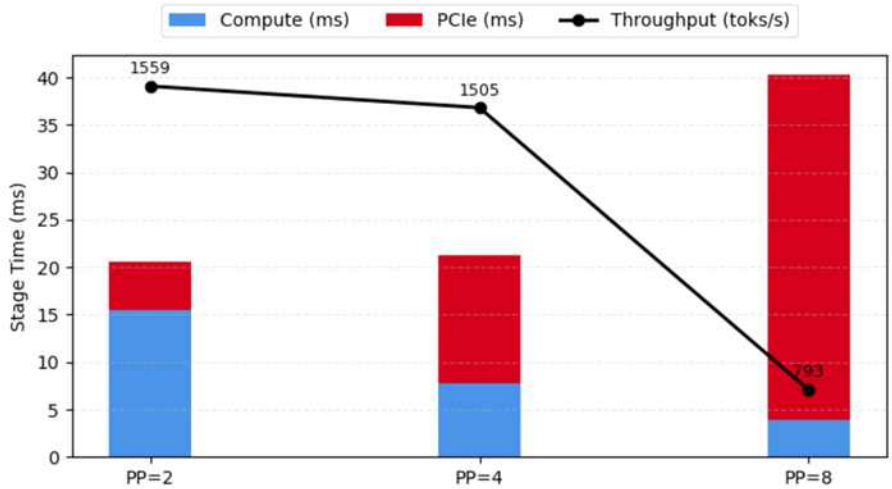


그림 618 파이프라인 병렬화 수준에 따른 각 디바이스의 연산 및 통신 소요 시간

- 이를 해소하기 위해 Tenstorrent NPU의 Ethernet-core를 활용한 hidden state 전송 방식을 고려함
- TT-NN은 point-to-point 통신 API를 제공하지만, 이는 디바이스 간 동기 실행에서만 사용 가능하고, 비동기 실행 구조인 PP에서는 기존 API 사용이 불가능함
- Ethernet-core 간 비동기 전송을 지원하는 통신 커널을 구현하고, 이를 TT-NN API 수준에서 사용할 수 있도록 래핑 (표 84)
  - ttnn.p2p_transfer(input_tensor, output_tensor)는 디바이스 A의 DRAM→Ethernet-core→디바이스 B의 Ethernet-core→DRAM으로 비동기 데이터 전송을 수행함
  - 두 텐서가 서로 다른 디바이스에 할당되어 있어야 하며, 향후 텐식스-코어(Tensix-core)의 SRAM으로 전송 범위를 확장할 수 있음

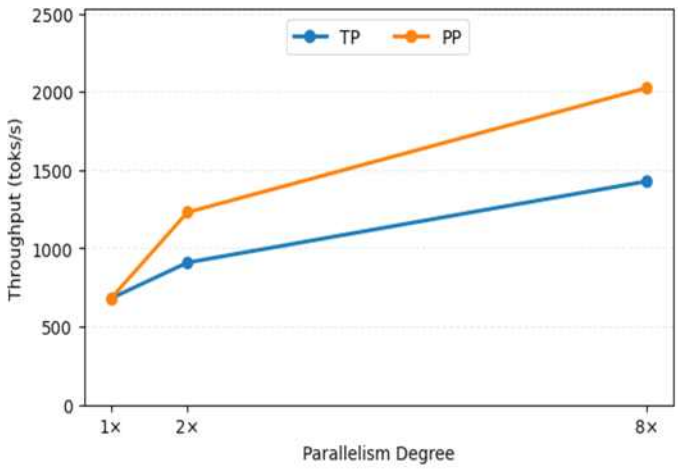
<table>
<tr><th>수행기관</th><th>개발 목표</th><th>개발내용 및 범위</th></tr>
<tr><td></td><td></td><td>
표 84 TT-NN 비동기 P2P 통신 API
<table>
<tr><th>항목</th><th>내용</th></tr>
<tr><td>API</td><td>ttnn.p2p_transfer(input_tensor, output_tensor)</td></tr>
<tr><td>역할</td><td>디바이스 A의 input_tensor 데이터를 디바이스 B의 output_tensor로 전송</td></tr>
<tr><td>조건</td><td>두 텐서가 서로 다른 디바이스에 할당되어 있어야 함</td></tr>
</table>
- 벤치마크 결과 P2P 전송 기능은 정상 동작했으나, TT-NN이 디바이스 간 비동기 실행 연산 등록 기능을 제공하지 않아 vLLM 엔진에 통합할 수 없는 구조적 제약이 존재함
- 본 연구에서는 P2P 통신 적용 효과를 확인하기 위해, vLLM 엔진에서 PCIe 전송 단계를 제거하고 측정된 P2P 지연 시간을 인위적으로 삽입하는 방식으로 실험을 구성함
- 실제 통합 없이도 파이프라인 구조를 유지하기 위해 더미 hidden state를 전달하도록 구성하고, 이를 통해 P2P 적용 시 예상 처리량 변화를 정량적으로 분석함
- 단일 디바이스 처리량 (680 toks/s) 기준, 2개 디바이스 구성에서 TP는 약 34% 증가(909 toks/s)에 그쳤지만 PP는 약 81% 증가(1230 toks/s)로 이상적 처리량 (1360 toks/s)의 약 90% 수준을 달성함
- 8개 디바이스에서는 TP가 단일 대비 약 110% 증가했지만, PP는 약 198% 증가 (2024 toks/s)하여 동일 구성에서 TP 대비 약 42% 높은 처리량을 보임
- 이는 디바이스 수가 증가할수록 TP보다 PP의 처리량 향상 가능성이 더욱 크다는 점을 시사함


그림 619 병렬화 수준에 증가 따른 병렬화 방법별 처리량 비교
</td></tr>
<tr><td>포항공과대학교
(위탁2)</td><td>이종 계산 자원 환경에서 긴 문맥 추론으로 인한 문제를 극복하는 동적 KV 캐시 관리 기술 연구</td><td>
(1) 긴 문맥 추론 지원의 문제점 및 KV 캐시 관리의 필요성
● LLM의 자기회귀 디코딩(autoregressive decoding)은 매 토큰마다 KV 캐시가 누적되며, 문맥 길이가 길어질수록 KV 캐시가 선형적으로 증가
● KV 캐시 증가량이 모델 파라미터보다 훨씬 크기 때문에 LLM 추론에서 가장 큰 메모리 소비 요소임
</td></tr>
</table>

수행기관	개발 목표	개발내용 및 범위

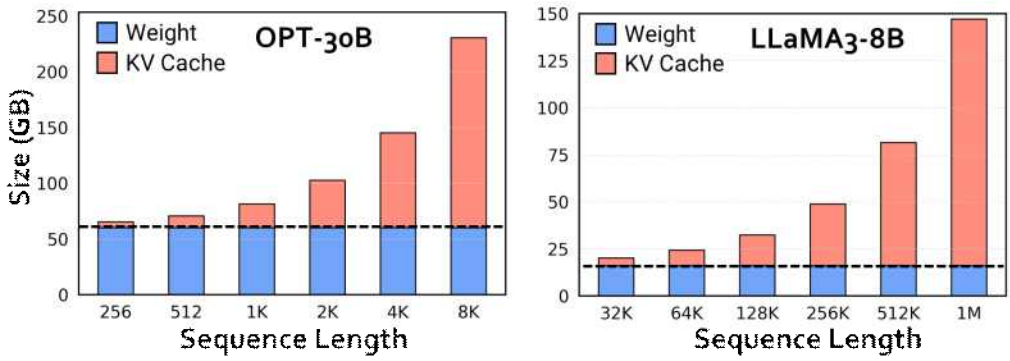


그림 620 LLM 추론에서 문맥(시퀀스) 길이에 따른 KV 캐시 크기 변화

- LLM의 활용이 폭발적으로 증가하는 상황에서 긴 문맥 지원이 모델의 품질과 응용 범위를 결정짓는 핵심 요소로 부상하고 있음
  - GPT-2가 약 1천 개 토큰 수준의 입력만 처리할 수 있었던 반면, 이후 등장한 Claude 계열 모델은 200K 이상 토큰 규모의 초장문 문맥을 지원하며, 최신 Gemini 계열 모델은 최대 백만 개 토큰까지 지원
- 긴 문맥 추론 과정에서 KV 캐시 증가로 인해 가속기 메모리 부족이 자주 발생하므로, 효율적인 KV 캐시 관리가 필요

(2) 선행 가지치기 기반 KV 전송량 감소 기법의 적용 및 성능 영향 분석

- 단순 오프로딩 스케줄링만으로는 긴 문맥 추론을 충분히 확장하는 데 구조적 한계가 존재하며, 통신 병목 완화를 위한 KV 캐시 가지치기 기법이 필수적임
  - LLM은 모델 가중치만으로도 이미 수 GB 이상을 점유하기 때문에, 문맥 길이가 일정 수준을 넘어서면 모든 KV 캐시를 GPU 메모리에 유지하는 것은 구조적으로 불가능
  - KV 캐시는 CPU 메모리로 오프로딩될 수밖에 없으며, 전송 비용이 성능 병목으로 작용하여 연산-통신 겹치기 기법 사용
    - GPU 메모리에 저장할 수 없는 KV 캐시를 CPU 메모리나 SSD로 오프로딩
    - 오프로딩된 KV 캐시를 GPU로 가져올 때 연산-통신 겹치기(prefetch overlap)

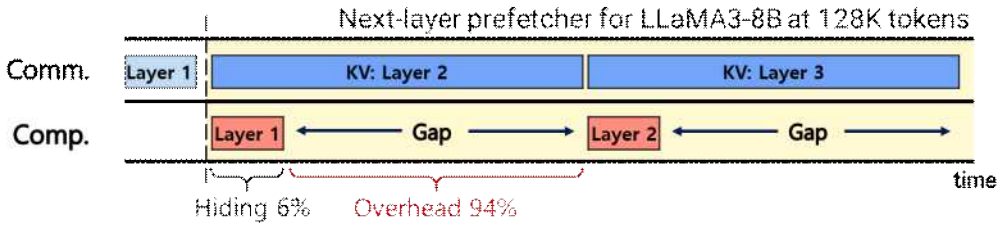


그림 621 다음 레이어 KV 캐시 프리패칭

  - 메모리 제약만을 고려할 경우, 오프로딩 전략을 적용함으로써 GPU 상에서 유지 가능한 KV 캐시 용량을 일정 수준까지 확장하는 것이 가능하나 프리패치 버퍼를 고려하면 처리 가능한 토큰의 한계가 더 줄어들게 됨
    - 오프로딩을 적용하지 않은 경우, GPU 여유 메모리가 8 GB일 때 처리 가능한 문맥 길이는 약 64K 수준에 그침
    - 반면 오프로딩 거리를 1로 설정하면 전체 레이어 중 절반의 KV 캐시만 GPU에 유지되므로, 메모리 기준으로는 최대 128K 토큰까지 확장 가능
    - 여기서 오프로딩 거리는 현재 처리 중인 레이어를 기준으로 GPU에 상주시켜 유지하는 인접 레이어들의 KV 캐시 범위를 나타내는 파라미터이며, 거리 1 설정은 연속된 1개 레이어의 KV 캐시만 GPU에 유지하고 나머지는 CPU 메모리로 오프로딩하는 구성을 의미
    - Llama3-8B(32 레이어) 기준 프리패치 버퍼는 전체 KV 크기의 추가 약 3.1% 필요

수행기관	개발 목표	개발내용 및 범위

표 85 메모리 내 최대 처리 가능 토큰(Llama3-8B)

GPU 여유 메모리	1 GB	2 GB	4 GB	8 GB
오프로딩 미적용	8K	16K	32K	64K
오프로딩 거리=1 (프리패치 버퍼 고려 없음)	16K	32K	64K	128K
오프로딩 거리=1 (프리패치 버퍼 고려)	15K	30K	60K	120K

- 서비스 수준 목표(Service Level Objective, 이하 SLO)를 고려했을 때는 최대 처리 가능한 토큰 개수가 상당히 감소
  - GPU 여유 메모리가 4 GB일 경우, 메모리 관점에서는 최대 64K 토큰 처리가 가능하지만, 실제 서비스 수준 목표 1.5× 기준에서는 약 45K, 서비스 수준 목표 2× 기준에서는 약 49K 수준에서 한계가 나타남
  - GPU 여유 메모리가 8 GB로 증가하더라도 이론적 처리 한계는 128K까지 확장되지만, 서비스 수준 목표 준수 기준에서는 각각 88K 및 90K 수준으로 제한됨
  - 이러한 결과는 메모리 확장으로 증가할 수 있는 처리량의 상당 부분이 CPU-GPU 간 통신 병목에 의해 상쇄되고 있음을 보여줌

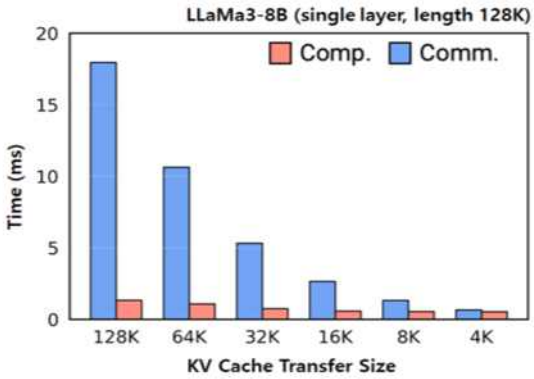
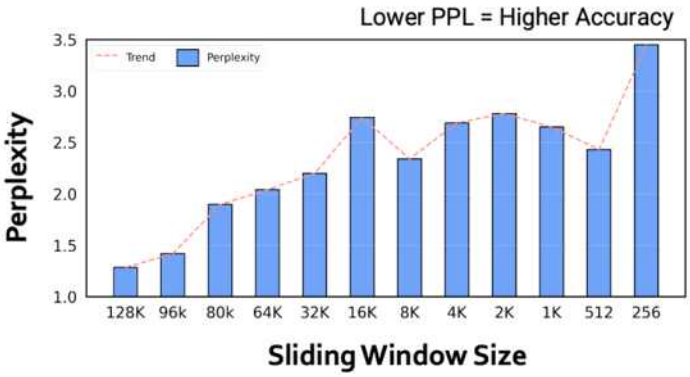
표 86 서비스 수준 목표 내 최대 처리 가능 토큰(Llama3-8B)

GPU 여유 메모리	1 GB	2 GB	4 GB	8 GB
오프로딩 거리=1 (프리패치 버퍼 고려 없음)	16K	32K	64K	128K
서비스 수준 목표 1.5x	한계 없음	30K	45K	88K
서비스 수준 목표 2x	한계 없음	한계 없음	49K	90K

- 메모리 제약만을 고려하여 산출된 이론적 최대 처리 문맥 길이와 실제 서비스 수준 목표를 만족하면서 처리 가능한 문맥 길이가 큰 차이를 보임

● 최신 가지치기 알고리즘 분석
- StreamingLLM[23]은 고정된 길이의 윈도우를 사용하여 오래된 토큰을 제거하되 초반부의 특정 중요 토큰을 항상 유지함으로써 무제한 길이의 입력에서도 성능 저하 없이 안정적으로 문맥 처리가 가능하게 함
  - 초기 몇 개의 토큰이 의미적으로 중요하지 않더라도, 모델 내부에서 이 토큰들에 매우 높은 어텐션 스코어(attention score)가 지속적으로 집중된다는 사실을 관찰하고 이를 어텐션 싱크(Attention Sink)라고 칭함
  - 어텐션 싱크 현상이 사라지면 모델의 어텐션 분포가 불안정해지고 성능이 급격히 저하되는 것을 실험적으로 확인
  - 모델은 항상 몇 개의 초기 토큰을 싱크(sink)로 사용하므로, 이 싱크 토큰들의 KV만 보존하면 어텐션 안정성이 유지되는데, 여기에 최근 생성된 슬라이딩 윈도우(sliding window) 구간의 KV를 합쳐, 전체 KV 중 일정 크기의 윈도우만 유지하고 이를 다음 레이어의 입력으로 사용
- CascadingKV[24]는 KV 캐시를 여러 개의 계단식 하위 캐시로 나누고 토큰의 중요도에 따라 각 하위 캐시에서 유지될 토큰을 동적으로 결정하여 전체 캐시 크기는 유지하면서도 유효한 문맥 길이를 지수적으로 확장

수행기관	개발 목표	개발내용 및 범위
		• StreamingLLM 방식이 통신 효율 측면에서 높은 이점을 제공하지만, 슬라이딩 윈도우 크기가 고정되어 있어 입력 문맥이 길어질수록 초반 토큰이나 중간 구간의 의미 정보가 사라지면서 모델이 장기 문맥 의존성을 충분히 활용하지 못한다는 한계를 보완하기 위한 접근 방식으로 제안된 구조 • 전체 문맥을 단일 윈도우로 관리하는 기존 방식과 달리, CascadingKV는 문맥을 여러 계층 단위로 분할하고 각 구간에 대해 서로 다른 보존 정책을 적용하는 계층적 메모리 구성 방식 • 최신 토큰 구간은 높은 우선순위로 보존하되, 오래된 문맥이라 하더라도 일정 부분을 축약한 형태로 저장함으로써, 모델이 장기 문맥을 보다 지속적으로 활용할 수 있도록 설계됨 • 이러한 방식은 단기 문맥과 장기 문맥의 균형을 유지하면서도, 전부를 GPU에 유지하지 않고 필요시 CPU와 GPU간 전달을 수행한다는 점에서 StreamingLLM 방식과 차별점을 가짐 - ShadowKV[25]는 Key는 GPU에, Value는 CPU로 오프로딩하고 필요시 프리패치하여 GPU 메모리 부담과 전송 병목을 줄이는 방식으로 긴 문맥 추론을 효율적으로 수행 • 기존 장기 문맥 처리 방식에서 발생하는 GPU 메모리 부담과 전송 병목 문제를 해결하기 위해 제안된 비대칭적 KV 캐시 관리 방식 • Key 토큰은 GPU에 유지하되, Value 토큰은 가지치기 및 CPU 오프로딩 대상이 됨 • GPU에는 양자화된 형태의 Key만 남아 있어 어텐션 경로를 유지하고, Value는 계산에 필요한 시점에만 GPU로 불러오는 방식 • 이 과정에서 Value는 단순 삭제되는 것이 아니라 CPU 측에 보존되며, 모델이 해당 문맥을 재참조할 가능성이 감지되면 Value를 미리 GPU로 전송하는 프리패치 기반 접근 방식을 적용 • 이를 통해 GPU 연산이 수행되는 동안 CPU에서 GPU로의 데이터 전송이 병렬화되어, 시스템은 추가적인 전송 비용을 거의 느끼지 않으면서 긴 문맥 모델 추론 수행 가능 - InfiniGen[26]은 쿼리-키 패턴을 예측해 꼭 필요한 KV 항목만을 선택적으로 GPU로 가져오는 방식으로 전송량을 줄임 • 모든 KV 캐시를 CPU 메모리에 풀(pool) 형태로 유지하되, GPU로는 어텐션 계산에 실제로 필요한 소수의 중요 토큰만 선택적으로 프리패칭하는 방식을 채택함 • InfiniGen의 핵심 아이디어는 현재 레이어의 어텐션 입력을 활용하여 다음 레이어의 어텐션 패턴을 미리 추측하는 것임 • 이를 위해 오프라인 단계에서 특이값 분해(Singular Value Decomposition, SVD)를 적용하여 Query 및 Key 가중치 행렬을 스큐잉(Skewing)하고, 소수의 열이 훨씬 큰 값을 갖도록 재구성함 • 이렇게 수정된 가중치는 수학적으로 원래 계산과 동일한 결과를 보장하면서도, 부분 가중치만으로 전체 어텐션 패턴을 효과적으로 근사할 수 있게 함 • 디코딩 단계에서는 레이어의 어텐션 입력에 레이어의 부분 Query 가중치와 부분 Key 캐시를 곱하여 추측된 어텐션 스코어를 계산하고, 최댓값에서 임계값을 뺀 값보다 큰 스코어를 가진 토큰들의 KV 캐시만 GPU로 전송함 • 이 과정에서 InfiniGen은 레이어별, 쿼리 토큰별로 필요한 KV 캐시 양을 동적으로 조정하여, 평균적으로 전체 KV 캐시의 약 10%만 GPU로 가져오면서도 모델 정확도를 거의 손실 없이 유지함 ● StreamingLLM 기반 가지치기에서 슬라이딩 윈도우 크기에 따른 모델 성능 변화 분석 - 캐시 크기가 증가할수록 통신 시간은 전송 데이터량 증가에 따라 급격히 증가하는 경향을 보이나, 연산 시간은 거의 일정하게 유지됨

수행기관	개발 목표	개발내용 및 범위
		 그림 622 StreamingLLM에서 슬라이딩 윈도우 사이즈에 따른 연산과 통신 비용 변화 - KV 캐시 가지치기 수준의 증가는 통신 시간을 효과적으로 단축시키는 반면, 일정 임계값 이하로 축소될 경우 모델이 유지해야 할 핵심 문맥 정보가 손실되면서 추론 정확도가 급격히 저하되는 상충 관계를 형성함 - 윈도우 크기가 커질수록 더 많은 문맥 정보가 유지되어 퍼플렉시티가 감소하는 경향을 보이는 반면, 윈도우 크기를 128K에서 256 토큰 수준으로 급격히 축소할 경우 핵심 문맥 정보의 상당 부분이 제거되면서 PPL이 급격히 상승하는 현상이 나타남  그림 623 StreamingLLM에서 슬라이딩 윈도우 사이즈에 따른 모델 정확도 ● 가지치기 전략에 따른 KV 캐시 단편화 및 통신 성능 영향 분석 - KV 캐시를 줄이기 위해 적용되는 가지치기 전략은 단순히 전송해야 할 총 데이터량뿐 아니라, KV 캐시의 메모리 배치 구조와 단편화 특성을 변화시켜 실제 통신 효율에 중대한 영향을 미침 - 동일한 전송량 조건에서도 캐시가 어떻게 분할·배치되는지에 따라 CPU-GPU 간 데이터 전송 과정에서 발생하는 호출 오버헤드와 PCIe 대역폭 활용 효율이 현저히 달라짐 - StreamingLLM과 CascadingKV의 KV 캐시 저장 구조 차이는 CPU에서 GPU로 KV 캐시를 전송하는 과정에서 메모리 복사 호출 횟수(memcpy call) 차이로 직결됨 • StreamingLLM은 슬라이딩 윈도우와 싱크 영역으로 구성된 연속적인 KV 블록을 유지하여, GPU로 전달될 캐시가 비교적 조밀한 단일 연속 메모리 구간으로 관리 • CascadingKV는 토큰 중요도에 따라 KV 캐시를 여러 개의 하위 집합으로 분리 저장하는 구조를 채택하고 있어, 메모리 상에서 다수의 분산 블록으로 관리되는 단편화된 배치(fragmented layout)가 형성됨

수행기관	개발 목표	개발내용 및 범위

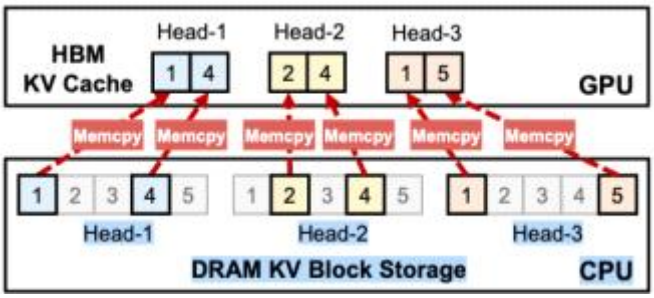


그림 624 가지치기로 인한 메모리 파편화 개념

- LLaMA3-8B 단일 레이어, 입력 길이 10K 환경에서 각 방식의 KV 캐시 전송 시간 비교

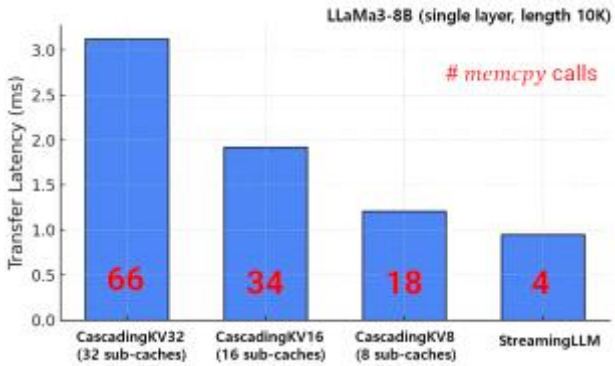
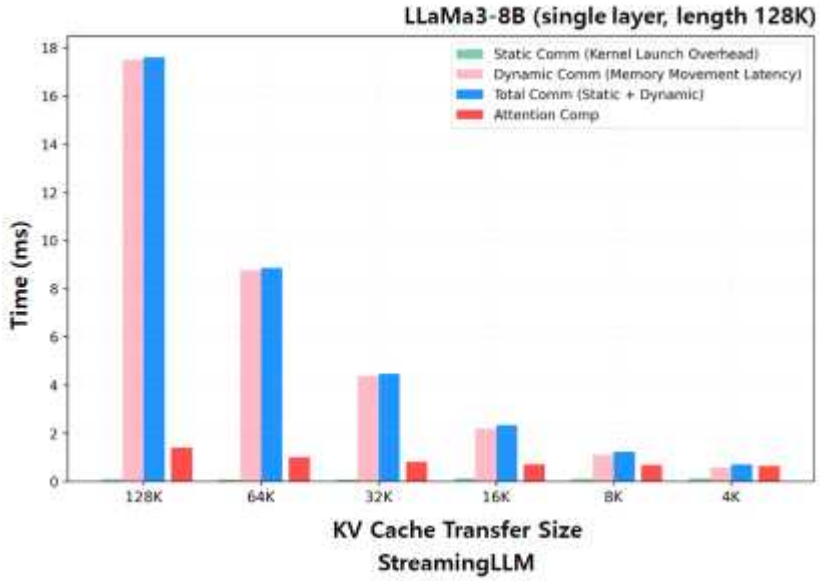


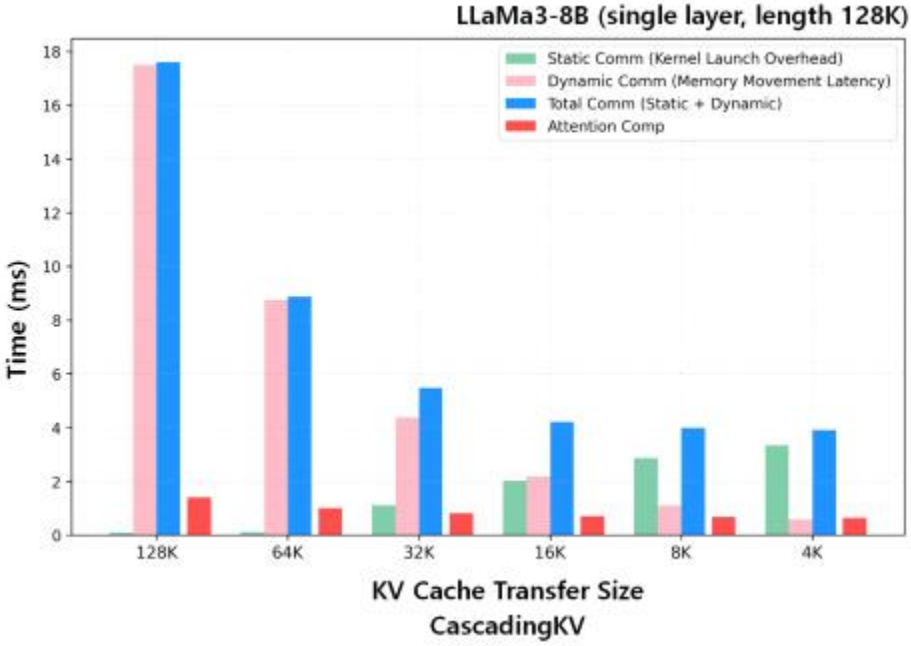
그림 625 가지치기 전략에 따른 통신 시간

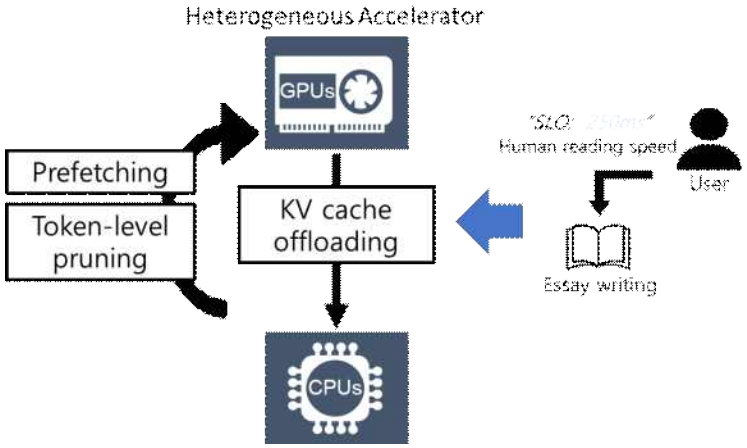
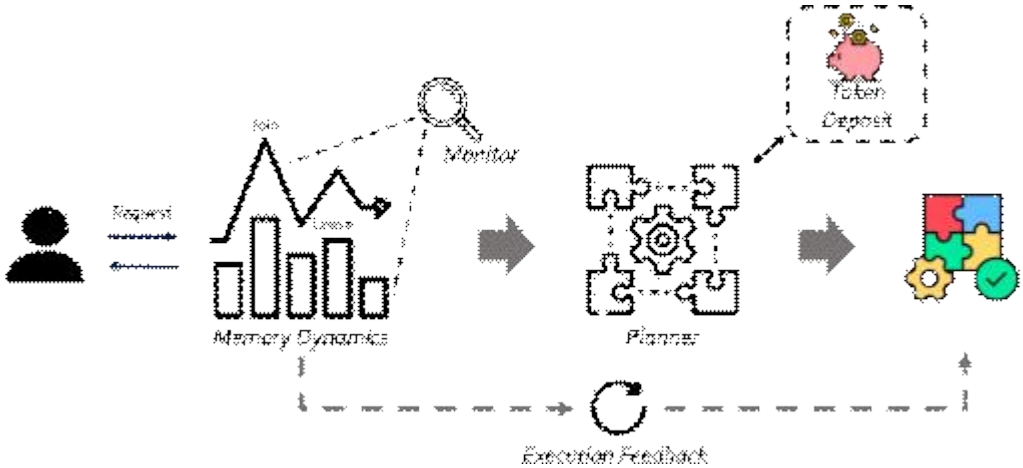
- StreamingLLM은 GPU로의 캐시 로딩 과정이 총 4회의 메모리 복사 호출로 완료되며, 입력 길이나 캐시 크기 변화와 무관하게 해당 호출 횟수가 일정하게 유지됨
- CascadingKV는 KV 캐시를 여러 하위 캐시 집합이 개별적으로 전송되면서, 하위 집합 수에 비례해 메모리 복사 호출이 누적됨
- 실험에 사용된 CascadingKV8, CascadingKV16, CascadingKV32에서는 각각 8개, 16개, 32개의 분리된 캐시 집합이 유지되며, 이에 따라 호출 횟수가 각각 18회, 34회, 66회로 증가함
- 이는 하위 집합 분할 규모가 커질수록 호출 오버헤드가 누적되며, 전송 단계에서의 비효율이 급격히 확대됨을 의미
- 통신 지연이 전송 바이트 수 자체보다는 메모리 단편화로 인해 발생하는 호출 오버헤드의 누적 효과에 의해 결정됨을 보여줌

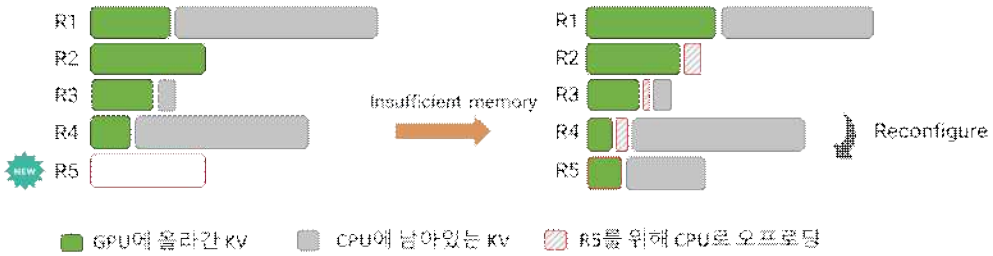
● 가지치기에 따른 단편화 증가가 메모리 복사 호출 횟수 및 디코딩 전송 지연에 미치는 영향 분석

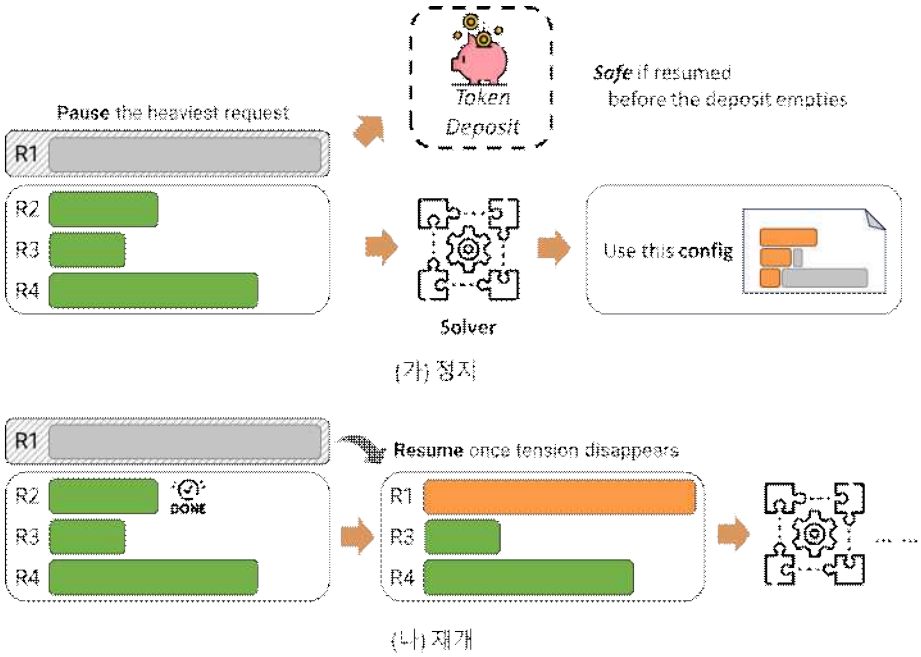
- 정적인 통신 준비 과정(Static Communication)과 실제 KV 캐시 전송 과정(Dynamic Communication)
  - Static Communication(Static Comm)은 GPU로 KV 캐시가 복사되기 이전에 수행되는 준비 비용으로 전송 요청이 실행 환경에 등록되고, 해당 요청을 수행할 커널이 활성되는 과정에서 발생
  - Static Comm은 데이터 크기와 직접적으로 연동되지 않으며, 전송 수행을 위해 발생하는 메모리 복사 호출 횟수에 따라 누적되는 특성을 가짐
  - KV 캐시 전송량이 감소하더라도 메모리 복사 횟수가 유지되거나 증가하는 구조에서는 Static Comm이 전체 지연의 증가 요인으로 작동할 수 있음
  - Dynamic Communication(Dynamic Comm)은 CPU에 저장된 KV 캐시를 GPU로 실제 복사하는 전송 단계에서 발생하는 지연으로, 이 비용은 전송되는 KV 캐시의 양과 선형적으로 비례

수행기관	개발 목표	개발내용 및 범위
		• Static Comm이 "전송을 몇 번 수행해야 하는가"를 반영한다면, Dynamic Comm은 "한 번의 호출에서 얼마나 많은 데이터를 전송하는가"를 나타내는 비용 구조 - StreamingLLM 기반에서 KV 캐시 전송량에 따른 디코딩 레이어 단위 KV 캐시 전송 시간 및 어텐션 연산 시간 비교 • 유지해야 하는 KV 캐시는 슬라이딩 윈도우 영역과 싱크 영역으로 제한되어 있으며, Key와 Value가 독립적으로 처리됨 • 이로 인해 레이어당 CPU에서 GPU로 KV 캐시 메모리 복사 호출 수는 4회로 고정되며, 이는 KV 캐시 전송 양에 대한 변동과 무관하게 유지되는 구조적 속성임 • 초기 128K 캐시 구간에서는 전송해야 하는 KV 캐시 양이 크기 때문에 Dynamic Comm이 전체 통신 비용의 대부분을 차지함 • 슬라이딩 윈도우 크기를 64K에서 32K, 이후 16K와 8K로 단계적으로 줄이고, 최종적으로 4K 수준까지 축소하자 KV 캐시 전송량이 선형적으로 감소하며, 이에 따라 Dynamic Comm 또한 동일한 비율로 감소함 • 반면 Static Comm은 호출 수가 변화하지 않기 때문에 일정 범위에서 유지되고, 전체 통신 비용은 KV 캐시 크기 축소에 따라 정비례하게 감소하는 구조적 패턴을 보임  그림 626 StreamingLLM 기반에서 KV 캐시 전송양에 따른 디코딩 레이어 단위 KV 캐시 전송 시간 및 어텐션 연산 시간 비교 - CascadingKV 기반에서 KV 캐시 전송량에 따른 디코딩 레이어 단위 KV 캐시 전송 시간 및 어텐션 연산 시간 비교 • CascadingKV는 하나의 연속된 캐시 구간이 아니라, 중요도에 따라 분리된 총 32개의 하위 KV 캐시 집합으로 구성된 구조이며, 각 하위 집합은 Key와 Value가 독립적으로 전송되는 단위로 관리됨 • 여기에 디코딩 과정에서 유지해야 하는 싱크 영역까지 포함되면서, 한 레이어 기준 발생하는 KV 캐시 전송 작업은 단일 호출이 아닌 다수의 메모리 복사 호출 형태로 수행됨 • 그 결과, CascadingKV32에서는 한 레이어 당 호출 수가 총 66회에 이르고, 이는 KV 캐시 크기가 줄어들더라도 메모리 복사 호출 수가 감소하지 않는 특성을 가짐으로 인해 캐시 축소가 곧바로 통신 지연 개선으로 이어지지 않는 구조적 한계가 나타남 • 실제로 128K에서 시작해 전송되는 KV 캐시 크기를 32K, 16K, 8K, 4K로 단계적으로 축소했음에도 불구하고, Static Comm은 캐시 크기 변화와 무관하게 일정한 수준에서 유지되며, 오히려 전체 전송 지연에서 차지하는 비중이 점차 증가하는 패턴을 보임 • CascadingKV 구조에서는 호출 횟수를 결정하는 요인이 캐시 크기가 아니라 캐시 단

수행기관	개발 목표	개발내용 및 범위
		편화 정도와 하위 KV 캐시 단위 구성 방식이기 때문임 • 미세화된 전송 단위를 유지한 상태에서 데이터양만 줄이는 경우, Dynamic Comm은 감소하더라도 Static Comm은 제거되지 않기 때문에 전송 지연이 구조적으로 남아 있게 됨 • 예를 들어 4K 크기의 KV 캐시를 전송해야 하는 상황에서도, StreamingLLM은 4회의 메모리 복사 호출로 작업이 완료됐지만 CascadingKV는 최대 66회의 호출을 해야 했으며, 그 결과 전체 전송 시간에서 두 배 이상의 차이가 발생함 • 이는 가지치기가 단순히 KV 캐시 크기를 줄이는 문제가 아니라, 전송할 KV 캐시가 어떤 구조적 형태로 저장되고, 어떤 단위로 전송되며, 그 과정에서 몇 번의 메모리 복사 호출이 수행되는가가 성능을 좌우하는 핵심 설계 변수임을 의미함 • 긴 문맥 기반 추론 환경에서는 캐시 크기보다는 캐시의 단편화 정도와 호출 구조의 효율성이 모델의 실제 처리 성능을 결정하는 본질적 요인으로 작용함  그림 627 CascadingKV 기반에서 KV 캐시 전송양에 따른 디코딩 레이어 단위 KV 캐시 전송 시간 및 어텐션 연산 시간 비교 (3) 실시간 모니터링 기반의 동적 오프로딩 및 프리패칭 기술 개발 ● 실시간 모니터링 기반으로 KV 캐시의 배치와 이동을 동적으로 재조정하는 동적 KV 캐시 관리 프레임워크를 개발 - GPU 메모리 활용 효율을 높이고 서비스 수준 목표 달성률을 안정화하는 것을 목표로 함

수행기관	개발 목표	개발내용 및 범위
		 그림 628 동적 KV 캐시 관리 개념 - 토큰 생성 과정과 출력 스케줄링을 분리하여 연산 진행과 사용자 응답 전달을 독립적으로 제어함으로써 서비스 수준 목표를 안정적으로 유지 • 디코딩 초기 단계에서 서비스 수준 목표 대비 여유 시간이 확보되는 경우, 생성된 토큰을 임시 보관 버퍼에 저장해 둠 --> 토큰 예치(Token Deposit) • 이후 특정 요청에서 KV 캐시 전송 지연 등으로 응답 지연 증가가 감지되면, 해당 요청의 디코딩을 일시적으로 정지시키는 한편, 버퍼에 저장된 토큰을 일정한 속도로 방출하여 사용자에게 연속적인 응답을 제공 - 런타임 상태를 모니터링하고 이를 바탕으로 배치 계획을 수립한 뒤, 계획에 따라 실행을 수행하고 그 결과를 다시 제어 신호로 환류하는 순환 구조로 지속해서 동작 • Memory dynamics를 모니터링 • Planner가 KV 배치 및 offloading 전략을 생성 • Execution feedback을 통해 계획을 계속 갱신 • 필요할 때 Token Deposit을 사용해 SLO를 유지  그림 629 실시간 모니터링 기반 동적 오프로딩 프리패칭 실행 흐름 ● 오프로딩과 프리패칭 전략의 동적 재구성과 제어 기법 - 오프로딩 전략이 고정된 규칙에 의해 사전에 결정되는 방식이 아니라 GPU 메모리 사용 현황과 배치 구성 변화를 실시간으로 반영하여 지속적으로 재구성 • 각 요청의 디코딩 진행 상황, 가용 GPU 메모리 용량, 통신 병목 상태 등을 종합적으로 고려하여, 요청 별 KV 캐시의 GPU 상주 비율을 동적으로 조정함으로써 전체 시스템 지연을 최소화 - 배치에 포함된 특정 요청이 추론을 완료하고 종료될 경우의 GPU 메모리 회수에 따른

수행기관	개발 목표	개발내용 및 범위
		KV 캐시 재배치 • 추론 완료된 요청이 점유하던 GPU 메모리가 즉시 해제되고, 이때 기존의 메모리 제약으로 인해 일부 KV 캐시만 GPU에 유지하던 다른 요청들은 새롭게 확보된 공간을 활용하여 자신의 KV 캐시를 GPU로 로딩할 수 있게 됨 그림 630 요청 R4 종료 시 GPU 메모리 회수에 따른 KV 캐시 재배치 - 새로운 요청이 배치에 추가되는 경우의 KV 캐시 재배치 • 현재 배치에 포함된 모든 요청의 메모리 사용 상태를 재평가하고, 일부 요청의 KV 캐시를 CPU로 추가 오프로딩하는 방향으로 배치 계획을 조정 • 메모리 제약, 전송 비용, 각 요청의 서비스 수준 목표 잔여 여유 시간을 종합적으로 고려하여 최적 조합을 탐색  그림 631 요청 R5 유입 시 GPU 메모리 압박에 따른 KV 캐시 재배치 - 정지 후 재개(Pause and Resume) • GPU 메모리가 지나치게 부족하거나, PCIe 전송 혼잡이 심화되어 어떠한 오프로딩 조합으로도 모든 요청의 서비스 수준 목표를 동시에 만족시킬 수 없는 상황을 위해 정지 후 재개 기법을 안전장치로 도입 • 전체 지연을 가장 크게 유발하는 요청을 일시적으로 중단시켜 GPU 메모리 및 전송 대역폭 자원을 환수한 뒤, 잔여 요청을 대상으로 새로운 오프로딩 계획을 재계산하여 서비스 수준 목표 충족 가능성을 회복하는 방식 • 시스템 자원이 충분히 회복되면 중단된 요청의 처리가 다시 이어지게 하는데, 예를 들어 다른 요청이 종료되어 GPU 메모리에 여유가 생기면, 해당 시점의 중단 요청의 재개 가능 여부를 판단하여 새 오프로딩 계획 하에 디코딩을 이어 진행함 • 정지 후 재개 기법은 극한의 자원 제약 환경에서도 시스템 전체의 서비스 수준 목표 달성을 안정적으로 유지하기 위한 보완 장치 기능을 함

수행기관	개발 목표	개발내용 및 범위
		 그림 632 정지 후 재개 기법 동작 과정 ● 오프로딩과 프리패칭 전략별 성능 비교 - 개발한 스케줄링 기법의 실효성을 검증하기 위해 최근 제안된 동적 오프로딩·프리패칭 기반 추론 시스템들(FlexGen+, Dynamic Heuristic)과 긴 문맥 추론에서의 성능을 비교 • FlexGen+ : 배치 구성 변경 시마다 KV 오프로딩 정책을 재설정함으로써 FlexGen[27]의 정적 KV 캐시 방식의 배치 적응성을 개선하나, 토큰 단위 최적화, 서비스 서비스 수준 목표 인지 스케줄링, 런타임 기반 동적 캐시 관리는 여전히 제공하지 않음 • Dynamic Heuristic : 본 연구의 핵심 의사결정 모듈에 ILP 기반 솔버 대신 단순 휴리스틱 정책을 적용한 것으로, 오프로딩되는 레이어 수를 최소화하는 방향으로만 의사결정을 수행하여 계산 비용은 낮아지지만, 요청 자원 상태를 반영한 정교한 스케줄링은 구현하지 못함 - 실험 설계 • 전체 디코딩 과정 동안 각 토큰 생성 시점의 지연 시간이 서비스 수준 목표 기준을 만족한 비율을 주요 성능 지표로 사용(즉, 토큰 단위로 측정된 실측 지연 값이 서비스 수준 목표를 달성한 비율) • 입력 문맥 길이를 8K, 32K, 128K 환경으로 구분해 평가 • 기준 서비스 수준 목표는 오프로딩없이 처리 가능한 추론 속도를 바탕으로 설정 • 서비스 요구 수준이 강화되는 상황을 모사하기 위해 해당 기준을 각각 1.5배, 2.5배 강화한 조건에서도 동일한 실험을 수행 - 실험 결과 분석 • FlexGen+와 Dynamic Heuristic은 문맥 길이가 증가함에 따라 서비스 수준 목표 달성률이 급격히 저하되는 경향을 보이며, 특히 128K 환경에서는 각각 69.2%와 32.9%까지 하락 • 본 연구의 기법(OrbitFlow)은 128K 긴 문맥 조건에서도 85% 이상의 서비스 수준 목표 달성률을 안정적으로 유지하는데 이는 요청별 상태 변화와 메모리 압력, 통신 지연을 단계별로 반영하여 토큰 배치를 정밀하게 조정하는 구조의 효과임 • 긴 문맥 환경으로 갈수록 토큰 개수나 배치 내 요청 구성의 급격한 변화 현상이 심화되어, 요청 간 진행 속도 편차가 커지고 KV 캐시 증가 패턴도 불규칙해짐으로 인해 단순한 휴리스틱이나 배치 단위 정책만으로는 적절한 오프로딩 전략을 유지하기 어려움 • 요청별·스텝별 상태를 반영한 실시간 최적화 기반 스케줄링 구조가 긴 문맥 추론 서비스에 특히 효과적임을 보여줌

수행기관	개발 목표	개발내용 및 범위

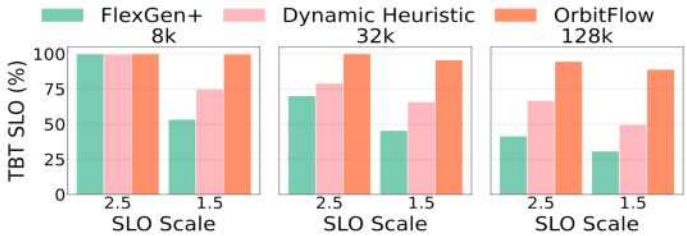


그림 633 동적 오프로딩·프리패칭 프레임워크의 서비스 수준 목표 달성률 비교

(4) 이종 가속기 환경에서 KV 캐시 관리 전략의 확장 가능성 평가

- 실제 LLM 추론 서비스 운영 환경에서 연산 자원은 동일한 구성으로 제공되기 어렵고, 성능이 다른 GPU가 혼합된 이기종 환경에서 실행되는 경우가 일반적임
- StreamingLLM 기반 최적화 기법이 동일 GPU 환경과 이기종 GPU 환경 모두에서 일관된 성능 향상을 보이는지, 그리고 하드웨어 구성에 따른 처리량 변동이 어느 수준에서 발생하는지를 확인하고자 실험을 수행함
  - 실험 환경 구성
    - 동종 GPU 환경 : 동일 사양의 A6000 GPU 두 개를 사용
    - 이종 GPU 환경 : A6000과 A5000 각각 1개를 조합하여 사용
  - LLaMA3-8B 모델을 대상으로 32K Context에 대해 파이프라인 병렬화 방식으로 수행
  - KV 캐시 슬라이딩 윈도우 크기를 16K에서 8K 4K로 점진적으로 감소시키며, 윈도우 크기가 처리량에 미치는 영향을 측정
  - 실험 결과 분석
    - 두 환경 모두 슬라이딩 윈도우 크기가 감소할수록 처리량이 증가하는 동일한 추세를 보였으나 이종 GPU 환경은 동종 GPU 환경 대비 처리량이 크게 떨어짐
    - compute/memory 차이 때문이며 더 낮은 capability를 가진 GPU가 전체 batch의 straggler로 동작함

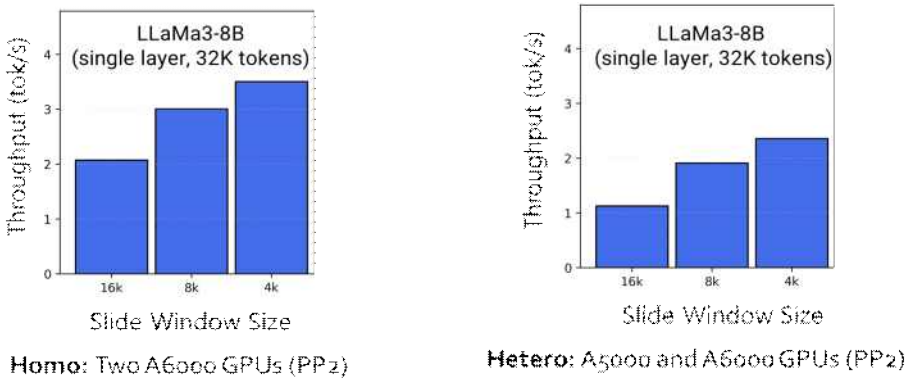
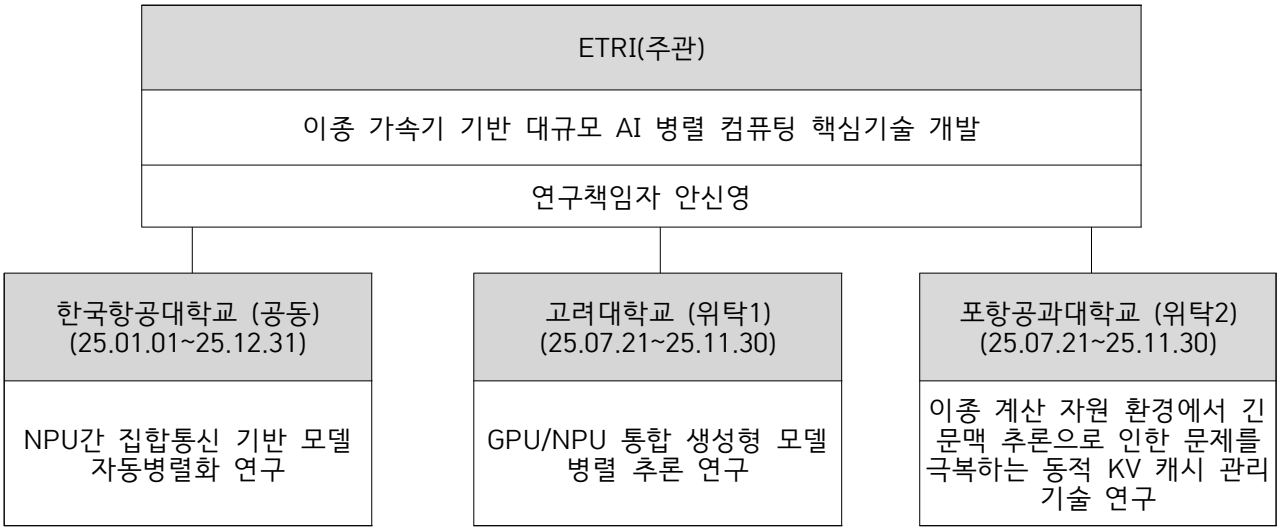


그림 634 동종/이종 GPU 환경에서의 처리량 비교

- 이종 가속기 환경에서는 동일한 KV 캐시 최적화 정책으로는 좋은 성능을 낼 수 없으므로 능력 인지형(Capability-aware) KV 배치 및 가지치기 전략이 필요함
  - 메모리 용량과 연산 성능이 다른 가속기(GPU)가 혼재된 실제 서빙 환경에서는 단일 오프로딩 또는 가지치기 정책을 적용할 경우, 가장 낮은 성능의 가속기가 전체 처리량을 제한하게 됨
  - 각 가속기의 연산 성능, 메모리 제약, 통신 특성을 정량적으로 모델링하고, 이를 기반으로 요청 배치, 레이어 분할, KV 캐시 배치 및 가지치기 정책을 장치별로 차등화하여 가속기 자원 특성에 부합하는 맞춤형 KV 캐시 관리
    - 고성능 GPU에는 상대적으로 큰 슬라이딩 윈도우를 유지하여 연산 자원을 적극 활용
    - 메모리 용량이 제한된 NPU 또는 저사양 GPU에는 강화된 가지치기 전략을 적용
  - 이를 통해 이종·분산 환경에서도 통신 부하, 연산 효율, 추론 정확도 간의 균형을 안정

수행기관	개발 목표	개발내용 및 범위
		적으로 유지하는 확장형 긴 문맥 추론 실행이 가능할 것으로 생각됨



3. 과제수행기간 추진체계 및 방법

가. 과제수행 추진체계

○ 인공지능연구소 초성능컴퓨팅연구본부는 ETRI R&R 2-1 페타바이트급 스케일의 메모리 중심 고성능 컴퓨팅 시스템의 개발과 그에 요구되는 IPR 확보 주도

- ETRI 초성능 컴퓨팅 연구본부에서 데이터 중심 새로운 컴퓨팅 시스템 구조 및 시스템SW 원천 기술을 연구
- ETRI 초성능 컴퓨팅 연구본부와 대학이 공동연구를 통해 대용량·비휘발성 메모리 기반 지능정보 데이터 처리 가속 기술을 연구
- ETRI 초성능 컴퓨팅 연구본부에서 EdgeCPS를 위한 새로운 컴퓨팅 시스템 구조 및 시스템SW 원천 기술을 연구
- ETRI 초성능 컴퓨팅 연구본부에서 저자원 디바이스간 연산 능력 차이를 극복하는 디바이스리스 컴퓨팅 시스템 구조 및 시스템SW 원천 기술을 연구하여 프로토타입을 개발
- 대학은 EdgeCPS 실행엔진의 원활한 운영을 위한 디바이스 트윈 명세 기술 및 탄력적 스케줄링 알고리즘을 연구
- 대학은 자원 제약 환경에서의 디바이스 관리 및 자원 증강에 대한 명세 기술 및 탄력적 운영을 지원하는 동적 스케줄링 알고리즘을 연구
- 기업은 EdgeCPS 실행엔진의 결과를 가시화하기 위한 실세계 디바이스 모델링 방법을 연구

[메모리 중심]

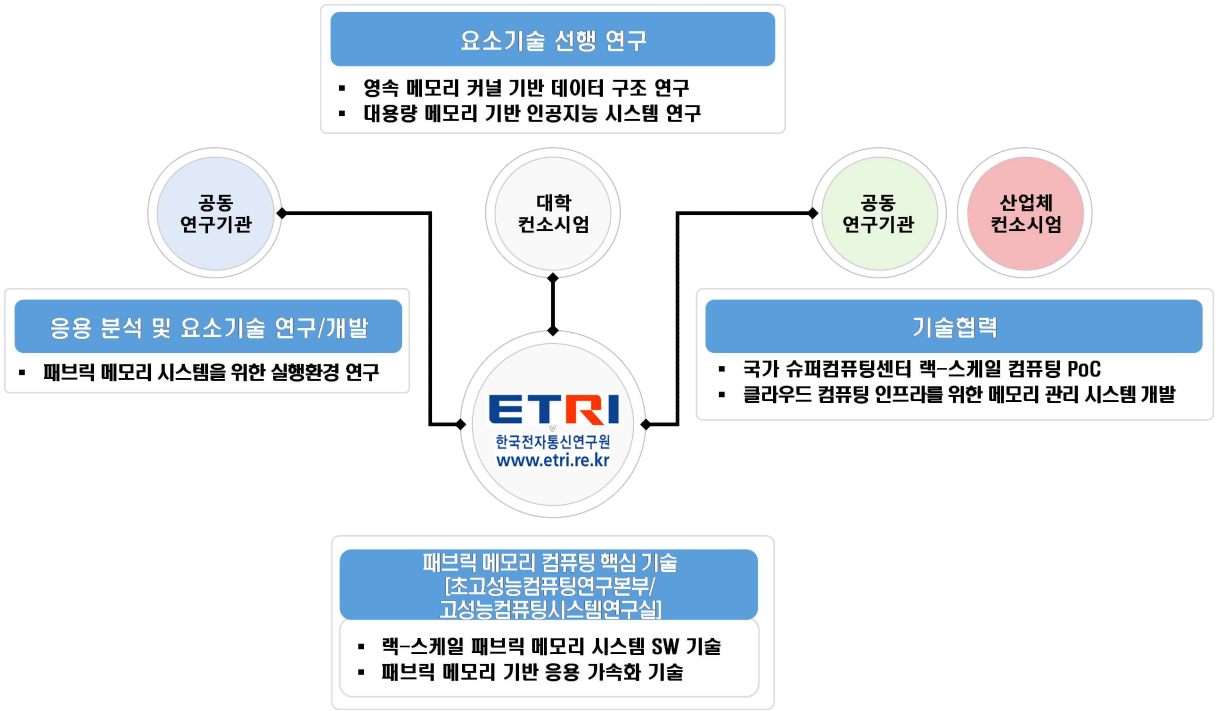


그림 635 추진 체계도-메모리 중심

○ 공동연구를 통해 주요 요소기술 연구/개발을 수행

- 인공지능을 위한 시스템 연구를 전문으로 하는 컴퓨팅 시스템 연구자와의 공동 연구를 통하여 메모리 중심 컴퓨팅 시스템 기반의 인공지능 플랫폼 연구
- 비휘발성 메모리 및 인메모리 저장소를 전문으로 하는 컴퓨팅 시스템 연구자와의 공동 연구를 통하여 비휘발성 메모리 기반 데이터 관리 기술 연구

- 분산 시스템을 전문으로 하는 컴퓨팅 시스템 연구자와 공동 연구를 통하여 메모리 중심 컴퓨팅 시스템 기반 쓰레드 병렬화 기술 연구

○ 대학 협력을 통해 요소기술 선행연구를 수행
- 국내외 고급 인공지능 및 시스템 연구자와 연구 협력 진행를 위해 관련 분야 전공 교수(대학)에 위탁하여 수행 후 메모리 중심 컴퓨팅 시스템의 요구사항으로 활용
- 국내외 그래픽 분석 및 시스템 연구자와 연구 협력 진행을 위해 관련 분야 전공 교수(대학)에 위탁하여 수행 후 메모리 중심 컴퓨팅 시스템의 적용 사례로 활용

○ 기관(기업, 기타 출연연) 협력을 통해 개발 시스템 검증 및 대용량 데이터 분석 응용 개발을 공동 수행
- 정부 출연 연구소들과의 협업을 통해서 과학 계산으로의 메모리 중심 컴퓨팅 기술 연구 적용처 발굴
- 산업체 컨소시엄과 함께 메모리 중심 컴퓨팅을 활용하는 클라우드 컴퓨팅 인프라를 위한 메모리 관리 시스템 개발 수행
- 산업체 컨소시엄과 함께 메모리 중심 컴퓨팅을 활용하는 대용량 디지털포렌식 응용 개발 수행

[EdgeCPS]

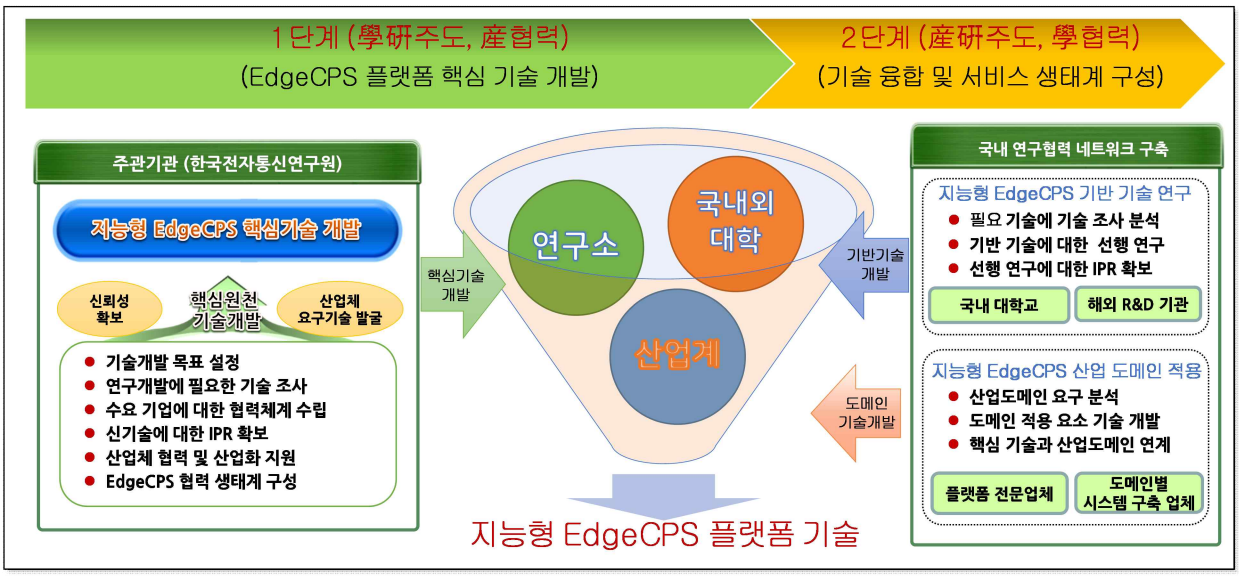


그림 636 추진체계도-EdgeCPS

○ 세계 기술을 선도하고 있는 국내외 대학과의 공동연구를 통해 요소기술의 연구/개발을 진행
- 저자원 디바이스간 연산 능력 차이를 극복하는 디바이스리스 컴퓨팅 시스템 구조 및 시스템SW 원천 기술을 공동으로 연구하여 프로토타입을 개발
- 대학은 자원 제약 환경에서의 디바이스 자원 증강을 위한 가상화된 스토리지 자원의 동적 스케줄링 알고리즘과 다양한 디바이스의 성능차이를 극복하기 위한 실디바이스 연계형 태스크 분할 실행 기술을 연구
- 국내외 고급 군집지능 및 실세계 디바이스의 디지털화를 위한 데이터 모델 연구자와의 협

력 연구 진행을 위해 관련 분야 전공 교수(대학)에 위탁하여 수행 후 EdgeCPS 플랫폼의 핵심 모듈로 활용

- 실세계에서 동작하는 디바이스 HW의 가상화 연구를 위해 근거리 가상공간 내에서 사용자들간 멀티모달 상호작용에 따른 성능을 분석하고 관련 디바이스를 활용하여 연구결과물을 검증

○ 기관(기업, 기타 출연연) 협력을 통해 디바이스리스 컴퓨팅 기술의 활용 및 개발 시스템 검증 등을 추진하고 향후 상용화 공동 수행을 위한 협력 체계 마련

- 다양한 산업 분석을 통한 수요자요구형기술을 개발하여 디바이스 HW 관련 기업으로부터 차세대 IoT 및 센서 등 다양한 디바이스 플랫폼에 적용 추진
- 분야별 시장 지배적인 사업자와 기술 진입이 용이한 제품군을 선택한 후 산업내 주요 제품에 필요한 핵심 기술 개발 및 상용화 추진
- 산업분야별로 다양한 기술을 보유한 연구소(한국생산기술연구원, 선박해양플랜트연구소, 포항산업과학연구원 등)들과의 협업을 통해서 대규모 디바이스 가상화 플랫폼을 구축하기 위한 협력 방안을 마련하고 기연구 결과와 통합하여 범국가적으로 활용가능한 플랫폼으로 개발

[이종가속기AI컴퓨팅]

그림 637 추진체계도-이종가속기AI컴퓨팅

○ 국내 대학과의 협력을 통해 요소기술의 선행 연구를 수행

- 국내 고성능 시스템 연구자들과 연구 협력 진행를 위해 관련 분야 전공 교수(대학)와 위탁 연구를 수행하여 그 결과를 AI 병렬 컴퓨팅 시스템의 요소 기술 및 적용 사례로 활용

○ 기관(기업) 협력을 통해 AI 병렬 컴퓨팅 핵심 기술의 활용 및 검증이 가능하도록 요구사항을 수집하는 등의 협력 체계 구축

- NPU 관련 기업과의 관련 협업을 통해 시장 요구 사항을 발굴하여 과제에 활용

나. 과제수행 방법

○ [메모리 중심] 기수행 연구 및 기보유 기술에 기반한 기성 하드웨어 기반 연구 추진

- 고성능 패브릭 메모리 시스템 구축 연구 수행
  - 기성 하드웨어 및 운영체제등을 지원하는 관리 용이성을 고려한 설계 수행
  - 요구 분야에 따른 패브릭 메모리 시스템 가속화 방안 연구
- 패브릭 메모리 시스템 자원 관리 기술 연구
  - 사용자 라이브러리 수준 혹은 가상머신 커널 수준의 자원 관리 분석
- 패브릭 메모리 활용 응용 가속화 연구
  - 인공지능/금융/유전체/빅데이터 등 대용량 데이터 처리 응용 가속화 연구

○ [EdgeCPS] 주관기관인 ETRI를 중심으로 産學硏 협업형 연구개발 체계를 구축하고, 기수행 연구 및 기보유 기술에 기반하여 EdgeCPS를 위한 새로운 아키텍쳐 연구 및 플랫폼 개발을 추진

- 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 기술 연구
  - GPU, FPGA 등 다양한 가속기 자원의 활용 방안 연구
  - 디바이스별로 다른 하드웨어 사양을 분석하여 적절한 증강 기능을 제공하기 위한 설계
  - 대규모 디바이스 오케스트레이션 및 복잡한 AI서비스를 위한 협력적 실행환경 고려
- 엣지 사이버물리시스템을 위한 서버자원융합형 대규모 디바이스 가상화 기술 연구
  - 다양한 디바이스, 네트워크, 운영체제의 운영 방식을 고려한 디바이스 프로파일 연구
  - 사용자 니즈에 따른 실세계 디바이스와 가상화된 자원의 실시간 동기화를 고려한 연동 방안 고려

○ [이종가속기AI컴퓨팅] 기성 하드웨어 기반 연구 추진

- 소규모 AI 컴퓨팅 시스템 프로토타입 구축 연구 수행
  - 2노드급의 CPU, 메모리, GPU를 고려
- 고성능 AI 분산/병렬 컴퓨팅 기술 연구
  - 동종의 다중 가속기를 이용한 대규모 생성형 모델의 병렬 학습 기술 연구
  - 다양한 세대 GPU(예, V100, A100, H100 등)를 이용한 LLM의 병렬 추론 기술 연구
  - 이종 가속기(예, GPU와 텐스토렌드 NPU)를 활용한 LLM 병렬 추론 기술 연구
- PCIe 기반 가속기간 집합통신 기술 연구
  - PCI 가속 장치간 데이터가 이동하는 경로를 줄일 수 있는 데이터 처리 연구 수행

다. 과제수행 편성도(세부기술 수행체계)

협약(세부)과제 주관책임자
김강호

1세세부 (20.01.01~23.12.31)

패브릭 메모리 컴퓨팅 핵심 기술 연구

고광원

- Peta-scale 메모리 중심 컴퓨팅 핵심 기술 연구
- 메모리 중심 컴퓨팅 기반 스케일러블 데이터 분석 가속 기술 연구

아주대학교
- 메모리 중심 컴퓨팅 시스템을 위한 기반 소프트웨어 기술 연구
- 공동연구개발기관 책임자 김상훈 외
- 고속 데이터 접근을 위한 소프트웨어 통합설계 기술 연구

옵살라대학교
- 대규모 그래프 분석 가속 기술 연구
- 공동연구개발기관 책임자 박창현 외
- 패브릭 메모리 기반 그래프 분석 가속 기술 연구

성균관대학교
- 비휘발성 메모리 기반 스케일러블 데이터 관리 연구
- 공동연구개발기관 책임자 남범석 외
- 비휘발성 메모리 기반 인메모리 저장소 연구

UNIST
- 메모리 중심 컴퓨팅 시스템 기반 인공지능 플랫폼 연구
- 공동연구개발기관 책임자 전명재 외
- 공유 메모리 기반 분산 인공지능 플랫폼 연구

2세세부 (21.01.01~23.12.31)

Massive Things의 초근접 서비스를 위한 지능형 EdgeCPS 플랫폼 개발

전인걸

- 저자원 디바이스간 연산능력 차이를 극복하는 디바이스리스 컴퓨팅 기술 연구
- 엣지 사이버물리시스템을 위한 대규모 디지털트윈 기술 연구
- 엣지 사이버물리시스템을 위한 대규모 디바이스 제어 플랫폼 프로토타입 개발

UT 달라스 대학교
- Multimodal interaction performance test between EdgeCPS based virtual spaces
- 공동연구개발기관 책임자 김진용 외
- 근거리 가상공간 사용자들간 멀티모달 상호작용 기술 연구

경북대학교
- 스토리지 증강을 위한 동적 프로비저닝 기술 연구
- 위탁과제 책임자 서영균 외
- 가상 스토리지 할당과 탄력성있는 동적 프로비저닝 지원 기술 연구

경상대학교
- EdgeCPS 기반의 자원 제약 디바이스의 분할 실행 기술 연구
- 위탁과제 책임자 이성진 외
- 다양한 디바이스의 성능차이를 극복하기 위한 실디바이스 연계형 태스크 분할 실행 기술 연구

경희대학교
- 로컬 모델 기반의 디바이스 데이터 증강 기술 연구
- 위탁과제 책임자 전석희 외
- 로컬 디바이스트윈을 통한 원격지 디바이스의 불충분한 멀티모달 데이터 증강 기술 연구

3세세부 (24.01.01~25.12.31)

이종 가속기 기반 대규모 AI 병렬컴퓨팅 기술 개발

안신영

- 고성능 AI 분산/병렬 컴퓨팅 기술 연구
- 이종 가속기 통합 AI 병렬 컴퓨팅 기술 연구
- 이종 가속기간 고성능 집합통신 기술 연구

한국항공대학교('25)
- NPU간 집합통신 기반 모델 자동병렬화 연구
- 공동연구개발기관 책임자 이재환 외
- 노드레벨 NPU간 집합통신 기술 및 NPU기반 자동병렬화 핵심기술 연구

충북대학교('24)
- CXL 기반 계층 메모리 플랫폼에서 커널 데이터 배치 연구
- 위탁과제 책임자 조희승 외
- CXL 기반 계층 메모리 플랫폼에서 효율적인 커널 데이터 배치 방법에 대한 연구

항공대학교('24)
- 데이터 처리 장치 기반의 AI 분산 컴퓨팅 SW 구조 연구
- 위탁과제 책임자 이재환 외
- 데이터 처리 장치 기반 고성능 분산 AI 프레임워크 구조 연구 및 데이터 처리 장치와 가속기간 직접 통신 기반의 커널 실행 기술 연구

고려대학교('25)
- GPU/NPU 통합 생성형 모델 병렬 추론 연구
- 위탁과제 책임자 안정섭 외
- 수B급 생성형 언어 모델의 병렬 추론 기술 및 NPU 내부 구조의 최적 활용 지원 고성능 커널 라이브러리 연구

포항공과대학교('25)
- 이종 계산 자원 환경에서 긴 문맥 추론으로 인한 문제를 극복하는 동적 KV 캐시 관리 기술 연구
- 위탁과제 책임자 전명재 외
- KV 캐시를 위한 메모리 계층 활용 및 전송 지연 최소화 기술 및 이종 가속기 런타임 환경을 고려한 전송량 축소 전략 동적 조정 기술 연구

3 과제 수행결과 및 목표달성도

1. 과제 수행결과

<table>
<tr><th>구분</th><th>1단계 (성과)</th></tr>
<tr><td>정량</td><td>
<table>
<tr><th colspan="2">논문</th><th colspan="4">특허</th><th colspan="2">기술이전</th></tr>
<tr><th rowspan="2">SCI(건)</th><th rowspan="2">비SCI(건)</th><th colspan="2">해외(건)</th><th colspan="2">국내(건)</th><th rowspan="2">건수</th><th rowspan="2">금액(백만원)</th></tr>
<tr><th>출원</th><th>등록</th><th>출원</th><th>등록</th></tr>
<tr><td>4</td><td>14</td><td>5</td><td>5</td><td>5</td><td>4</td><td>3</td><td>190</td></tr>
</table>
•(사업화) 기술이전
- ㈜시스기어(1억원): 2-계층 메모리 시스템 기술 v1
- ㈜시스기어(0.7억원): 2-계층 메모리 시스템 기술 v2
- ㈜테라텍(0.2억원): PCIe 기반 확장 메모리 기술
•(기타) 국제MoU체결: 미국, TidalScale ('20.02)
•(기타) 실증사업: 에너지기술연구원, 한컴위드, ㈜녹십자지놈
</td></tr>
<tr><td>정성</td><td>•2021년 국가연구개발사업 우수성과 100선 선정</td></tr>
</table>

<table>
<tr><th>구분</th><th>2단계 (성과)</th></tr>
<tr><td>정량</td><td>
<table>
<tr><th colspan="2">논문</th><th colspan="4">특허</th><th colspan="2">기술이전</th></tr>
<tr><th rowspan="2">SCI(건)</th><th rowspan="2">비SCI(건)</th><th colspan="2">해외(건)</th><th colspan="2">국내(건)</th><th rowspan="2">건수</th><th rowspan="2">금액(백만원)</th></tr>
<tr><th>출원</th><th>등록</th><th>출원</th><th>등록</th></tr>
<tr><td>10</td><td>44</td><td>9</td><td>4</td><td>22</td><td>7</td><td>2</td><td>60</td></tr>
</table>
•(사업화) 기술이전
- ㈜크래프트X(3천만원) : 컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술
- ㈜알티스트(3천만원) : 컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술
•(표준화) ITU-T SG16 Q5 (SG16-C243R1 : Reference model of AI knowledge sharing for AI computing platform)
•(기타) IPR확보를 위한 프로그램 등록 7건
</td></tr>
<tr><td>정성</td><td>•패브릭 메모리 컴퓨팅 핵심 기술 사업화를 위한 애로기술지원사업 진행 (1건)
•5G 연계형 디바이스의 혁신적 서비스 개발 지원 협력 (경기도경제과학진흥원)
•다양한 디바이스가 설치되는 대규모 생산환경에 적용 가능성 검토 협력 (RIST, POSCO)
•GPU 가속 기술을 이용한 대규모 생산 설비 데이터 분석 기술 협력 (한국생산기술연구원)
•레거시 및 신규 제작하는 자동화 설비의 지능 제어 기술 협력 (비엠아이)
•본부내 디바이스시뮬레이션연구실 및 ㈜뉴블라: 장치간 집합통신 기술 관련 연구 협력 추진
•텐스토렌트와 연구협력 추진(NPU서버 구축 및 Metalium 오픈소스 기반 CCL 연구)
•NPU IP 업체인 Semidynamics (RISC-V Core 및 Tensor 엔진 IP 기술 보유 기업)와 LoI 체결
•㈜딥인스펙션: EDDIS 분산 학습 플랫폼 관련 기술 지원</td></tr>
</table>

<table>
<tr><th>구분</th><th>3단계 (성과)</th></tr>
<tr><td>정량</td><td>
<table>
<tr><th colspan="2">논문</th><th colspan="4">특허</th><th colspan="2">기술이전</th></tr>
<tr><th rowspan="2">SCI(건)</th><th rowspan="2">비SCI(건)</th><th colspan="2">해외(건)</th><th colspan="2">국내(건)</th><th rowspan="2">건수</th><th rowspan="2">금액(백만원)</th></tr>
<tr><th>출원</th><th>등록</th><th>출원</th><th>등록</th></tr>
<tr><td>2</td><td>9</td><td>2</td><td>0</td><td>3</td><td>0</td><td>0</td><td>0</td></tr>
</table>
•(기타) IPR확보를 위한 프로그램 등록 8건
</td></tr>
<tr><td>정성</td><td>•㈜하이퍼엑셀과 연구협력 신규 추진(HyperAccel LPU 기반 CCL 및 추론기술 연구)
•텐스토렌트와 연구협력 지속 추진(NPU 추가 구축 및 추론 기술 개발)
•㈜딥인스펙션에 EDDIS 분산 학습 플랫폼 관련 기술 지원 및 기술이전 추진중</td></tr>
</table>

가. 세부 정량적 성과

[과학적 성과]

□ 논문(국내외 전문 학술지) 게재

번호	논문명	학술지명	주저자명	호	국명	발행기관	SCIE 여부 (SCIE/비SCIE)	게재일	등록번호 (ISSN)	기여율

번호	논문명	학술지명	주저자명	호	국명	발행기관	SCIE 여부 (SCIE/비SCIE)	게재일	등록번호 (ISSN)	기여율
1	B3-Tree: Byte-Addressable Binary B-Tree for Persistent Memory	ACM ToS	차호근	16	미국	ACM	SCIE	2020.07.16	1553-3077	25%
2	InK: In-Kernel Key-Value Storage with Persistent Memory	MDPI Electronics	하민종	9	Switzerland	MDPI	SCIE	2020.11.13	2079-9292	50%
3	Failure-Atomic Byte-Addressable R-tree for Persistent Memory	IEEE TPDS	조수정	32	미국	IEEE	SCIE	2021.03.01	1045-9219	33%
4	A Case for Low-Latency Communication Layer for Distributed Operating Systems	TOIS	김상훈	E104-D	Japan	IEICE	SCIE	2021.12.01	1745-1361	100%
5	대규모 디바이스의 자율제어를 위한 EdgeCPS 기술 동향	ETRI 전자통신동향분석	전인걸	제37권 제1호	대한민국	ETRI	비SCIE	2022.02.01	1225-6455	100%
6	CCoW: Optimizing Copy-on-Write Considering the Spatial Locality in Workloads	MDPI Electronics	하민종	11	스위스	MDPI	SCIE	2022.02.03	2079-9292	70%
7	On-the-Fly Repairing of Atomicity Violations in ARINC 653 Software	Applied Sciences	Eu-teum Choi	12	스위스	MDPI	SCIE	2022.02.15	2076-3417	33%
8	CCoW: 커버리지를 활용한 지역성 기반 Copy-on-Write 기법	정보과학회 컴퓨팅의 실제 논문지 (KTCP)	하민종	Vol 28 No 3	대한민국	한국정보과학회	비SCIE	2022.03	2383-6318	100%
9	Pivotal B+tree for Byte-Addressable Persistent Memory	IEEE Access	JONGHYEON YOO	10	미국	IEEE	SCIE	2022.04.28	2169-3536	25%
10	EdgeCPS 플랫폼을 위한 지식 공유 그래프를 활용한 컴포넌트 기반 AI 응용 지원 시스템	한국정보통신학회논문지	김영주	26	대한민국	한국정보통신학회	비SCIE	2022.08.01	2234-4772	100%

번호	논문명	학술지명	주저자명	호	국명	발행기관	SCIE 여부 (SCIE/비SCIE)	게재일	등록번호 (ISSN)	기여율
11	Semi-supervised Robust Domain Adaptation for Multi-label Classification on Nonintrusive Load Monitoring	sensors	Cheong-Hwan Hur	22	스위스	MDPI	SCIE	2022.08.04	1424-8220	100%
12	EdgeCPS를 활용한 사용자 인증 및 임무 자동화를 통한 드론 배송 시스템 개선	대한임베디드공학회 논문지	조민근	18	대한민국	대한임베디드공학회	비SCIE	2023.08.31	1975-5066	100%
13	Empowering Storage Systems Research with NVMeVirt: A Comprehensive NVMe Device Emulator	ACM Transactions on Storage (ToS)	Sang-Hoon Kim	Volume: 19 Issue: 4	미국	ACM	SCIE	2023.10.31.	1553-3077	50%
14	BioEdge: Accelerating Object Detection in Bioimages with Edge-Based Distributed Inference	Electronics	Hyunho Ahn	Vol.12, No. 21	스위스	MDPI	SCIE	2023.11.05	2079-9292	100%
15	Virtual Environment Model Generation for CPS Goal Verification using Imitation Learning	Transactions on Embedded Computing Systems	신용준	23(1)	미국	ACM	SCIE	2024.01.01	1539-9087	50%
16	ScissionLite: Accelerating Distributed Deep Learning With Lightweight Data Compression for IIoT	IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS	안현호	20(10)	미국	IEEE	SCIE	2024.06.24	1551-3203	50%
17	Encrypted Traffic Decryption Tools: Comparative Performance Analysis and Improvement Guidelines	Electronics	Minwoo Jo	13	스위스	MDPI	SCIE	2024.07.22	2079-9292	100%

번호	논문명	학술지명	주저자명	호	국명	발행기관	SCIE 여부 (SCIE/비SCIE)	게재일	등록번호 (ISSN)	기여율
18	PyCAN: Open-source Python software of N-dimension Content-Addressable Network	SoftwareX open access	Yuchan Lee	28	네덜란드	SoftwareX	SCIE	2024.11.12	2352-7110	50%
19	HitShield: Enhanced Swap Eviction Scheme for Multi-Generational LRU	IEEE Access	MINWOO JO	Vol.13	미국	IEEE	SCIE	2025.04.07	2169-3536	100%
20	LLM 추론 성능 향상을 위한 Disaggregated Prefill 기술 동향	전자통신동향분석	안후영	Vol.40 No.5	한국	한국전자통신연구원	비SCIE	2025.10.01	1225-6455	100%
21	LACX: Locality-Aware Shared Data Migration in NUMA + CXL Tiered Memory	Electronics	Hayong Jeong	14	스위스	MDPI	SCIE	2025.10.29	2079-9292	50%

□ 국내 및 국제 학술회의 발표

번호	회의 명칭	발표자	발표 일시	장소	국명
1	HPCA 2020	김세권	2020.02.26	샌디에고	미국
2	ICDE 2020	오세현	2020.04.22	달라스	미국
3	ICDE 2020	임강묵	2020.04.23	달라스	미국
4	EuroSys 2020	오형준	2020.04.30	Virtual	Virtual
5	RAID 2020	Xiaoguang Wang	2020.10.14	Virtual	Virtual
6	ICDCS 2020	김상훈	2020.12.02	Virtual	Virtual
7	Middleware 2020	김동의	2020.12.10	Virtual	Virtual
8	ISET 2021	김영주	2021.05.27	제주	대한민국
9	2021년 한국컴퓨터종합학술대회	박주원	2021.06.24	제주	대한민국
10	2021년 한국컴퓨터종합학술대회	안홍조	2021.06.24	제주	대한민국
11	2021년 한국컴퓨터종합학술대회	하민종	2021.06.25	제주	대한민국
12	USENIX ATC 2021	임강묵	2021.07.15	Virtual	Virtual
13	ICTC 2021	이준희	2021.10.22	제주	대한민국
14	ICCD 2021	배한여름	2021.10.26.	Virtual	Virtual
15	CCGrid 2022	박형준	2022.05.17	타오르미나	이탈리아
16	ISET 2022	김영주	2022.05.21	제주 그라벨	대한민국
17	SIGMETRICS 2022	서영균	2022.06.06	뭄바이	인도
18	2022년 항공우주전자심포지엄	하상일	2022.07.07	부산 아난티 힐튼	대한민국
19	2022년 항공우주전자심포지엄	김창희	2022.07.07	부산 아난티 힐튼	대한민국
20	OSDI 2022	김원배	2022.07.11	칼즈배드	미국
21	ATC 2022	최상진	2022.07.12	칼즈배드	미국
22	DAC 2022	이수비	2022.07.14	샌프란시스코	미국
23	ICPP 2022	김창대	2022.08.30	보르도	프랑스

번호	회의 명칭	발표자	발표 일시	장소	국명
24	VLDB 2022	Lam-Duy Nguyen	2022.09.08	시드니	호주
25	2022년 임베디드공학회	나갑주	2022.11.02	제주 그라벨	대한민국
26	Middleware 2022	이준희	2022.11.09	퀘벡	캐나다
27	2022년 한국통신학회 추계종합학술발표회	석종수	2022.11.16	라한셀렉트 경주	대한민국
28	VRST 2022	Ayush Bhardwaj	2022.12.01	쓰쿠바시	일본
29	VRST 2022	Henry Kim	2022.12.01	쓰쿠바시	일본
30	한국소프트웨어 종합학술대회(KSC) 2022	서정인	2022.12.21	제주 라마다프라자 호텔	대한민국
31	KSC 2022	고동범	2022.12.21	제주 라마다프라자 호텔	대한민국
32	KSC 2022	용예원	2022.12.23	제주	대한민국
33	FAST 2023	우호빈	2023.02.22	산타클라라	미국
34	FAST 2023	김상훈	2023.02.23	산타클라라	미국
35	SAC 2023	하상일	2023.03.27	탈린	에스토니아
36	SAC 2023	이진훈	2023.03.28	탈린	에스토니아
37	2023년 대한전자공학회 하계학술대회	이문규	2023.06.28.	제주	대한민국
38	2023년 대한전자공학회 하계학술대회	성시훈	2023.06.28.	제주	대한민국
39	ICRA 2023	Mudassir Ibrahim Awan	2023.06.29	런던	영국
40	ATC 2023	송원욱	2023.07.11	보스턴	미국
41	ATC 2023	최상진	2023.07.12	보스턴	미국
42	VLDB 2023	김태훈	2023.08.29	밴쿠버	캐나다
43	VLDB 2023	Carl Duffy	2023.08.31	밴쿠버	캐나다
44	Euro-Par 2023	나갑주	2023.08.31	리마솔	키프로스
45	SECON 2023	장영우	2023.09.11	마드리드	스페인
46	SMC 2023	김영주	2023.10.01	호놀루루	미국
47	MEMSYS 2023	Alireza Haddadi	2023.10.05	워싱턴	미국
48	MEMSYS 2023	Muhammad Hassan	2023.10.05	워싱턴	미국
49	MobiCom 2023	Xinyue Ma	2023.10.05	마드리드	스페인
50	ISMAR 2023	Haokun Wang	2023.10.16	시드니	호주
51	CLUSTER 2023	김태훈	2023.11.01	산타페	미국
52	제1회 한국햅틱스학회 학술대회	Mudassir Ibrahim Awan	2023.11.24	부여	대한민국
53	SIGGRAPH Asia 2023	Yatharth Singhal	2023.12.12	시드니	호주
54	IPDPS 2024	안신영	2024.05.30	샌프란시스코	미국
55	2024 한국컴퓨터종합학술대회	조민우	2024.06.27	제주	대한민국
56	2024 한국컴퓨터종합학술대회	정하용	2024.06.27	제주	대한민국
57	BigdData 2024	안후영	2024.12.16	워싱턴	미국
58	BigdData 2024	안신영	2024.12.16	워싱턴	미국
59	CCGrid 2025	임아현	2025.05.20	트롬쇠(Tromsø)	노르웨이
60	ICPP 2025	장수호	2025.09.09	샌디에고	미국
61	Micro 2025	박준혁	2025.10.21	서울	대한민국
62	ICCE-Asia2025	윤정은	2025.10.30	부산	대한민국
63	NeurIPS 2025	전성호	2025.12.04	샌디에고	미국
64	Bigdata 2025	이유찬	2025.12.10	마카우(Macau)	중국
65	한국소프트웨어 종합학술대회 2025	김하영	2025.12.16	부산	대한민국
66	HiPC 2025	송빈원	2025.12.19	하이데라바드	인도

□ 기술 요약 정보

연도	기술명	요약 내용	기술 완성도	등록 번호	활용 여부	미활용사유	연구개발기관 외 활용여부	허용방식

□ 보고서 원문

연도	보고서 구분	발간일	등록 번호

□ 생명자원(생물자원, 생명정보)/화합물

번호	생명자원(생물자원, 생명정보)/화합물 명	등록/기탁 번호	등록/기탁 기관	발생 연도

[기술적 성과]

□ 지식재산권(특허, 실용신안, 의장, 디자인, 상표, 규격, 신품종, 프로그램)

번호	지식재산권 등 명칭 (건별 각각 기재)	국명	출원				등록			기여율	활용 여부
			출원인	출원일	출원 번호	등록 번호	등록인	등록일	등록 번호		
1	Distributed system for distributed lock management and method for operating the same	미국	한국전자통신연구원	2020.03.31	16/836246	11500693	한국전자통신연구원	2022.11.15	11500693	100%	
2	Method for genome sequence alignment and apparatus thereof	미국	한국전자통신연구원	2020.10.14	17/070452	12002548	한국전자통신연구원	2024.06.04	12002548	100%	
3	Method and apparatus for managing memory in memory disaggregation system	미국	한국전자통신연구원	2021.05.19	17/324883	11586549	한국전자통신연구원	2023.02.21	11586549	100%	
4	Method and apparatus for memory integrated management of cluster system	미국	한국전자통신연구원	2021.11.02	17/517284	12118394	한국전자통신연구원	2024.10.15	12118394	100%	
5	Quantum diagnostic circuit and quantum characteristic diagnostic method	미국	한국전자통신연구원	2021.12.03	17/542324	12487896	한국전자통신연구원	2025.12.02	12487896	100%	
6	Method and device for accessing memory	미국	한국전자통신연구원	2022.05.10	17/741187	12008261	한국전자통신연구원	2024.06.11	12008261	100%	
7	Apparatus and method for genome sequence alignment acceleration	미국	한국전자통신연구원	2022.06.03	17/832252					100%	
8	Method and apparatus for accelerating network transmission in memory-disaggreated environment	미국	시스기어, 한국전자통신연구원	2022.12.06	18/076051	11849005	시스기어, 한국전자통신연구원	2023.12.19	11849005	90%	
9	Method and apparatus for migrating virtual machine in memory-disaggregated environment	미국	시스기어, 한국전자통신연구원	2022.12.06	18/076163					90%	
10	Method and apparatus for verifying integrity in memory disaggragation environment	미국	시스기어, 한국전자통신연구원	2023.03.24	18/126229	12153525	시스기어, 한국전자통신연구원	2024.11.26	12153525	90%	
11	Method and apparatus for memory management in memory disaggregation environment	미국	시스기어, 한국전자통신연구원	2023.03.24	18/126232	12223186	시스기어, 한국전자통신연구원	2025.02.11	12223186	90%	

번호	지식재산권 등 명칭 (건별 각각 기재)	국명	출원				등록			기여율	활용 여부
			출원인	출원일	출원 번호	등록 번호	등록인	등록일	등록 번호		
12	Artificial intelligence application provision method and apparatus for supporting edge computing for cyber-physical systems	미국	한국전자통신연구원	2023-07-05	18/347352					100%	
13	Apparatus and method for Virtual device mirroring between Edges	미국	한국전자통신연구원	2024.02.23	18/586329					100%	
14	Efficient space management technique for GPU resources using reinforcement learning	미국	한국전자통신연구원	2024.06.03	18/732130					100%	
15		미국	한국전자통신연구원	2025.06.18.	19/242,034					30%	
16		미국	한국전자통신연구원	2025.10.29	19/372,868					100%	
17	양자 진단 회로 및 그것의 양자 특성 진단 방법	대한민국	한국전자통신연구원	2020.12.21	10-2020-0179529	10-2799607	한국전자통신연구원	2025.04.18	10-2799607	100%	
18	프로세스 포크 시의 카피-온-라이트 최적화 장치 및 방법	대한민국	아주대학교산학협력단	2020.12.30	10-2020-0187135	10-2491966	아주대학교산학협력단	2023.01.19	10-2491966	100%	
19	메모리 노드의 데이터 전송 방법	대한민국	한국전자통신연구원	2021.03.05	10-2021-0029331					100%	
20	클러스터 시스템의 메모리 통합 관리 방법 및 장치	대한민국	한국전자통신연구원	2021.03.22	10-2021-0036414	10-2591315	한국전자통신연구원	2023.10.16	10-2591315	100%	O
21	메모리 접근 방법 및 장치	대한민국	한국전자통신연구원	2021.05.11	10-2021-0060779	10-2841875	한국전자통신연구원	2025.07.30	10-2841875	100%	
22	마이크로서비스 기반의 지능형 지식 공유 방법 및 시스템	대한민국	한국전자통신연구원	2022.01.04	10-2022-0000862	10-2728027	한국전자통신연구원	2024.11.05	10-2728027	100%	O
23	시뮬레이션 실행 시간 예측 방법	대한민국	한국전자통신연구원	2022.03.02	10-2022-0026831					100%	
24	유전체 서열 정렬 가속 장치 및 방법	대한민국	한국전자통신연구원	2022.04.19	10-2022-0048190	10-2882821	한국전자통신연구원	2025.11.04	10-2882821	100%	
25	마이크로서비스 기반의 워크플로우 관리 방법 및 시스템	대한민국	한국전자통신연구원	2022.05.27	10-2022-0065351					100%	
26	동적 오프로딩을 통한 인공지능 모델 추론 방법 및 장치	대한민국	한국전자통신연구원	2022.08.25	10-2022-0106725					100%	
27	추가 모듈을 이용한 인공지능 모델 추론 방법 및 장치	대한민국	한국전자통신연구원	2022.08.29	10-2022-0108185					100%	
28	모듈 공유 관리 장치 및 방법	대한민국	한국전자통신연구원	2022.09.21	10-2022-0119470					100%	
29	자율 사물로 구성된 엣지 환경에서 가이드 장치를 통한 실시간 제어 방법 및 장치	대한민국	한국전자통신연구원	2022.10.11	10-2022-0129306					100%	
30	데이터 로컬 정보를 고려하는 컨테이너 기반 동적 워크로드 처리 시스템, 장치 및 방법	대한민국	한국전자통신연구원	2022.11.09	10-2022-0148945	10-2814160	한국전자통신연구원	2025.05.23	10-2814160	100%	

번호	지식재산권 등 명칭 (건별 각각 기재)	국명	출원				등록			기여율	활용 여부
			출원인	출원일	출원 번호	등록 번호	등록인	등록일	등록 번호		
31	메모리 분리 환경의 가상머신 이주 방법 및 장치	대한민국	시스기어, 한국전자통신연구원	2022.11.21	10-2022-0156424	10-2813871	시스기어, 한국전자통신연구원	2025.05.23	10-2813871	90%	
32	분리 메모리 환경의 네트워크 송신 가속화 방법 및 장치	대한민국	시스기어, 한국전자통신연구원	2022.11.21	10-2022-0156303	10-2813876	시스기어, 한국전자통신연구원	2025.05.23	10-2813876	90%	
33	EdgeCPS 지원을 위한 인공지능 응용 제공 방법 및 장치	대한민국	한국전자통신연구원	2022.11.29	10-2022-0162995					100%	
34	분리 메모리 환경의 무결성 검증 방법 및 장치	대한민국	시스기어, 한국전자통신연구원	2023.02.03.	10-2023-0014978					90%	
35	분리 메모리 환경의 메모리 관리 방법 및 장치	대한민국	시스기어, 한국전자통신연구원	2023.02.20	10-2023-0022075	10-2844387	시스기어, 한국전자통신연구원	2025.08.05	10-2844387	90%	
36	디스크 기반 양자 컴퓨팅 시뮬레이션 장치 및 방법	대한민국	한국전자통신연구원	2023.03.10	10-2023-0031631					100%	
37	에지 간 가상 디바이스 미러링 장치 및 방법	대한민국	한국전자통신연구원	2023-03-22	10-2023-0037049	10-2885353	한국전자통신연구원	2025-11-07	10-2885353	100%	
38	엣지 클라우드 기반의 자율 주행 보조 상황 예측 방법 및 시스템	대한민국	한국전자통신연구원	2023-11-13	10-2023-0156466					100%	
39	멀티 클러스터 기반의 워크플로우 동적 오프로딩 방법 및 이를 위한 장치	대한민국	한국전자통신연구원	2023-11-21	10-2023-0161918					100%	
40	강화학습을 이용한 GPU 자원 관리 방법 및 이를 위한 장치	대한민국	한국전자통신연구원	2023.11.27	10-2023-0166527					100%	
41	환경 반응형 자율비행체 비행경로를 동적으로 생성하는 시스템 및 방법	대한민국	한국전자통신연구원	2024.02.06	10-2024-0018438					100%	
42	메모리 접근 패턴 인식 기반 분리 메모리 관리 장치 및 방법	대한민국	한국전자통신연구원	2024.07.31	10-2024-0101522					100%	
43	분리 메모리 시스템에서 포크 지원을 위한 쓰기 시 복사 처리 장치 및 방법	대한민국	한국전자통신연구원	2024.08.12	10-2024-0107441					100%	
44		대한민국	한국전자통신연구원	2025.05.22.	10-2025-0066665					30%	
45		대한민국	한국전자통신연구원	2025.08.26.	10-2025-0119001					100%	
46		대한민국	한국전자통신연구원	2025.09.18	10-2025-0134155					100%	

ㅇ 지식재산권 활용 유형

※ 활용의 경우 현재 활용 유형에 √ 표시, 미활용의 경우 향후 활용 예정 유형에 √ 표시합니다(최대 3개 중복선택 가능).

번호	제품화	방어	전용실시	통상실시	무상실시	매매/양도	상호실시	담보대출	투자	기타
20				√						
22				√						

□ 저작권(소프트웨어, 서적 등)

번호	저작권명	창작일	저작자명	등록일	등록 번호	저작권자명	기여율
1	대용량 메모리를 활용한 유전체 분석 가속	2020.11.18	김창대	2020.11.18	C-2020-047684	한국전자통신연구원	100%
2	리눅스 커널 타임 프로파일러	2020.11.19	김창대	2020.11.19	C-2020-047690	한국전자통신연구원	100%
3	2-계층 메모리 관리기 기술	2020.12.09	김창대	2020.12.09	C-2021-005969	한국전자통신연구원	100%
4	가상머신 주소공간 관리 기술	2020.12.09	김태훈	2020.12.09	C-2021-005970	한국전자통신연구원	100%
5	원격 메모리 관리기	2020.12.09	김창대	2020.12.09	C-2021-005971	한국전자통신연구원	100%
6	가상머신 주소공간 관리 기술 v2	2021.10.01	고광원	2021.11.24	C-2021-050103	한국전자통신연구원	100%
7	2-계층 메모리 관리기 기술 v2	2021.10.11	고광원	2021.11.24	C-2021-050107	한국전자통신연구원	100%
8	가용 메모리를 활용하여 가속하는 유전체 서열 정렬 프로그램	2021.10.28	김창대	2021.11.24	C-2021-050165	한국전자통신연구원	100%
9	원격 메모리 관리기 v2	2021.10.29	고광원	2021.12.21	C-2021-056807	한국전자통신연구원	100%
10	가상자원관리플랫폼 기반의 지식공유미들웨어 실행 백엔드 인터페이스	2021.11.19	김영주	2021.12.21	C-2021-056868	한국전자통신연구원	100%
11	가상자원관리플랫폼 기반의 지식공유미들웨어 실행 웹 인터페이스	2021.11.19	김영주	2021.12.21	C-2021-056867	한국전자통신연구원	100%
12	스토리지 증강을 위한 컨테이너 기반 가상화 플랫폼	2022.11.02	나갑주	2022-12-20	C-2022-054739	한국전자통신연구원	100%
13	인공지능 정보 지식 그래프 구축기	2022.11.01	김영주	2022-12-20	C-2022-054737	한국전자통신연구원	100%
14	유저 레벨 패브릭 메모리 컴퓨팅 시스템	2023-10-30	고광원	2023-11-22	C-2023-053613	한국전자통신연구원	100%
15	유저 레벨 패브릭 메모리 연동 미멀록 라이브러리	2023-10-30	고광원	2023-11-22	C-2023-053620	한국전자통신연구원	100%
16	유저 레벨 패브릭 메모리 연동 머슬 라이브러리	2023-10-30	고광원	2023-11-22	C-2023-053621	한국전자통신연구원	100%
17	엣지 사이버-물리 시스템 디바이스리스 컴퓨팅 프레임워크를 위한 장치 제어 매니저	2023-09-30	석종수	2023-11-22	C-2023-053593	한국전자통신연구원	100%
18	지식 그래프 기반의 질의 추천기	2023-10-25	김영주	2023-11-22	C-2023-053557	한국전자통신연구원	100%

번호	저작권명	창작일	저작자명	등록일	등록 번호	저작권자명	기여율
19	프로그램 (엣지 사이버-물리 시스템 연계형 실디바이스 구동 모듈)	2023-10-20	강성주	2023-11-22	C-2023-053519	한국전자통신연구원	100%
20	프로그램 (엣지 사이버-물리 시스템 응용 서비스 워크플로우 실행 인터페이스)	2023-10-26	신용준	2023-11-22	C-2023-053552	한국전자통신연구원	100%
21	에프피지에이 기반 피씨아이이 가속기용 라이브러리	2024-10-21	김경민	2024-12-05	C-2024-050693	한국전자통신연구원	100%
22	에프피지에이 기반 피씨아이이 가속기용 디바이스 드라이버	2024-10-21	김경민	2024-12-05	C-2024-050712	한국전자통신연구원	100%
23	피씨아이이 가속기 기반 집합통신 라이브러리	2024-11-13	이수광	2024-12-12	C-2024-052522	한국전자통신연구원	100%
24	분산 딥러닝 파이썬 인터페이스 버전 1.6	2025-02-04	안신영	2025-03-11	C-2025-008907	한국전자통신연구원	100%
25	분산 딥러닝 유틸리티 라이브러리 버전 1.6	2025-02-04	안신영	2025-03-11	C-2025-008908	한국전자통신연구원	100%
26	통합 공유 메모리 버퍼 프레임워크 버전 2.1	2025-01-16	안신영	2025-03-11	C-2025-008910	한국전자통신연구원	100%
27	분산 딥러닝 코어 라이브러리 버전 1.6	2025-02-04	안신영	2025-03-11	C-2025-008911	한국전자통신연구원	100%
28	에프피지에이 기반 피씨아이이 가속기용 디바이스 드라이버 버전 1.1	2025-08-25	김경민	2025-09-18	C-2025-037841	한국전자통신연구원	100%
29	알디엠에이 헤이치에이 라이브러리 버전 1.0	2025-08-27	박준혁	2025-09-18	C-2025-037845	한국전자통신연구원	100%
30	에프피지에이 기반 피씨아이이 가속기용 피어 메모리 클라이언트 버전 1.0	2025-08-26	박준혁	2025-09-18	C-2025-037846	한국전자통신연구원	100%
31	에프피지에이 기반 피씨아이이 가속기용 라이브러리 버전 1.1	2025-08-25	김경민	2025-09-18	C-2025-037847	한국전자통신연구원	100%

□ 신기술 지정

번호	명칭	출원일	고시일	보호 기간	지정 번호

□ 기술 및 제품 인증

번호	인증 분야	인증 기관	인증 내용		인증 획득일	국가명
			인증명	인증 번호		

□ 표준화

ㅇ 국내표준

번호	인증구분[1]」	인증여부[2]」	표준명	표준인증기구명	제안주체	표준종류[3]」	제안/인증일자

* 1」 한국산업규격(KS) 표준, 단체규격 등에서 해당하는 사항을 기재
* 2」 제안 또는 인증 중 해당하는 사항을 기재
* 3」 신규 또는 개정 중 해당하는 사항을 기재

ㅇ 국제표준

번호	표준화단계구분[1]」	표준명	표준기구명[2]」	표준분과명	의장단 활동여부	표준특허 추진여부	표준개발 방식[3]」	제안자	표준화 번호	제안일자
1	NP	F.AI-AKS "Reference model of AI knowledge sharing for AI computing platform" (New): Initial draft	ITU-T	SG16	X	X	국제표준	김영주	TD179	2023.7.10

* 1」 국제표준 단계 중 신규 작업항목 제안(NP), 국제표준초안(WD), 위원회안(CD), 국제표준안(DIS), 최종국제표준안(FDIS), 국제표준(IS) 중 해당하는 사항을 기재
* 2」 국제표준화기구(ISO), 국제전기기술위원회(IEC), 공동기술위원회1(JTC1) 중 해당하는 사항을 기재
* 3」 국제표준(IS), 기술시방서(TS), 기술보고서(TR), 공개활용규격(PAS), 기타 중 해당하는 사항을 기재

[경제적 성과]

□ 시제품 제작

번호	시제품명	출시/제작일	제작 업체명	설치 장소	이용 분야	사업화 소요 기간	인증기관 (해당 시)	인증일 (해당 시)

□ 기술 실시(이전)

번호	기술 이전 유형	기술 실시 계약명	기술 실시 대상 기관	기술 실시 발생일	기술료 (해당 연도 발생액)	누적 징수 현황
1	경상계약	가변블록관리 기반 2-계층 메모리 시스템 기술	㈜시스기어	2020.11.12	100 백만원	-
2	경상계약	PCIe 기반 확장 메모리 기술	㈜테라텍	2021.12.13	20 백만원	-
3	경상계약	가변블록관리 기반 2-계층 메모리 시스템 기술 v2	㈜시스기어	2021.12.15	70 백만원	-
4	정액기술료	컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술	크래프트X	2022.02.17	30 백만원	-

번호	기술 이전 유형	기술 실시 계약명	기술 실시 대상 기관	기술 실시 발생일	기술료 (해당 연도 발생액)	누적 징수 현황
5	정액기술료	컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술	알티스트	2022.08.29	30 백만원	-

□ 사업화 투자실적

번호	추가 연구개발 투자	설비 투자	기타 투자	합계	투자 자금 성격*

* 내부 자금, 신용 대출, 담보 대출, 투자 유치, 기타 등에서 해당하는 사항을 기재

□ 사업화 현황

번호	사업화 방식1」	사업화 형태2」	지역3」	사업화명	내용	업체명	매출액		매출 발생 연도	기술 수명
							국내 (천원)	국외 (달러)		

* 1」 기술이전 또는 자기실시 중 해당하는 사항을 기재
* 2」 신제품 개발, 기존 제품 개선, 신공정 개발, 기존 공정 개선 등에서 해당하는 사항을 기재
* 3」 국내 또는 국외 중 해당하는 사항을 기재

□ 매출실적(누적)

사업화명	발생 연도	매출액		합계	산정 방법
		국내(천원)	국외(달러)		
합계					

□ 사업화 계획 및 무역 수지 개선 효과

성과					
사업화 계획	사업화 소요기간(년)				
	소요예산(천원)				
	예상 매출규모(천원)		현재까지	3년 후	5년 후
	시장 점유율	단위(%)	현재까지	3년 후	5년 후
		국내			
		국외			
	향후 관련기술, 제품을 응용한 타 모델, 제품 개발계획				
무역 수지 개선 효과(천원)	수입대체(내수)		현재	3년 후	5년 후
	수 출				

□ 고용 창출

순번	사업화명	사업화 업체	고용창출 인원(명)		합계
			yyyy년	yyyy년	
합계					

□ 고용 효과

구분			고용 효과(명)
고용 효과	개발 전	연구인력	
		생산인력	
	개발 후	연구인력	
		생산인력	

□ 비용 절감(누적)

순번	사업화명	발생연도	산정 방법	비용 절감액(천원)
합계				

□ 경제적 파급 효과

(단위: 천원/년)

구분	사업화명	수입 대체	수출 증대	매출 증대	생산성 향상	고용 창출 (인력 양성 수)	기타
해당 연도							
기대 목표							

□ 산업 지원(기술지도)

순번	내용	기간	참석 대상	장소	인원

□ 기술 무역

(단위: 천원)

번호	계약 연월	계약 기술명	계약 업체명	계약업체 국가	기 징수액	총 계약액	해당 연도 징수액	향후 예정액	수출/ 수입

[사회적 성과]

□ 법령 반영

번호	구분 (법률/시행령)	활용 구분 (제정/개정)	명 칭	해당 조항	시행일	관리 부처	제정/개정 내용

□ 정책활용 내용

번호	구분 (제안/채택)	정책명	관련 기관 (담당 부서)	활용 연도	채택 내용

□ 설계 기준/설명서(시방서)/지침/안내서에 반영

번호	구 분 (설계 기준/설명서/지침/안내서)	활용 구분 (신규/개선)	설계 기준/설명서/ 지침/안내서 명칭	반영일	반영 내용

□ 전문 연구 인력 양성

번호	분류	기준 연도	현황										
			학위별				성별		지역별				
			박사	석사	학사	기타	남	여	수도권	충청권	영남권	호남권	기타

□ 산업 기술 인력 양성

번호	프로그램명	프로그램 내용	교육 기관	교육 개최 횟수	총 교육 시간	총 교육 인원

□ 다른 국가연구개발사업에의 활용

번호	중앙행정기관명	사업명	연구개발과제명	연구책임자	연구개발비

□ 국제화 협력성과

번호	구분 (유치/파견)	기간	국가	학위	전공	내용

□ 홍보 실적

번호	홍보 유형	매체명	제목	홍보일
1	학술대회 특별세션 개최	2021 대한임베디드공학회 추계학술대회	대규모자율제어 시스템을 위한 EdgeCPS 기술 워크샵	2021.11.11
2	공영방송 홍보	KBS 대전	인간 유전체 분석 시간 단축 컴퓨팅 시스템 개발	2022.01.14
3	메타버스 전시	ETRI Conference 2022	메모리 중심 고성능 컴퓨팅 기술	2022.05.24
4	학술행사 발표	ETRI Conference 2022	패브릭메모리 - 하드웨어 경계를 초월하는 메모리 기술	2022.05.25
5	부스 전시	2022 대한민국 소프트웨어대전	메모리 중심 고성능 컴퓨팅 기술	2022.12.07.
6	기업체 기술협력 발표	㈜딥인스펙션	딥러닝 분산 학습 플랫폼 EDDIS 3.0 개발계획	2025.08.13
7	기관협력 기술교류회 발표	AICA	이종 가속기 기반 대규모 AI 병렬컴퓨팅 기술	2025.09.22

□ 포상 및 수상 실적

번호	종류	포상명	포상 내용	포상 대상	포상일	포상 기관
1	우수성과선정	국가연구개발 우수성과 100선	정보전자분야 우수	김강호 외 4명	2021.11.01	과학기술정보통신부
2	학술상수상	2021년 한국컴퓨터종합학술대회 우수발표논문상	우수발표논문상	하민종, 박주원, 김상훈	2021.07.23	한국정보과학회
3	학술상수상	ICPP'22 The Best Paper Awards	The Best Paper Awards	김창대, 고광원, 김태훈, 한대규, 서지원	2022.09.01	ICPP
4	우수논문상	2022 항공우주전자심포지엄 학술대회 우수논문상	우수논문	하상일	2022.07.07	항공우주학회

번호	종류	포상명	포상 내용	포상 대상	포상일	포상 기관
5	우수논문상	2025년 한국소프트웨어종합학술대회 우수논문상	우수논문	김하영 외 2명	2025.12.18	한국정보과학회

[인프라 성과]

□ 연구시설・장비

구축기관	연구시설/연구장비명	규격(모델명)	개발여부(○/×)	연구시설・장비 종합정보시스템* 등록여부	연구시설・장비 종합정보시스템* 등록번호	구축일자(YY.MM.DD)	구축비용(천원)	비고(설치 장소)
ETRI	멀티코어 시스템	R2308WFTZS	○	×	-	20.08.25	14,148	ETRI 7동 6층 서버실
ETRI	멀티코어 시스템	R2308WFTZS	○	×	-	20.08.25	14,148	ETRI 7동 6층 서버실
ETRI	멀티코어 시스템	R2308WFTZS	○	×	-	20.08.25	14,148	ETRI 7동 6층 서버실
ETRI	멀티코어 시스템	R2308WFTZS	○	×	-	20.08.25	14,148	ETRI 7동 6층 서버실
ETRI	멀티코어 시스템	PowerEdge R7525	○	×	-	20.08.25	14,129	ETRI 7동 6층 서버실
ETRI	멀티코어 시스템	PowerEdge R7525	○	×	-	20.08.25	14,129	ETRI 7동 6층 서버실
ETRI	멀티코어 시스템	PowerEdge R7525	○	×	-	20.08.25	14,129	ETRI 7동 6층 서버실
ETRI	멀티코어 시스템	PowerEdge R7525	○	×	-	20.08.25	14,129	ETRI 7동 6층 서버실
ETRI	High-Speed Network Switch	MQM8700-HS2F	○	○	NFEC-2020-08-264540	20.08.25	41,490	ETRI 7동 6층 서버실
ETRI	실험용 대용량 비휘발성 메모리 기반 멀티코어 시스템	ESC8000 G4	○	×	-	21.11.01	18,860	ETRI 7동 6층 서버실
ETRI	실험용 대용량 비휘발성 메모리 기반 멀티코어 시스템	ESC8000 G4	○	×	-	21.11.01	18,860	ETRI 7동 6층 서버실
ETRI	고속IO지원 비휘발성 메모리 시스템	ESC8000A-E11 2200W	○	×	-	22.06.24	18,600	ETRI 7동 6층 서버실
ETRI	고속IO지원 비휘발성 메모리 시스템	ESC8000A-E11 2200W	○	×	-	22.06.24	18,600	ETRI 7동 6층 서버실
ETRI	메모리 인터커넥트 지원 확장가능한 메모리 시스템	M50FCP2UR208	○	×	-	23.06.13	9,800	ETRI 7동 6층 서버실
ETRI	메모리 인터커넥트 지원 확장가능한 메모리 시스템	M50FCP2UR208	○	×	-	23.06.13	9,800	ETRI 7동 6층 서버실
ETRI	메모리 인터커넥트 지원 확장가능한 메모리 시스템	M50FCP2UR208	○	×	-	23.06.13	9,800	ETRI 7동 6층 서버실
ETRI	메모리 인터커넥트 지원 확장가능한 메모리 시스템	M50FCP2UR208	○	×	-	23.06.13	9,800	ETRI 7동 6층 서버실
ETRI	클러스터 자원 가상화 지원을 위한 EdgeCPS 플랫폼 서버	KTNF KH591S2	○	×	-	23.10.30	23,491	ETRI 7동 L층 L60 서버실

구축기관	연구시설/ 연구장비명	규격 (모델명)	개발여부 (○/×)	연구시설・장비 종합정보시스템* 등록여부	연구시설・장비 종합정보시스템* 등록번호	구축일자 (YY.MM.DD)	구축비용 (천원)	비고 (설치 장소)
ETRI	EdgeCPS 플랫폼 서버	SC4000-E10	○	×	-	23.07.04	23,340	ETRI 7동 L층 L60 서버실
ETRI	단일 시스템 PoC 서버	ASUS ESC8000A-E12	○	×	-	24.07.08	23,400	ETRI 7동 651-1 서버실
ETRI	단일 시스템 PoC 서버	ASUS ESC8000A-E12	○	×	-	24.07.16.	23,400	ETRI 7동 651-1 서버실
ETRI	병렬추론 개발서버	ASUS ESC8000A-E13P	○	○	NFEC-2025-10-309179	25.10.30	47,670	ETRI 7동 651-1 서버실
ETRI	병렬추론 개발서버	ASUS ESC8000A-E13P	○	○	NFEC-2025-10-309180	25.10.30	47,670	ETRI 7동 651-1 서버실

* 「과학기술기본법 시행령」 제42조제4항제2호에 따른 연구시설・장비 종합정보시스템을 의미

[그 밖의 성과] (해당 시 작성)

○
○

나. 계획하지 않은 성과 및 관련 분야 기여사항 (해당 시 작성)

○
○

2. 목표달성도

가. 공통지표

구분	기본지표							
	지표명	총사업연도	1단계		2단계			3단계
			′20년도	′21년도	′22년도	′23년도	′24년도	′25년도
과학적 성과	SCI(E) 논문	16	2	2	4	2	4	2
기술적 성과	국내특허(출원)	30	2	3	12	7	3	3
	국내특허(등록)	11	0	0	0	2	1	8
	국제특허(출원)	16	2	3	4	3	2	2
	국제특허(등록)	9	0	0	1	2	4	2
경제적 성과	기술이전(건)	5	1	2	2	0	0	0
	기술료(억원)	2.5	1.0	0.9	0.6	0	0	0

구분	심화지표							
	지표명	총사업연도	1단계		2단계			3단계
			′20년도	′21년도	′22년도	′23년도	′24년도	′25년도
과학적 성과	표준화된 IF 상위 20% SCI 논문(건)	2	0	1	0	0	1	0
기술적 성과	특허활용률 (기술이전건수/ 특허등록보유건수)	25%	-	-	200%	0	0	0
	국제표준승인표준 기고서(건)	0	0	0	0	0	0	0
	3극 특허(건)	0	0	0	0	0	0	0
경제적 성과	연구비 대비 기술료 수입(%)	1.72	7.5	3.08	2.09	0	0	0

나. 자율지표

구분	자율지표			
	지표명	연간 평균 목표	1단계	
			′20년도	′21년도
과학적 성과	(최)우수 국제학술대회*	평균 2.2건	7건	2건

구분	자율지표				
	지표명	연간 평균 목표	2단계		
			′22년도	′23년도	′24년도
과학적 성과	(최)우수 국제학술대회*	평균 1.7건	10건	16건	3건

구분	자율지표		
	지표명	연간 평균 목표	3단계
			′25년도
과학적 성과	(최)우수 국제학술대회*	평균 2건	6건

* 한국정보과학회, “2020년 한국정보과학회 우수학술대회 (개편)목록,” 2020.12.08

3. 목표 미달 시 원인분석

가. 목표 미달 원인(사유) 자체분석 내용

○ 기술이전 :
- (2단계 : 0.6억/2.2억) 본 사업은 롤링플랜으로 연구 목표가 변경되어 '22~'23년 수행된 연구 개발이 중지되었음. 따라서 기술이전을 위한 추가 기술 개발 및 기술 이전 활동이 중지됨.
- (3단계 : 0.0억/1.8억) 3단계 2차년도 이후에 기술 이전 추진 예정이었으나 3단계가 1년만에 종료(중지)되어 연구 방향의 가능성을 파악하는 PoC 수준의 기술 개발만 이루어짐.

나. 자체 보완활동

○ 본 사업의 최종년도 내용을 이어서 연구 개발하는 신규 사업('26~30)이 진행될 예정으로, 본 사업에서 PoC 수준으로 개발한 기술들을 발전시켜 기술이전 추진 예정

다. 연구개발 과정의 성실성

○ 과제 세부 연구 기술 담당자들은 각자의 연구 분야에 대한 관련 문헌 및 최신 기술 동향 분석을 적극적으로 수행하고, 연구원간의 지속적인 소통을 통해 성실하게 연구개발을 진행하였음

4 관련 분야에 대한 기여

1. 과학적·기술적·경제적·사회적 파급효과

(1) 과학적 성과

[메모리 중심]

- 메모리 중심 컴퓨팅 기반 데이터 처리 기술 연구 선도
  - 상위20% SCI 저널(Cho et al., IEEE TPDS, Mar. 2021) 1편 게재 완료
  - (최)우수국제학술대회 (USENIX ATC'21 등) 9편 발표 완료
  - SCIE 저널 3편 게재 완료
- 유전체 분석 속도 향상 (최신 기술 대비 3.34배 향상)으로 유전체 관련 연구 가속화
  - 최우수국제학술대회 (ICPP'22) 발표
  - ㈜녹십자지놈 실증사업 수행 완료 (기술성/사업성 최고 평가)
- 메모리 중심 컴퓨팅 병렬성 향상 연구 관련 국제 협업(MoU)으로 국제적 홍보
  - 미국 실리콘벨리 TidalScale 국제 MoU 체결 (Feb. 2020)
  - 26회 비대면 연구협력 회의 수행 (2020~2021)
- 메모리 확장 기반 대규모 2-계층 기반 양자 컴퓨팅 시뮬레이션 환경 (ETRI內협력) 제공
  - 기존 시뮬레이션 대비 용량 효율성 64배 달성 및 양자상태-인지 압축 기법으로 559배 성능 향상
  - 45-큐빗 양자 시뮬레이션 (8,192 Xeon Phi, 0.5 페타 메모리 사용)을 실험실 수준 장비로 시뮬레이션
- 거대 데이터 기반 대용량 메모리 컴퓨팅의 효율적인 수행환경 제공 기술 연구 추진
  - (최)우수국제학술대회 (USENIX ATC'23 등) 9편 발표 완료
  - SCIE 저널 1편 게재 완료
- 대규모 메모리 기반 그래프 분석 가속 연구: ETRI연구사업계획서 정량 목표 달성
  - 1 trillion edge 그래프 분석 소요 시간: 559초

[EdgeCPS]

- 저자원 디바이스 성능 한계 극복을 지원하는 디바이스리스 컴퓨팅 기술 및 대규모 디바이스 가상화 플랫폼 연구 선도
  - (최)우수국제학술대회 (SIGMETRICC '22 등) 8편 발표 완료
  - SCIE 저널 1편 게재
- 자원 및 연산 능력 제한으로 인해 고성능 인공지능 기술 지원이 불가능한 디바이스들의 성능 한계를 극복하는 EdgeCPS 플랫폼 원천 기술확보
  - 초대규모 디바이스들 연계를 통해 온디바이스 컴퓨팅과 서버 중심 컴퓨팅 기술의 한계를 넘는 군집/복합 인공지능 서비스를 실현
- 적용 분야별 다양한 성능 요구사항 만족을 위해 AI 반도체, 엣지 서버, 클라우드가 혼재된 환경에서 하이퍼오토메이션을 제공하는 컴퓨팅 컨티뉴엄의 새로운 패러다임 선도
  - 다양한 플랫폼을 통해 저성능 단말에 초저지연성, 고대역폭, 이종 가속기 지원 등을 안정적으로 지원하는 오케스트레이션 기술 확보
- 엣지디바이스, MEC, 클라우드가 혼재된 환경에서 컴퓨팅 컨티뉴엄의 새로운 패러다임 선도
  - EdgeCPS에 대한 관심도 제고 및 원천기술 홍보를 위한 EdgeCPS 워크샵 개최 및 다수의 국내외 학술행사 참석
  - 대규모 디바이스 제어기술 및 가속기 공유모델 연구 : ETRI연구사업계획서 정량 목표 달성

[이종가속기AI컴퓨팅]

- 이종성능GPU 기반 분산학습 기술 선도
  - (최)우수국제학술대회 (IPDPS'24, BIGDATA'24) 2편 발표 완료

- 다수의 이종 가속기간 데이터 이동을 최적화 기반 기술 확보
  • SCIE 2편 게재, (최)우수국제학술대회(BIGDATA'24) 1편 발표
- LLM 학습과 추론 과정에서 발생하는 메모리 병목을 완화하여 자원 활용 효율과 메모리 접근 지연 단축에 기여
  • SCIE 논문 2편, (최)우수국제학술대회 3편
- 지속 학습(Continual Learning)의 프롬프트 선택 과정과 업데이트 과정에서 요구되는 자원과 정확도 간의 trade-off를 분석하여 자원 효율성을 극대화함으로써, 제한된 자원 환경에서의 지속 학습 수행에 기여
  • (최)우수국제학술대회 1편
- 성능이 다른 가속기(GPU)가 장착된 다수의 노드를 통합 활용하여 분산 병렬 처리함으로써 저비용·고효율 AI 모델의 학습/추론에 기여
  • (최)우수국제학술대회 2편

(2) 기술적 성과

[메모리 중심]

- ETRI IP-R&D 프로그램 기반 특허 동향 분석 및 지식재산 확보 전략 수행
  • 메모리 분리 (Memory Disaggregation) 기술 및 유전체 분석 기술 특허 동향 분석
  • 관련 특허동향 분석 및 특허 확보 전략 수립
- 차세대 메모리 분리 시스템, 거대 데이터 지능정보 처리 및 유전체 분석 핵심 특허 확보
  • 국제 특허 (미국) 총 14건 출원하여 9건 등록 완료, 국내 특허 13건 출원하여 8건 등록 완료
- 2021년 국가 연구개발사업 우수성과 100선 선정 (정보·전자 분야)
  • 과학기술정보통신부와 한국과학기술기획평가원 주관 우수성과 선정평가위원회 평가
- CXL 디바이스 기반 메모리 확장 기술 개발: REDHAT 연구 협력

[EdgeCPS]

- 엣지 사이버물리시스템 기반 서비스는 타 산업과의 연계성이 매우 높은 기술로 5G/6G등을 활용할 국방, 로봇, 도시, 자동차 등 다양한 분야에 활용 가능
  • 산업 분야 별로 요구되는 다양한 종류 AI 디바이스들의 특성 활용을 지원함으로써 산업 분야별 지능화 및 지식의 융합을 제공하는 기회 제공
- 기존의 클라우드 산업에 적용할 수 없었던 실시간 학습 및 추론 분야에서 중요 매개체 역할을 함으로써 클라우드/엣지의 서비스 커버리지를 극대화해 부가가치를 창출
  • 데이터를 수집하는 엔드포인트 영역에서 컴퓨팅을 자체적으로 수행하는 엣지 컴퓨팅을 통해 자율주행자동차 산업, 스마트팩토리 산업 등의 발전에 기여
- 적용 분야별 다양한 성능 요구사항 만족을 위해 AI 반도체, 엣지 서버, 클라우드가 혼재된 환경에서 대규모 디바이스를 안정적으로 제어하는 기술 확보
  • 국제 특허 (미국) 3건 출원하여 등록 절차 진행 중, 국내 특허 14건 출원하여 등록 절차 진행 중
  • 국제 표준 (ITU-T SG16) 1건 제안
- 국제 연구 협력(미국 UTD)을 통한 물리적 공간 및 사물에 대한 대규모 디바이스 제어 기술 유효성 입증

[이종가속기AI컴퓨팅]

- 다수의 물리 노드들의 가속기 자원을 통합하여 하나의 생성형 모델을 분산처리 할 수 있는 SW 기반 기술 확보
- PCI 기반 가속기의 집합통신 구조연구를 통한 기반 기술 확보
  • 1건의 HW PoC 및 7건의 SW 등록
- 이종성능 GPU기반 분산학습 기술 보완 개발 및 E-패밀리기업을 통한 수자원공사 실증
  • 이종 OS(Ubuntu, Rocky) 지원 및 FP16/BF16학습 지원 기능

• 국제 특허 1건 / 국내 특허 1건 출원 완료
• SW 4건 등록(EDDIS관련 3건, USMB 1건)
- 서로 다른 노드에 장착된 가속기간 데이터를 CPU 개입없이 저지연·저비용으로 전달하는 방법에 기반한 AI 모델 학습/추론 가속 기술 확보
• 국내 특허 1건 출원 완료, 국제 특허 1건 연구원 내부 출원 절차 진행 중
- 메모리 병목을 완화하여 자원 활용 효율을 높이는 기술 확보
• 국내 특허 1건 출원 완료, 국제 특허 1건 출원 완료
- LLM 추론 서빙의 이종 가속기 통합 활용에 관한 기술 선점
• 국내 특허 2건, 국제 특허 2건 연구원 내부 출원 절차 진행 중

(3) 경제적 성과

[메모리 중심]
- 메모리 확장 시스템 관련 제품 개발 및 사업화 지원
• 기술이전 3건 (1.9억원) 계약 완료
+ ㈜시스기어: “탄력적 블록 관리 기반 2-계층 메모리 시스템 기술” 이전 (총 1.7억원)
. 국내 클라우드 기업의 국제 규모 대용량 메모리 가상머신 서비스 지원 (0.5 TB → 24 TB)
. 수자원공사 대상 메모리 관리 기술 사업화 진행 (그린IT를 위한 On-Premise 시스템 구축)
. 이전 기술 사업화 검증을 위한 실증 사업 완료 (코오롱베니트 ‘21.11.)
. 메모리 자원 활용률 증가를 통한 저탄소 데이터센터 구축 기술
. 해당 사업 부문 평가 금액 (2021.11.) 240억원 → 추가 투자 유치
+ ㈜테라텍: “PCIe 기반 확장 메모리 기술” 이전 (2천만원)
. “메모리 중심 컴퓨팅 연결망 원천 기술” 과제 협력 기술 이전 수행 (ETRI內협력)
. 차세대 메모리 인터커넥트 기반 메모리의 계층 관리 기술 제공
• 기술이전 업체의 제품 개발 및 사업화 지원, 추가 기술이전을 위한 시장 요구사항 분석
+ ㈜시스기어의 시스얼라인 애로기술지원사업 진행, TTA 인증 지원
+ 클라우드 컴퓨팅 풀 서비스 모델 지원 플랫폼 제공
• 에너지기술연구원 PoC: 대규모 과학계산 및 빅데이터 분석 비용 절감 확인
• 한컴위드 PoC: 대용량 데이터 분석을 위한 대용량 디지털포렌식 적용시험 수행
- 유전체 분석 성능 향상 (3.34배)으로 인한 유전자 분석 대중화
• ㈜녹십자지놈 PoC: 유전체 분석 기술 평가 → 기술성/사업성 최고점 획득
• 생명공학연구원 PoC: 국가 바이오 빅데이터 구축 (K-DNA) 사업 등 대용량 유전체 분석 사업 적용 논의

[EdgeCPS]
- 기술이전을 통한 컨테이너 가상화 기반의 워크플로우 실행 기술 관련 제품 개발 및 사업화 지원
• 기술이전 2건 (0.6억원) 계약 완료
+ ㈜크래프트X: “컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술” 이전 (총 0.3억원)
+ ㈜알티스트: “컨테이너 가상화 기반의 워크플로우 실행 인터페이스 기술” 이전 (총 0.3억원)
- IoT/5G 시장의 성장 방향에 부합하는 서비스 플랫폼 기술 개발을 통해 혁신적인 AI서비스 제공 및 구축·운영비절감이 가능하며, 새로운 비즈니스 모델 창출 가능
• 저성능 단말과 고성능 엣지 시스템을 사용한 시스템 자원 공유 서비스를 개발하여 관련 매출 및 세계 시장 점유 효과 확보
- 데이터를 수집하는 엔드포인트 영역에서의 컴퓨팅 성능을 개선하는 디바이스리스 컴퓨팅을 통해 자율주행자동차 산업, 스마트팩토리 산업 등의 발전에 기여
• 신규 망 투자비(CAPEX)와 기존 망 운영비 절감 효과와 산업 전반의 새로운 비즈니스 모델 창출 가능

[이종가속기AI컴퓨팅]
- 다수의 이종성능 GPU 기반 분산학습 기술 개발로 기투자 가속기들을 통합한 대규모 AI 학습이 가능하게 되

어 신규 GPU 투자비용 감소
- GPU와 NPU를 통합한 컴퓨팅이 가능하게 되면 이미 GPU를 가지고 있는 국내 기업들도 가속기의 이종성 때문에 망설이던 국산 NPU 구매가 가능해져 특정 글로벌 대기업으로의 락인 효과 감소
- AI 분산 학습 플랫폼 관련 기술에 대한 수요기업(E-패밀리기업)의 협력 요청 대응을 통한 성장 유도
- 국내외적으로 NPU 자원의 투자에 더 적극적으로 바뀌어 국내 NPU 기업들의 성장 견인 가능

(4) 사회적 성과

[메모리 중심]
- 유전자 기술 대중화: 유전체 분석 비용 절감으로 의료 환경 개선 및 신속한 감염병 진단 도구 개발
  - 영유아 조기 선별검사 확대로 높은 의료비용으로 인한 사회적 비용 절감
  - 오미크론 진단 등에 사용되는 전장 혹은 타겟 DNA 시퀀싱 성능 향상으로 인한 신속한 감염병 진단
- 데이터 주권을 위한 대응 기술 확보: 대규모 국내 클라우드 산업 활성화
  - 국내 발생 대용량 데이터 처리에 대한 해외 클라우드 의존성 제거
  - 대규모 데이터 처리를 위한 저비용 대용량 정보처리 기기 구축 기술의 개발도상국 수출
- 데이터 기반 사회 촉진: 대규모 과학계산 및 빅데이터 분석 비용 절감
  - 동적 컴퓨팅 자원 구성에 따른 데이터센터 내 자원 활용도 개선
  - 요구 메모리 기반 자원 할당 기술로 새로운 클라우드 가격 모델 제시
- 해외 슈퍼컴퓨터 기반 양자 시뮬레이션 대체하여 국내 양자 연구의 대중화
  - 실험실 수준 컴퓨팅 자원으로 대규모 양자 시뮬레이션 수행
- 국내 메모리 업체의 HW-SW 통합 솔루션 기반 국제 기술경쟁력 확보
  - 차세대 인터커넥트 (예시: CXL) 메모리 기반 HW-SW 통합 기술로 메모리 기술 경쟁력 선점
- 대용량 메모리 기반 기술을 통해서 범죄 예방등 사회 안정화 기여
  - 대용량 메모리 기반 고속의 유사 이미지 검색 알고리즘을 활용하여 CCTV등과 같은 IoT장치로부터 받은 이미지들을 빠르게 처리가능

[EdgeCPS]
- 사용자 체감 품질을 극대화할 수 있는 EdgeCPS기반의 초저지연 서비스를 구축함으로써 기존 산업의 활성화뿐만 아니라 창의적인 아이디어를 가진 국민들의 새로운 서비스 창출·창업을 통한 신산업 육성 및 신규 고용 확대가 가능
  - 초고속 차세대 이동통신과 저지연 엣지 서비스를 통해 제조업, ICT, 헬스 케어, 에너지 등 4차 산업혁명과 관련된 광범위한 분야에서 다양한 가치 창출 가능
- 데이터 이용에 관한 규제 혁신과 개인정보 보호 협치를 목적으로하는 데이터 3법과 같이 안전한 데이터 이용을 위한 사회적 규범 정립이 진행되는 기조에 이바지
  - 엣지 사이버물리시스템이라는 분야별/지역적 최적화를 통해 민감한 정보는 내부에서만 처리하는 지역성 서비스가 가능하고, 보안이 필요한 사용자에게 안전한 통신서비스 제공
  - 광범위한 디바이스 및 데이터센터에 프로세싱, 스토리지, 응용들을 분산시켜 단일 장애로 인한 네트워크 다운 방지나 안정적 서비스를 통해 정보 보호 강화

[이종가속기AI컴퓨팅]
- 저전력의 고성능 NPU들을 통합하여 운용 가능한 컴퓨팅 기술을 통해 GPU활용대비 국내 인공지능 컴퓨팅 전력 소비를 줄여 이산화탄소 배출 감소 가능
- 전국적으로 설치되어 있는 CCTV등과 같은 IoT 장치로부터의 정밀하고 고속의 영상 이미지를 획득, 생성형 AI 기반 미래 사회 안전 지원 AI 서비스를 비용효율적으로 제공하여 범죄 예방 등 사회 안정화에 기여 가능

2. 후속 과제에 도움을 줄 수 있는 연구 결과

[메모리 중심]

(1) 메모리 중심 컴퓨팅 기반 대용량 메모리 관리 기술 연구 결과
- 저탄소 데이터센터를 위한 자원관리 기술 연구
- 클라우드 기반 메타버스 인프라 구현 기술 연구
- 대용량 메모리 기반 분산 인공지능 학습/추론 기술 연구
- CXL기반 대규모 메모리 기술/ 비휘발성 메모리 기술 연구

(2) 유전체 분석 가속 기술 연구 결과
- 차세대 유전자 분석 기반 실시간 감염병 추적 기술 연구
- 영유아 조기 선별 검사 확대를 위한 저비용 유전체 분석 기술 연구

(3) 대용량 양자 시뮬레이션 기술 연구 결과
- 양자 암호화 알고리즘 연구
- 양자 암호화 기반 통신 기술 연구

[EdgeCPS]

(1) 디바이스리스 컴퓨팅 기술 연구 결과
- 이종의 무인/자율형 이동체를 활용한 도심 교통/물류 제어 기술 연구
- 상황 적응형 컴퓨팅 오프로딩을 통한 자율형 디바이스 고장 감내/대응 기술 연구
- 임베디드-엣지-클라우드 연계형 컴퓨팅 컨티뉴엄 플랫폼 연구

(2) 대규모 디바이스 가상화 기술 연구 결과
- 마이크로셀 단위의 디지털 트윈 중심형(DT-centric) 메타버스 플랫폼 연구
- 시간 동기화된 가상 자원을 통한 실시간 아크버스(AI-Robot-Cloud Verse) 플랫폼 연구
- 대규모 물리 디바이스 연계형 군집 제어 기술 연구

(3) 저지연 멀티모달 데이터 처리 기술 연구 결과
- 로컬/원격지 간 상호작용을 위한 택타일 인터넷 중계 서버 기술 연구
- 예측 불가능한 원격 환경에 대한 실시간 모델링/동기화 기술 연구
- 양단간 네트워크 상태 적응형 하이브리드 멀티모달 데이터 처리 기술 연구

[이종가속기AI컴퓨팅]

(1) 고성능 AI 분산/병렬 컴퓨팅 기술 연구 결과
- 이종성능GPU 기반 데이터 병렬 분산 학습 기술
- 이종성능GPU 기반 데이터 병렬 분산 추론 기술

(2) 이종 가속기 통합 AI 병렬 컴퓨팅 기술 연구 결과
- 이종 가속기 기반 LLM 분산/병렬 학습/추론 기술
- 데이터 처리 장치 기반의 AI 분산 컴퓨팅 구조 연구

(3) 이종 가속기간 고성능 집합통신 기술 연구 결과
- PCIe기반 가속기 PoC 기반의 집합통신 연구
- 이종 가속기 기반 LLM 분산/병렬 학습/추론 기술

(4) 분리자원 노드 추상화 기술 연구 결과
- CXL 기반 계층 메모리 플랫폼에서 커널 데이터 배치 연구

5 성과관리 및 활용계획

1. 성과관리 현황

※ 데이터 관리 계획(DMP : Data Management Plan) 참고

[메모리 중심]

<table>
<tr><th>구분</th><th colspan="4">주요내용</th></tr>
<tr><td rowspan="3">데이터
생산 및 관리</td><td colspan="4">o 다계층 메모리 관리 기술 관련 성능 분석 데이터
- 다계층 메모리 활용에 따른 벤치마크 수행에 따른 성능 데이터
- 다계층 메모리 활용시 메모리 구성에 따른 벤치마크 성능 데이터
- 다계층 메모리 활용시 추가 기능 적용시 발생하는 벤치마크 성능 데이터</td></tr>
<tr><td>데이터 유형</td><td>소스코드, 수치, 챠트 데이터</td><td>연구데이터
파일 포맷</td><td>텍스트, 스프레드시트, PDF 등</td></tr>
<tr><td colspan="4">o 데이터 수집/생산 방법
- 수행 평가를 위한 벤치마크 응용 그룹 선정
- DRAM, 원격DRAM, HPSSD등을 활용하여 다계층 메모리를 구성
- 벤치마크 응용의 수행을 통해 메모리 구성에 따른 성능 결과를 저장</td></tr>
<tr><td>데이터
저장 및 보존</td><td colspan="4">o 데이터 보존 및 백업 방안
- 연구 수행 중 자체 연구 개발 관리 서버에 연 2회 데이터 저장
- 기관 내부 연구 데이터 리포지터리에 과제 종료 후 데이터 저장
- 데이터 훼손을 대비한 버전 제어 저장 관리 시스템에 저장</td></tr>
<tr><td rowspan="3">데이터
공동활용</td><td colspan="4">o 공동활용 가능한 연구데이터 종류 (해당되는 연구데이터에 표시, 기타는 별도 기재)
①SW ②소스코드 ●수치 ●텍스트 ⑤이미지 ⑥음향 ⑦동영상 ⑧기타( )</td></tr>
<tr><td colspan="4">o 공개 범위
●원외공개 ●②원내공개 ③제목만 공개(원내, 원외) ④비공개(사유선택)

o 공개의 경우, 공유 시점
①당해 연구수행 중 ②당해 연구종료 후 바로 ●당해 연구종료 후 엠바고 지나서 (기간: 3달 후)

o 공개의 경우, 공개·공유 방법
- 기관 절차에 따른 데이터 개방 및 공유</td></tr>
<tr><td colspan="4">o 연구데이터의 공개·공유 제한 사항
- 저작자표시-비영리-동일조건 변경 허락 (BY-NC-SA)
- 연구목적시 무료 활용, 상업용 활용시 별도 계약 또는 데이터 이용 협약서를 통해 사용</td></tr>
</table>

[EdgeCPS]

<table>
<tr><th>구분</th><th colspan="4">주요내용</th></tr>
<tr><td rowspan="3">데이터
생산 및 관리</td><td colspan="4">o EdgeCPS를 위한 AI 전용 지식 공유 데이터
- 연구과제에서 진행하는 EdgeCPS 기반 지식 공유 미들웨어에서 필요한 구성요소로써 사용되는 지식 공유 그래프에서 AI 응용 서비스 지원을 위한 필수 데이터인 학습 데이터, 학습 모델, 지원 디바이스 등에 데이터와 AI 응용 수행 후에 생성된 데이터 (시스템 정보, 추론 결과, 성능 등)를 수집</td></tr>
<tr><td>데이터 유형</td><td>이미지 데이터, 학습모델 성능, 자원 상태 수치</td><td>연구데이터
파일 포맷</td><td>image file (jpg)
binary file (pt)
text file (csv, txt)</td></tr>
<tr><td colspan="4">o 데이터 수집/생산 방법
- AI 응용에 필요한 학습 데이터, 학습모델, 그리고 지원 디바이스 등을 서버 저장장치에 주기적으로 수집하여 저장하고, 이들 데이터들을 기반으로 수행한 상태 정보 및 성능 정보들을 서버 저장장치에 저장</td></tr>
<tr><td>데이터
저장 및 보존</td><td colspan="4">o 데이터 보존 및 백업 방안
- 연구 수행 중 자체 연구 개발 관리 서버에 연2회 이상 데이터 저장
- 기관 내부 연구 데이터 리포지터리에 과제 종료 후 데이터 저장
- 데이터 훼손을 대비한 재생성 문서화 진행</td></tr>
<tr><td rowspan="3">데이터
공동활용</td><td colspan="4">o 공동활용 가능한 연구데이터 종류 (해당되는 연구데이터에 표시, 기타는 별도 기재)
①SW ②소스코드 ●수치 ●텍스트 ⑤이미지 ⑥음향 ⑦동영상 ⑧기타( )</td></tr>
<tr><td colspan="4">o 공개 범위
①원외공개 ●원내공개 ③제목만 공개(원내, 원외) ④비공개(사유선택)

o 공개의 경우, 공유 시점
①당해 연구수행 중 ②당해 연구종료 후 바로 ●당해 연구종료 후 엠바고 지나서 (기간: 2년 후)

o 공개의 경우, 공개·공유 방법
* 기관정차에 따른 데이터 개발 및 공유</td></tr>
</table>

	o 연구데이터의 공개·공유 제한 사항 * 저작자표시-비영리-동일조건 변경 허락(BY-NC-SA) * 연구목적인 경우 무료로 활용하고, 상업용으로 활용할 경우 별도의 계약 또는 데이터 이용 협약서를 통해 사용

[이종가속기AI컴퓨팅]

구분	주요내용			
데이터 생산 및 관리	o 이종가속기 AI컴퓨팅 분석 데이터 - 시스템 레벨의 성능 실험 스크립트 및 결과 데이터(예, GPU/NPU 성능 수치, 메모리 사용량, 네트워크 이용률, 병렬 추론 시간 등)			
	데이터 유형	소스코드, 수치, 텍스트	연구데이터 파일 포맷	텍스트, 스프레드시트, PDF 등
	o 데이터 수집/생산 방법 - 시스템 성능 모니터링 도구를 이용하여 수집 - 직접 개발한 성능 측정 프로그램의 표준 출력을 캡처하여 수집하고 텍스트 파일로 저장함			
데이터 저장 및 보존	o 데이터 보존 및 백업 방안 - 연구 수행 중 자체 연구 개발 관리 서버에 연 2회 데이터 저장 - 원내 연구데이터 리포지터리(edr.etri.re.kr)를 통한 데이터 관리			
데이터 공동활용	o 공동활용 가능한 연구데이터 종류 (해당되는 연구데이터에 표시, 기타는 별도 기재) ①SW ②소스코드 ●수치 ●텍스트 ⑤이미지 ⑥음향 ⑦동영상 ⑧기타( )			
	o 공개 범위 ①원외공개 ●원내공개 ③제목만 공개(원내, 원외) ④비공개(사유선택) o 공개의 경우, 공유 시점 ①당해 연구수행 중 ②당해 연구종료 후 바로 ●당해 연구종료 후 엠바고 지나서 (기간: 3달 후) o 공개의 경우, 공개·공유 방법 - 기관 절차에 따른 데이터 개방 및 공유			
	o 연구데이터의 공개·공유 제한 사항 - 저작자표시-비영리-동일조건 변경 허락 (BY-NC-SA) - 연구목적시 무료 활용, 상업용 활용시 별도 계약 또는 데이터 이용 협약서를 통해 사용			

2. 성과 관리 및 활용 계획

(1) 기술적

- 기존 시스템의 한계를 극복하는 대용량 컴퓨팅의 효율적 수행 환경 제공으로 국내 클라우드 서비스 산업과 대규모 과학 계산, 금융 분석 등의 산업의 글로벌 경쟁력 증대에 활용
- EdgeCPS 플랫폼 기술을 메타버스에 적용하여 실세계에서 가상세계로 이어지는 단방향에서 가상세계에서 실세계로의 반영이 실현되어 다양한 사회적 문제해결의 기반기술로 활용
- 다양한 가속 HW 자원들의 공유 기술을 통해 최대한 많은 수의 저성능 단말이 인공지능 기술을 처리 할 수 있도록 지원하여 초대규모 디바이스들 연계를 통해 온 디바이스 컴퓨팅과 서버 중심 컴퓨팅 기술의 한계를 넘는 군집/복합 인공지능 서비스를 실현에 활용
- 초거대 생성형 AI 모델의 학습 및 추론을 위한 고효율 분산 병렬 컴퓨팅 핵심 기술로 활용
- GPU 중심의 고성능 컴퓨팅 구조를 이종 NPU 장치가 추가된 이기종 컴퓨팅 아키텍처로 전환하여 지능정보 데이터의 비용 효율적인 저전력 고속 처리를 가능하게 하는데 활용

(2) 사회문제해결

- 대규모 데이터에 대한 효율적인 분석 시스템을 제공함으로써 유전자 분석, 사회 데이터 분석 등의 비용이 낮추어 사회문제 해결에 기여
- 기존 해외 클라우드에 의존하였던 대규모 데이터 처리 및 저장의 국산화를 통해 데이터 주권 확보와 데이터 안보에 기여
- 고성능 엣지 시스템의 가속 HW 자원 공유로 국가적으로 전력 및 CO2 절감에 기여
- 거동이 불편한 사회적 약자들의 생활 체험을 간접 경험할 수 있는 서비스 개발이 가능하며, 코로나 바이러스와 같은 간염병으로 인한 온라인 학습의 확장을 통해 평등한 교육 기회 제공에 기여
- LLM 기반 미래 사회 안전 지원 AI 서비스를 비용효율적으로 제공하여 범죄 예방 등 사회 안정화에 기여

(3) 확보된 기술의 사업화 전략

- 패브릭 메모리 확장 기술을 국내 클라우드 업체 등에 이전하여 대용량·저비용 메모리 가상머신 서비스 개발 (예시: 대용량·고효율 인메모리 저장소 가상머신 서비스)
- 메모리 중심 컴퓨팅 기술 이전을 통해 대규모 디지털 포렌식 및 빅데이터 분석 솔루션 개발 (예시: 현장 투입 가능한 이동식 대용량 디지털 포렌식 플랫폼)
- 기술 이전을 통한 차세대 메모리 보드와 패브릭 메모리 확장 기술을 결합한 HW-SW 통합 솔루션 개발 (예시: PCIe 기반 메모리 보드와 패브릭 메모리 확장 기술을 결합한 저비용·대용량 메모리 시스템 통합 솔루션)
- EdgeCPS 플랫폼 기술을 관련 기업에 이전하여 엣지 사이버물리시스템 기반 디바이스리스 컴퓨팅을 위한 전용 HW 및 최적화 SW 솔루션을 개발 (예시: 디바이스리스 컴퓨팅 전용 저사양 디바이스, 디바이스트윈을 통한 저비용 학습 솔루션 등)
- 디바이스리스 컴퓨팅 기술을 국내 디바이스업체, 엣지 클라우드 업체, 솔루션 개발업체 등에 이전하여 5G/6G 연동 대규모 디바이스를 이용한 사용자 초근접 서비스 개발 (예시: 스마트공장 생산성 향상, 스마트시티 모니터링, 군집드론 제어, 나노로봇 제오 등)
- 이종가속기 AI 컴퓨팅 기술을 NPU 개발 기업, LLM 추론 서비스 제공 기업 등에 기술 이전하여 이종 가속기 기반 AI 컴퓨팅 솔루션 개발(예시 : 이종 성능 GPU 기반 데이터 병렬 분산 학습 솔루션, 이종 성능 GPU 기반 LLM 추론 서빙 솔루션, 이종가속기 기반 LLM 추론 서빙 솔루션 등)

6 연구개발비 사용실적

연구개발비 사용실적 현황

과제책임자	성명	김강호	직위	본부장
과제수행기간	전체	2020. 01. 01 - 2025. 12. 31 (6년 0개월)		

연구개발기관의 연구개발비 (단위: 천원)		정부지원 연구개발비	기관부담 연구개발비		그 외 기관 등의 지원금				합계		
					지방자치단체		기타( )				
		현금	현금	현물	현금	현물	현금	현물	현금	현물	합계
총계		14,516,000	-	19,200	-	-	-	-	14,516,000	19,200	14,535,200
1단계	1년차	1,315,000	-	19,200	-	-	-	-	1,315,000	19,200	1,334,200
	2년차	2,921,000	-	-	-	-	-	-	2,921,000	-	2,921,000
2단계	3년차	2,874,000	-	-	-	-	-	-	2,874,000	-	2,874,000
	4년차	2,953,000	-	-	-	-	-	-	2,953,000	-	2,953,000
	5년차	2,236,000	-	-	-	-	-	-	2,236,000	-	2,236,000
3단계	6년차	2,217,000	-	-	-	-	-	-	2,217,000	-	2,217,000

7 중요 연구변경 사항

구분[1]	변경 전	변경 후	변경사유 및 조치사항	변경근거[2]

* 1」 연구개발목표, 연구개발비, 연구개발기관, 연구책임자 등에서 해당 사항을 기재합니다.
* 2」 문서번호 또는 승인일자 중에서 해당 사항을 기재합니다.

[뒷면지]

주 의

1. 이 보고서는 한국전자통신연구원의 기본사업으로 수행한 개발과제 최종보고서이다.

2. 이 연구개발내용을 대외적으로 발표할 때에는 반드시 한국전자통신연구원에서 시행한 기본사업 결과임을 밝혀야 한다.