Raspberry Pi와 클라우드 환경을 이용한 실습실 설계 및 구현

Design and Implementation Laboratory
Using Raspberry Pi and Cloud system

지도 교수 민 덕 기 (인)

--- 환종조 ---

건국대학교 컴퓨터공학과 김 민 환	건국대학교 중어중문학과 이 종 하

이 논문을 컴퓨터공학 학사 학위 청구 논문으로 제출함

2019년 06월

건국대학교 소프트웨어융합학부 컴퓨터공학과

Raspberry Pi와 클라우드 환경을 이용한 실습실 설계 및 구현

김민환[1], 이종하[2]
[1] 건국대학교 컴퓨터공학과
[2] 건국대학교 중어중문학과
mail@nerd.kim, dlwhdgk9@konkuk.ac.kr

Design and Implementation Laboratory Using Raspberry Pi and Cloud system

Minhwan Kim[1], Jongha Lee[2]
[1] department of computer engineering
[2] department of chinese language and literature

요 약

현재 대부분의 초, 중, 고등학교 및 대학교 등의 교육기관에서는 컴퓨터 관련 교육을 위한 컴퓨터 실습실을 운영하고 있다. 또한 이러한 실습실은 시립/국립 도서관, 학원 등에서 운영되고 있다. 각 실습실에 있는 컴퓨터는 가능한 동일한 OS(운영체제)와 소프트웨어들을 설치하여 운용하고 있으며, 컴퓨터의 하드웨어의 경우 specification을 동일하게 설정하고 있다. 컴퓨터 실습실 관리는 관리자의 수작업을 통해 여러 대의 컴퓨터에 대한 환경설정 및 OS, 소프트웨어 설치 작업을 진행한다. 하지만 사람이 수작업으로 진행하는 일에는 실수가 발생할 수 있고, 완벽히 같은 환경을 구성하기가 힘들다. 이를 개선하기 위하여 많은 비용과 많은 시간이 소요되는 것이 현실이다. 이러한 문제를 해결하기 위하여 Raspberry Pi와 클라우드 환경을 이용하여 VDI 솔루션을 구현한다.

1. 서 론

라즈베리파이(영문: Raspberry Pi)는 영국 잉글랜드의 라즈베리파이 재단이 학교와 개발도상국에서 기초 컴퓨터 과학의 교육을 증진시키기 위해 개발한 신용카드 크기의 싱글 보드 컴퓨터이다. 가격이 저렴한 관계로 대량 생산되고 있으며, 크기가 매우 작기 때문에 주변에서 구하기 쉬우므로 성능이 좋지 않지만 가장 접근성이 뛰어난 컴퓨터라고 평가받는다. 클라우드 컴퓨팅은 인터넷 기반 컴퓨팅의 일종으로 정보를 자신의 컴퓨터가 아닌 인터넷에 연결된 다른 컴퓨터가 아닌 인터넷에 연결된 다른 컴퓨터로 처리하는 기술을 의미한다. 컴퓨터와 다른 장치들이 요청 시 데이터를 제공해준다. 이를 통해 고성능 연산에 대해서 라즈베리파이 자체 내에서 실시하지 않고 클라우드 컴퓨팅 기술을 활용하여 고성능 연산을 서버에 넘기고 결과만 수신하는 방안을 제시하였다. 논문에서의 데이터 처리 송신 및 수신은 직접 설계 및 구현하는 과정을 통해 설명하였다. 실험 과정에서 라즈베리파이는 3세대 B+ 버전을 사용하였으며, 클라우드 컴퓨팅 기술은 OpenStack을 사용하여 구현하였다. 본 논문은 접근성이 우수하지만 성능이 떨어지는 라즈베리파이를 클라우드 컴퓨팅 기술을 통해 최고의 성능을 갖춘 하드웨어로 변모시켜, 실습실이 저비용으로 고효율을 낼 수 있도록 방안을 제시한다.

2. 관련연구

2.1 Cloud Platform

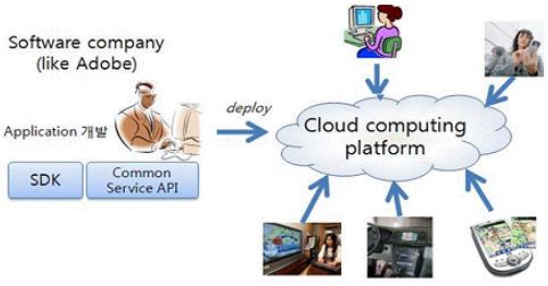
Cloud Platform(클라우드 환경)은 사용자의 환경 밖에서 서비스로서 제공된 확장 가능한 컴퓨팅 자원을 의미한다. 사용자는 클라우드 환경에 있는 모든 자원을 언제 어디서나 인터넷을 통해 액세스할 수 있다.

클라우드 플랫폼은 기업이나 개인이 클라우드 환경을 구축할 때 쉽게 개발 가능하도록 프로비저닝, 자동화 및 자동 스케일 업/다운(auto scaling)을 비롯한 가상 서버 관리, 스토리지 관리, 네트워크 관리, 보안관리, 데이터베이스, 도커 등 필요한 요소를 제공하는 소프트웨어를 의미한다.

현재 수 많은 클라우드 플랫폼이 운영되고 있다. 글로벌 IT 기업인 아마존의 AWS, 마이크로소프트의 애저(Azure), 구글의 구글 클라우드 플랫폼 이외에도, 한국에서는 삼성의 S클라우드, 네이버의 네이버 클라우드 플랫폼, KT의 Ucloud Biz등이 운영되고 있다.

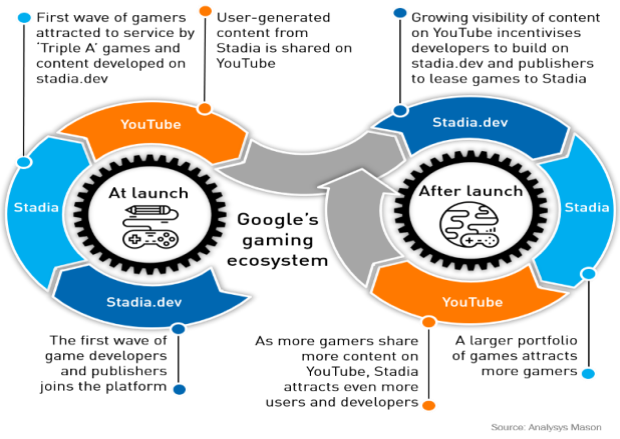
글로벌 IT기업들은 클라우드 플랫폼에 큰 관심을 갖고 적극적으로 투자를 하고 있다. 특히, 구글은 Firebase(데이터 동기화), Orbitera(클라우드소프트웨 어), Apigee(API 관리 및 예측 분석), 캐글(데이터 사이언스)와 같은 기업들을 인수하여 구글 클라우드 플랫폼과 합병하면서, 클라우드 플랫폼을 적극적으로 지원하고 있다.

많은 기업들이 비용절감 및 특정한 기능 사용을 위해 클라우드 플랫폼을 사용하고 있으며, 그 수가 급격하고 증가하고 있는 추세이다. 최근에는 특히 도메인 특화 기술이 기업용 특화 서비스를 넘어서 클라우드 서비스 기술로 발전이 예상됨에 따라서, 이에 대한 중/장기적인 기술개발 및 표준화 작업이 진행중이다.



▲ 그림 1. 클라우드 컴퓨팅 구성도

특히, 구글에서는 클라우드 플랫폼 기술을 이용한 게임 스트리밍 서비스를 개발 중이며 서비스할 예정이다. 이 서비스의 이름은 STADIA다. 유튜브 기반으로, 사실상 유튜브 동영상을 보며 조작을 하는 컨셉이기 때문에 유튜브 재생만 가능하다면 저 사양 PC든 스마트폰이든 태블릿 PC든 실시간 스트리밍을 통해 FHD 60프레임으로 게임을 할 수 있게 된다. 정식 출시때는 무려 4K 60프레임으로 게임을 즐길 수 있게 만들겠다고 발표하며 저 사양 PC 유저들에게 충격을 주었다. 기존의 게임들은 사용자의 기기에 게임 파일을 설치한 후 실행하는 형태로 되어 있다면, Stadia는 게임 파일 및 계산과 처리를 모두 서버가 담당하며 유저는 마우스나 키보드 움직임의 조작 신호를 서버에 보낸 뒤 서버로부터 게임 화면을 실시간으로 전송 받는 식으로 플레이 된다. 게임을 진행하는데 필요한 데이터를 모두 서버를 통해 처리한다는 점에서 클라우드 플랫폼 활용의 적절한 사례이다.



▲ 그림 2. Google STADIA Algorithm[15]

2.2 OpenStack

오픈스택은 2010년 7월 미국의 호스팅업체인 Rackspace와 NASA가 공동으로 개발하여 첫 번째 릴리즈 버전인 Austin을 내놓으면서 시작되었다. 이는 클라우드 자원을 관리하고, 정의하고, 활용하기 위한 프레임워크이다. 주요 기능은 다음과 같다 :

a) Nova
Compute Service로 OpenStack 시스템에 연결된 hypervisor에 행하는 모든 행위(VM생성, 삭제 등)를

시행 및 관리하는 역할을 수행한다.

b) Horizon
Dashboard로 OpenStack 시스템 관리자를 위한 WebUI를 제공한다.

c) Swift
StorageService로 오브젝트 스토리지(ObjectStorage) 환경을 구축하고 관리하기 위한 서비스로, OpenStack과 별개로 독립적 구축이 가능하다.

d) Glance
Imaging Service로 가상머신 이미지의 저장, 등록, 관리, 전달을 위한 서비스로 Nova와 스토리지 간의 중계역할을 담당한다.

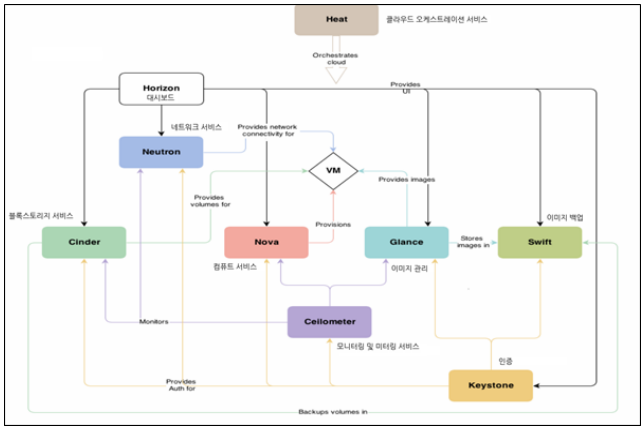
e) Heat
Orchestration으로 템플릿 형태의 클라우드 자동화를 위한 인터페이스를 제공한다.

f) Neutron
Networking Service로 오픈스택내의 IPaddress들과 네트워킹을 관리하는 기능을 제공함. SDN의프레임을 제공하고 OpenStack에서의 인스턴스 네트워킹을 위한 서비스이다.

오픈스택 소프트웨어는 하나의 소프트웨어가 아닌 여러 개의 소프트웨어(또는 컴포넌트)로 구성되어 있다.

최근 화웨이가 오픈스택을 이용한 클라우드 솔루션 제공업체로 나섰으며 HP, AT&T, Intel을 포함한 삼성과 KT 등 많은 대기업에게 지원을 받고 있다.

가장 최근(2018)에 Rocky 버전이 출시되었다.



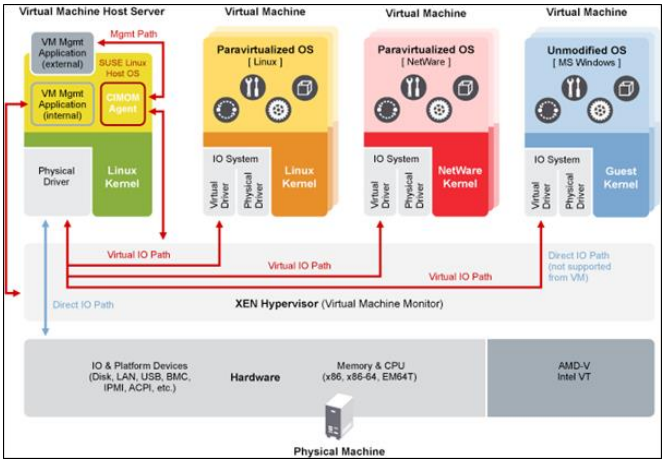
▲ 그림 3. OpenStack Architecture[10]

2.3 Xen Hypervisor

케임브리지 대학교에서 개발이 시작되어 2003년에 첫 공개 버전이 발표되었다. Xen 하이퍼바이저는 하나의 컴퓨터에서 여러 개의 운영체제를 동시에 실행하도록 도와주는 소프트웨어 계층으로, 3가지 요소로 구성된다. 매우 높은 퍼포먼스 가상화를 지원하며, 오픈소스 OS 지원 및 무료로 사용가능하다는 장점이 있다. 초기에는 반가상화만을 지원하여 게스트 운영체제를 실행하기 위해서는 게스트 운영체제를 젠에서 실행할 수 있도록 수정해 주어야 했으나 3.0부터는 게스트 운영 체제를 수정하지 않아도 젠에서 실행할 수 있게 되었다. 다만 QEMU같은 CPU 에뮬레이터가 아닌 전통적인 하이퍼바이저이기 때문에 호스트와 다른 아키텍처의 게스트를 실행할 수는 없다.

GPU지원 확장, 스냅샷 수명 주기 단순화를 통한 관리 환경 성능 형상, 응용프로그램과 데스크탑의 성능 통합, 보안 강화와 같은 보완이 이루어진 7.4버전까지 출시했다.

최근 Xen은 인텔 칩셋에서 발견된 MDS 취약점으로 인해 추측 실행과 관련된 부 채널 공격을 막기 위한 연구를 진행하고 있다.



▲ 그림 4. Xen hypervisor architecture[11]

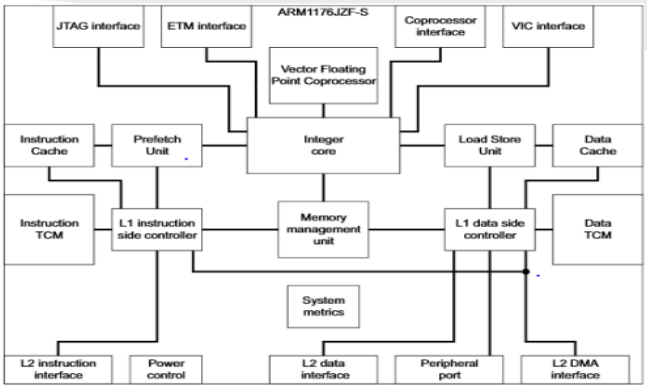
2.4 Raspberry Pi

라즈베리파이는 2012년에 출시된 영국의 라즈베리 파이재단에서 만든 초소형/초저가의 컴퓨터이다. 교육용 프로젝트의 일환으로 만들어진 것이나, 최근에는 IOT(사물인터넷)분야에서도 많이 사용되고 있다. 그 예시로는 텔레그램 미세먼지 알림 채널 운영, 토렌트 검색봇, 파이썬 테스트, 실내온도 측정, LED 제어, 220v전원 제어, 카메라 제어 등이 있다.

eMMC NAND를 제외하여 단가를 엄청나게 낮추어 지속적으로 많은 사랑을 받고 있다. 필요한 기능이 있으면 검색만 하면 바로 나올 정도로 자료가 많아서, 사용하는데 있어서 진입 장벽이 낮고, 그로 인해 출시부터 현재까지 엄청 나게 많은 양이 판매되었다.

운영체제로는 오픈소스인 리눅스 OS를 사용하여, 별도의 운영체제 이용료를 낼 필요가 없다. 초기 라즈베리파이는 USB 포트 하나에 700MHz 프로세서, 256MB RAM을 장착하고 있었으며 무선 접속 기능은 없었다. 그러나, 2018년에 라즈베리파이 3 모델 A+가 발표되었다. CPU 클럭은 B+와 동일한 1.4 GHz이지만 메모리는 512 MB, 2.4 GHz 및 5 GHz 대역의 b/g/n/ac 와이파이 및 블루투스 4.2/BLE 연결이 가능하며, 풋프린트 사이즈가 B+ 에 비해 2/3 수준으로 줄어든 대신 큰 포트들이 많이 빠졌다. 라즈베리파이는 초기 모델에 비해서 비약적인 발전을 이루었으며 그 발전은 지금도 진행중이다.

현재 라즈베리파이는 개발사의 개발에 관심이 집중되기 보다는 유저들의 활용에 대해 관심이 집중되고 있다. 라즈베리파이의 성능 업데이트는 시간이 지남에 따른 하드웨어 기술 발전에 의해 이루어지는 것으로 한정적이지만, 유저들의 활용 방안은 무궁무진하게 많기 때문이다.



▲ 그림 5. Raspberry Pi architecture[12]

3. 목표

본 과제를 요약하면, 다음과 같은 시스템을 설계 및 구현한다 :

- 본 과제는 Cloud 환경을 구성하여 교내 실습실 환경을 구축할 수 있도록 한다.
- SaaS형태의 VDI(Virtual Desktop Infra) 환경을 구축하고, Raspberry Pi를 ThinClient로 활용하여 실습실 환경을 구축한다.
- 적용처(실습실, 기업 등)의 개인용 컴퓨터는 모두 Raspberry Pi(ThinClient로 활용)로 대체하여, 이를 교내 실습실의 메인 컴퓨터(Client)로 사용한다. 이 때, Raspberry Pi는 VDI Server의 Instance로 접속하여 가상의 원격 환경을 사용할 수 있도록 구현한다.
- 실습실에서 사용하기에 성능/비용면으로 최적화된 구조로 설계한다.
- VDI Server들은 다수의 ThinClient들을 관리하기 위해 다수의 서버로 구성하며, OpenStack을 도입한다. 다수의 VDI Server들을 관리하기 위하여 Management Server를 운영한다.
- VDI Server는 ThinClient가 원격 데스크톱 환경을 사용할 수 있도록 특정 OS를 보유한 Instance를 올려주고, ThinClient와 Instance는 RDP 통신으로 원격 데스크톱 환경을 사용할 수 있다.
- 도입 및 유지보수 비용을 최대한 낮게 설정하고, 안정성은 기존 타사 제품을 유지할 수 있도록 한다.

본 과제를 실 사용처에 도입하게 될 경우, 다음과 같은 비용적 효과를 확인할 수 있다 :

(가) OS license 비용 절감

타 기업의 솔루션에서는 Client-side와 Server-side Instance 모두에 동일한 OS를 사용해야 한다는 단점이 있다(호환성의 문제). 본 과제에서는 Client-side에서 Linux(ex. Ubuntu)를 사용하고, Server-side Instance에만 Windows OS license를 적용하여 유료 OS(Windows)의 license 비용을 대폭 줄인다.

(나) Hardware 비용 절감

높은 비용을 투자하여 Client 컴퓨터를 구매 및 Scale-

up 하지 않고, 시가 한 대당 3만원대의 Raspberry Pi만을 이용하여 낮은 비용으로 시스템 구성을 제안한다. 또한 Client 컴퓨터들의 전체적인 Upgrade를 진행해야 할 경우, 실습실 관리용 Web 사이트에서 일괄적으로 원하는 사양으로 업그레이드를 진행하면 된다.

기타 이점은 다음과 같다 :

(가) 실습실 환경 자동 구성 및 빠른 instance 생성
타 기업의 솔루션에서는 ThinClient가 Power-on 될 때 instance 생성을 시작하여 boot-storm 현상이 발생한다. 본 프로젝트에서는 ThinClient에서 Power-Off를 진행하여도 instance 자체를 Power-Off 시키지 않고, 관리자가 지정해둔 image로 roll-back 이후 해당 instance를 suspend 상태로 두어 실습실 환경의 빠른 Power-On을 지원한다.

(나) 실습실 환경에 대한 배포 용이
필요한 image를 생성해 두었다가 특정 실습실에서 해당 image로 instance를 직접 생성할 수 있다.

(다) S/W License 관리
전체적인 라이선스 관리가 가능하다. 개개인이 별도로 설치하는 소프트웨어가 굉장히 많고 다양한데, instance가 Power-Off 될 때 자동으로 특정 image로 초기화 되므로 관리실에서 관리하는 S/W외에는 라이센스가 없는(혹은 불법) S/W가 설치되어 지속적으로 유지될 일이 없다.

(라) 관리의 용이성
항상 관리실이 원하는 최신 데스크톱 환경을 제공하거나, 관리자 전용 Web Application으로 특정 instance에 대해 RAM을 추가하거나 제거하고, CPU를 추가하거나 제거할 수 있다. 이로 인하여 주기적으로 PC를 교체해야 하는 비용적/시간적 부담에서 해소된다. 또한 원하는 OS(Windows 10 ~ Windows 7, Ubuntu, CentOS 등)를 직접 제공할 수 있다.
따라서 사용자 단말기에 대해서 개별적인 유지보수가 불필요하며, 기업의 경우 조직 개편 및 사원별 자리 이동, 부서 이동 등의 사무환경에 따라서 관리를 편리하게 할 수 있다.

(마) Anywhere
실습실 뿐만이 아닌 기업이나 공공기관 등에서도 본 솔루션의 구조를 알맞게 개선하여 도입이 가능하기에 확장성이 높다.

(바) 저탄소 녹색성장
VDI를 구축함에 따라 Client PC가 차지하는 전력 소모를 획기적으로 줄여 저탄소 녹색성장의 원동력이 될 수 있다.

기존 환경(개별 PC 환경)	VDI 환경
개인별 PC에 업무용/실습용 OS 및 S/W 설치	VDI Server의 가상화 PC에 OS 및 S/W 설치 * Client PC는 ThinClient로 대체되고, Client는 입출력 역할만 수행한다.
바이러스, 불법 S/W 설치 등으로 인한 관리 어려움	OS및 S/W를 중앙에서 설치하므로 관리가 편리함
내/외부 침입으로 인한 정보 유출 위험	데이터 중앙관리로 정보 유출 위험 차단

▲ 그림 6. 기존 환경과 VDI 도입 환경 간략 비교

4. 시스템 설계

본 설계에서 길게 고민한 내용은 다음과 같다 :
- OpenStack을 사용해야 하는가? Xen Hypervisor를 사용해야 하는가?
- Scale-up 방식과 Scale-out 방식 중 어떠한 것이 맞는가?
- 디스크 관리는 어떻게 해야 하는가?
- 사용자 별로 구조를 다르게 진행해야 하는데(cost-effective), Scale 확장을 위해서는 구조를 동일하게 가져가야 하지 않나? (Flexibility).

먼저, High Availability 구성과 디스크 안정성을 위한 구성을 진행한다. 서버들은 Active-Standby 2중화 구성을 진행하여 안정성을 높이며, 디스크의 경우(Storage node) raid-1 구성 및 Ceph를 통한 disk 3-copy 원칙을 사용한다.

다수의 서버를 OpenStack을 이용하여 가상화 환경을 구성한다. 여기서 Infra, Compute, Storage node들을 분리하여 구성하며, 이 과정에서 자동화 및 빠른 node 확장을 위하여 openstack-ansible(오픈스택 앤서블)과 ubuntu maas(우분투 마스)를 이용한다.

- Infra node는 OpenStack의 전체적인 인프라를 관리하는 서버의 집단을 의미한다.
- Compute node는 OpenStack에서 Instance를 올리기 위한 RAM과 CPU를 보유한 서버의 집단을 의미한다.
- Storage node는 OpenStack의 전체 인프라에서 사용되는 디스크들의 집단을 의미한다.
- Ubuntu maas는 network booting을 사용하여 서버들에 ubuntu설치를 담당한다.
- openstack-ansible은 Rocky 버전을 사용하고 있으며, openstack에 알맞은 node를 구성하여 설치를 진행한다.

서버를 내부망에 연결하여 network booting을 진행하면, 이 booting end-point가 ubuntu maas 서버로 잡혀 ubuntu가 설치되고, 이 ubuntu auto start-up script가 바라보는 end-point가 openstack-ansible 서버로 잡혀 있어 openstack설치까지 진행된다.

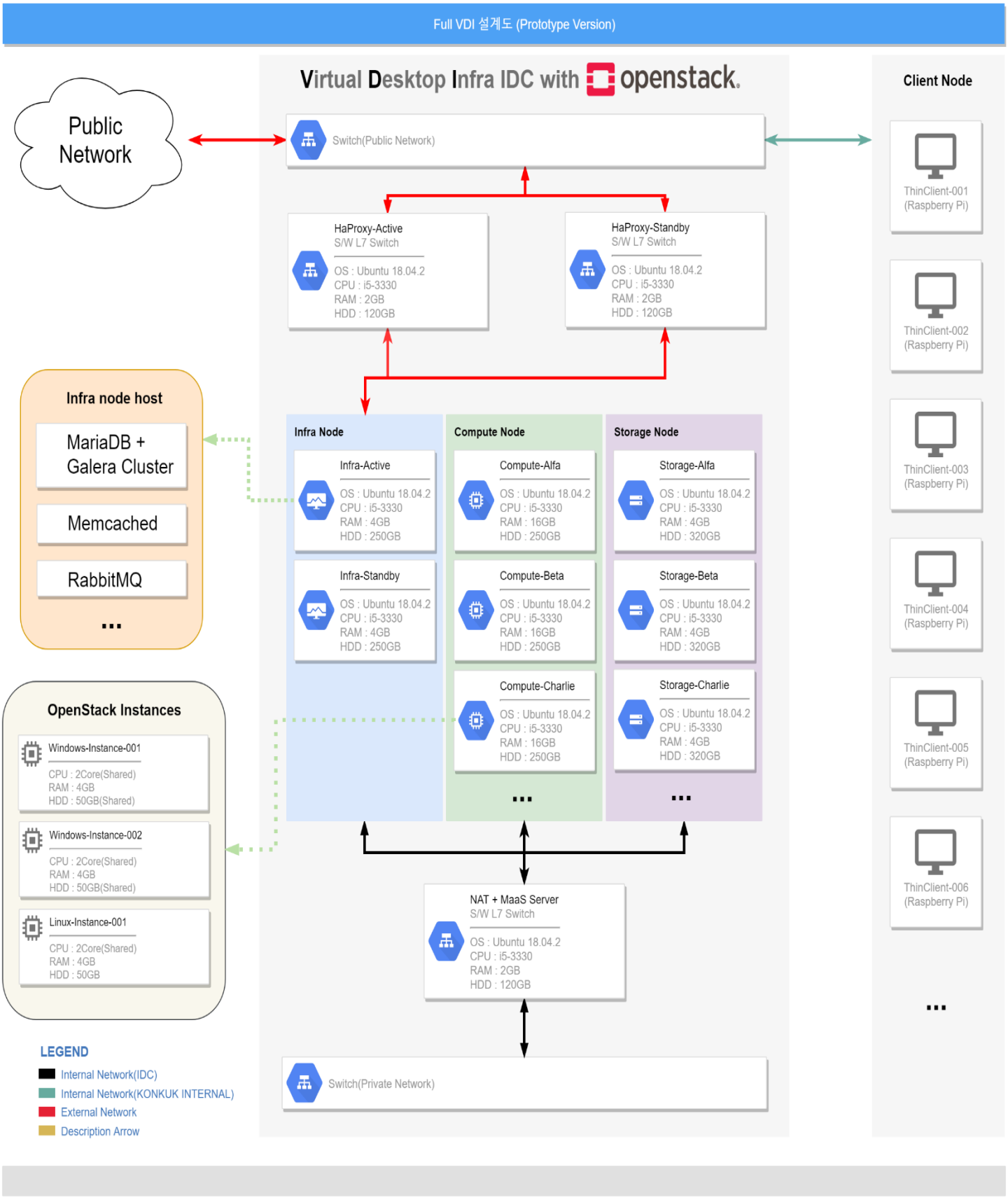
Client(ThinClient) 집단과 Public Network는 Public network switch를 통해 접근이 가능하며, HaProxy, 소프트웨어 L7 스위치가 부하를 분산한다. 이후 Infra active node와 standby node로 HA 구성을 진행한다.

HaProxy 서버는 S/W L7 역할을 진행하며, Active Switch와 Standby Switch로 구분된다. Active Switch가 정상적으로 동작하지 않으면 Standby Switch가 그 역할을 대신한다.

Infra node는 Infra-Active와 Infra-Standby로 구성되며, Infra-Active가 정상적으로 동작하지 않으면 Infra-Standby가 그 역할을 대신한다.

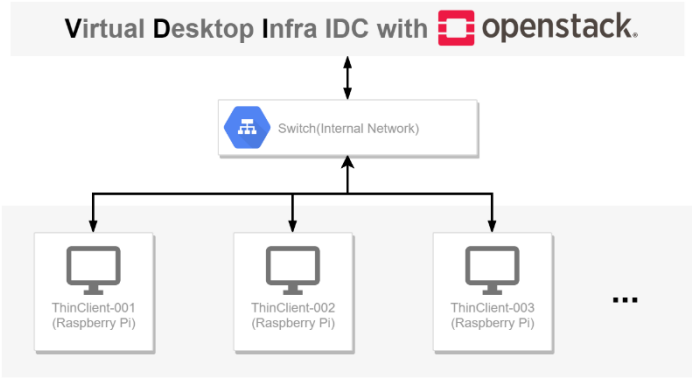
Compute node와 Storage node는 IDC내 internal network 통신만 진행한다. Infra node를 통해 외부망 혹은 ThinClient들과 통신을 진행한다.

결과적으로 서버 한 대의 기능이 상실되어도 정상적으로 동작할 수 있도록 설계 및 구현을 진행하며, 서버를 안정적이고 빠르게 늘릴 수 있도록 설계를 진행한다.



▲ 그림 7. Full VDI 설계도

추가적으로, 실습실 설계는 다음과 같이 진행한다.



▲ 그림 8. VDI IDC with OpenStack

실습실은 Internal Network를 통해 VDI 서버단과 통신을 진행하며, Client는 Raspberry Pi를 사용하여 구현한다.

6. 서버 구성 및 실험 결과

6.1 서버 인프라 구성에 대한 결과

본 프로젝트는 사용하지 않는 컴퓨터들을 수거하여 전체적으로 조립 및 검사, 교체 등의 작업을 진행하여 서버의 node로 사용하게 되었고, 전체적으로 낮은 사양의 서버들을 사용하여 server node들을 구성하게 되었다. 대부분 i5 2세대의 CPU를 보유 중이었고, 2GB정도의 RAM을 보유중이었다.

여기서 램을 추가로 주문하고 하드디스크를 주문하여 조립을 진행하였고, 연구실에서 사용하던 데스크탑 서버를 모두 OpenStack node로 활용하여 구성을 진행하였다.

▲ 그림 9. 작업이 완료된 컴퓨터(서버로 활용)

본 논문에서는 OpenStack을 구성하기 위하여 총 34대의 서버로 인프라를 구축하게 되었고, 용도 별 서버의 분배는 다음과 같다 :

- MainStack Server : 1대
- NAT/MaaS Server : 1대
- HaProxy Server : 2대
- Infra node : 2대
- Compute node : 20대
- Storage node : 4대
- Temp node : 4대

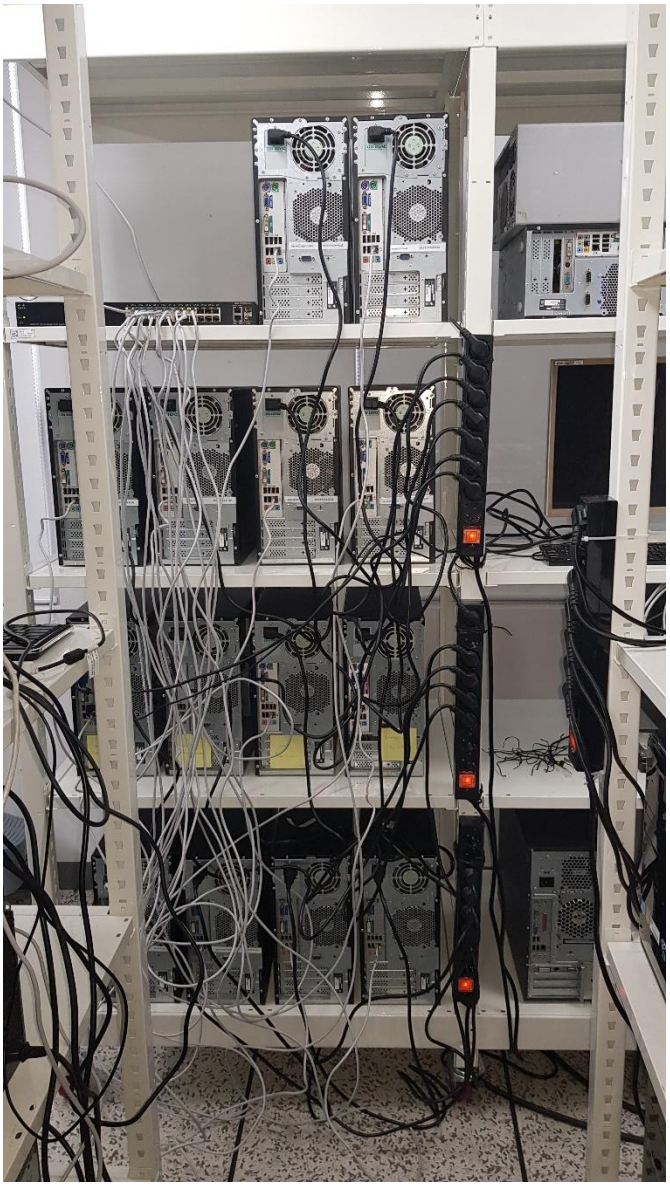
위와 같이 설계를 진행한 후, 서버실에 전체적으로 배치 및 구성을 진행하면서 작업을 시작하였다. 설치는 Ubuntu MaaS 서버에 NAT 작업을 진행할 수 있도록 설정한 이후, 각 node의 bios에서 network booting을 진행하면 설정(yml)대로 ubuntu os가 설치될 수 있도록 설정하여 두었다. 여기에 OpenStack-ansible까지 포함하여 Ubuntu 설치와 동시에 OpenStack이 자동으로 설치될 수 있도록 구현하였다.

서버 인프라 구성에는 다양한 trouble-shooting을 진행하게 되었고, 완벽히 구성하기 까지는 약 2개월 정도가 걸렸다.

▲ 그림 10. 서버실(건국대학교 신공학관 12층)

▲ 그림 11. RACK-001: Proxy, Infra, Nat/MaaS 서버, HaProxy 서버, Compute node들

▲ 그림 12. RACK-002: Compute node와 Storage node들



▲ 그림 13. RACK-003: Compute node와 Storage node들

▲ 그림 14. RACK-004: Compute node 중 GPU node들

FQDN	Power	Status	Owner	Cores	RAM (GiB)	Disks	Storage (GB)
heroic-marten.maas	Unknown	Ubuntu 18.04 LTS	admin	4	3.0	1	320.1
merry-camel.maas	Unknown	New		0	0.0	0	0.0
modern-weasel.maas	Unknown	Ubuntu 18.04 LTS	admin	4	3.0	1	320.1
moved-lemur.maas	Unknown	Failed deployment	admin	4	4.0	2	2320.5
neat-crane.maas	Unknown	Ubuntu 18.04 LTS	admin	4	3.0	1	320.1
open-chimp.maas	Unknown	Ubuntu 18.04 LTS	admin	4	4.0	2	2320.5
picked-rabbit.maas	Unknown	New		0	0.0	0	0.0
pretty-goose.maas	Unknown	New		0	0.0	0	0.0
proper-falcon.maas	Unknown	Ubuntu 18.04 LTS	admin	16	16.0	1	250.1
pumped-falcon.maas	Unknown	New		0	0.0	0	0.0
pumped-trout.maas	Unknown	Ubuntu 18.04 LTS	admin	4	3.0	1	320.1
rapid-kodiak.maas	Unknown	Ubuntu 18.04 LTS	admin	16	16.0	1	250.1
ruling-mantis.maas	Unknown	Ubuntu 18.04 LTS	admin	8	28.0	1	1000.2
safe-lynx.maas	Unknown	Ubuntu 18.04 LTS	admin	6	16.0	1	2000.4
simple-amoeba.maas	Unknown	Ubuntu 18.04 LTS	admin	16	16.0	1	250.1
up-moth.maas	Unknown	New		0	0.0	0	0.0
useful-cub.maas	Unknown	New		0	0.0	0	0.0
valid-dog.maas	Unknown	Ubuntu 18.04 LTS	admin	6	16.0	1	1000.2
vocal-gibbon.maas	Unknown	Ubuntu 18.04 LTS	admin	4	3.0	1	320.1

▲ 그림 15. Ubuntu MaaS 등록 서버 리스트 (참고1)

이름	호스트	Zone	Status	State	마지막 업데이트됨
nova-compute	aware-cougar	nova	활성화됨	Down	2주, 1일
nova-compute	vocal-gibbon	nova	활성화됨	Up	0분
nova-compute	bright-colt	nova	활성화됨	Up	0분
nova-compute	pumped-trout	nova	활성화됨	Up	0분
nova-compute	daring-fly	nova	활성화됨	Up	0분
nova-compute	enough-shark	nova	활성화됨	Up	0분
nova-compute	fleet-feline	nova	활성화됨	Up	0분
nova-compute	neat-crane	nova	활성화됨	Up	0분
nova-compute	safe-lynx	nova	활성화됨	Up	0분
nova-compute	valid-dog	nova	활성화됨	Up	0분
nova-compute	rapid-kodiak	nova	활성화됨	Up	0분
nova-compute	bold-insect	nova	활성화됨	Up	0분
nova-compute	simple-amoeba	nova	활성화됨	Up	0분
nova-compute	proper-falcon	nova	활성화됨	Up	0분
nova-conductor	ruling-mantis-nova-api-container-e03d736b	internal	활성화됨	Up	0분
nova-conductor	grand-quagga-nova-api-container-432fddd4	internal	활성화됨	Up	0분
nova-consoleauth	ruling-mantis-nova-api-container-e03d736b	internal	활성화됨	Up	0분
nova-consoleauth	grand-quagga-nova-api-container-432fddd4	internal	활성화됨	Up	0분
nova-scheduler	ruling-mantis-nova-api-container-e03d736b	internal	활성화됨	Up	0분
nova-scheduler	grand-quagga-nova-api-container-432fddd4	internal	활성화됨	Up	0분

▲ 그림 16. OpenStack Compute node 리스트 (참고2)

이름	호스트	Zone	Status	State	마지막 업데이트됨
cinder-scheduler	ruling-mantis-cinder-api-container-b34723e0	nova	활성화됨	Up	0분
cinder-volume	grand-quagga-cinder-volumes-container-fc9390ec@RBD	nova	활성화됨	Up	0분
cinder-volume	ruling-mantis-cinder-volumes-container-b8a3ebb9@RBD	nova	활성화됨	Up	0분
cinder-scheduler	grand-quagga-cinder-api-container-e14892a2	nova	활성화됨	Up	0분

▲ 그림 17. OpenStack Storage node 리스트 (참고3)

유형	이름	호스트	Zone	Status	State	마지막 업데이트됨
Linux bridge agent	neutron-linuxbridge-agent	daring-fly	-	활성화됨	Up	0분
DHCP agent	neutron-dhcp-agent	ruling-mantis	nova	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	pumped-trout	-	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	ruling-mantis	-	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	valid-dog	-	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	bright-colt	-	활성화됨	Up	0분
Metadata agent	neutron-metadata-agent	ruling-mantis	-	활성화됨	Up	0분
Metering agent	neutron-metering-agent	ruling-mantis	-	활성화됨	Up	0분
L3 agent	neutron-l3-agent	ruling-mantis	nova	활성화됨	Up	0분
Metadata agent	neutron-metadata-agent	grand-quagga	-	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	fleet-feline	-	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	vocal-gibbon	-	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	safe-lynx	-	활성화됨	Up	0분
Metering agent	neutron-metering-agent	grand-quagga	-	활성화됨	Up	0분
DHCP agent	neutron-dhcp-agent	grand-quagga	nova	활성화됨	Up	0분
L3 agent	neutron-l3-agent	grand-quagga	nova	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	aware-cougar	-	활성화됨	Down	3주, 3일
Linux bridge agent	neutron-linuxbridge-agent	enough-shark	-	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	grand-quagga	-	활성화됨	Up	0분
Linux bridge agent	neutron-linuxbridge-agent	neat-crane	-	활성화됨	Up	0분

▲ 그림 18. 네트워크 에이전트 (참고4)

6.2 서버 인프라 구성에 대한 근거

(1) 서버 역할 별 이중화 HA 구성을 진행한 이유

본 논문에서는 서버 역할 별로 HA(고 가용성, High Availability)구성을 위하여 이중화를 진행하였다. HA 란 서버와 네트워크, 프로그램 등의 정보 시스템이 상당히 오랜 기간 동안 지속적으로 정상 운영이 가능한 성질을 말하며, '가용성이 높다'는 뜻으로서, 서비스 중단 시간을 획기적으로 줄여주는 것을 의미한다. HA 는 흔히 가용한 시간의 비율을 99%, 99.9% 등과 같은 퍼센테이지로 표현하는데, 1 년에 계획 된 것을 제외하고 5 분 15 초 이하의 장애시간을 허용한다는 의미의 5 nines, 즉 99.999%는 매우 높은 수준으로 고품질의 데이터센터들이 목표로 하고 있는 지표이다. 하나의 정보 시스템에 HA 가 요구된다면, 그 시스템의 모든 부품과 구성 요소들은 적합하고 알맞게 설계되어야 하며, 실제로 사용되기 전에 완벽히 시험되어야 한다.

HA 솔루션을 이용하면, 각 시스템 간에 공유 디스크를 중심으로 집단화 하여 클러스터로 엮어지게 만들 수 있다. 동시에 다수의 시스템을 클러스터로 연결할 수 있지만 주로 2 개의 서버를 연결하는 방식을 많이 사용한다. 만약 클러스터로 묶인 2 개의 서버 중 1 대의 서버에서 장애가 발생할 경우, 다른 서버가 즉시 그 업무를 대신 수행하므로, 시스템 장애를 불과 수 초에서 수 분 안에 복구할 수 있다.

HA 솔루션은 서비스 가용성을 높이기 위한 솔루션으로 기본적으로 데이터 무결성과 장애 감시 기능이 요구된다. 데이터 무결성은 기본적으로 HA 구성된 서버 중 1 번 서버가 장애가 발생시, 2 번 서버가 대상 서비스를 바로 서비스를 하기 위해서는 양쪽의 데이터는 항상 100% 동일 해야 하는 무결성을 보장 해야 한다. 이와 같이 데이터를 동일하게 맞추기 위해서는 데이터 복제(data replication)기능이 반드시 필요하다. 또한 장애 감시 기능의 경우, 앞에서 설명한 서버 HA 구성시 1 번 서버는 서비스 운영을 맞게 되며, 2 번 서버는 1 번 서버가 장애 발생시 서비스를 운영하기 위한 대기 상태로 구성이 된다. 이러한 구성은 Active-Stand By HA 구성이라고 한다.

장애 발생은 크게 네트워크 장애, OS 및 서비스 프로그램(프로세스) 장애, 서버 하드웨어 장애로 나눌 수 있다. 장애 감시 Agent 는 각 서버에 설치가 되며 이러한 Agent 는 1 번 서버와 2 번 서버에 각각 설치되어, 자기 자신의 서버의 장애 포인트 감시 및 크로스로 1 번 서버는 2 번 서버를 감시, 2 번 서버는 1 번 서버를 상호 감시(Heartbeat Check) 하고, 문제가 발생할 시 자동 전환 기능(fail over)을 이용하여 HA 를 유지하게 된다. 같은 장비가 두 대 탑재되므로 비용이 두 배로 발생하는 문제가 존재하지만, 본 논문의 경우 실습실 환경의 안정성을 유지하기 위해서는 HA 가 필수 조건이며, 이를 위해 HaProxy 와 Infra node 에 HA 구성을 진행하게 되었다.

(2) Load balancer 의 선택 이유 (HAProxy)

Load balancer 란, network 를 사용하는 서비스에서 발생되는 트래픽이 상당히 높아 한 대의 서버에서 처리 불가능한 수준일 경우, 여러 대의 서버가 분산처리하여 서버의 load 증가, 부하량, 속도 저하 등을 고려하여 적절히 분산처리하여 해결해주는 장비(혹은 서비스)를 의미한다.

IT 관련 업체에서는 TCP 혹은 UDP 에 대한 load balancing 을 진행하기 위해서는 Layer 4(이하 L4) 스위치를 사용한다. L2 와 L3 대비 HA(High Availability)지원이 되기도 하고, 포트 기반의 filtering, mirroring, load balancing 등이 지원되며, 무엇보다 ASIC 기반의 빠른 속도로 운영되면서 L7 대비 저비용으로 활용이 가능하기 때문이다.

L7 스위치의 경우, L4 대비 고비용이면서 CPU 처리 속도에 따른 성능 저하가 있지만, URL(혹은 contents) 기반의 load balancing 과 모든 network layer 에 대한 filtering 을 진행할 수 있다는 장점을 보유하고 있다.

	L4	L7
목적	Layer 4 까지의 load balancing 및 HA 지원.	URL(Contents)기반 load balancing 및 HA 지원
특징	포트 기반 filtering, mirroring, load balancing 등	L4 Switch 기능 및 컨텐츠 기반 제어
성능	동시 연결 수, 처리 용량, 임계치	동시 연결 수, 처리 용량, 임계치
보안 기능	Layer 4 까지의 보안 기능 지원	URL(Contents)기반 보안 기능 지원
장점	ASIC 기반 빠른 속도	URL
단점	서비스 유무만 확인 Contents 기반 제어 불가	L4 대비 고비용 CPU 처리 성능 저하
활용 사례	SLB, FWLB, VPNLB	(L4 Switch 기능 포함) 및 URL 기반의 load balancing, filtering

▲ 그림 19. L4 스위치와 L7스위치 간략 비교

L4 스위치와 L7 스위치의 가격은 생산 업체별로 차이가 굉장히 크지만, L4 스위치의 경우 한 대당 평균 10,000,000 원의 비용을 지불해야 하며, L7 스위치의 경우 한 대당 58,000,000 원 ~ 330,000,000 원을 지불하여야 한다(Cisco 제품 기준).

본 논문에서는 실습실 환경을 위한 구성을 진행하였으므로, 모든 contents 에 대한 제어를 진행하기 위해 L7 스위치가 반드시 필요한 상황이나, 비용적인 문제로 인하여 탑재가 불가하였다. 따라서 대안으로 소프트웨어 스위치로 이중화 구성을 진행하게 되었다.

소프트웨어 스위치는 대표적으로 Nginx 와 HAProxy 가 사용된다. Nginx 의 경우, 대표적인 웹서버인 Apache 의 문제점을 해결하면서 만들어진 웹서버로, 비동기 방식으로 개발되어 가볍고 빠른 것으로 유명한 오픈소스 어플리케이션이다. Nginx 는 http 나 reverse proxy 기능 외에도 load balancer 기능 또한 강력하다. Nginx 로드 밸런싱을 지원하며, 연결이 가장 적은 서버로 트래픽을 전달하는 역할을 진행해준다. 하지만 Nginx 의 경우 health-check 기능이 지원되지 않으므로, 실제 서버의 상태를 체크하는 API 를 만들어서 상황에 따라 스위칭하는 고가용성을 위한 동작에는 용이하지 않다.

HAProxy(High Availability Proxy)는 웹 서버가 들어오는 요청을 여러 end-point 로 분산 할 수있게 해주는 TCP / HTTP load balancer 및 proxy server 이다. HAProxy 는 너무 많은 동시 연결이 단일 서버의 기능을 과포화하는 경우에 유용하다. 클라이언트가 모든 요청을 처리하는 단일 서버에 연결하는 대신 클라이언트는 HAProxy instance 에 연결한다. 이 인스턴스는 reverse proxy 를 사용하여 load balancing algorithm 에 따라 요청을 사용 가능한 end-point 중 하나로 전달한다. 우리가 브라우저에서 사용하는 proxy 는 클라이언트 앞에서 처리하는 기능으로, forward proxy 라 한다. reverse proxy 의 역할을 간단히 설명하면, 실제 서버 요청에 대해서 서버 앞 단에 존재하면서, 서버로 들어오는 요청을 대신 받아서 서버에 전달하고 요청한 곳에 그 결과를 다시 전달하는 것이다.

HAProxy 는 기본적으로 VRRP(Virtual Router Redundancy Protocol)를 지원한다. HAProxy 의 성능상 초당 8 만 건 정도의 연결을 처리해도 크게 무리가 없지만, 소프트웨어 기반의 솔루션이기 때문에 HAProxy 가 설치된 서버에서 문제가 발생하면 하드웨어 기반의 L7 스위치 보다 불안정할 수 있다. 하지만 기존의 L7 스위치에 비해 굉장히 저렴한 비용과 더불어 구성에 대한 어려움을 탈피하며 코드로 모든 설정을 제어할 수 있는 점이 장점이라 할 수 있다. 또한 기존의 L7 스위치의 경우 Master-Slave 방식의 이중화 구성이 비용적으로나 기술적으로나 어려움에 비해, HAProxy 의 경우 공식적으로 본 방식을 지원하므로 구현이 상대적으로 쉽다. 따라서 Load balancer 는 HAProxy 를 사용하여 구성하게 되었다

7. 클라이언트 구성에 대한 결과

위 서버 인프라 구성에 접속하여 가상 데스크탑 환경을 사용하기 위한 클라이언트(이하 ThinClient)는 Raspberry Pi를 이용하여 제작하였다. ThinClient에는 다음과 같은 내용이 포함된다 :

- Ubuntu Desktop 18.04.2 Stable version을 Main OS로 사용한다.
- ThinClient는 서버 환경의 Instance에 연결하여 원격으로 주어진 환경을 사용할 수 있다.
- 사양은 관리자가 OpenStack Dashboard에 접속하여 자유롭게 변경할 수 있다.
- OS가 부팅된 이후, 실습실 서버로 instance 상태를 Power On 상태로 변경하고, Instance가 정상적으로 On 되었음을 확인한 이후 특정 instance에 RDP Protocol로 접속시킨다.

기본적으로 다음과 같은 내용들을 기반으로 제작을 진행하였다 :

- RDP Server로 연결이 가능하여야 한다.
- Keyboard input, Mouse input등이 가능하여야 한다.
- 최대한 가볍게 제작하여 ThinClient에서 부하가 일어나는 경우가 없도록 제작한다.

ThinClient 출고 전에는 다음의 bash 스크립트를 사용자 루트 디렉토리[~/] 에 작성하여 두고, 이를 root권한으로 실행할 수 있도록 한다.

```
#!/bin/bash


### Are you a root user?
if [[ $EUID -ne 0 ]]; then
        echo "[SYSTEM]You must be a root
user." 2>&1
        exit 1
fi


### Is network connection ok?
while true; do
        echo -e "GET http://google.com
HTTP/1.0\n\n" | nc google.com 80 >
/dev/null 2>&1
        if [ $? -eq 0 ]; then
                break
        else
                echo "[SYSTEM]We could not
verify your internet connection. Try again
in 5 seconds." && wait
                sleep 5 && wait
        fi
done

echo "[SYSTEM]Internet connection is
verified!" && wait



# SETTING : Change
repository(archive.ubuntu.com ->
mirror.kakao.com)
sed -i
's|arhive.ubuntu.com|mirror.kakao.com|g'
/etc/apt/sources.list && wait

# SETTING : Update repository
apt-get update && wait

# SETUP : Install ubuntu-desktop
apt-get install --no-install-recommends
ubuntu-desktop -y && wait

# SETTING : Change root password
echo 'root:****' | chpasswd && wait

# SETTING : Auto login setting
sed -i.bak -e '3d' /etc/pam.d/gdm-password
&& wait
sed -i.bak -e '3d' /etc/pam.d/gdm-autologin
&& wait
echo -e
"[daemon]\nAutomaticLoginEnable=true\nAutom
aticLogin=root\n\n[security]\nAllowRoot=tru
e\n\n[xdmcp]\n\n[chooser]\n\n[debug]\n" >
/etc/gdm3/custom.conf && wait


### … 부분 생략 … ###

# SETTING : Autostart this
application(test)
```

```
mkdir ~/.config/
mkdir ~/.config/autostart/
touch ~/.config/autostart/vdi.desktop
echo -e "[Desktop
Entry]\nType=Application\nName=SimpleVDI\nE
xec=/home/ubuntu/start.sh\nX-GNOME-
Autostart-enabled=true" >
~/.config/autostart/vdi.desktop

# SETTING : [20190603][BUG]mesg: ttyname
failed
echo -e "tty -s && mesg n" > /root/.profile
&& wait

# SETTING : Power saving(blank screen)
gsettings set org.gnome.desktop.session
idle-delay 0

# SETTING : Automatic suspend
sudo systemctl mask sleep.target
suspend.target hibernate.target hybrid-
sleep.target

# SETTING : Automatic screen lock
gsettings set org.gnome.desktop.screensaver
lock-enabled false

# Install Complete
touch $CheckFile && wait
echo '[SYSTEM]Rebooting system(after 5
seconds).'
sleep 5 && wait

reboot
```

위 스크립트가 실행되면 OpenStack instance에 접근이 가능한 config 파일이 생성된다. 이후 다음의 코드로 RDP 접속을 진행한다(pseudo code 포함).

```
#!/bin/bash

# Config 파일이 존재하는가?

# Config 파일에 기재되어 있는 OpenStack
instance 로 접근이 가능한가?

# Start RDP program
```

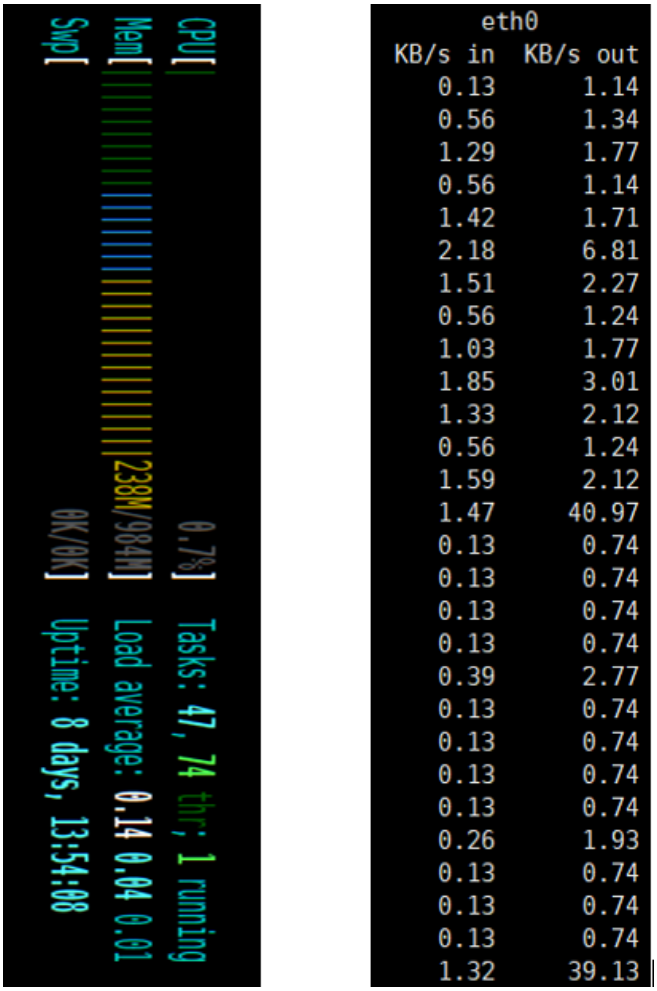
```
echo Y | ./client /v:${config::host}
/port:${config::port} /u:Administrator
/compression /gfx /rfx /k:${config::key} /f
/sound +auto-reconnect && wait

# ThinClient 가 종료되면 OpenStack
instance 에 초기화 명령을 내린다.

# Shutdown thinclient
shutdown -h now
```

위와 같은 내용을 기반으로 OpenStack 서버 인프라 내 특정 instance와 ThinClient가 RDP 프로토콜로 통신하며 가상 환경을 사용할 수 있게 된다.

결과적으로, 클라이언트는 장기 사용 과정(8 일 13 시간 연속 사용)에서도 굉장히 낮은 load 수치를 띄었으며, 사용 트래픽 역시 굉장히 낮은 수치를 보였다.



▲ 그림 20. 성능 분석

7. 결론 및 향후 연구

7.1 결론

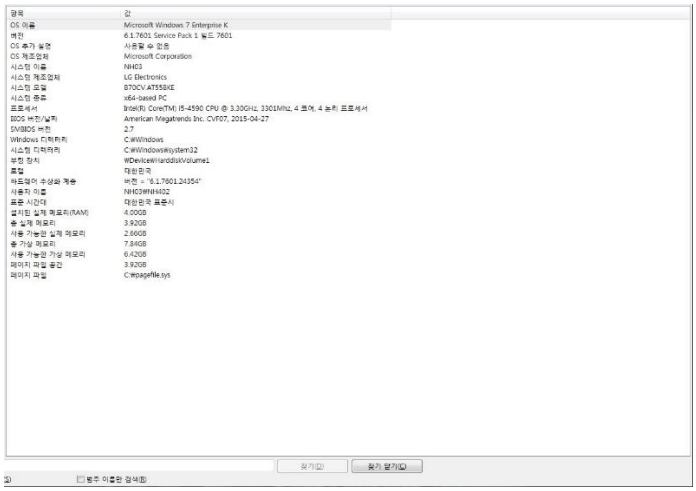
가상화는 단일 물리적 자원을 논리적으로 분할하여 여러 개의 컴퓨팅 환경을 구성할 수 있다. 또한 물리적 자원을 통합하여 사용할 수 있는 기술로, 다양한 분야에서 사용이 가능하다. 본 논문에서는 실제로 사용할 수 있는 실습실 환경에 대한 서버 구성 및 클라이언트 구성을 진행하였다. 또한 서버 구조와 클라이언트의 안정성과 기능성 면에서 적절하다는 결과를 볼 수 있었다.

7.2 향후 연구 과제

(1) 기존 환경 개선

현재는 구형 CPU를 탑재한 PC를 활용하여 가상화 node를 구성하였다. 예전 실습실에서 사용하던 Samsung i5 4세대 컴퓨터들을 가지고 가상화 node를 구성하여야 하니 느릴 수 밖에 없었다. 또한 Disk에 대한 raid를 엮지 못하여 디스크 안정성에 대한 문제도 확인된다.

이에 대한 해결방안으로, HardDisk를 SSD로 교체하고, 고 사양 서버 몇 대를 node로 엮는다면 더욱 좋은 performance를 낼 수 있을 것이라 기대된다.



▲ 그림 21. 건국대학교 구형 실습실 컴퓨터 SPEC

(2) OverSpec문제 해결

현재 구현한 모델이 OverSpec이라는 가정하에보다 compact한 모델의 설계가 필요하다고 판단된다.

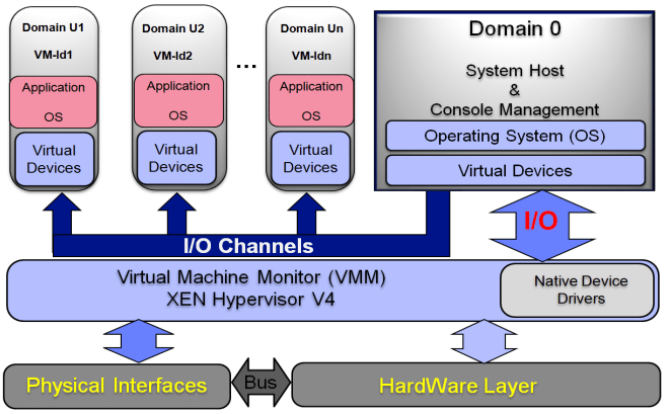
클라이언트 부분에서, ThinClient의 단가를 낮추기 위해 더 낮은 사양의 single board computer를 적용하되, 기존 모델의 성능을 그대로 유지하거나 더 좋은 성능을 가질 수 있도록 연구 과제로 남긴다. 대표적인 낮은 사양의 single board computer로는 Asus Tinker Board가 있다.

▲ 그림 22. Asus Tinker Board [16]

또한, 더욱 빠른 부팅과 RDP접속을 위하여 더 가벼운 형식의 TinyCore OS로 변경하거나, OS가 존재하지 않는 ZeroClient(제로 클라이언트) 방식을 채택하되, 기존 모델의 성능을 그대로 유지하거나 더 좋은 성능을 가질 수 있도록 연구 과제로 남긴다.

▲ 그림 23. Tiny OS [17]

서버 쪽에서는, 가상화 환경을 더욱 가볍게 운영하기 위해서 Xen Hypervisor를 이용하여 자체 제작한 Web management service를 탑재하되, 기존의 성능을 유지하거나 더 좋은 성능을 낼 수 있도록 연구한다.

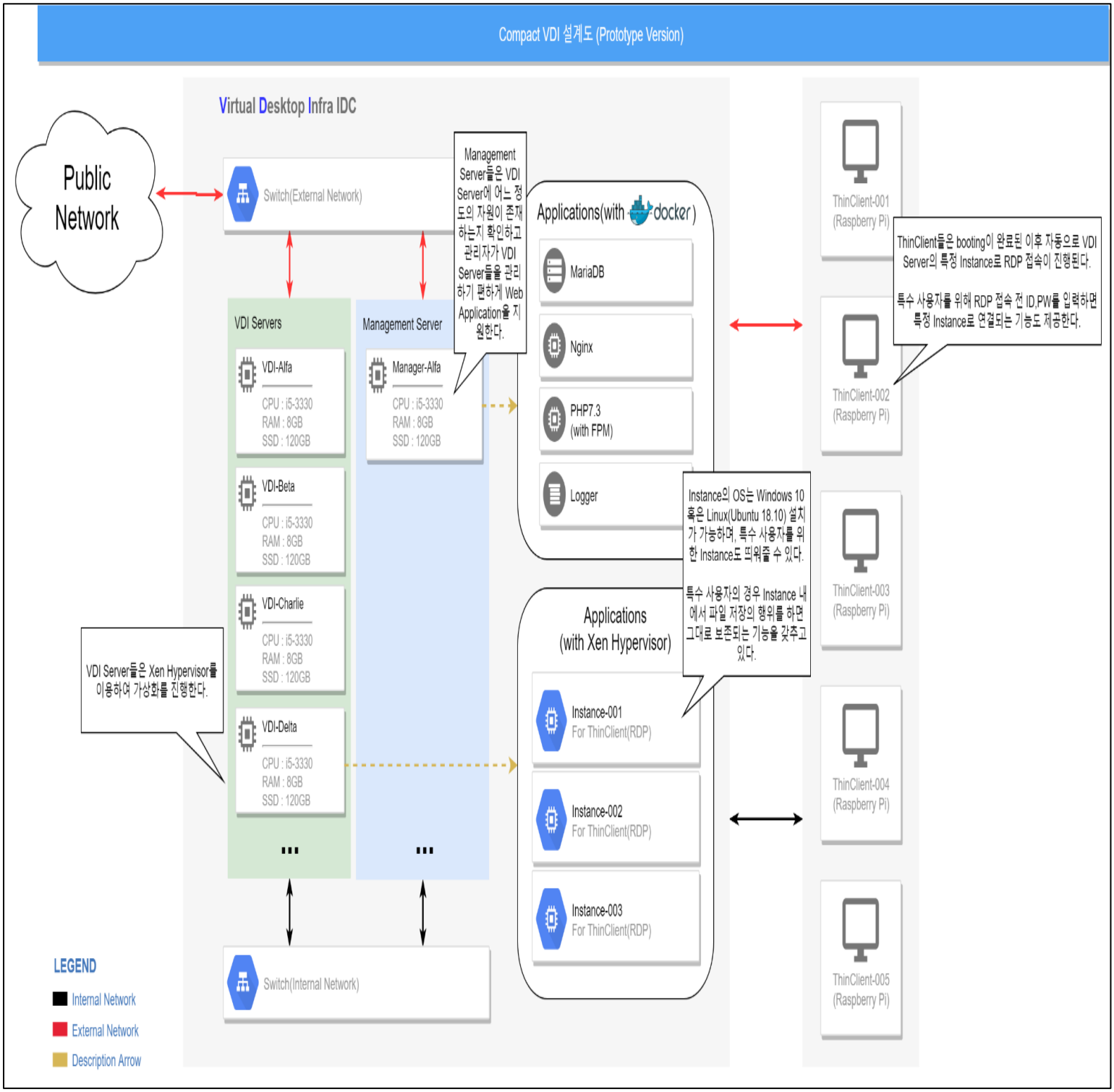


▲ 그림 24. Xen Hypervisor Architecture [18]

Compact버전의 VDI 설계는 다음과 같이 진행한다. 실습실 환경은 고유하기 때문에, node를 굳이 나눌 필요는 없으므로 기존에 존재하던 Infra, Compute, Storage node들을 없애고 하나의 node로 통합한다.

이어서, node들에 미리 instance들을 띄워두고, ThinClient가 사용하지 않고 있는 instance에 접속하여 해당 Desktop 환경을 사용하는, 간단한 환경이다.

최종적으로, 시스템을 최대한 Compact하고 가볍게 만들면서, 기존과 동일한 성능을 내거나 그보다 더 좋은 성능을 낼 수 있도록 하는 것을 연구 과제로 삼는다.



▲ 그림 25. Compact VDI 설계도

참고문헌

[1] 정보통신산업진흥원, 클라우드 컴퓨팅 서비스 플랫폼 기술 동향, 최성(남서울대학교 컴퓨터공학과 교수)
[2] 한국정보과학회, PathSavanna: Xen 기반 가상 라우터에서의 GPGPU를 이용한 실제적인 패킷 라우팅, 박근영외 2인
[3] 한국정보과학회, Desing and Implementation of HPC Facility, Dereje Ragassa
[4] 위키백과, “라즈베리파이“, https://ko.wikipedia.org/wiki/라즈베리_파이
[5] 위키백과, “클라우드컴퓨팅“, https://ko.wikipedia.org/wiki/클라우드_컴퓨팅
[6] 위키백과, “네트워크스위치“, https://ko.wikipedia.org/wiki/네트워크_스위치
[7] 나무위키, “컴퓨터실“, https://namu.wiki/w/컴퓨터실
[8] “AWS-애저-구글, 기업용 클라우드 플랫폼 최강자는 누가 될까?“, CIO, 2017.09.22, https://www.ciokorea.com
[9] “전 시트릭스 CTO의 역설적 주장 ’가상화로 보안문제 해결’“, CIO, 2011.07.05, https://www.ciokorea.com
[10] KenseiIT, “OpenStack“, https://kensei.co.kr/469
[11] Novell, “Zen hypervisor“, https://www.novell.com
[12] Raspberry Pi, “Raspberry Pi“, https:// www.raspberrypi.org
[13] SW중심사회, “SW교육의무화“, https://software.kr
[14] “중고교 국내 컴퓨터실 보유 현황및 문제점“, 뉴스줌, 2015.04.22, http://news.zum.com/
[15] STADIA, “About“, https://www.stadia.dev/
[16] ASUS, “Single Board Computer”, https://www.asus.com/
[17] 위키백과, “TinyOS”, https://ko.wikipedia.org/wiki/TinyOS
[18] ReseachGate, “Xen Hypervisor Architecture”, https://www.researchgate.net