Chapter
03

클라우드 데이터센터 구축을 위한 메모리 풀링 기술 동향

윤지욱_한국전자통신연구원 책임연구원

본 고에서는 인공지능 응용 서비스의 확대와 데이터 집약적 애플리케이션의 급격한 성장에 대응하기 위한 자원 중심 데이터센터의 핵심 기술인 메모리 풀링(memory pooling) 기술 동향을 분석한다. 기존의 서버 중심 구조는 물리적 제약으로 인한 메모리 용량의 한계와 자원 파편화 문제에 직면해 있으며, 특히 클라우드 환경에서 발생하는 CPU 및 메모리 자원의 극단적인 비대칭성은 시스템 효율성을 저하시키는 주요 원인이 되고 있다. 이러한 문제를 해결하기 위해 등장한 메모리 풀링 기술은 컴퓨팅 자원과 메모리 자원을 분리하여 독립적으로 관리함으로써 자원 활용률을 극대화하고, 설비 투자 비용(CapEx) 및 운영 비용(OpEx)을 절감할 수 있는 혁신적인 해결책으로 주목받고 있다.

I. 서론

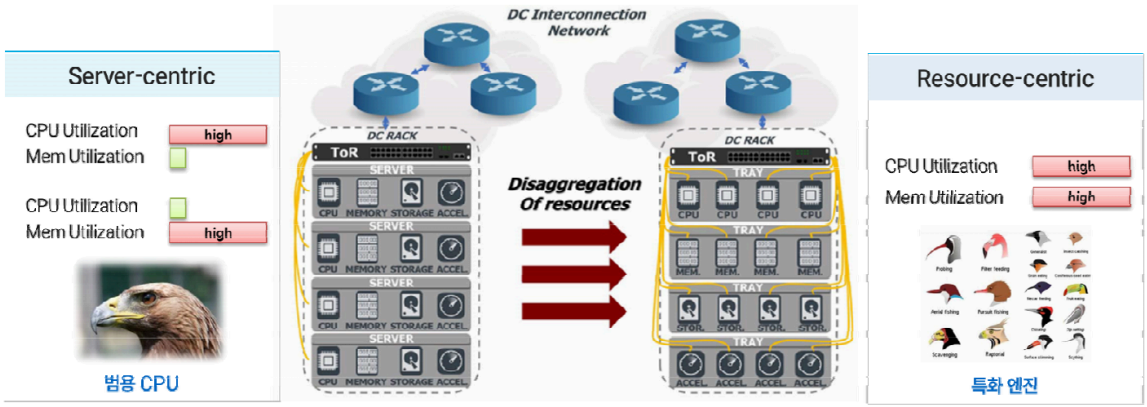
오늘날 사회 전반적으로 인공지능 응용 서비스가 확대됨에 따라서 이를 뒷받침하는 데이터센터는 거대 언어 모델(Large Language Model: LLM), 대규모 데이터 분석, 딥러닝 트레이닝 등과 같은 데이터 집약적 애플리케이션의 급격한 성장에 직면하게 되었다. 데이터 집약적 애플리케이션은 전례 없는 수준의 메모리 용량과 대역폭을 요구하고 있어 기존의 서버 중심(server centric) 데이터센터 구조는 메모리 용량 벽에 부딪히게 되었으며, 컴퓨팅 자원과 메모리 자원의 심각한 불균형으로 자원 파편화 문제가 발생하게 되었다. 이러한 자원 비효율성을 해결하기 위해 자원 중심(resource centric) 데이터센터 구조가 전 세계적으로 활발히 연구되고 있다[1]-[4]. 본 고의 I장에서는 데이터센터 진화 방향에 대해 살펴보고, II장에서는

* 본 내용은 윤지욱 책임연구원(☎ 042-860-1214, younjw@etri.re.kr)에게 문의하시기 바랍니다.
** 본 내용은 필자의 주관적인 의견이며 IITP의 공식적인 입장이 아님을 밝힙니다.
***본 고는 2026년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2019-0-00002, 광 클라우드 네트워킹 핵심기술 개발)

현재 연구되고 있는 메모리 풀링 기술별 주요 특징에 대해 살펴보겠다. III장에서는 제안 방식별 주요 특징들을 비교하였다.

1. 데이터센터 진화 방향

[그림 1]은 데이터센터 네트워크 구조가 서버 중심에서 자원 중심으로 진화하는 것을 보여준다[2]. 기존의 서버 중심 구조에서는 CPU, 메모리, 스토리지가 하나의 서버에 함께 구현된다. 반면에, 자원 중심 구조에서는 자원을 각각 분리(disaggregation)하여 CPU 풀, 메모리 풀, 스토리지 풀을 구현하고 이들을 초저지연, 초광대역 인터커넥션 기술로 연결한다. 클라우드 환경에서는 CPU 집약적 서비스와 메모리 집약적 서비스가 공존하며, 이들 간 극단적인 비대칭 특성이 발생한다. 구글 데이터센터 자료에 따르면, 서버 중심 구조로 이들 서비스를 수용할 경우, 1.5%의 메모리 집약적 애플리케이션이 데이터센터 내 전체 메모리 자원의 약 98.2%를 점유한다[5]. 현재 자원 중심 데이터센터 구조에서 가장 주목받고 있는 기술은 메모리 풀링(memory pooling) 기술이다. 실제 클라우드 환경에서의 시스템 구축 비용은 메모리에 크게 좌우된다. 연구에 따르면, Azure 서버 가격의 약 50%, Meta 랙 가격의 약 40%를 메모리가 차지한다. 메모리 풀링 기술은 데이터센터의 자원 활용률을 극대화하여 운영 비용(OpEx)과 설비 투자 비용(CapEx)을 획기적으로 절감할 수 있는 해결책으로 주목받고 있다.



〈자료〉 N. Terzenidis, et al., "High-Port and Low-Latency Optical Switches for Disaggregated Data Centers: The HipoλaosSwitch Architecture [Invited]", J. Opt. Commun. Netw, 2018. 재구성

[그림 1] 데이터센터 네트워크 진화 방향

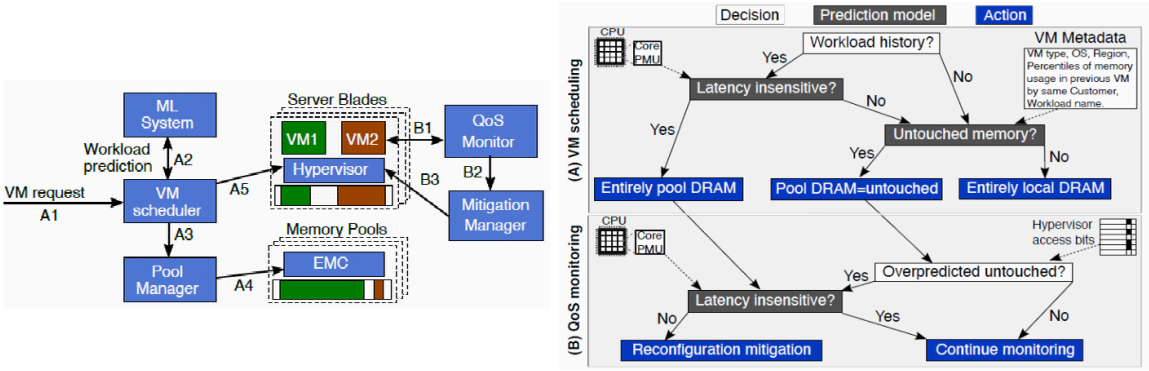
II. 메모리 풀링 시스템

기존의 메모리 풀링 시스템은 크게 서버 기반 방식과 원시 장치(raw device) 기반 방식으로 연구되었다. 서버 기반 방식은 상용 서버를 메모리 노드로 사용하고 RDMA(Remote Direct Memory Access)와 같은 기술을 통해 호스트 노드와 원격 메모리 노드를 상호 연결한다. 하드웨어 구현이 단순하다는 장점은 있으나, 메모리 노드마다 고성능 CPU와 복잡한 운영체제 스택이 필요하여 설치 및 운용 비용이 증가한다는 단점이 있다. 원시 장치 기반 방식은 컴퓨팅 능력이 없는 순수 메모리 장치만으로 메모리 노드를 구성하고 이를 네트워크에 직접 연결하는 방법이다. 이 방식은 비용 측면에서는 큰 장점이 있으나 물리 메모리 주소를 네트워크에 직접 노출하기 때문에 보안에 매우 취약하며, 다수의 호스트가 동일한 메모리 풀을 공유할 때 발생할 수 있는 충돌을 회피하기 위해 복잡하고 느린 글로벌 조정 프로토콜이 필요하다. 상기의 문제점들을 해결하기 위해서 CXL(Compute Express Link) 컨소시엄에서 표준화가 활발히 진행되고 있는 CXL 기술을 이용하여 메모리 풀링 시스템을 구축하기 위한 다양한 연구가 진행되고 있다[6]-[10]. 본 고에서는 CXL 기술을 이용하는 메모리 풀링 기술 중 최근에 발표된 것들을 중심으로 고찰해 본다.

1. Pond

Pond는 클라우드 시스템의 메모리 스트랜딩(memory stranding)과 언터치드 메모리(untouched memory) 문제를 해결하여 전체 시스템 비용을 절감하기 위해 제안된 CXL 기반의 메모리 풀링 시스템이다[6]. Pond는 기계학습(Machine Learning: ML) 기반 예측 모델을 사용하여 가상머신이 실행되기 전, 해당 워크로드가 지연시간에 민감한지와 언터치드 메모리 용량을 예측하여 언터치드 메모리와 동일한 용량을 메모리 풀에 할당한다. 메모리 풀은 CXL 메모리로 구성된다. Pond의 핵심 아이디어는 언터치드 메모리 용량을 정확히 예측할 수만 있으면, 가상머신이 실제로는 사용하지 않는 메모리이기 때문에 이를 메모리 풀에 할당하여도 성능 저하가 발생하지 않는다는 것이다.

또한, 메모리 풀을 zNUMA(zero core NUMA: 프로세싱 파워 없이 메모리만으로 구성된 NUMA)에 할당하여 가상머신이 로컬 메모리를 우선적으로 사용하게 설계하였다. [그림 2] (가)는 새로운 가상머신이 생성될 때 로컬 메모리와 풀 메모리를 어떻게 예측하고 할당하는



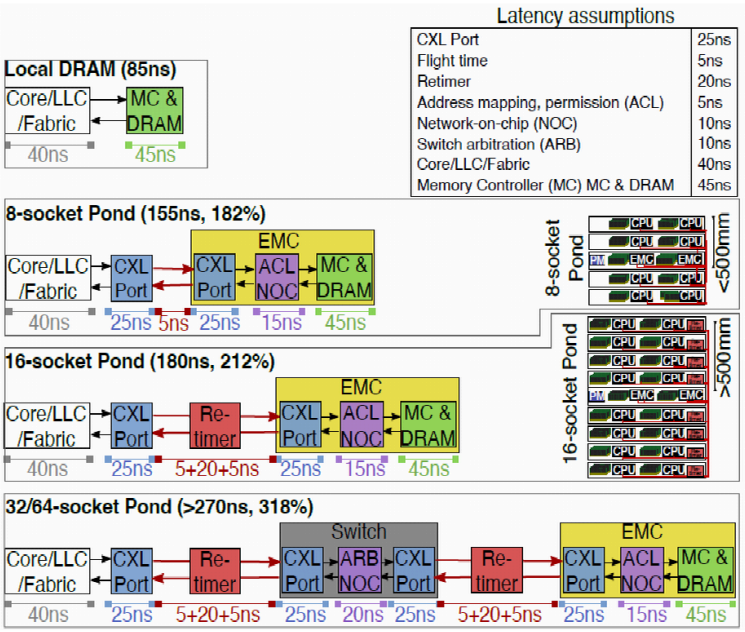
(가) 제어 평면 (나) 예측 모델

〈자료〉 Huaicheng Li et al, “Pond: CXL based memory pooling systems for cloud platforms”, ASPLOS’23, 2023.

[그림 2] POND 작업 흐름도

지에 대한 제어 평면 작업 흐름도를 보여준다. 사용자가 가상머신 생성을 요청하면(A1), 가상머신 기반 메모리 예측 모델을 통해 로컬 메모리 용량과 언터치드 메모리 용량을 예측한다(A2). 예측된 메모리 요구량을 바탕으로, 로컬 메모리와 풀 메모리 각각에 필요한 만큼의 슬라이스를 할당한다(A4, A5). 호스트 OS 관점에서 살펴보면, 풀 매니저는 풀 메모리 주소 범위를 hot-pluggable but not enabled로 설정하고 Add_capacity/Release_capacity 명령어를 사용하여 1GB 슬라이스 단위로 추가/삭제한다. 이후, QoS(Quality of Service) 감시를 통해 필요시 로컬 메모리 용량을 변경한다(B1~B3). [그림 2] (나)는 가상머신 기반 예측 모델을 이용하여 지연 민감도에 따라 워크로드를 로컬 메모리 또는 풀 메모리에 할당하는 흐름도를 보여준다. 예측은 워크로드의 과거 이력, 가상머신 종류 등과 같은 메타데이터를 바탕으로 수행된다. 지연에 민감한 워크로드는 로컬 메모리에 할당하고, 지연에 둔감한 워크로드 중 가상머신 메타데이터를 기반으로 언터치드 메모리로 예측된 것만 풀 메모리에 할당한다. [그림 3]은 Pond 규모별 하드웨어 연결 구조와 이에 따라 추가되는 세부적인 지연시간을 보여준다. 로컬 DRAM은 NUMA 로컬 메모리 접근시간으로 본 논문에서 성능 비교의 기준점이 된다. Pond는 지연시간 최소화를 위해 16개 이하의 작은 규모로 구성하는 것을 제안한다.

현재 상용 기술로 구현 가능한 EMC(External Memory Controller)는 AMD사의 Genoa를 기준으로 최대 128개의 PCIe 5.0 lane과 12개의 DDR5 채널을 가지기 때문이다. Pond를 구성하는 모든 호스트 CPU가 8개의 CXL 링크(PCIe lane)를 사용할 경우, 최대 16개의 호스트 CPU를 별도의 스위치 없이 EMC만을 이용하여 최대 12개의 DDR5와 연결할 수



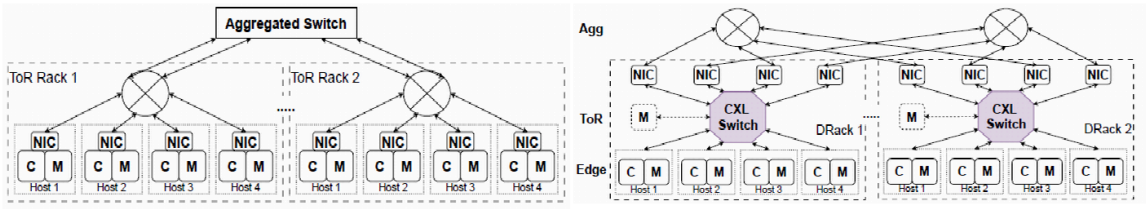
〈자료〉 Huaicheng Li et al., "Pond: CXL based memory pooling systems for cloud platforms", ASPLOS'23, 2023.

[그림 3] Pond 규모별 하드웨어 연결 및 지연시간

있다. CXL 스위치를 사용하면 최대 64개까지 확장이 가능하며, EMC만을 사용하는 구조 대비 100ns의 지연시간이 추가된다. 16 소켓 Pond는 로컬 메모리만을 사용하는 경우와 비교하여 5% 이하의 성능 감소로 약 7%의 DRAM 메모리를 절감하였고 전체 클라우드 서버 가격의 약 3.5%를 절감한다.

2. DRack

DRack은 인공지능 서비스 활성화로 폭증하고 있는 랙 간(inter-rack) 트래픽을 효과적으로 제공하기 위한 CXL 기반 분리형(disaggregation) 랙 구조를 제안한다[7]. 랙 내부의 모든 NIC과 로컬 메모리를 호스트로부터 분리하여 랙 수준의 NIC 풀과 메모리 풀을 형성하여 랙 간 통신에 활용한다. 대역폭 확장을 위한 별도의 하드웨어 추가나 재구성의 복잡도 없이도 데이터 집약적 애플리케이션 환경에서 랙 간 트래픽을 효과적으로 처리할 수 있다는 장점이 있다. 현재의 서버 중심 구조는 랙 내부 집중도가 큰, 즉 거의 모든 트래픽이 랙 내부에서만 처리되는 애플리케이션 환경에서는 높은 성능(low latency, high throughput)을 보인다.

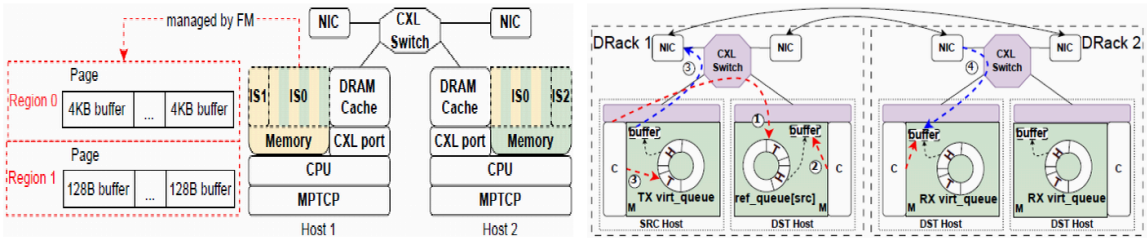


(가) 랙 중심 구조 (나) DRack 구조

〈자료〉 Xu Zhang et al., "DRack: A CXL disaggregated rack architecture to boost inter-rack communication", USENIX'25, 2025.

[그림 4] 데이터센터 구조

그러나 랙 간 통신이 빈번하게 발생하는 LLM과 같은 데이터 집약적 애플리케이션에서는 호스트가 처리하는 데이터 용량이 NIC 용량을 초과하게 되어 성능 저하가 발생한다. [그림 4] (가) 랙 중심 구조에서 랙은 서비스 기반으로 구성된다. 즉, 서비스 종류에 따라서 캐시 서버 랙, 웹 서버 랙, 연산 서버 랙 등으로 분리되어 구성된다. 따라서 동시에 여러 종류의 서버와 다양한 자원을 사용하는 LLM과 같은 데이터 집약적 애플리케이션이 수행되면 랙 간 트래픽 용량이 급격하게 증가하여 호스트 카드에 있는 NIC 대역폭을 초과하게 된다. 또한, 로컬 자원과 원격 자원을 동시에 사용하는 빈도수 증가로 자원 단편화가 발생하여 급격한 성능 저하로 이어진다. 이를 해결하기 위해 [그림 4] (나) DRack에서는 랙 안에 있는 모든 NIC와 메모리를 호스트로부터 분리해서 NIC 풀과 메모리 풀을 생성하고 이들을 CXL 프로토콜을 사용하여 호스트와 연결한다. NIC 풀이 ToR 계층의 상위 링크 역할을 하도록 계층을 재배치하여 랙 간 경로를 크게 확장하였다. 이를 위해 기존의 NIC는 그대로 사용하면서 호스트 메모리를 풀 용도로 사용한다. 자원 단편화를 억제하고 현재 적용된 호스트



(가) 메모리 할당 및 접근 (나) 랙 내 및 랙 간 통신

〈자료〉 Xu Zhang et al., "DRack: A CXL disaggregated rack architecture to boost inter-rack communication", USENIX'25, 2025.

[그림 5] DRack 개념도

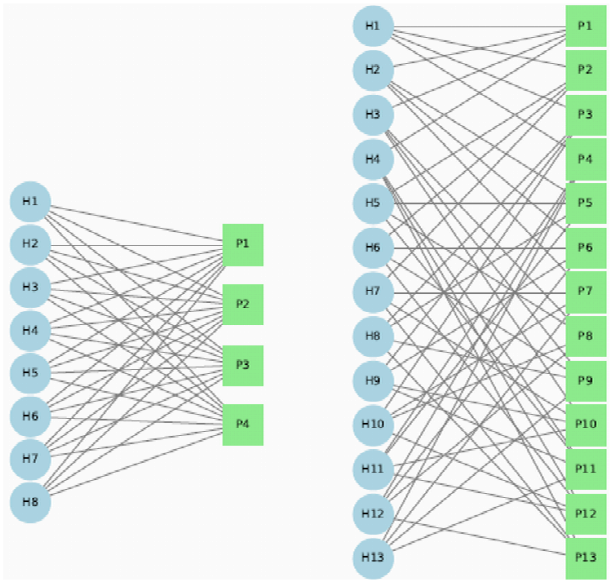
단에서의 스케줄링과의 충돌을 회피하기 위해 NIC 풀은 호스트별 스케줄링 대신 vNIC (Virtual NIC) 단위로 스케줄링을 수행한다. [그림 5] (가)는 DRack에서 메모리를 할당하고 메모리 풀에 접근하는 방식을 설명하기 위한 개념도이다.

호스트 CPU가 NIC 풀과 메모리 풀에 직접 접근하기 위해서 CPU에 CXL 포트를 구현하였다. CXL 포트에 붙어있는 DRAM 캐시는 CXL 스위치에서 발생하는 지연시간을 최소화하기 위해 추가된 것으로 로컬 메모리와는 별도의 DRAM으로 구성된다. 랙에 있는 모든 호스트는 CXL 포트(CXL.mem)를 통해 다른 호스트 메모리를 직접 ld/st 한다. 메타데이터 정보를 갖는 Reference(IS1, IS2)만 인접 호스트에 전달하고 실제 데이터는 메모리 풀로 사용하는 IS0에 저장된다. [그림 5] (나)에서 랙 내부 통신은 CXL 스위치만을 사용하여 연결되며, 랙 간 통신은 CXL 스위치와 NIC 풀을 통해 연결된다. 그림에 표시된 붉은색 점선은 메모리 ld/st 동작을 나타내며, 파란색 점선은 DMA 읽기/쓰기 동작을 보여준다. 랙 간 통신은 NIC 풀에 있는 다수의 NIC를 사용하여 병목 현상을 회피한다. DRack은 ToR 중심 구조 대비 통신 단계를 평균 37.3% 단축하여 Redis와 같이 지연시간에 민감한 서비스에서 꼬리 지연시간을 62.2% 감소시킨다. 반면에, 다수의 NIC를 사용하기 위해 MPTCP(Multi-Path TCP)를 사용하기 때문에 NIC 수가 증가하면 CPU 인터럽트 및 커널 처리 부하가 급증하는 문제가 있다.

3. Octopus

Octopus는 CXL 프로토콜 기반의 메모리 풀링 시스템을 실제 데이터센터에 구현하는 경우, 가격과 규모에 대한 현실적인 네트워크 구조를 제안한다[8]. 현재 CXL 기반 메모리 풀링은 CXL 스위치 또는 큰 포트 수를 가지는 MHD(Multi-Headed Device)를 사용하는 대칭형 구조로 과도한 구축 비용과 최소 500ns 이상의 추가 지연시간이 발생하여 메모리 계층으로 사용하기에 부적합하다는 문제점이 있다. 이를 해결하기 위해 CXL 스위치를 사용하는 대신 최대한 작은 수의 포트를 가지는 MHD를 비대칭 구조로 연결함으로써 작은 규모에서 가격 경쟁력 있는 CXL 메모리 풀링 시스템을 구현하였다. 가격과 지연시간을 모두 고려하면 실제적인 구현은 4개 또는 8개의 CXL 포트를 갖는 MHD를 사용하는 것이 합리적이다. 그러나 기존의 대칭형 토폴로지에서 호스트 수는 이와 연결되는 MHD 포트 수와 동일하다. 따라서 MHD의 포트 수가 적어지면 호스트 수 또한 줄어들어 큰 규모의 Pod(Performance

optimized datacenter)를 구현할 수 없다는 문제점이 발생한다. 이러한 문제점을 해결하기 위해 MHD를 비대칭 토폴로지로 연결하여 호스트가 서로 다른 MHD 집합에 연결되는 구조를 제안한다. [그림 6]은 대칭형 Pod 구조와 Octopus에서 제안하고 있는 비대칭형 Pod 구조를 보여준다. 대칭형 토폴로지에서는 모든 호스트(H)는 모든 메모리 풀(P)을 같은 비율로 공유한다. 즉, 8개의 호스트(H1~H8)는 4개의 메모리 풀(P1~P4)과 모두 연결된다. 현재 상용 기술로 구현 가능한 MHD의 최대 8-lane CXL 포트 수는 16개이며, 이에 따라 최대 호스트 수도 16개로 제한된다. 호스트 수를 16개 이상으로 증가시키기 위해서는 별도의 CXL 스위치를 추가해야 한다. 반면에, 비대칭형 토폴로지에서 호스트는 메모리 풀 전체가 아닌 일부만을 공유한다. 호스트가 사용할 수 있는 메모리 풀 용량이 작아지는 단점은 있으나 성능과 가격 측면에서 현실적인 대안을 제시한다. [그림 6] (나)는 비대칭형 토폴로지를 사용하여 4개의 CXL 포트 수를 갖는 MHD를 사용하여 총 13개의 호스트를 연결할 수 있음을 보여준다. 호스트의 CXL 포트 수, MHD 포트 수에 따른 연결 가능한 최대 호스트 수와 MHD 수는 (식 1)로 구해진다.



(가) 대칭형 (나) 비대칭형

〈자료〉 Daniel S. Berger et al., "Octopus: Scalble low-cost CXL memory pooling", Azure Research Preprint 2025.

[그림 6] Pod 구조

$$H = 1 + X \times (N-1),\ M = \frac{H \times X}{N} \qquad \text{(식 1)}$$

H: 호스트 수, M: MHD 수, X: Host port 수, N: MHD port 수

[표 1]은 MHD 포트 수와 호스트 포트 수별 최대로 구성 가능한 CXL 기반 Pod 크기와 가격을 대칭형 토폴로지 구조와 Octopus 구조에서 비교한 것이다. 7번과 8번을 비교해 보면, 8개의 포트 수를 갖는 호스트와 MHD를 사용하여 동일한 비용으로 Pod를 구축하는 경우, Octopus 방식이 7배 이상 큰 Pod를 구성할 수 있다.

[표 1] MHD 및 호스트 포트 크기별 CXL Pod 설계

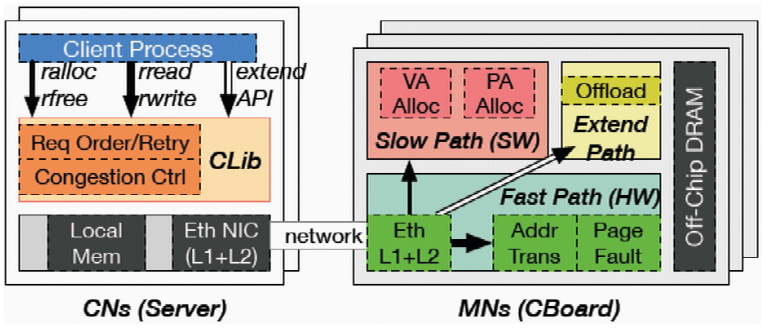
순번	MHD 포트	토폴로지	호스트 포트	Pod 크기	MHD 수량	Pod 가격(달러/호스트)
1	2	대칭형	2	2	2	300
2	2	Octopus	2	3	3	300
3	2	Octopus	4	5	10	600
4	4	대칭형	4	4	4	670
5	4	Octopus	4	13	13	670
6	4	Octopus	8	25	50	1,340
7	8	대칭형	8	8	8	1,600
8	8	Octopus	8	57	57	1,600

〈자료〉 Daniel S. Berger et al., "Octopus: Scalble low-cost CXL memory pooling", Azure Research Preprint 2025. 재구성

4. Clio

Clio는 최소한의 프로세싱 능력만을 가지는 하드웨어 기반의 지능형 메모리 노드(MN)와 컴퓨팅 노드(CN)용 사용자 공간 라이브러리인 CLib으로 구성된 통합 시스템을 제안한다[9]. Clio의 핵심 철학은 메모리 노드의 하드웨어에서 최대한 상태(State)를 제거하여 확장성과 성능을 극대화하는 것이다.

[그림 7]은 컴퓨팅 노드와 메모리 노드로 구성된 Clio 구조를 보여준다. 컴퓨팅 노드의 CLib은 원격 메모리 관리를 위해 애플리케이션에 비 투명(non- transparent) API를 제공한다. 또한, 요청 ID 관리, 재전송 버퍼 유지, 혼잡 제어와 같은 네트워크 전송 로직과 상태 관리를 모두 컴퓨팅 노드로 이동시켜 메모리 노드를 전송계층으로부터 자유롭게 만든다.



〈자료〉 Zhiyuan Guo et al., "A hardware-software co-designed disaggregated memory system", ASPLOS'22, 2022.

[그림 7] Clio 구조도

메모리 노드는 하드웨어 데이터 평면(Fast Path: FP), 소프트웨어 제어 평면(Slow Path: SP) 및 빠른 연산을 위한 하드웨어 가속장치(Extend Path: EP)로 구성된다. FP는 100Gbps의 대역폭과 마이크로초 단위의 지연시간을 보장하기 위해 하드웨어 로직(ASIC/FPGA)으로 구현되어 주소 변환, 권한 검사, 데이터 읽기/쓰기 등 실시간 데이터 요청을 처리하며 메타데이터를 저장한다. FP는 확정적 성능 보장을 위해 파이프라인 구조로 설계된다. SP는 저전력 ARM 기반 SoC에서 실행되는 소프트웨어로 구성된다. 메모리 할당, 해제, 가상 주소 생성과 같이 지연에는 민감하지 않지만, 복잡한 로직이 필요한 메타데이터 연산을 수행한다. EP는 애플리케이션에 특화된 연산을 메모리 노드에서 직접 수행할 수 있도록 해주는 프레임워크이다. 포인터 추적(pointer chasing), 데이터 필터링과 같은 작업을 처리함으로써 네트워크 트래픽 용량을 줄이고 전체 시스템 성능을 향상시킨다. [표 2]는 Clio 구성 요소별 주요 기능을 보여준다.

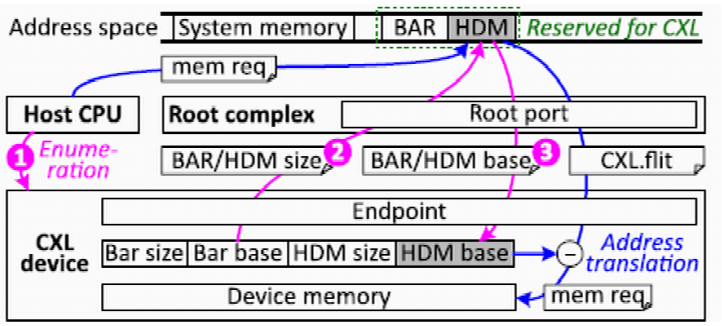
[표 2] Clio 구성 요소별 주요 기능

노드	구성 요소	주요 기능	구현 방식
컴퓨팅 노드	CLib	요청 순서 보장, 재전송, 혼잡 제어, 애플리케이션 API 제공	사용자 소프트웨어
메모리 노드	Slow Path	물리/가상 주소 할당, 메타데이터 관리, ARM SoC 제어	SoC 기반 소프트웨어
메모리 노드	Fast Path	가상 주소를 물리 주소로 변환, 권한 검사, 페이지 폴트 처리	전용 하드웨어 로직
메모리 노드	Extended Path	KV 저장, Radix tree 탐색 등 계산 오프로딩	하드웨어/소프트웨어 혼합

〈자료〉 Daniel S. Berger et al., "Octopus: Scalble low-cost CXL memory pooling", Azure Research Preprint 2025. 재구성

5. DirectCXL

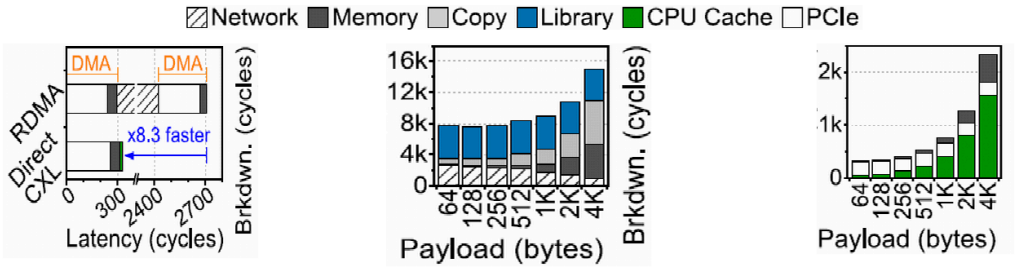
DirectCXL은 CXL 프로토콜을 사용하여 호스트와 원격 메모리를 직접 연결하여 데이터센터의 메모리 자원을 효율적으로 풀링하는 방식을 제안한다[10]. 기존의 메모리 풀링 기술이 주로 RDMA를 사용하거나 소프트웨어 기반 방식을 사용함으로써 발생하는 빈번한 데이터 복사, 페이지 폴트, 문맥 교환 오버헤드와 같은 다양한 문제점을 해결하기 위해 제안되었다. DirectCXL은 상용 FPGA 기반으로 CXL 2.0 시스템을 구축하여 CXL 디바이스를 메모리 풀링 용도로 사용할 수 있음을 실험적으로 검증하였다. RDMA에서 발생하는 비효율적인 데이터 복사 없이 원격 메모리를 호스트 시스템 메모리 공간에 직접 매핑하여 별도의 소프트웨어 개입 없이 ld/st 명령어로 접근할 수 있게 해준다. 또한, CXL의 캐시 일관성 기능을



〈자료〉 Donghyun Gouk et al., "Memory pooling with CXL", IEEE Micro, 2023.

[그림 8] DirectCXL 연결 방식

활용하여 하드웨어 수준의 낮은 지연시간으로 메모리 자원을 관리하고 cxl-namespace를 통해 원격 메모리를 mmap 방식으로 애플리케이션에 할당하는 전용 소프트웨어 스택을 제공한다. [그림 8]은 DirectCXL이 호스트 CPU와 원격 메모리를 연결하는 방식을 보여준다. 호스트 커널 드라이버는 CXL RP(Root Port)에 연결된 CXL 장치들을 탐색하고, PCIe 트랜잭션 계층을 통해 장치의 BAR(Base Address Register) 정보와 내부 메모리인 HDM(Host managed Device Memory) 정보를 확인한다. 커널 드라이버는 확인된 BAR, HDM 정보를 호스트의 시스템 메모리 공간에 매핑한 후 시스템 메모리 매핑 정보를 해당하는 CXL 장치에 알려준다. 이후, 호스트가 HDM을 사용할 필요가 발생하면 호스트는 ld/st 명령을 통해 시스템 메모리에 매핑된 HDM 주소에 접근한다(파란색). 이 요청은 해당 RP에서 CXL flit으로 변환되어 CXL 메모리에 전달된다(파란색). CXL 장치의 CXL 컨트롤러는 수신된 HDM 주소에서 시작 주소(base address)를 빼는 작업을 통해 실제 물리적 주소를 찾아낸다. [그림 9]는



(가) 하드웨어 (나) RDMA (다) DirectCXL

〈자료〉 Donghyun Gouk et al., "Memory pooling with CXL", IEEE Micro, 2023.

[그림 9] 지연시간 분석

RDMA와 DirectCXL의 지연시간을 상세히 비교 분석해 보여준다. [그림 9] (가)는 순수 하드웨어에서만의 지연시간을 나타낸다. 64바이트 읽기 기준으로 DrectCXL이 RDMA보다 약 8.3배 빠름을 알 수 있다. [그림 9] (나)와 (다)는 각각 RDMA와 DirectCXL에서 지연시간을 세부 분석한 그래프이다. RDMA는 데이터 복사와 소프트웨어 라이브러리 오버헤드가 큰 비중을 차지하는 반면, DirectCXL은 CPU 캐시와 PCIe가 주요 원인이며 소프트웨어 및 복사 오버헤드가 전혀 없음을 알 수 있다.

III. 제안 방식별 성능 비교 분석

[표 3]은 본 고에서 고찰해 본 CXL 기반 메모리 풀링 기술별 주요 특징을 보여준다. 현재, 대부분의 메모리 풀링 기술은 CXL.mem 프로토콜만을 사용하고 있다. 이는 캐시 일관성과 같은 복잡한 문제를 회피하면서 기존의 NUMA 노드와 쉽게 연동하기 위해서다. 향후, 메모리 풀링 기술은 MH-LD(Multiheaded Logical Device)를 지원하는 상용 버전의 CXL3.0 스위치가 출시되는 시점에 맞춰, 메모리 공유(sharing) 기술로 발전될 것으로 예측된다.

[표 3] 제안 방식별 주요 특징

구분	Pond	DRack	Octopus	Clio	DirectCXL
규모	소규모 (Max.64)	소규모	중규모 (57개 Pod)	대규모	대규모
프로토콜	CXL.mem	CXL.mem	CXL.mem	-	CXL.mem
NUMA	사용	사용	사용	-	사용
메모리 풀	CXL 메모리	로컬 DRAM	CXL 메모리	DRAM(CBoard)	CXL 메모리
스위치	EMC, CXL 스위치	CXL 스위치	MHD only	ToR 스위치	CXL 스위치
주요 특징	- ML 예측 모델을 이용하여 언터치드 메모리를 예측하여 zNUMA에 할당 - DRAM 비용 7% 절감	- NIC와 메모리를 분리하여 랙 간 트래픽 병목현상을 해결	- 저비용의 MHD만을 사용하여 중규모의 메모리 풀 구성	- HW 데이터 평면과 SW 제어 평면을 분리하여 확장성과 성능 극대화	- FPGA로 CXL 시스템 구현하여 원격 메모리를 호스트 메모리 공간에 직접 매핑 - RDMA 대비 7배 빠름

〈자료〉 한국전자통신연구원 자체 작성

IV. 결론

본 고에서는 인공지능 응용 서비스 확산으로 초거대 규모(hyperscale)의 클라우드 중심 구조로 변화하고 있는 데이터센터의 자원 활용도를 높이고 클라우드 시스템 구축 가격을 획기적으로 절감하기 위한 메모리 풀링 기술에 대해 고찰해 보았다. 데이터센터 내 워크로드는 서버 가상화와 SDN, NFV 사용 확대, AI/ML 분산 학습 트래픽 증가로 동적 · 대규모 · 분산 특성으로 급격히 변화하고 있다. 메모리 풀링 기술은 CXL 표준과 함께 사용될 경우, 기존의 NUMA 환경에서 큰 장점이 있다는 것이 실험적으로 증명되고 있다. 현재의 메모리 풀링 기술은 단일 랙에서 소수의 서버 간에 CXL 메모리를 공유하는 정도로만 적용되고 있으나, 가까운 미래에 실제 데이터센터 Pod 규모에 적용될 것으로 예측된다.

참고문헌

[1] Christian Pinto et al., “ThymesisFlow: A Software-Defined, HW/SW co-Designed Interconnect Stack for Rack-Scale Memory Disaggregation”, In Proc. IEEE/ACM Int. Symp. on Microarchitecture (MICRO), 2020, pp.868-880.

[2] N. Terzenidis et al., “High-Port and Low-Latency Optical Switches for Disaggregated Data Centers: The HipoλaosSwitch Architecture [Invited]”, J. Opt. Commun. Netw., Vol.10, No.7, 2018, pp.A85-A93.

[3] Ryohei Urata et al., “Mission Apollo: Landing Optical Circuit Switching at Datacenter Scale”, 2022, pp.1-13.

[4] L. Poutievski et al., “Jupiter Evolving: Transforming Google’s Datacenter Network via Optical Circuit Switches and Software-Defined Networking”, In Proc. of ACM SIGCOMM’22, Aug. 2022.

[5] Muhammad Tirmazi et al, “Borg: the Next Generation”, In Proc. of EuroSys’20, Apr. 2020,

[6] Huaicheng Li er al., “Pond: CXL-Based Memory Pooling Systems for Cloud Platforms”, In Proc. of ACM ASPLOS’23, Mar. 2023, pp.573-587.

[7] Xu Zhang et al., “DRack: A CXL-Disaggregated Rack Architecture to Boost Inter-Rack Communication”, In Proc. of 2025 USENIX ATC, Jul. 2025, pp.1261-1277

[8] Daniel S. Berger et al., “Octopus: Scalable Low-Cost CXL Memory Pooling”, 2025.

[9] Zhiyuan Guo et al., “Clio: A Hardware-Software Co-Designed Disaggregated Memory System”, In Proc. of ACM APSLOS’22, Mar. 2022, pp.417-433.

[10] Donghyun Gouk et al., “Direct Access, High-Performance Memory Disaggregation with DirectCXL”, In Proc. of 2022 USENIX ATC, Jul. 2022, pp.287-294.