오픈스택 및 오픈소스 영향력 [1)]

백동명* 강동재** 정성인*** 이범철****

현재의 정보통신산업(ICT)를 이끄는 동인이 MBCS(Mobile, Big Data, Cloud, Social Network)라고 한다. 모바일이 주는 자유로움, 빅데이터란 새로운 서비스, ICT 자원이 집중되는 클라우드, 사람과 사람의 정보를 실시간으로 연결해주는 소셜 네트워크를 말한다. 그 중 클라우드 플랫폼으로 유명한 오프스택을 소개하고자 한다. 코어 프로젝트 9개를 요약하면서 기술 동향을 요약하고, 그에 못지 않는 중요성을 가진 커뮤니티를 오픈소스 활동의 관점으로 기술한다. 또한 오픈스택에서 사용하는 파이썬 언어를 소개하여 Java 중심 시각을 벗어나고자 했다. 끝으로 발전 방향을 다양한 시각에서 기술하였다.

목 차

** ETRI 스마트노드플랫폼연구실/선임연구원
** ETRI 고성능컴퓨팅 SW 연구실/선임연구원
*** ETRI 고성능컴퓨팅 SW 연구실/책임연구원
**** ETRI 스마트노드플랫폼연구실/실장

Ⅰ. 오픈스택 기술 동향

오픈스택은 아파치 라이센스를 가진 IaaS 서비스를 제공하는 클라우드 OS 이다. 2010년에 Rackspace 와 NASA 가 연합하여 오픈소스 클라우드 SW 를 시작하였다. 이후 2012 년에 설립된 오픈스택 재단에 의해서 현재 SW 와 커뮤니티를 운영한다. Rackspace, HP, IBM, Dell, RedHat, Canonical, Cisco, VMware 등 200 개 이상의 회사들이 참여하고 있다. 주요 기능인 데이터센터 내의 컴퓨팅 자원, 스토리지 자원, 네트워킹 자원 풀 관리를 대쉬보드를 통해서 한다. 6 개월마다 릴리즈를 발표할 정도의 빠른 성장을

1) 본 연구는 미래창조과학부가 지원한 2013 년 정보통신·방송(ICT) 연구개발사업(SNP, CSB)의 연구결과로 수행되었음

하는데 알파벳순서로 이름을 정한다. Greezly 버전(2013 년 4 월) 후, 현재의 Havana (2013 년 10 월)가 출시되었고, Icehouse(2014 년 4 월)가 예정되어 있다.

이번 Havana 버전에는 템플레이트 기반의 오케스트레이션을 다루는 Heat, 빌링에 있어서 필수적인 메터링을 다루는 Ceilometer 가 정식 프로젝트로 추가되었다. HEAT, Ceilometer 가 추가되어 9 개의 코어 프로젝트(코드명: nova, swift, cinder, neutron, horizon, heat, ceilometer, glance, keystone)가 있다. 크게 컴퓨팅 리소스, 스토리지 리소스, 네트워크 리소스, 대쉬보드, 공통 서비스(Shared Services)로 분류한다[1]. 그 외에도 Icehouse 버전에 인큐베이트 프로젝트로 들어올 Trove(Database service), Ironic(Bare Metal), Marconi(Queue Service), Savannah(Data Processing) 등이 있다[2]. 유수 벤더 및 커뮤니티간 공동협력을 통한 작업이 되므로 최신 클라우드 기술의 각축장이 되고 있다.

풀 패키지 설치가 아닌 프로젝트 단위의 방식으로 개발되었기에 초기 수동 설치가 비교적 어렵다는 평이 있지만 모듈 독립성과 코드 간결성이 매우 좋다. RESTful 방식으로 각 독립된 프로젝트에서 서비스 받을 수 있다. 모두 Python 언어로 작성되었으며, 대부분의 데몬들이 WSGI(Web Server Gateway Interface)로 구현되어 비슷한 소스 구조를 가진다.

1. 컴퓨팅 리소스(코드명: Nova)

Nova 는 IaaS 시스템의 주요 파트인데 클라우드 컴퓨팅의 페브릭 컨트롤러이다. 스케줄러에 의해 컴퓨팅, 스토리지, 네트워크 등의 정보를 획득하여 하이퍼바이저 API(XenAPI for XenServer/XCP, libvirt for KVM or QEMU, VMwareAPI for VMware 등)를 통해 가상머신을 생성하고 종료시킨다. KVM, XenServer 가 많이 사용되며 윈도계열인 Hyper-V 도 최근 추가되었다. 베어메탈과 HPC(High Performance Computing)의 구성도 고려된다. 모듈 간 통신은 AMQP 등의 메시지 큐 방식이 사용되어 대용량 구현에 제한이 된다. 설계 특징은 특별한 HW 가 없이 레거시 하드웨어를 사용해서 확장성(Scalability)을 가지며, 주로 X86 머신 중심으로 개발되나 최근 X86 머신에 이어 ARM 머신에도 개발 중이다. 대용량 컴퓨팅 풀을 만들려면 확장성을 확보해야 하며 Regions, Cells, Availability zones, Host aggregates 등의 개념이 나오고 있다. 모듈 내의 데이터베이스, 메시지 큐, 스케줄러 등을 안전하고(High Availability: HA, 고가용성) 확장성 있게 사용하기 위해 포커스를 맞추고 있다.

2. 스토리지 리소스

가. Object Storage(코드명: Swift)

Swift 는 Rackspace 에서 개발한 안정된 기술로 이미지들, 볼륨 백업, 스냅샷 등의 대용량 스토리지를 분산·저장한다. 하둡(Hadoop)같이 빅데이터 서비스에게 필요한 분산 저장 시스템을 제공한다. 데이터 손실을 막기 위해 오브젝트와 파일들을 통상 3 개의 복사본을 이용해서 Ring 레이아웃에 따라서 분산 저장한다. Swift-proxy 를 통해 사용자 요청을 받아들이는 것이 다른 프로젝트와 차이가 있다. 오픈스택은 현재 KT, Rackspace, HP 에서 퍼블릭 클라우드로 사용되며, 이에 맞는 HA 가 중요시되면서 선택이 아닌 주요 기능으로 부상 중이다. SwiftStack, Ceph, Riak, Gluster-swift 등의 다른 스토리지 선택사항이 나타난다[3]. 오브젝트와 블록 스토리지를 둘 다 지원하는 Ceph 가 주목 받았다. 이 역시 평이한 사양의 HW(commodity HW)로서 쉽게 확장할 수 있는 구조를 택하고 있다.

나. Block Storage(코드명: Cinder)

Cinder 는 생성된 VM 에 부착되어 사용자 데이터를 지속적으로(persistent) 저장하기 위한 블록레벨의 스토리지이다. 대쉬보드를 통해 생성 혹은 탈부착할 수 있고, Ceph, CloudByte, Coraid, EMC, GlusterFS 등 많은 기술이 있다. Swift 와는 달리 데이터베이스 스토리지, 확장성 있는 파일 시스템, Raw 블럭 레벨 스토리지를 접근하는 서버 등의 빠른 성능과 민감한 시나리오에 적합하다. Nova-scheduler 처럼 cinder-scheduler 를 통해 최적의 스토리지 장소를 찾아내어 Grizzly 버전에서 큰 발전을 했다. 즉, Nova 에서 독립하여 다양한 벤더의 제품(IBM, SolidFire, NetApp, Nexenta, Zadara, linuxiSCSI)이 수용되었다.

3. 네트워킹 리소스(코드명: Neutron)

과거 코드명은 퀀텀이었으나 개명하였다. 가장 기본적인 API 사용과 다양한 벤더의 요구사항을 수용하기 위해 플러그인-에이전트 구조로 구성된다. 주요 제품은 Cisco virtual and physical switches, Nicira NVP product, NEC OpenFlow products, OVS(Open vSwitch), Linux bridging, Ryu Network Operating System, Midokua 등이 있으며 점점 많아지고 있다. 또한 L3, DHCP, Load Balancer, Firewall 등의 기능도 제공하여 물리 네트워크 기능과 비슷해지고 있으며 다수의 네트워킹 모델이 있다. 가장 기본적인 것은

VLAN, Flat, FlatDHCP 모델이 있다. 그 중 FlatDHCP 모드는 VM 이 생길 때마다 fixed IP 를 부여하고 공인 IP 격인 동적인 IP 를 부여한다. 네트워크 노드에서 SPOF(Single Point of Failure)가 되지 않도록 HA 목적의 Multi-host 를 고안했는데, 그 기능이 작동되지 않은 Greezly 의 한계가 있었다.

4. 대쉬보드(코드명: Horizon)

관리자 혹은 사용자는 대쉬보드 GUI 를 통해서 클라우드 기반의 자원들을 접근하고, 프로비저닝하고 자동화 한다. Django 프레임워크를 이용한 MVT(Model, View, Template)로 구성되어 있다. 빌링, 모니터링 등과 같은 것은 써드파티 제품 등을 이용할 수 있다. 서비스 제공자나 상용 벤더의 취향에 맞게 GUI 를 디자인하여 사용할 수 있다. 대쉬보드 접근말고도 OpenStack API 혹은 EC2 호환 API 를 통해서 접근을 자동화하거나 리소스를 관리하는 툴을 만들 수 있다.

5. 공통 서비스

가. Orchestration Service(코드명: Heat)

클라우드 서비스에서는 자동화 및 오토 스케일링 등이 중요하다. 아마존의 경우 모니터링 서비스가 존재하고, 모니터링 파라메터를 기반으로 트리거링해서 VM 등의 ICT 자원을 프로비저닝 받고, 오토 스케일링하는 오케스트레이션 툴(AWS Cloudformation)이 있다. RedHat 이 이런 기능을 오픈스택에서도 구현하도록 개발하여 Havana 버전에 추가하였다. Heat 의 경우 단수 VM 의 추가/삭제가 아닌 자신의 클라우드 시스템에 맞는 서버 스택을 구상해서 템플릿 기반으로 자동 구축하도록 하는 것이다.

나. Metering Service(코드명: Ceilometer)

Heat 와 결합되어서 모니터링 기능을 제공한다. 각 개별 오픈스택 프로젝트의 발생하는 다양한 파라메터를 모니터링해서 빌링의 원천 데이터를 제공한다.

다. Image Service(코드명: Glance)

Glance 는 가상 디스크 이미지를 위한 카탈로그 및 저장소를 제공한다. 디스크와 서버

이미지들을 탐색, 등록, 전달(delivery)하는 기능이다. 저용량일 경우는 로컬 서버를 사용하고, 용량이 큰 경우에는 Swift 를 사용하기도 한다. 다른 프로젝트와 마찬가지로 API 를 이용하여 표준적인 REST 인터페이스를 이용한다. 직접 OS CD 를 이용하거나 이미 만들어진 이미지의 URL 을 이용해서 이미지를 생성할 수 있다. 이 외에 글랜스가 아닌 유틸리티로서 사용하는 경우도 있으며 Server3, uShareSoft 등이 있다[3]. 기본적인 OS 설치뿐 아니라 관련된 애플리케이션을 쉽게 설치하고 관리되도록 하는 것까지 포함되어 있다.

라. Identity Service(코드명: Keystone)

Keystone 은 클라우드 OS 전반에 걸친 공통의 인증작업을 제공한다. 즉, 사용자에게 오픈스택 정책, 카탈로그, 토큰 및 인증을 위한 일원화된 접속점을 제공한다. 백엔드 기술로는 LDAP(Lightweight Directory Access Protocol), SQL, KVS(Key Value Stores) 등이 있다[3].

II. 오픈스택 커뮤니티

오픈스택은 클라우드 SW 이지만 커뮤니티 이름이기에 커뮤니티에 대한 이해가 필수적이다. 클라우드계의 리눅스를 목표로 국제적인 커뮤니티 활동이 오픈스택을 만드는 원동력이기 때문이다. 물건 제조가 아닌 SW 를 작성하는 것은 거대 자본과 거대 시설보다는 인간의 순수한 창작력에 좌우되고, 인터넷으로 국제적 공동작업이 가능해지며, 리눅스 활동으로 인한 오픈소스 작업 프로세스가 잘 마련되었기 때문이라고 판단된다. 클라우드의 대세로 인해 리눅스에 이어 오픈스택은 오픈소스 역사상 가장 빠르게 성장하고 있다. 여기서는 오픈스택을 오픈소스의 일종이라는 큰 관점에서 오픈소스 활동과 커뮤니티의 특징, 영향력을 기술하고자 한다.

1. 오픈소스 활동[4]

오픈소스의 정의는 변천하지만 초기에는 “자유롭게 열람하고 사용하고 고치고 또 다른 사람에게 배포할 수 있는 소프트웨어”란 면에서 지속 가능하게 하는 공동체의 의미가 강해져 “공동체에서 오픈 프로젝트로서 진행하면서 집단협업으로 만든 소프트웨어”로 정의되고 있다[4]. 예로서 Debien, Apache, KDE, 구 MySQL 등이 있다. 상용제품을 전략적

목적에 의해 공개된 예도 있으며 Solaris, Android, Fedora, OpenSuse, MySQL 등이 있다. 주요 사이트로서는 www.sourceforge.net, www.blackducksoftware.com 등이 있다. 이것은 익숙한 윈도계열의 상용 SW 에서 접하던 개발전략과는 매우 다르다. 시장에서 가치있는 SW 개발을 위해 프로젝트팀이 폐쇄적 개발 방법으로 일정기간 동안 연구 개발하여 출시하는 방법과는 동기, 개발구조, 개방성이 큰 차이를 보인다. 시장은 반드시 이기적 인간의 탐욕으로 지배된다는 일반적 믿음을 깨뜨린다. 빅브라더의 위험 속에서 "개방과 공유"의 가치가 "폐쇄와 통제"의 가치를 넘는 현상이 여기서 일어나고 있다. 접속비용이 점차 낮아져 계층적 계급적 사회가 평평하게(flat) 변화해가는 증거라고 보여진다. 국가, 기업의 높은 벽을 뚫고 국제적 집단협업을 통해서 SW 를 만들어가는 것이다. 실제로 낮에는 Microsoft 사에서 일하면서 밤에는 리눅스 오픈소스 SW 를 작성하는 사람이 많다고 한다. 팀 동료와의 정보교환보다 타국의 커뮤니티 회원들끼리 정보 교환이 더 빠른 예가 점점 많아질 것이다.

이런 추세로 인해 오픈소스의 초기 이미지인 아마추어, 비영리, 상용화되지 못한 프로토타입의 인상을 극복하고 있는 중이다. 개발에 대한 배타권 행사인 라이센스 문제가 아직은 큰 이슈이나 점차 많은 분야에서 비용절감을 계기로 사용이 급속화되고 있다. 세계적인 벤더라 하더라도 이런 국제적 흐름을 무시할 수 없게 되었다. 그 예로 클라우드 컴퓨팅에서 오픈스택, 빅데이터의 Hadoop, 자동차 IT 융합의 AGL(Automobile Grade Linux), SDN 의 OpenDayLight 등이 그 예이다. 따라서 이제는 이곳에 참여하지 못하면 고립 혹은 도태의 인상까지 준다.

2. 오픈스택 커뮤니티 활동

오픈스택의 시작은 Rackspace 사와 NASA 로부터 시작된다. Rackspace 의 스토리지 솔루션과 NASA 의 컴퓨팅의 결합으로 시작되어 현재 12886+ 참가자와 131+ 국가가 참여 (2013 년 11 월 18 일)하고 있다. 2012 년에 OpenStack Foundation 재단이 만들어져 인력과 자금을 운영하고 있다. 플래티넘 멤버쉽으로는 AT&T, IBM, HP, Rackspace, RedHat, Canonical, SUSE, Nebula 등이 있고, 골드 멤버쉽으로는 Cisco, Dell, NetApp, ClearPath, Cloudscaling, DreamHost, ITRI, Mirantis, Morphlabs, Piston, Cloud, Yahoo, VMware, NEC, Intel 등이 있다. 이들은 계약조건에 따라 일정기간 인력과 자금을 지원해

야 한다. 주 커뮤니티 사이트는 http://www.openstack.org 이고 쉬운 인스톨을 위한 devstak.org 가 있고, 개발한 OpenStack API 의 호환성을 체크하기 위해 샌드박스 서비스를 제공해주는 trystack.org 가 있다. 한국은 www.openstack.or.kr 공식 사이트와 페이스북상에 'OpenStack Korea Group'가 있다.

3. 오픈소스의 영향력[4]

오픈소스 활동은 기존 R&D 개발체제에 큰 영향을 줄 것이다. 몇 가지를 알아보자. 여기서 기술된 많은 내용은 참고문헌 4 번 문서를 기반으로 한 세미나(김명준 발표)에서 받은 내용과 영감을 기반으로 함을 먼저 밝혀둔다.

가. 라이센스 문제

오픈소스를 사용하여 개발한 독자적인 개발코드에 대한 라이센스를 누가 가지느냐 하는 문제이다. 현재 소프트웨어는 저작권, 특허권, 영업비밀, 상표 등의 지적재산권법에 의해 보호받고 있으며, 오픈소스 라이센스는 사용자의 자유로운 사용, 복제, 배포, 수정을 보장하고 있다. 저작권 관련 문구 유지, 제품명 중복 방지, 서로 다른 라이센스의 조합 등의 공통적 준수사항이 있고 사용 여부 명시, 소스코드 공개, 특허 등은 라이센스마다 다르다. 주요 라이센스는 GPL 2.0, LGPL 2.1, BCD Licence, Apache License 등이 있다. GPL 이 제일 많이 채택하는 라이센스인데 소스코드 공개 범위가 크다. 이에 비해 LGPL 2.1 은 좀 더 완화된 라이센스를 통해 배포를 장려하고 소스코드를 공개할 필요가 없다. BSD 라이센스는 미국정부에서 제공한 재원으로 운영되므로 소스코드를 공개할 필요가 없어서 상용 소프트웨어에 무제한 사용이 가능하다. Apache license 는 아파치 재단의 모든 소프트웨어에 적용되며, 특허권에 관한 내용이 포함되어 BSD 라이센스보다는 좀더 법적으로 완결된 내용을 담고 있다[5].

오픈스택의 경우 2012 년 창설된 비영리 단체인 OpenStack Foundation 에서 유지보수 하고 있으며 Apache License 하에 배포된다. 따라서 연구소 혹은 기업에서 오픈소스를 활용해서 R&D 를 한다면 라이센스 문제를 염두에 두어야 한다.

나. 버전 업 문제

활동이 활발한 커뮤니티 오픈소스일수록 릴리즈 향상 속도가 빠르다. 만약 현 A 버전

을 통해서 연구 개발하여 A dash 를 만들었다고 가정하자. 그러나 곧 B 버전이 나오면 A dash 가 묻히고 만다. 계속 고객의 요구사항에 따라서 유지보수, 개선할 벤더 혹은 커뮤니티가 없다면 사장된다. A dash 가 매우 뛰어난 이노베이션 요소를 가지고 있다면 소스를 만드는 커뮤니티 활동에 적극적으로 개입하여 A dash 의 방향으로 B 가 나오도록 해야 한다. 그러나 기업 혹은 연구소가 프로젝트 기반으로 재정과 인력을 운영한다면 프로젝트 후에는 재정과 인력을 제공할 수 없어 지속되지 못한다. 그렇다고 지금의 오픈소스를 벗어나기도 쉽지 않다. 그만큼 오픈소스는 많은 기간과 인력의 수고로 이루어진 지적 결과물이기 때문이다. 따라서 연구소 혹은 기업에서는 일정한 버전의 오픈소스를 잘 선택해서 사용해야 한다.

다. 개방적 R&D 체제

기업이란 재화와 서비스를 생산하기 위한 생산요소들을 일정한 조직과 지역 내에 밀집시킨 집단이라 할 수 있다. 근거리에 있어야 인적·물적 자원을 충분히 활용해서 생산 효과를 낼 수 있다는 전제가 있다. 그래서 기업은 과-부-회사, 연구소는 팀-부-부문-원 등의 계층적 구조를 가지고 인적, 물적 자원을 통제한다. 그러나 정보통신기술이 발달하면서 이 전제들이 깨어지는 예들이 많아지고 있다. 폐쇄된 조직 내의 정보를 이용하기보다 소셜 네트워크를 통해서 고급정보를 얻고, 실시간 미팅과 의사결정을 할 수 있다. 전통적인 조직에서는 얼굴과 얼굴을 대면하면서 실시간 회의와 토론을 통해 신속히 진행될 수 있지만, 운영 방식에 따라서 매우 비효율적이 될 가능성이 높다. 부서간 벽, 계층 구조로 인한 의사결정의 느림, 조직문화로 인한 느슨한 혹은 너무 긴장된 분위기 등이 생산성에 큰 영향을 미치기 때문이다. 전통적으로 월급을 주는 경제적 장벽이 근로자를 보호 혹은 통제하였지만 현재는 창의적 분위기, 복지의 혜택, 자발성 등도 기업활동에 큰 요소로 등장했다. 이런 시대적 분위기를 가장 표면으로 보여주는 곳이 오픈소스 커뮤니티이지 않을까 판단된다. 대기업의 스폰서를 받고, 나름 경력을 쌓기 위해 가장 코어 기술을 숨긴 채 오픈소스 활동을 하지만 역시 컴퓨터와 인터넷만 있으면 할 수 있다. 이러한 순수한 SW 창작을 통해서 생산하는 특징은 개방과 공유에 큰 가치를 두게 한다. 국제 공용어인 영어로 이메일, 메일링 리스트, 실시간 채팅 등을 통해서 소스가 개발된다. 회사의 계층적 수직구조를 벗어난 수평적 통제구조를 접하게 된다.

라. 개발-검증-홍보의 일원화

폐쇄적 수직 구조에서 소스를 개발한다면 개발자와 평가자가 구분되어 계획 일정에 따라 개발하고 평가하게 된다. SW 를 저장하는 저장소를 따로 두고 일정기간 보호하면서 업데이트하여 개발하게 된다. 완료되면 베타 버전으로 공개하여서 버그를 잡게 된다. 홍보팀을 통해 이 제품에 대해 홍보하게 된다. 그러나 개발자 커뮤니티가 형성된 곳이라면 개발하면서 검증하고 그 경험이 인터넷 사이트에 게시판 글로서 쌓이게 되면 그 자체가 홍보가 된다. 그 코드의 작동 여부는 직접 다운받아서 설치해보면 알게 된다. 만약 문제가 생긴다면 곧바로 사이트에 이슈 제기를 하면 개발자는 그 피드백을 통해 실시간으로 수정한다. SW 란 지적창작물이 인터넷의 빠른 속도로 인한 공유가 가능하기 때문이다.

마. 소셜 공동체의 강한 힘

실제적으로 오픈소스에 참여하는 개발자들의 개방의식, 통신기기를 이용한 정보의 수집 능력은 뛰어나다. 페이스북, Linkedin, 트위트 등의 소셜 네트워크를 통해서 교류가 활발하다. 회사란 곳은 재원을 기반으로 프로젝트에 따른 사람의 이산집합이 이루어지며 강한 결합(strong coupling)만큼이나 단절도 크나(조직을 벗어나면 관계도 약해진다는 뜻), 소셜 공동체를 통한 만남은 매우 약한 결합(weak coupling)이나 목적성이 분명하고 접속비용이 낮아서 매우 오래 지속될 가능성도 높다. 그런 현대의 모습을 볼 수 있는 곳도 바로 오픈소스 공동체이다. 그만큼 전통조직에서는 위계질서가 높아서 상하 교류의 접속비용이 높아져 있다는 반증이기도 하다.

분명 이런 ICT 기술의 발달로 인한 소셜 네트워크로 인해 사람과 사람, 사람과 정보의 접속비용이 현저히 낮아져 기술의 융합이 쉽게 일어나, 서로 다른 기술이 융합되어 새로운 서비스가 많이 일어나게 될 것이다. 이로 인한 SW 수요가 증가할 것으로 보여진다. 그 선두에 오픈소스 활동이 있다고 보여진다. 또한 그런 운영방식이 여러 분야에 적용되어 조직의 큰 변화를 이루게 될 것이다. 속도의 문제일 따름이다.

III. 파이썬 언어

Python 언어는 1989 년 후반에 Guido van Rossum 에 의해서 개발되었다. CWI 란 기업에서 개발한 ABC 를 통해 많은 경험을 쌓은 후이다. Amoeba 분산 운영체제를 통한 시

스템 호출 권한 접근을 원했다고 한다. 특징은 우선 매우 쉽다. Perl 에서 보던 $:, ~ 등의 산만한 표시가 없고 많은 라이브러리가 존재하며 신속한 프로토타입 도구를 제공한다. Powerful programming language 이기도 하다. Perl 의 경우 문자열 패턴 매칭이 우수하고 강력한 정규식(Regular expression)을 제공하며, 텍스트 스트림필터링, 인식, 추출이 쉬우나 어떤 면에선 애매하고 어렵고 혼란스러워 늘 참고서가 필요하다. Java 의 경우는 기본 문법이 엄격한 객체 지향을 추구하여서 늘 부담스러우나 Python 은 Perl 의 장점이 충분하면서 객체지향적인 스크립트이다. Tcl/Tk 를 지원해서 GUI 를 지원한다. JavaScript 와 가장 비슷하다. 쉬운 언어이나 LIST 에서 제공하는 매우 효율적인 고수준의 데이터 타입을 제공한다. 매우 심플하지만 객체지향 프로그램에 대해 효율적 접근이 가능하도록 하지만 꼭 OOP 가 필수는 아니다. 확장성과 이식성이 우수해서 '접착제 언어'라 불린다. Python 자체가 C 로 작성되어 CPython 이라 하고, Java 로 작성된 것은 JPython 도 있다. 스크립트 언어이지만 *.pyc, *.pyo 등으로 바이트 컴파일이 된다. Java 처럼 훌륭한 메모리 관리를 할 수 있고 exception handlers 를 통해 예외처리되어 디버깅에 도움을 준다.

인터프리터 언어이므로 코딩하면서 반응을 보고 진행할 수 있다. 그래서 본격적인 시스템을 만들게 되면 성능이 문제될 수 있다고 생각되지만 반드시 그런 것은 아니다. python 을 C 로 전환하는 방법으로 성능 향상이 크다고 한다. C 확장 모듈을 기반으로 한 패키지의 대표적인 예가 바로 Numeric Python 이다. 각종 수학적 연산(행렬, FFT, 랜덤수, 선형대수 등)을 쉽게 처리해 주며, 성능 향상의 효과뿐 아니라 코드의 은닉이 가능하다. Python 의 경우 스크립트 언어여서 누구나 볼 수 있는 약점이 있지만 코드를 숨기고자 할 경우에는 C 모듈이나 C 확장형을 만들 수 있다. 문법에 의거한 프로그래밍 교육을 벗어나 프로그램 작성의 본질을 배울 수 있으므로 대학에서 초보 프로그래밍 교육과정으로 활용하는 일이 증가하고 있다.

인터프리터로 가장 간단하게 활용하는 것이 바로 계산기처럼 사용할 수 있다. 각종 수식을 적고 리턴키를 치면 값이 나온다. 복소수 연산도 가능하고, 문자열의 계산을 쉽게 만들어서 언어 처리에도 편리하다. 제어문도 간단한 편이며 If, for, while 문이 기본이고 range, break, continue, pass 등도 있다. 함수(function)를 나타내는 def 가 있다. 순수한 객체언어의 경우 직접 함수를 사용할 수 없으나, Python 은 직접 함수를 사용할 수 있어서 수월한 느낌을 준다. 정규 함수 말고도 Default Argument values, keyword Arg, Arbitary

Argument Lists, Unpacking Argument Lists, Lambada Forms, Documentation Strings 등이 있다. 데이터 구조도 List, Sets, Dictionaries 등이 있다. 사전형의 경우 C 로 구현하려면 상당한 노력이 드는데 비해, Python 은 기본형으로 제공하므로 쉬운 High-level 언어라는 증거로 볼 수 있다.

함수보다 약간 큰 함수 묶음 단위가 Modules 이다. 모듈 등의 묶음이 Packages 이며 import 명령을 통해 찾도록 되어 있다. 스크립트 언어가 쉽다는 것은 역으로 말하면 매우 많은 라이브러리가 존재하고 다양한 프레임워크가 있다는 이야기가 된다. C 의 경우는 어렵지만 집중적으로 소스를 보면 파악되나, Python 의 경우는 광범위한 Modules, Package, framework 를 학습해야 원활하게 작성할 수 있다는 말이 된다. 실제적으로 os, glob, sys, sys, stderr, write, re, math, urllib2, smtplib, daetime, data, zlib, timeit, Timer, average 등을 Python 창시자 매뉴얼로 보면 부록으로 소개하여 라이브러리의 중요성을 역설하고 있다[6]. 객체지향 언어라는 것은 클래스를 사용하는 것인데 Java 처럼 엄격하지 않아서 쉬운 느낌을 준다. 이런 쉬운 Python 언어 특징에 의해 한층 더 오픈스택의 발전이 빨라진다.

IV. 오픈스택 발전 방향

정보통신산업이란 본질적으로 사물의 추상화된 개념인 정보를 프로세싱・저장・전달하는 산업이다. 따라서 서버, 스토리지, 네트워크란 ICT 자원이 필요하다. ICT 자원이 개인 소유에서 메시업을 통한 개방된 형태로 클라우드센터에 집중되고 있다. 이중 클라우드 플랫폼 기술 중 커뮤니티와 벤더들을 통해 공동 개발되는 오픈스택이 단연 우세이다. 그러나 다양한 기술, 많은 규격, 설치의 어려움 등으로 방만하고 복잡하게 보인다. 이런 속에서 현재의 모습과 바람직한 발전 방향을 고찰하고자 한다.

① 우선 오픈스택은 IaaS 를 목표로 하므로 PaaS, SaaS 로의 발전은 분명하다. 즉, 단순 자원할당의 인프라 서비스에서 개발환경까지 제공하는 플랫폼 서비스, 애플리케이션까지 제공하는 서비스로 발전할 것이다. PaaS 로서는 구글의 Google App, SaaS 로서는 Google Doc 를 들 수 있다. ② 오픈스택의 기술적 리더쉽을 가진 Rackspace 사의 제공 서비스를 보면 오픈스택의 모자란 면이 보일 것이다. 데이터베이스, 로드밸런스, 백업 등의 기능은 현재 오픈스택은 지원하지 않는다. 그러나 현대의 상용 웹 서비스에는 위의 기

능은 필수이다. ③ 상용화 정도를 보면 분명하다. 오픈스택의 유명세에 비해서 대규모 상용 서비스는 Rackspace 와 HP 정도 밖에 없다. 클라우드 스택의 경우 수백의 벤더에서 상용 서비스를 하는 것과는 대조적이다. 실제적으로 클라우드 스택의 경우 소스 수정이 어렵고 모듈 독립성이 결여되었지만 중형 규모에서 안정적인 네트워크 기능을 제공해서 상용에 적합하다는 평이다. 그렇다고 오픈스택에 대해 비관적인 평은 이르다. 워낙 발전속도가 빠르기 때문이다. ④ 가상화의 선두주자는 역시 VMware 이다. 클라우드 관리 SW는 아니지만 VMware vSphere 관리 SW를 사용하면 역시 안정적이고 각종 자원에 대한 모니터링이 뛰어남을 확인할 수 있다. VM migration, HA 등의 고급기능이 안정적으로 수행되고 GUI 등의 상용화 느낌이 들지만, 오픈스택의 경우 기초적 디자인 수준이라 보여진다. 그러나 무료의 힘과 커뮤니티의 발전속도, 오픈 개발을 위한 소스 공개란 장점을 생각해보면 무시할 수 없다. ⑤ 오픈스택의 모자란 점이 무엇일까란 질문에 대한 답은 함께 사용하는 오픈소스를 보면 분명해진다. 모니터링 SW 은 zabbix, collected, nagios 를 사용하고, 자동화 툴로서는 chef, puppet, crawbar(from DELL), 스토리지 파일 시스템으로는 ZFS, GlusterFS 등이 있다. 이것으로 보아 역시 모니터링 기능, 자동화 기능이 이슈임을 알 수 있다. 특히 현재는 VM 상의 모니터를 VNC 기술을 이용해서 볼 수 있으며 VMware에 비해 불안정하고 느리다. 그러나 SPICE 기술로 일 전진하였다. ⑥ 오픈스택의 가장 큰 어려움은 설치의 어려움이다. 9 개의 코어 프로젝트가 따로 개발해서 통합하는 형식을 택할 수 밖에 없어서 초기 설치가 생각보다 쉽지 않다는 것이 첫 인상일 것이다. 그래서 Devstack 등의 커뮤니티를 통해서 자동설치를 계속 개발 중에 있다. 또한 수백대의 서버에 설치하기 위해서는 Chef, Puppet 등을 사용한다. 따라서 설치의 용이성이란 필수적 기능을 늘 살펴보아야 한다. 아예 이미지로서 설치하는 베어메탈 방식도 계속 발전 중이다. ⑦ 획기적인 발전은 오픈스택의 단일 클라우드 센터를 넘어 이기종간 클라우드 연결을 가능하게 하는 Service Broker 의 등장 단계일 것이다. 단일 클라우드가 우세한 미국이나(Google, Facebook, iCloud 등), 국토가 좁은 한국의 경우 큰 중요성이 부각되지 못하나 유럽 같은 환경에서는 이기종 클라우드 간 연동이 중요하게 부상한다. ⑧ 좀 다른 관점으로는 서비스 목적에 맞게 적절히 사이즈, 성능, 배치 등을 쉽게 구성할 수 있느냐는 것이다. 퍼블릭 혹은 프라이비트 구성, 싱글 호스트 및 멀티 호스트 구성, 의료-교육-미디어 전달용 클라우드를 생각할 수 있다. Cisco 의 경우 차세대 데이터센터 전략인 ACI(Application-

Centric Infrastructure)를 보면 애플리케이션마다 다른 요구사항을 제시한다. 이를 참고로 생각해보면 오픈스택도 향후 애플리케이션에 따른 유연한 자원할당에까지 포커스를 맞출 수 있을 것이다.

Ⅴ. 결 론

OpenStack 의 접근방법에 대한 몇가지 커맨트를 하고 마무리하고자 한다. 첫째, OpenStack 은 현재 9 개의 프로젝트로 구성되어 있고 소스 구조가 비슷하며 Python 단일 언어로 기술되었으므로 총체적인 관점으로 접근해도 되리라고 보여진다. 한 프로젝트를 깊이 보는 방법도 좋지만 총체적인 관점을 항상 지니고 있어야 된다. 둘째, 기업에서 이것을 사용할 경우에는 라이선스, 버전, 커뮤니티 활동, 동향 등을 잘 살펴보고 접근해야 한다. 기존의 개발체계와는 사뭇 다른 것이 있음을 주의해야 한다. 이런 오픈소스 개발론이 사회전체에도 영향을 미칠 것이라는 조심스런 주장도 해보았다. 셋째, 오픈스택을 기술하는 Python 언어는 C 와 Java 처럼 어렵지는 않지만 다수의 라이브러리와 프레임워크란 관문을 통과해야 되는 것을 잊어서는 안된다. 넷째, 오픈스택은 전세계 클라우드 기술의 각축장이므로 이를 연구 개발하는 것은 여러 면에서 아이디어를 얻을 수 있다. 다섯째, 오픈스택 미래의 모습 혹은 로드맵을 여러 관점으로 예측해서 자체 개발 혹은 기존 기술 흡수를 잘 판단해서 타이밍을 놓치지 않아야 한다.

<참 고 문 헌>

[1] http://www.openstack.org/software/havana
[2] http://www.openstack.org/software/roadmap/
[3] Ken Pupple, “Real-world Openstack v1.1”(2013. 8. 23 서울강의) 자료.
[4] 김명준, 정성인, “공개SW 재해석과발전방안 -연구개발측면-”, ETRI 내세미나자료, 2013. 9. 13.
[5] http://wiki.kldp.org/wiki.php/OpenSourceLicenseGuide#s-2.1Cisco
[6] Guido van Rossum, “Python Tutorial Release 2.6.4”, Jan. 4. 2010.

* 본 내용은 필자의 주관적인 의견이며 NIPA 의 공식적인 입장이 아님을 밝힙니다.