# 클라우드 컴퓨팅 환경의
# 웹 서비스 개발 가이드

정보통신산업진흥원 부설
nipa SW공학센터

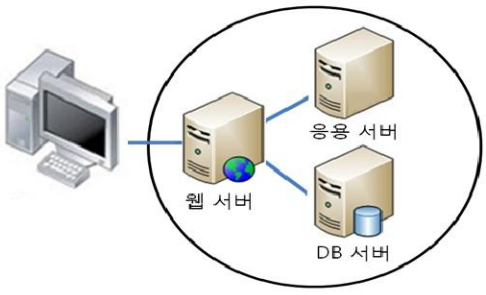
# 목 차

# 1. 클라우드 컴퓨팅 환경

## 1.1. 클라우드 컴퓨팅의 개요

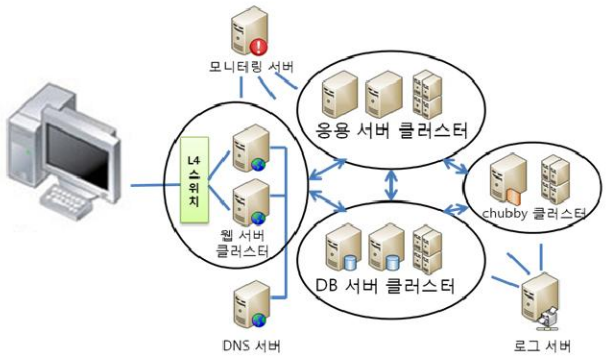
클라우드 컴퓨팅이란 한 마디로 정의하기가 매우 까다로운 용어로서 개발자뿐만 아니라 시스템 관리자, 하드웨어 제공업체, 심지어 사용자까지 포함되는 매우 광범위한 개념으로서의 컴퓨팅 서비스를 의미한다. 모바일 관점에서 보면 단순한 모바일 웹 서비스를 모바일 클라우드라고 말하기까지 한다. 틀린 것은 아니지만 모바일 웹 서비스라고 해서 모두 클라우드 컴퓨팅이라고 하지는 않는다.

일반적으로 클라우드 컴퓨팅에서 사용되는 기술들은 컴퓨팅에 필요한 자원을 공유, 분배하여 사용하는 기술들이다. 또한 그 메커니즘을 클라이언트에서 보기엔 구름처럼 흐릿하고 애매하게 보인다고 하여 클라우드(cloud)라고 불리우기도 한다. 하지만 클라우드 컴퓨팅에서 말하는 개념이나 구성하는 요소들이 과거에 없었던 것은 아니다. 인터넷 상에서 개인용 스토리지 서비스를 제공하기도 했고 ASP(Application Service Provider)라는 개념도 있었다. 이런 과거의 각각의 서비스들을 클라우드라고 부르지는 않았다. 클라우드는 새로운 개념은 아니지만 이미 존재하던 시스템이 가져야할 특징들에 대해서 과거보다 더 높은 수준과 각 요소간의 긴밀한 융합을 요구하는 컴퓨팅이라 할 수 있다. 클라우드 컴퓨팅이란 이와같은 다양한 각각의 요소들을 모두 포괄하는 개념이라고 할 수 있다.




| 단순 웹 서비스 환경 |
| --- |
| ● 구성 요소가 적음 |
| ● 성능과 용량이 제한적 |
| ● 유지보수의 범위가 전체 |
| ● 구축이 쉽고 간단함 |
| ● 서버 구매 비용이 높음 |

[그림 1 : 웹 서비스 환경의 변화 - 단순 웹 서비스]




| 분산 클러스터 컴퓨팅 환경 |
|---|
| ● 구성 요소가 많음 |
| ● 성능과 용량이 확장성을 가짐 |
| ● 유지보수의 단위가 분리됨 |
| ● 구축이 어렵고 복잡함 |
| ● 구축 비용이 매우 높음 |

[그림 2 : 웹 서비스 환경의 변화 - 분산 클러스터 컴퓨팅 ]


| 클라우드 컴퓨팅 환경 |
|---|
| ● 구성 요소를 헤아릴 수 없음 |
| ● 성능과 용량 제한이 거의 없음 |
| ● 유지보수의 단위가 세분화됨 |
| ● 기존 구축 환경을 빌려서 사용 |
| ● 사용량 과금으로 비용이 적음 |

[그림 3 : 웹 서비스 환경의 변화 - 클라우드 컴퓨팅]

가트너 자료에 의하면 '클라우드 컴퓨팅은 인터넷 기술을 이용하여 내외부 고객들에게 확장성있고 탄력적인 IT 서비스가 제공되는 방식이다(Cloud is a style of computing where scalable and elastic IT-related capabilities are provided as a service to customers using Internet technologies.)'라고 정의되고 있다[1]. 여기서 확장성과 탄력성은 단순히 늘어나는 것만이 아니라 줄어들 수도 있음을 의미하며, 이를 위해서는 컴퓨팅 자원이 무한대로 늘어나거나 필요한 수준만큼 마음대로 줄일 수 있어야 한다. 또한 이러한 작업이 수분~수십분 이내로 진행이 가능하다는 것이 클라우드 컴퓨팅 서비스의 큰 특징이라고 할 수 있다.

## 1.2. 클라우드 컴퓨팅의 종류

클라우드 컴퓨팅은 일반적으로 제공 방식과 제공 대상에 의해 분류될 수 있다.

### 1.2.1. 제공 방식에 의한 분류

어떤 방식의 서비스를 제공하느냐에 따라서 클라우드 서비스는 SaaS , PaaS , IaaS로 나눌 수 있다.

### ● SaaS (Software as a Service)

사용자가 네트워크로 접속하여 메일, CRM, 워드프로세스 등과 같은 애플리케이션을 사용할 수 있는 서비스를 말한다. 서비스 사용자는 실행되는 소프트웨어를 설치할 필요가 없으며, 실행되는 컴퓨터나 저장장치에 대해서도 알지 못해도 된다. 이와같은 환경들은 소프트웨어 제공자에 의해 관리된다. 소프트웨어를 설치하는 것이 아니라 서비스 형태로 제공한다는 의미에서 소프트웨어 서비스(SaaS)라고 불리운다. 세일즈포스닷컴이 대표적인 예이다.

### ● PaaS (Platform as a Service)

사용자(개발자)는 애플리케이션이나 서비스가 실행되는 환경을 서비스 제공자로부터 제공받아 사용한다. 제공자로부터 제공받지 않는 실행 환경에 대해서는 제한적이지만, 제공되는 API 등으로 확장성이 가능한 실행환경이 표준화되기 때문에 이를 통해 여러 가지 다양하고도 서로 연동하는 어플리케이션들의 개발이 가능하게 된다. 구글의 App엔진과 같은 플랫폼 서비스(PaaS)가 여기에 해당된다.

### ● IaaS (Infrastructure as a Service)

서버, 스토리지, 데이터베이스 등과 같은 IT 자원을 서비스 형태로 제공한다. 위의 두 서비스와는 달리 개인 사용자에게 직접적인 서비스를 하기보다는 위의 두 서비스를 공급하는 제공자들이 관련 시스템을 구축할 수 있도록 자원을 할당해 주는 서비스로서 각 자원에 대해 사용한 만큼만 비용을 지불하도록 하는 경제적인 서비스이다. 아마존의 S3(Simple Storage Service)가 대표적인 예이다.


[그림 4 : 제공 방식에 의한 클라우드 계층 분류와 그 대표적인 서비스][2]

### 1.2.2. 제공 대상에 의한 분류

클라우드 컴퓨팅을 분류하는 또 다른 방법으로 제공 대상에 따른 분류를 사용하기도 한다. 클라우드 서비스 자원을 어디에 위치시키고 누구를 위해 서비스를 시행할 것인가로 다음과 같이 분류한다.

● **퍼블릭 클라우드 (Public Cloud)**

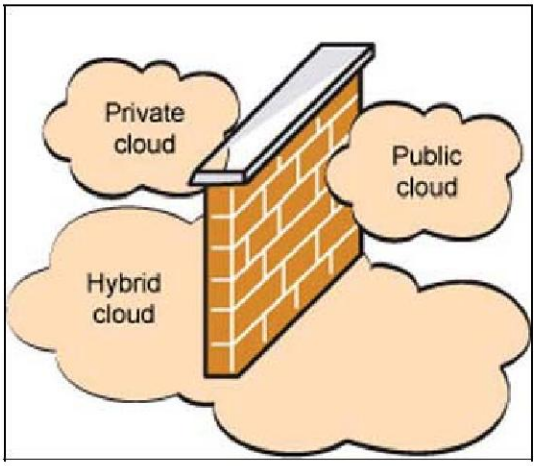
특정 기업이나 사용자를 위한 서비스가 아닌 인터넷에 접속 가능한 모든 사용자를 위한 클라우드 서비스다. 모든 사용자를 위한 서비스이지만 서비스 내부에 저장된 데이터나 서버와 같은 자원은 각 서비스에서 사용자별로 권한 관리가 되거나 격리되어 사용자 간에는 전혀 간섭이 없다.

● **프라이빗 클라우드 (Private Cloud)**

특정 기업이나 특정 사용자만을 대상으로 하는 클라우드 서비스다. 컴퓨팅 자원과 클라우드 내에 저장되는 데이터는 기업 내부에 저장되기 때문에 자원의 제어권을 기업 자체에서 가지고 있으며, 물리적인 데이터 보안 측면이 퍼블릭 클라우드보다 강조되었다.

● **하이브리드 클라우드 (Hybrid Cloud)**

퍼블릭 클라우드와 프라이빗 클라우드를 병행하여 사용하는 클라우드 서비스다. 데이터 보안이 중요하거나 컴퓨팅 자원에 대한 제어를 가져야 하는 서비스나 시스템은 프라이빗 클라우드를 사용하고 그렇지 않은 경우에는 퍼블릭 클라우드를 사용하여 연동하는 방식이다.




[그림 5 : 제공 대상에 의한 클라우드 분류 방식][3]

기업에서 클라우드 컴퓨팅을 적용할 때에는 클라우드 컴퓨팅의 개념이 서버 가상화 기반의 서버 통합과 구분하기가 어렵다는 것을 숙지해야 한다. 프라이빗 클라우드를 도입할 경우, 일반적으로 기업 내부 시스템의 자원은 거의 변화가 없으며 서비스 자체가 측정이 가능하여 통계 관리가 이미 이루어지고 있을 뿐 아니라 자원의 분배가 가상 머신을 이용하여 할당이 가능하게 된다. 이럴 경우 프라이빗 클라우드와 서버 가상화 도입은 거의 차이가 없어진다고도 볼 수 있다. 그러나 서버 가상화의 경우는 추가적인 서버 확장에 대한 유연성이나 외부 네트워크에 대한 자원 분배를 지원하지 않으며, 일반적인 기업의 경우 전체적인 시스템의 구조를 변경하는 것보다 점진적으로 변경해나가는 것을 선호하기 때문에 클라우드 컴퓨팅의 모든 요소를 적용하는 경우는 드물다.

## 1.3. 클라우드 컴퓨팅 환경의 아키텍처

클라우드 컴퓨팅 환경의 장점은 서버, 스토리지, 데이터베이스와 같은 인프라를 가상화하여 그 전체 시스템 구조와 내부를 알지 못하더라도 어플리케이션 개발자와 사용자가 서비스를 손쉽게 서비스를 개발하고 이용할 수 있다는 데에 있

다. 하지만 그 인프라를 이해한다면 더욱 최적화된 서비스가 가능할 것이다. 컴퓨팅 시스템의 인프라 아키텍처가 클라우드로 진행하기 위해 단계별로 갖추어야 할 요소들은 다음과 같다.

### 1) 웹 어플리케이션 모듈화

단순한 웹 서비스의 경우 웹 서버 내부에 웹 어플리케이션을 상주시키는 경우가 대부분이다. 하지만 시스템 내부에서도 어플리케이션의 기능에 따라 모듈로 분리하는 아키텍처를 가져야 하는 것이 개발과 유지보수에 있어서 필수적이다. 분리된 모듈은 하나의 프로세스일 수도 있고 다른 프로세스로 분리되어 호출을 통해 정보를 주고 받을 수도 있다. 각각의 모듈은 일반적으로 부하와 성능을 고려하여 외부와의 통신과 데이터 I/O 작업 위주로 분류한다.

### 2) 서버 확장

서비스 사용자가 많아져 시스템에 부하가 증가하면 서버를 확장할 수 있어야 한다. 서버 확장은 각 모듈을 다른 서버에 두고 모듈간 호출은 원격 호출로 처리한다. 원격호출에는 RPC, HTTP와 같은 프로토콜을 이용한다. TCP, UDP와 같은 프로토콜을 이용할 수도 있으나, 이기종간 호출 혹은 내부망 호출에서 심각한 네트워크 트래픽이 발생할 수도 있으므로 주의해야 한다. 이를 방지하기 위하여 시스템 내부 기능 사이의 호출에는 최소한으로 간단한 프로토콜과 경량 서버 솔루션을 사용한다.

### 3) 서버 클러스터 구성

모듈별로 서버가 확장되었음에도 불구하고 시스템의 부하가 많아지면 하나의 서비스 기능에 대해서도 여러 대의 서버 클러스터를 구성한다. 서비스 클러스터를 구성하는 서버는 가상 머신도 고려할 수 있다. 일반적으로 간편한 부하 분산을 위해서 로드밸런서(Load Balancer)를 사용한다. 로드밸런서를 사용하면 부하가 증가함에 따라 쉽게 대응이 가능하다는 이점이 있다. 부하가 증가하는 서비스를 찾아 해당 서비스 클러스터에 서버를 추가로 투입하고 로드밸런서 설정만 변경하면 된다. 이와같이 부하 상황에 따라 서버를 간단히 증가하거나 줄일 수 있는 것도 클라우드 컴퓨팅의 대표적인 특징이다.

### 4) 데이터 분산 저장

서버 클러스터를 통해 서비스 서버가 많아지게 되면 여러 서버들로부터 데이터

베이스 서버로의 접속이 집중되어 병목현상이 발생되게 된다. 또한 데이터 자체의 양도 증가하여 하나의 데이터베이스 서버로는 이를 감당할 수 없게 된다. 이 단계에서는 하나의 서비스에서 실행되는 데이터베이스가 아닌 여러 서버에서 데이터를 서비스하는 분산 데이터 관리 시스템과 분산 파일 시스템을 도입하여 서비스의 가용성을 높이고 데이터의 복제본을 생성하여 안정성을 높인다.

### 5) 캐시 구성

데이터의 읽기가 빈번하게 발생되는 서비스의 경우 캐시 계층을 추가함으로써 데이터 서비스의 부하를 줄이면서 서비스의 응답 속도를 향상시킬 수 있다. 어플리케이션 서버에서는 데이터 계층으로 데이터를 요청하기 전에 캐시 계층에 존재하는지 확인해 캐시에 있으면 바로 서비스하고 캐시에 없을 때만 데이터 계층으로 데이터를 요청해 캐시에 저장하고 서비스로 전달한다. 캐시를 사용하는 방법으로는 일반적으로 LRU(least recently used) 알고리즘을 많이 사용하지만, 서비스의 추세와 통계적 특성을 반영하여 여러 가지 다양한 알고리즘을 활용하는 편이다.

### 6) 외부 서비스 제공

여기까지 컴퓨팅 환경이 갖추어지면 충분히 클라우드 서비스의 특징을 갖추었다고 볼 수 있다. 하지만 자체적인 클라우드 서비스만이 아니라 여유가 있는 서비스 자원을 활용하여 다른 외부 서비스와 연동할 수도 있다. 특정 기능을 활용할 수 있는 API를 외부에 노출하여 다른 개발 업체와 협력해 기존에 없는 기능을 제공하거나, 다른 서비스와 연합한 서비스를 개발하는 것이 가능해진다. 서비스를 외부에 공개할 경우, 해당하는 서비스에 대해 보안이나 요금에 관련된 정책을 고려하도록 한다.

## 2. 클라우드 컴퓨팅 환경의 웹 서비스 기술

클라우드 환경의 근간을 이루는 기술 중에서 대표적인 것은 구글에서 발표한 분산 플랫폼 기술이다. 구글은 2004년부터 내부 시스템을 구성하는 시스템 소프트웨어의 구조를 논문으로 공개하기 시작했다.[4] 소스 코드 자체가 아닌 논문으로만 공개했음에도 불구하고 그 파급 효과는 매우 컸으며, 이것을 구현하려는 시도가 많이 이루어지면서 클라우드 컴퓨팅 환경의 구성을 위한 기반 기술들이 되었다.

구글은 공식적인 발표를 하지 않았지만 수십만대 이상의 리눅스 서버를 운영하고 있는 것으로 추측된다. (발표된 논문들을 통해 5000대 이상의 서버를 운영하고 있음을 예상할 수 있으며, 백만 대를 넘었을 것이라는 예상도 나오고 있다.) 또한 이 리눅스 서버들은 기업용 서버들보다 훨씬 저렴한 일반 가정용 PC에 가까운 사양을 가지고 있기 때문에 안정성이 매우 떨어진다. 구글은 이러한 하드웨어의 문제점을 소프트웨어로 해결하였으며, 그 성공 모델과 기술이 클라우드 컴퓨팅의 모태가 되었다.

구글에서 발표한 주요 논문에는 GFS[5], MapReduce[6], Bigtable[7], Chubby[8] 등이 있으며, 이를 구현하기 위해 야후를 비롯한 오픈 소스 진영에서 수많은 시도들이 이루어지며 여러 가지 기술들이 개발되었고, 또한 이 환경에서 동작하기 위한 어플리케이션을 개발하기 위한 시도들도 점진적으로 이루어지고 있다.

대표적인 구현 기술들은 다음과 같다.

- ▶ 분산 데이터 베이스 시스템 NoSQL
- ▶ 분산 파일 시스템 I/O 가상화
- ▶ 분산 시스템 운영 관리 기술
- ▶ 병렬 처리 기술 Mapreduce
- ▶ 로그 수집 기술
- ▶ 캐시 기술

보다 나은 클라우드 컴퓨팅 환경에서의 어플리케이션 구현을 위해 필요한 기술들은 이외에도 여러 가지가 있을 수 있으나 여기서는 위의 6가지 기술들에 대해서만 설명하도록 한다.[9]

## 2.1. 분산 데이터 베이스 시스템 NoSQL

관계형 데이터 베이스 시스템은 1970년에 IBM에서 개발된 이후 지금까지 수십 년에 걸쳐서 확고한 데이터 저장 솔루션으로서 유지되어왔다. 그러나 최근 인터넷과 하드웨어의 발전으로 인하여 검색 서비스, 소셜 네트워크 서비스, 데이터웨어하우스와 같은 다양한 서비스가 출현하였고, 그 사용자 수와 그 사용자들에 의해 생성되는 데이터의 양이 기하급수적으로 증가하였다. 이를 뒷받침하기 위해서 데이터에 대해 정합성이나 견고성보다는 손쉬운 확장성, 시스템 확장 과정이나 장애 상황에서도 서비스를 유지할 수 있는 고가용성, 보다 경제적인 비용이 강조되게 되었다. 결과적으로 확장성과 고가용성을 제공하면서도 표준 SQL이나 엔티티 간의 관계를 지원하지 않는 새로운 데이터 관리 시스템이 등장하게 되었으며, 이런 데이터 관리 시스템을 NoSQL이라고 부른다. 클라우드 컴퓨팅의 아키텍쳐적인 요구 사항을 NoSQL 솔루션이 만족시켜주고 있으므로 클라우드 컴퓨팅 플랫폼의 데이터 서비스 계층에 NoSQL을 많이 사용한다.

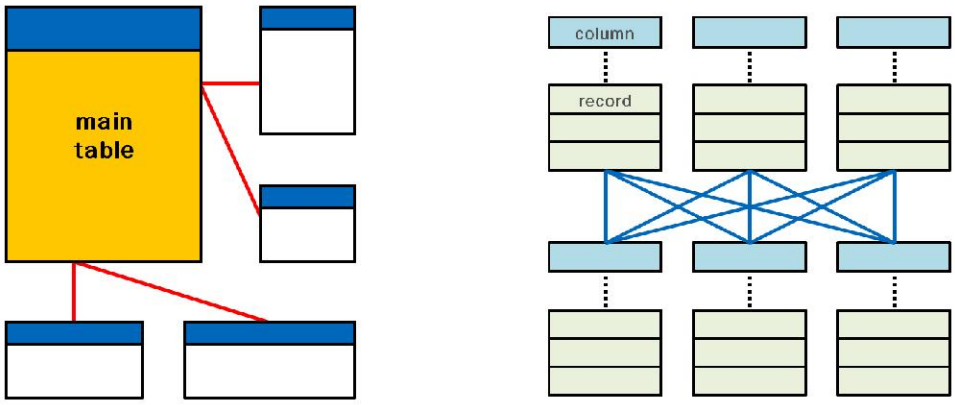
관계형 데이터베이스는 요즘도 구조적 데이터 저장소의 솔루션으로서 널리 사용되고 있으나, 글로벌하게 웹 서비스를 제공하는 구글, 야후, 페이스북 등과 같은 업체에서는 더이상 관계형 데이터 베이스만으로는 대용량 데이터 서비스를 효율적으로 제공할 수 없다는 것을 인식하고 서비스의 요구 사항에 부합되는 데이터 저장소를 별도로 개발해 사용하기 시작했다.

관계형 데이터베이스는 그 자체적으로는 기존의 데이터를 유지하면서 대규모의 데이터를 저장하기 위해 확장을 하기 어렵다. 물론 오라클, DB2 와 같은 상용 데이터베이스 솔루션과 테라데이타와 같은 기존의 분산 관계형 데이터베이스 솔루션에서는 페타바이트(petabyte)[1]이상으로 데이터 저장 공간을 확장할 수 있다. 하지만 구글, 야후, 페이스북 같은 인터넷 서비스 업체들은 대부분 오픈 소스 기반의 데이터 관리 시스템과 저가의 스토리지나 분산 파일 시스템 등을 이용하며, 이들 서비스가 저장 중인 대용량의 데이터는 고가의 솔루션에 저장할 수준의 고가용성을 요구하지 않는 경우가 많다. 이런 인터넷 서비스에서는 수십억건 이상의 데이터에 대해 join 같은 연산은 불필요하며, 데이터 정합성보다는 필요에 따라 데이터를 복제해서 안정적으로 보관하며, 장애 상황에서도 즉시 데이터

1) 1 petabyte = 1000 terabyte

를 조회할 수 있는 저장소가 필요했다.

또한 수십억건 이상의 데이터를 저장하고 있는 테이블에 대해 인덱스의 재구성, 칼럼의 추가 등과 같은 작업은 시스템이 운영 중인 상황에서는 현실적으로 불가능하다. 따라서 인덱스와 데이터가 분리돼 별도로 운영되며, 스키마에 대한 정의도 자유롭거나 스키마 자체가 없는 요구사항도 추가됐다. 이렇게 기존의 관계형 데이터베이스와는 다른 형태의 데이터 저장소에 대한 요구 사항이 증가하게 됐으며, 구글의 Bigtable, 아마존의 Dynamo 등과 같은 저장소가 발표되면서 NoSQL에 대한 관심이 증가했다. 그동안 특별한 용도로만 사용되던 일부 오픈소스들도 NoSQL로 분류되면서 NoSQL이 새로운 트렌드가 됐다.




[그림 6 : 관계형 데이터 베이스(좌)와 NoSQL(우)]

NoSQL은 아직 성숙된 기술 분야는 아니며, NoSQL이라는 용어 자체도 정확하게 정의되어 있지 않지만, Not Only SQL의 약어로서 일컬어지고 있다. 구조적인 데이터를 관리하는 기능을 수행하는 솔루션 중 관계형 데이터베이스가 아니라고 해서 모두 NoSQL로 분류할 수는 없다. 일반적으로 다음의 속성을 가지고 있는 데이터 저장소를 NoSQL의 범주로 분류한다.

- 관계형 데이터 모델이 아닌 키-값 또는 키-값을 응용한 데이터 모델
- 안정적이고 고가의 하드웨어가 아닌 다수의 값싼 하드웨어 이용
- 데이터는 분산된 노드에 복제되어 저장
- 데이터의 정합성(consistency)[2]보다는 단절 내결함성(partial-tolerance)[3] 중시

2) 모든 클라이언트는 항상 동일한 데이터를 보장받는 속성
3) 특정 노드의 네트워크가 단절되어도 데이터가 여러 노드에 나누어 저장되어 조회가 가능한 속성. 대신 데이터의 수정이 발생하면 정합성을 보장할 수 없음

- 2단계 커밋(2phase commit)[4]보다는 정족수(quorum)기반[5] 트랜잭션

데이터 모델에 따른 분류로는 키-값, 컬럼, 문서, 그래프로 구분할 수 있다. 키-값 모델은 가장 단순한 모델로, 키와 바이너리 타잎의 값을 저장소에 저장하는 구조다. 이 경우에는 데이터를 조회할 때에도 키만 있으면 된다. 컬럼 모델은 관계형 데이터베이스와 비슷하지만 관계형 데이터베이스가 행(row)단위로 데이터를 저장하는데 반해, 데이터를 열(column) 단위로 저장하며 관리하는 모델이다. 문서 모델은 데이터의 저장 단위가 문서가 되며, 하나의 문서 내에는 여러 개의 필드와 필드에 대응하는 값이 있는 형태이다. 그래프 데이터 모델은 그래프에 있는 node와 edge를 저장하고, 이를 쉽게 탐색할 수 있는 모델을 제공한다.

대부분의 NoSQL 솔루션은 분산된 서버에 데이터를 배치시키는 특징을 대부분 갖고 있다. 데이터를 여러 서버에 배치하는 경우 데이터의 분산을 어떻게 하느냐에 따라 크게 메타 데이터 방식, P2P 방식으로 분류할 수 있다.

● 메타 데이터 방식

데이터의 배치 정보가 중앙 집중적인 데이터 서버나 마스터 서버에 저장돼 있으며, 클라이언트는 데이터 연산을 위해 메타 데이터나 서버를 경유해 실제 데이터를 처리할 서버로 접속하는 방식이다. 데이터 배치에 대한 정보를 중앙에서 관리하기 편하며 맵리듀스(MapReduce)등 타 클라우드 솔루션과 결합하기도 쉽다. 단점으로는 데이터를 관리하는 서버나 관리 데이터에 문제가 발생하면 전체 데이터에 접근할 수 없게 된다.

● P2P 방식

별도의 메타 데이터가 없으며 해시 함수를 이용해 특정 키를 서비스하는 서버를 찾는 방식이다. 메타 데이터나 이를 관리하는 서버가 없기 때문에 장애가 발생하더라도 부분적인 키 영역에만 나타난다. 하지만 저장된 데이터를 이용해 분석 작업을 하거나 데이터에 기반한 로드밸런싱이 어렵다.

기존의 관계형 데이터 베이스는 대부분의 업무나 시스템 환경에 사용 가능한 형태를 지닌데 비해, 새롭게 나타난 NoSQL은 특정 용도, 특정 조건에서만 사용할

4) 데이터의 수정에 대한 안정성을 보장하기 위해서 데이터의 수정이 실패할 경우 롤백하여 이전 상태로 되돌아가는 방식
5) 분산 데이터베이스의 데이터 복사본이 서로 일치하지 않아 정합성이 요구될 경우, 다수결 정족수로 결정하는 방식

수 있는 경우가 많다. 따라서 NoSQL을 사용할 때에는 NoSQL 솔루션 중 구축하려는 시스템에 적합한 솔루션인지 판단을 해야한다. 아울러, NoSQL은 대부분 분산 시스템으로 구성되기 때문에 데이터 모델 뿐만 아니라 데이터 분산, 클러스터 멤버십, 장애 대응 등 다양한 분산 이슈를 해결해야 하므로 구현 자체가 복잡한 경우가 많다. 그리고 문서가 부족해 사용하기 어려운 점이 많다. 특히 데이터를 저장하기 때문에 솔루션에 대한 이해가 되지 않으면 업무에 적용하기 어렵다.

NoSQL을 이해하기 위해서는 해당 오픈소스의 이론적 배경을 설명하는 논문을 참고하는 것을 추천한다. 대표적인 예로써 Bigtable에 대한 논문[7]과 Casandra에 대한 논문[10]이 있다.

## 2.2. 분산 파일 시스템 I/O 가상화

클라우드 컴퓨팅 서비스는 사용자의 모든 데이터가 클라우드 인프라 내부에 저장되기 때문에 기본적으로 대량의 데이터나 파일을 저장할 수 있는 저장소를 필요로 한다. 일반적으로 대량의 파일을 안전하게 저장하기 위해 NAS나 SAN 을 사용한다. NAS(Network Attached Storage)는 스토리지 장비와 데이터 전송 관리 모듈이 결합된 장치로, 이더넷 카드를 통해 서버와 연결된다 SAN(Storage Area Network)은 광채널로 스토리지만을 위한 별도의 네트워크를 구성한 후 이를 서버와 연결하는 구조를 일컫는다. 이 두 가지 장비가 스토리지 솔루션에서 가장 많이 사용되고 있었으나 클라우드 서비스에서 요구하는 서비스의 동적인 증가, 감소에 대한 유연성과 고가격의 비용적인 측면을 고려해서 클라우드 컴퓨팅에서는 주로 분산 파일 시스템이 개발되어 사용되게 되었다.

분산 파일 시스템은 SAN이나 NAS 과 비교하여 성능적으로 뛰어나다기 보다는 빠른 시간 내로 확장이 가능한 탄력적 확장성에 최적화된 솔루션으로서 요구되는 기능에 따라 대안으로 사용된다. 분산 파일 시스템의 특징으로는 서버의 일반 디스크 장비를 활용하기 때문에 상대적으로 저렴하다는 점과, OS의 커널 차원에서 파일 시스템을 제공하는 것이 아니라 어플리케이션으로서 파일 시스템을 구현하여 제공하는 것이기 때문에 POSIX 표준 지원을 따르지 않고 자체적인 I/O API를 제공하게 되는 점이 있다.

파일 시스템은 기본적으로 하나의 서버가 하나의 파일 시스템을 소유하고 관리하는 방식으로 사용되어왔다. 하지만 네트워크상에서 여러 서버가 특정 데이터를 서로 공유할 수 있게 지원할 수 있는 특수한 형태의 파일 시스템이 많이 개발되어 활용되고 있다. 이런 형태의 파일 시스템으로는 대표적으로 네트워크 공유 파일 시스템과 클러스터 파일 시스템이 있다.

● 네트워크 공유 파일 시스템

네트워크 상에서 여러 서버가 특정 데이터를 서로 공유할 수 있게 지원하는 특수 형태의 파일 시스템을 네트워크 파일 시스템이라고 하며, 클라이언트-서버 파일 시스템이라고도 한다. 네트워크 공유 파일 시스템에 연결된 사용자는 원격지에 있는 서버의 파일 시스템을 마치 자신이 사용하고 있는 컴퓨터에 있는 파일 시스템인 것처럼 사용할 수 있다.

네트워크 파일 시스템을 사용하려면 파일 서버 입장에서는 접근 허용을 선언하고, 클라이언트 입장에서는 사용요청(mount)이 필요하다. 파일 시스템의 특성상 데이터를 전송할 때 프로토콜 변환에 따르는 오버헤드가 있지만 사용상의 편리함으로 인해 많이 활용된다.

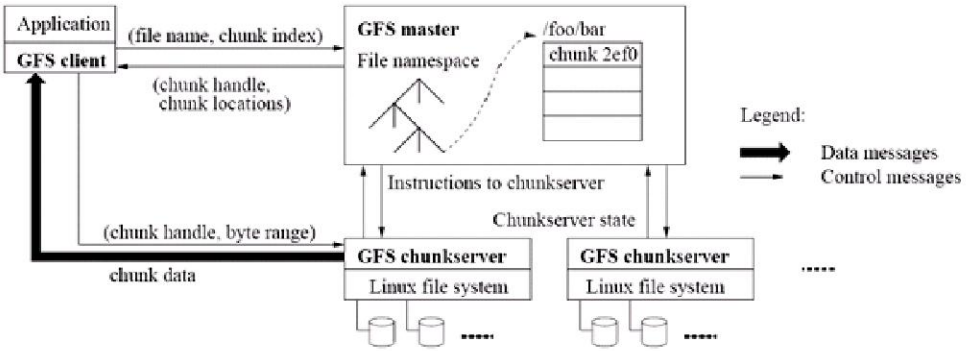
네트워크 파일 시스템의 대표 사례로는 NFS와 CIFS가 있다. NFS(Network File System)는 SUN사가 개발한 이기종 서버 사이의 파일을 공유하는 네트워크 프로토콜로 원격지에 있는 파일 시스템을 마치 자신의 컴퓨터에 있는 파일 시스템인 것처럼 사용할 수 있다. 리눅스를 포함한 유닉스 계열에서 많이 사용되고 있다. CIFS(Common Internet File System)은 마이크로 소프트에서 개발된 파일 시스템 프로토콜로서 클라이언트-서버 모델을 따르며 TCP/IP 프로토콜을 이용한다. 기존의 FTP 나 HTTP 프로토콜의 모자란 점을 보완하는 측면이 강하다.

● 클러스터 파일 시스템

클라우드 컴퓨팅에서 요구되는 파일 시스템은 서버 고장과 같은 시스템 장애가 발생하더라도 지속적으로 서비스를 제공할 수 있는 신뢰성과 가용성을 보장해야 한다. 따라서 위와 같은 NFS나 CIFS 의 단순한 클라이언트-서버 모델의 파일 시스템으로서는 이를 만족시키기 어려우며, 성능과 확장성에도 한계가 있다.

클러스터 파일 시스템은 성능과 확장과 가용성 면에서 이를 충족시키는 분산 파일 시스템 구조로, 최근에 연구와 개발이 활발히 진행 중이다. 클러스터 파일 시스템에서는 파일 메타데이터를 관리하는 전용 서버를 별도로 분리함으로써 메타데이터에 접근하는 경로와 데이터에 접근하는 경로를 분리한다. 그리고 이를 통해 파일 입출력 성능을 높이면서 독립적인 확장과 안전한 파일 서비스를 제공하고자 한다. 하지만 메타 데이터 서버에 부하가 집중될 수 있는 문제점도 내포한다.

클러스터 파일 시스템은 구글에서 GFS에 대한 논문을 발표하면서 주목을 받았으며, 이를 바탕으로 오픈 소스 프로젝트로서 아파치의 하둡(Hadoop) 파일 시스템이 등장하여 현재 야후, 아마존 등에서 널리 활용되고 있다. 또한 클러스터 파일 시스템이 대부분 분산 구조(Distributed Architecture)를 가지기 때문에 분산 파일 시스템이라고도 불리게 되었다.




[그림 7 : Google File System 아키텍처]

분산 파일 시스템에서 동작하는 어플리케이션을 개발하게 될 경우, 기존의 POSIX 표준을 따르는 read, write와 같은 I/O 함수들을 이용할 수 없기 때문에 기존의 소스 코드의 I/O 함수를 해당하는 분산 파일 시스템에서 제공하는 I/O API 함수로 적용하기 위한 가상화 작업이 요구된다. 아마존의 S3 (Simple Storage Service) 서비스가 분산 파일 시스템 위에서 동작하는 웹 스토리지 서비스를 구현한 대표적인 예라고 할 수 있다.

## 2.3. 분산 시스템 운영 관리 기술 (chubby)

한 대의 서버를 운영하는 단순한 환경에서는 서버의 배치와 관리가 매우 간단하여 고려할 필요가 없는 경우가 대부분이다. 하지만 분산 시스템을 구성하여 배치할 경우 일부 서비스 기능의 예상치 못한 중단이나 장애, 서비스 업그레이드의 동기화, 서비스의 우선 순위 등과 같은 예상하지 못한 다양한 상황이 발생할 수 있다. 따라서 분산된 환경에 서버를 배치할 경우 서버 노드들을 관리하기 위하여 네이밍 서비스와 부하 분산과 같은 항목들이 반드시 고려되어야 하고, 이를 해결하기 위하여 다양한 알고리즘과 구현 기법이 요구된다.

### ● 부하 분산 처리

기존의 환경에서 부하 분산을 위하여 사용되어온 방법으로 L4스위치나 DNS 서버를 이용하는 방식이 있다. 엄밀하게 말하자면 부하 분산을 위한 L4 스위치와 네이밍 서비스를 위한 DNS 서버는 용도가 다르지만, L4스위치가 출시되기 전까지는 DNS 서버에서 라운드 로빈(Round-Robin) 방식을 이용하여 서버 분산을 해결하는 것이 일반적인 부하 분산 방식이었다.

▷ L4스위치

일반적인 클러스터 환경에서는 부하 분산을 위해 로드밸런서로서 L4 스위치를 배치하여 클라이언트의 요청을 여러 대의 서버로 나누어 처리한다. 이와 같은 구성에서 클라이언트는 특정 웹 서버나 어플리케이션 서버로 직접 접속하는 것이 아니라 로드밸런서에 설정된 가상 IP를 이용해 접속한다. 특정 어플리케이션 서버에 장애가 발생할 경우 로드밸런서에서 장애가 발생한 서버를 제거함으로써 장애가 발생한 서버로 요청을 보내지 않는 구조이다. L4 스위치는 하드웨어로서 제공되기 때문에 구축과 관리가 간단하지만 비용이 많이 들고 지역적인 제약이 있다.

▷ DNS 서버

DNS 서버를 구성할 경우 어플리케이션 서버 클러스터를 하나의 도메인으로 설정하고, 도메인에 매칭되는 어플리케이션 서버들의 물리적인 서버 IP를 DNS 서버에 등록한 후 클라이언트는 도메인 룩업 과정을 통해 물리적인 서버 IP를 찾아서 접속하는 방법을 사용한다. 장애가 발생할 경우 DNS 서버에서 장애가 발

생한 서버를 해당 도메인에서 제거한다. 장애 상황을 모니터링하기 위해 별도의 관리 시스템을 구성하고 장애 발생 시 관리자에게 경고 메일이나 SMS 메시지 등을 보내는 방식으로 시스템을 관리한다. 소프트웨어로서 서버 노드들을 관리하기 때문에 L4 스위치 방식보다 느리다. DNS 서버의 서비스 룩업 과정은 복잡하기 때문에 클라이언트에서는 캐시를 많이 활용하는데, 이 경우 서버의 변경 사항이 즉시 반영되지 않는 문제가 있다.

### ● 동기화 락 관리

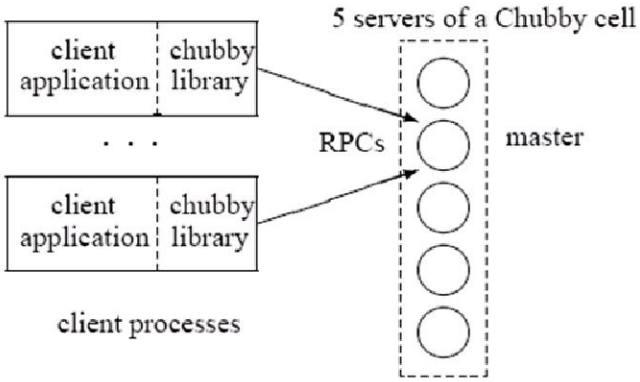
싱글 스레드로 수행되던 프로그램을 멀티스레드 환경으로 바꿀 경우 가장 먼저 생각해야 되는 부분이 전역 변수 등 스레드 사이에 공유되는 변수다. 공유 변수에 접근하여 값을 수정하거나 조회하는 경우에는 경쟁이나 충돌을 막기 위하여 락(lock), 원자성(atomicity)을 고려하여 동기화시켜 여러 스레드가 동시에 접근하지 못하도록 해야 한다. 분산 환경에서도 동일한 이슈가 발생하며, 해결 방법은 더 복잡하다. 여러 서버가 하나의 공유된 자원에 동시에 접근하려고 할 때 한번에 하나의 서버만 접근하도록 구현해야 한다. 가장 쉬운 방법은 락을 사용하는 것이지만 하나의 서버가 접근하고 있는 동안 심각한 트래픽이 발생될 수 있으며, 락을 획득한 서버가 정상적으로 해제를 하지 못할 경우 데드락(deadlock) 현상이 발생될 수 있다. 분산 환경에서의 동기화 락 문제는 매우 어려운 문제이다.

### ● 장애 상황 판단

분산 환경에서는 중요한 역할을 수행하는 서버의 경우 장애 상황에 대한 가용성을 위하여 이중화 구성을 한다. 이중화 구성 방식은 두 서버 모두 동시에 서비스를 수행하는 Active-Active 방식과 하나의 서버가 서비스를 수행하고 장애 발생을 대비하여 다른 서버는 대기하고 있는 Active-Standby 방식이 있다. Active-Active 방식에 비해 Active-Standby 방식이 구현하기 간단하여 더 널리 사용되는데, 이 구성에서는 Active 상태의 주 서버에서 장애가 발생했을때 Standby 상태의 보조 서버가 Active 상태로 바뀌면서 서비스를 수행해야 하기 때문에 장애 상황을 판단해내는 시점이 중요하다. 보통 이와 같은 구성에서는 관리 시스템으로 모니터링 서버를 추가하여 각 서버로부터의 하트비트(Heartbeat)를 받아서 장애 상황을 판단하게 된다. 그러나 네트워크의 단절로 인하여 관리 시스템이 정상적으로 동작하는 주 서버를 장애 상황으로 판단하게 되는 경우, 주 서버와 보조 서버가 동시에 서비스를 수행하게 되어 동시에 공유된 자원에 접근하는 문제가 발생하게 된다. 이와 같은 상황은 일반적인 어플리케이

션 개발자가 해결하기에는 쉽지 않은 문제이다.

이와같은 분산 환경의 서버 노드들 사이의 관리를 위한 솔루션으로서 구글에서 발표한 chubby 락 기술을 기반으로 개발된 아파치 오픈 소스 프로젝트 주키퍼(zookeeper)가 있다.




[그림 8 : chubby 시스템]

주키퍼는 그 자체도 여러 대의 서버로 구성된 분산 시스템이며, 분산 환경에서 락, 네이밍 서비스, 클러스터 멤버십 등을 쉽게 구현할 수 있도록 해준다.
주키퍼가 제공하는 기능들은 다음과 같다.

▷ 시간 동기화
▷ 노드 관리 (znode)
▷ 접근 권한 (ACL Access Control List) 제공
▷ 세션 관리
▷ 이벤트 처리 보증

주키퍼가 제공하는 위의 기능들이 직접적으로 분산 락관리, 네이밍 서비스, 클러스터 멤버십을 지원하는 것은 아니다. 다만 그러한 관리 기능들을 쉽게 만들 수 있는 메커니즘을 제공하며, 이를 기반으로 필요한 용도에 맞는 관리 기능을 개발자가 직접 개발해야 한다.

## 2.4. 병렬 처리 기술 (MapReduce)

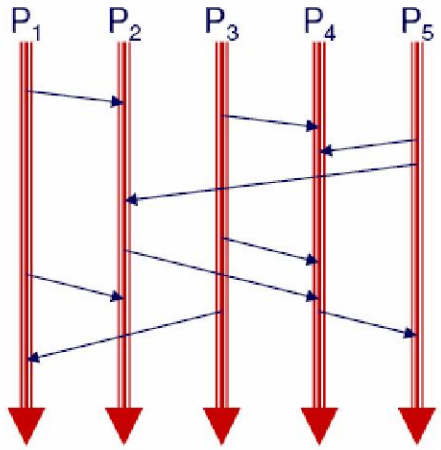
맵리듀스는 대용량 데이터를 병렬로 처리하기 위한 소프트웨어 프레임워크이며,

많은 저가의 장비로 페타바이트 이상의 데이터를 처리하는 메커니즘의 모델이다. 맵리듀스는 한 대의 서버에서는 처리하기 어려운 대용량 데이터와 데이터 처리에 걸리는 오랜 응답 시간을 해결하기 위한 소프트웨어 기법 중에서 병렬 처리 기법의 하나로서 구글에서 개발되었으며, 분산 환경 특히 구글 플랫폼 기술인 GFS 와 Bigtable 과 연동하여 동작하도록 최적화되었다. 구글의 기술을 응용한 오픈 소스 하둡에서도 동일한 형태로 구성되어 있으며, 현재 대용량 데이터 처리에 있어서 높은 효율성으로 많은 관심을 받고 있다.

대용량 데이터의 병렬 처리에는 맵리듀스가 등장하기 전까지 MPI (Message Passing Interface) 모델이 전통적으로 사용되어 오고 있었으며 아직까지도 작업 단위성(Task Granularity)과 기능 분할(Partitioning Function)을 고려하는 측면에서 보면 맵리듀스가 MPI 보다 나은 성능을 보장한다고 하기 어렵다. 그러나 분산 환경에서의 내결함성(fault-tolerance)과 키-값 모델을 제공함으로써 상대적으로 훨씬 더 어렵고 복잡한 MPI 메커니즘보다 폭넓게 활용되고 있다.

● MPI 모델

고속 연산을 중요시하는 슈퍼 컴퓨팅 분야에서 개발되었으며, 분산된 노드 사이에서의 계산을 위한 데이터 분배 및 취합 에 대한 기능을 정의한 프로그래밍 모델이다.



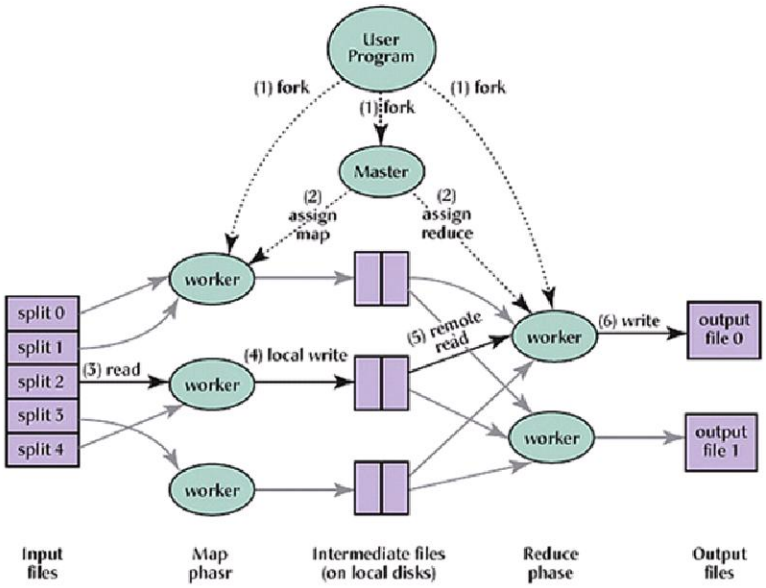

[그림 9 : MPI 모델 동작 원리]

MPI 용어 자체가 의미하듯이, MPI 모델에서는 각 노드가 서로 데이터를 주고받으면서 연산 처리를 진행하게 되며, 각 노드가 자신의 컴퓨팅 자원을 얼마나 효율적으로 소모하느냐에 중점을 두고 있기 때문에 처리가 끝나 컴퓨팅 자원의

여유가 있는 노드에 작업을 할당하는 메커니즘이 상당히 복잡해진다. 또한 이를 위해서 데이터의 전송이 많아지는데 서버-클라이언트 모델의 형태가 아니기 때문에 네트워크의 사용량이 많아 고속 네트워크 인프라가 필수적이다.
MPI 모델이 요구되는 대용량 데이터의 빠른 처리는 일반적인 비즈니스 어플리케이션에서는 많지 않으며, 일반 개발자들이 개발하기에는 어려움이 많다. 오픈 소스 솔루션으로는 MPICH가 있다.

● MapReduce 모델

구글의 맵리듀스는 소스 및 메커니즘이 공개되어있지 않으며, 오픈 소스 진영의 하둡 맵리듀스가 구글이 발표한 맵리듀스의 논문을 기반으로 개발되어 활용되고 있으며 동일한 메커니즘으로 동작한다고 볼 수 있다.
하둡 맵리듀스의 처리는 크게 맵 함수와 리듀스 함수로 구분될 수 있으며, 이 두 가지 함수의 처리가 동일한 노드에서 발생할 수도 있고 서로 다른 노드에서 발생할 수도 있다. 일반적으로 입력 데이터는 레코드 단위로 맵 함수에 전달된다. 각 레코드에 맵 함수를 실행하면 키와 값이 출력되는데, 키-값을 이용하여 데이터를 전달할 리듀스 노드가 결정된다. 리듀스 함수는 키-값 목록을 받아 사용자가 원하는 데이터 처리를 한 후 여러 개의 값을 출력한다. 맵 함수와 리듀스 함수의 동작이 끝나면 리듀스 함수가 동작한 노드의 로컬 파일 시스템에 결과 데이터값이 저장되는데, 이 데이터는 다른 맵리듀스 동작의 입력 데이터로서 활용될 수 있으며 이를 반복하여 더 복잡한 연산의 병렬 처리가 가능해진다.



[그림 10 : MapReduce 모델 구조]

하둡 맵리듀스를 이용하면 병렬 처리를 위해 요구되는 다음과 같은 작업들을 프레임워크에서 대신 수행해 준다.

▷ 입력 파일을 분할하여 맵 함수에 전달
▷ 맵 함수의 출력 결과를 정렬, 병합
▷ 정렬, 병합된 결과를 리듀스 함수에 전달
▷ 처리 연산을 분산된 서버에서 수행
▷ 분산 처리된 결과들을 리듀스 함수가 통합

이 복잡한 과정들을 개발자가 직접 처리하는 것이 아니라 맵리듀스 프레임워크에서 자동으로 처리해준다. 사용자는 맵 함수와 리듀스 함수만 용도에 맞게 어플리케이션에 적용하면 분산 병렬 처리를 수행할 수 있다.

오픈 소스 하둡의 경우 파일 시스템(HDFS)과 맵리듀스가 하나의 패키지로 같이 배포되며, JAVA로 제공되는 라이브러리를 통해 Mapper 클래스와 Reducer 클래스를 상속받아 개발할 수 있다.

## 2.5. 로그 수집 기술

한 대의 서버를 운영하는 단순한 환경에서는 로그를 따로 관리해야할 필요성이 거의 없다. 로그는 서버의 상태 정보나 작업 진행 상황, 사용자의 기록을 저장하는데 개발 단계에서는 디버깅에 이용되는 정도이지만, 운영 환경에서는 장애 원인 파악이나 장애가 발생할 상황에 대한 예측, 사용 패턴 분석, 부하 분산 및 통계 자료로 활용되며, 특히 분산 환경에서는 빠질 수 없는 중요한 기능이다.

일반적으로 로그는 로그를 생성하는 어플리케이션이 동작하는 서버의 로컬 파일 시스템에 저장한다. 분석이나 백업이 필요한 경우에는 주기적으로 하나의 서버나 공용 스토리지에 저장하고 분석 작업을 수행한다. 하지만 분산된 환경에서는 수많은 서버 클러스터에 로그가 쌓이고 이들 서버 중 일부는 장애가 수시로 발생하기 때문에 각 서버에 저장된 로그 정보가 유실될 수 있다. 또한 수십~수백대 이상의 서버로부터 로그 파일을 특정 서버로 모으려면 추가적인 작업이 필요하다. 야후, 페이스북과 같이 수많은 로그 데이터를 관리하고 분석해야 하는 서비스에서는 이를 위해 별도의 로그 관리 시스템을 개발하여 사용한다.

로그 관리 시스템에서 고려해야 할 사항은 다음과 같다.

▷ 확장성

로그 수집 대상을 수십, 수백, 수천 대의 규모로 확장할 수 있어야 한다. 하나의 특정 서버나 특정 스토리지로 수집된 로그가 집중되는 경우 해당 서버의 부하의 한계 때문에 확장성을 가지지 못한다. 로그 수집 시스템 자체적으로도 분산 처리 기술을 적용해야 확장성을 가질 수 있다.

▷ 실시간성

특정 서버에서 발생된 로그가 최종 저장소에 실시간으로 반영되어 즉시 확인이 가능한지 여부를 의미한다. 분산된 환경에서 로그의 실시간성을 만족시키기는 매우 어렵기 때문에 적절한 응답시간과 정확성을 결정해야 한다.

▷ 안정성

로그의 기록 자체가 중요한 정보는 아니지만 장애 판단이나 분석을 위한 참고를 위해 필요하며 특정 어플리케이션의 경우 없어서는 안될 자료가 될 수 있기 때문에 가능한 유실되지 않도록 백업을 유지하는 등의 안정성을 고려해야 한다.

이와같은 요구 사항들을 만족시키는 솔루션은 현재 매우 드물다. 하지만 오픈소스 프로젝트 중 척와(chukwa)와 스크라이브(scribe)와 같은 솔루션들이 로그 수집과 관리 기능을 제공하고 있으며 특정 용도에 맞게 활용되어 사용되면서 버전 업그레이드되고 있다.

● 척와 chukwa

척와는 하둡 프로젝트의 서브 프로젝트로 진행 중인 오픈소스 프로젝트로 분산된 서버의 로그를 하둡 파일 시스템에 저장하고 하둡 맵리듀스를 이용해 로그 분석을 수행하는 솔루션이다.

척와는 범용적인 로그 수집과 로그 관리를 위한 솔루션으로 개발되었지만 하둡 클러스터의 로그와 서버의 상태 정보를 관리하는 기능도 들어있다. 하둡 클러스터에서는 여러 대의 서버에 로그가 나누어져 저장되기 때문에 로그를 보기 위해서는 각 서버와 접속해야 하는 불편함이 있는데 척와를 이용하면 쉽게 로그를 수집할 수 있다.

척와의 장점으로는 어플리케이션과는 독립적으로 로그를 수집하는 기능을 제공

하며, 로그 파일을 다시 하둡 파일 시스템을 이용하여 저장하기 때문에 안정적이고. 또한 실시간에 가까운 준실시간(near real time) 수집과 분석을 지원한다는 점이 있다.
척와의 단점으로는 하둡 플랫폼을 구성하는 요소의 하나로, 하둡이 설치된 환경에서만 운영 및 동작이 가능하다는 제약이 있으며, 로그를 바이너리 파일의 형태로 저장하기 때문에 로그 정보를 보기 위해서는 별도의 뷰어가 필요하다는 점이 있다.

● 스크라이브 scribe

스크라이브는 페이스북에서 사용하는 로그 관리 시스템으로 아파치 라이센스를 가진 오픈 소스이다.
척와가 기본적으로 하둡 저장소에 대한 기능까지 모두 제공하는데 비해 스크라이브는 로그를 수신하는 수신 서버와 수신 서버에 접속하는 클라이언트 모듈만 제공한다. 클라이언트 모듈은 별도로 지원하는 게 아니라 인터페이스만 제공하기 때문에 다양한 프로그래밍 언어에서 이에 맞추어서 클라이언트를 개발하여야 사용가능하다. 그리고 로그 저장 스토리지와 시스템 구성은 사용자가 직접 구성해야 한다.
기존의 로그를 로컬 파일 시스템에 저장하는 시스템에 스크라이브를 적용하는 경우에는 우선 각 로그 생성 어플리케이션 서버에 스크라이브 서버를 설치하고, 스크라이브 서버로 로그 파일을 전송하는 클라이언트 모듈을 개발해야 한다. 어플리케이션 서버에서 생성된 로그 파일이 클라이언트 모듈을 통해 스크라이브 서버로 송신되고, 각 어플리케이션 서버의 스크라이브 서버로부터 스크라이브 중앙 서버로 다시 송신되어 중앙 서버에서 로그를 최종적으로 관리한다. 중앙 서버에서는 하둡, NAS 등과 같은 스토리지에 로그 파일을 저장하는데 스토리지에 저장하기 전 로컬 파일 시스템에 임시로 저장한 후, 스토리지에 저장한다.
스크라이브는 척와보다 단순하고 환경적인 제약이 없으나, 클라이언트 모듈 개발과 장애 상황 판단과 같은 추가적인 부분에서 커스터마이즈가 필요하다는 단점이 있다.

## 2.6. 캐시 기술

캐시란 이미 존재하는 데이터를 반복해서 조회하거나 시간이 오래걸리는 연산을

수행하는 등의 경우에 응답 시간을 줄이기 위하여 결과 값을 임시로 저장해놓은 복제본의 집합을 의미한다. 실제로 많은 양의 복잡한 데이터에 대한 처리를 수행할 경우 임시 저장소를 이용하여 캐시를 활용하느냐 마느냐에 따라 10~100배 이상의 성능의 차이가 나게 된다. 캐시는 비단 컴퓨팅 환경에서만이 아니라 여러 가지 용도로 널리 활용되고 있으며, 포괄적인 개념으로 일반 어플리케이션에서 사용하는 queue나 stack의 메모리 구조까지도 포함된다고 할 수 있다.
컴퓨팅 환경에서 캐시는 일반적으로 다음과 같이 분류된다.

● CPU 캐시

CPU 칩의 내부 혹은 가까이에 연결된 작은 메모리이며, 하드웨어로 구현된다. 저속의 DRAM에 대한 요청 속도를 높이는 용도로 쓰이며 L1, L2, L3 cache 가이에 해당된다.

● 디스크 캐시

디스크에 저장하는 정보 중 일부를 메모리에 저장함으로써 데이터에 접근할 때 빨리 처리하게 하는 메카니즘으로 하드웨어로서 하드 디스크에 부착되어 나오는 램이나 소프트웨어로서 OS 나 어플리케이션에서 메모리의 일부를 가상디스크처럼 사용하는 형태를 가리킨다. 디스크 버퍼, 페이지 캐시 등이 이에 해당된다.

● 웹 캐시

웹브라우저나 프락시 서버 혹은 DNS 서버에서 네트워크의 응답을 기다리지 않고 컨텐츠를 읽어오기 위해 사용되는 기록들을 웹 캐시라고 부른다. 디스크에서 읽어오는 응답 속도가 네트워크의 응답 속도보다 빠르기 때문에 디스크에 저장된다.

● 데이터베이스 캐시

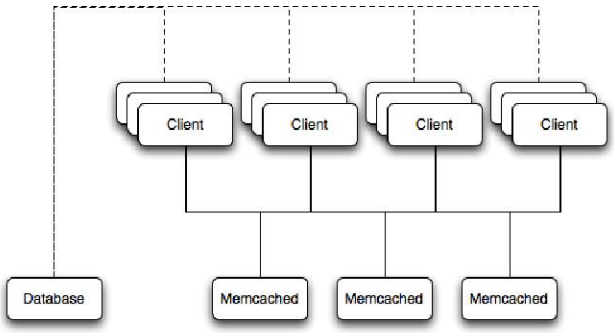
데이터베이스 내부에서 사용하는 캐시 기술로 주로 관계형 데이터 베이스에서 복잡한 처리를 요구하는 SQL 질의문에 대해서 응답 시간을 줄이기 위해 사용된다.

● 메모리 캐시

이전에 계산한 값을 메모리에 저장하여 매번 계산하지 않게 하는 최적화 기법으로 메모리를 가상 디스크처럼 사용하는 디스크 캐시와 유사하지만, 데이터 처리

를 위하여 캐시 메모리를 지속적으로 업데이트하는 기법들이 요구되는 특징이 있다. 검색어 자동 완성이나 데이터 검색을 위한 해시 테이블 등이 이에 속한다.

클라우드 컴퓨팅 환경에서 중요시 되는 캐시는 원격 네트워크를 지원하는 메모리 캐시에 해당하며 주로 데이터베이스 부하를 줄이고 웹 어플리케이션의 성능을 높이기 위한 용도로 사용된다. 대표적인 오픈 소스 솔루션으로서 memcached가 있다.




[그림 11 : memcached 시스템 구성]

memcached는 원격 분산 메모리 캐시 시스템으로, 네트워크 상에서 해시 테이블을 분배하여 사용할 수 있는 기능을 제공한다. memcached는 내부적으로 네트워크 라이브러리와 해시 테이블로 구성되어있으며 네트워크 라이브러리에서는 프로토콜을 처리하고, 해시 테이블에서는 요청 유형에 따라 데이터를 처리하고 통계정보를 기록한다. memcached 는 서버에서 분산 처리를 위한 별도의 기능을 갖고 있지 않으며, 분산 기능은 클라이언트에서 처리한다. 캐시 사용자는 자신의 환경에 맞는 클라이언트 라이브러리를 선택해 옵션에 분산 구성을 설정하면 클라이언트 라이브러리에서 데이터를 여러 대의 원격 memcached 서버에 분배하여 캐시로서 동작하게 한다.

개발자가 memcached를 시스템에 적용하려면 캐시로 처리하고 싶은 데이터를 선택하고 해당하는 데이터 처리에 적합한 해싱 기법(일반적으로 ketama기법을 이용한다)을 이용하여 데이터를 키-값 쌍으로 구성한 뒤 이를 memcached의 아이템으로 입력하여 캐시로서 동작하도록 하고, 해당하는 키-값 쌍을 memcached에서 읽어서 처리할 수 있도록 솔루션을 수정하여 적용해야 한다.

# 3. 클라우드 컴퓨팅 환경의 웹 서비스 적용 가이드

여기서는 기존의 웹 서비스를 클라우드 환경에서 동작하게 하기 위한 기술 구현을 실제 예제 적용을 통해 설명한다.

개발 환경은 다음과 같다

- ▷ APMSETUP 7 (Apache 2.2.14, PHP 5.2.12)
- ▷ SW공학센터 예제 (PHP)

SW공학센터 예제는 가이드와 함께 첨부되어있으며 소프트웨어 공학센터 홈페이지 자료실(http://www.software.kr/)에서 다운받을 수 있다.

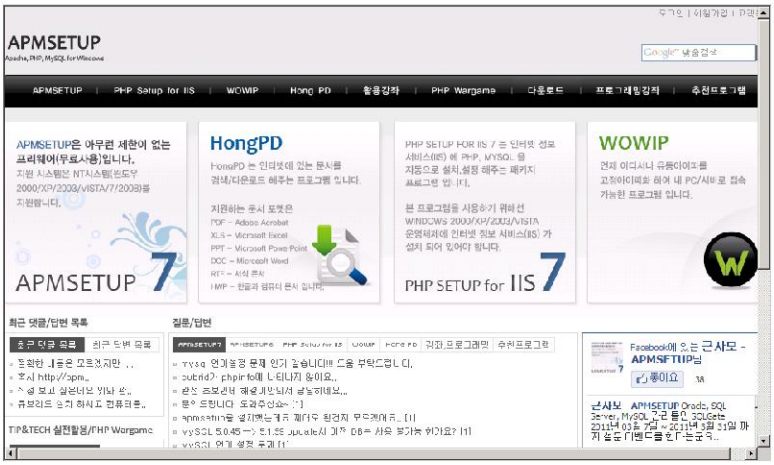
## 3.1. 설치 및 환경 구축

설치 과정 및 환경 구축은 다음과 같다

- ▷ APMSETUP 설치
- ▷ PHP library curl 및 pear package 설치
- ▷ AWS 계정 등록 및 결재

### 3.1.1. APMSETUP 설치

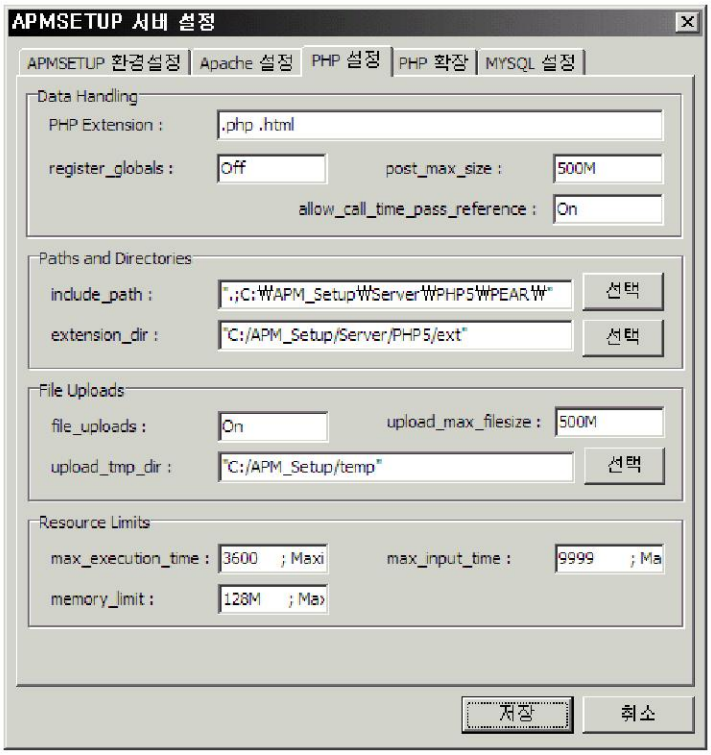
간단한 웹 서비스를 위한 서버로서 공개 배포 솔루션인 APMSETUP7 (Apache + PHP + MySQL)을 다운받아 설치한다.




[예제 그림 1 : APMSETUP 배포 홈페이지]

APMSETUP은 홈페이지(http://www.apmsetup.com/)에서 배포하고 있으며 windows/linux/unix 의 OS 환경이 모두 지원된다.


[예제 그림 2 : APM 정상 설치 후 동작 화면]




[예제 그림 3 : PHP 설정 화면]

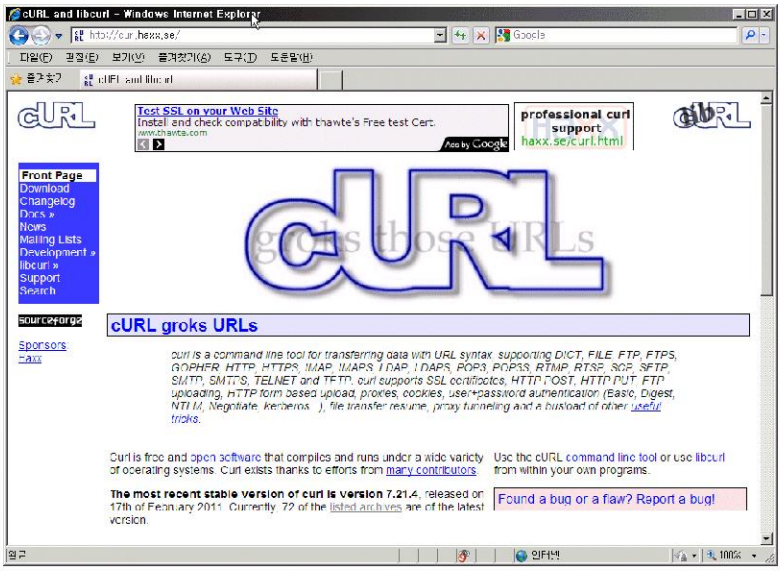
php 설정 화면에서 업로드 파일 사이즈의 제한을 500MB로 설정한다.
현재 php5에서 지원하는 파일 업로드, 다운로드의 최대 한도는 32비트 컴퓨터에서 지원하는 Integer size에 따라 2GB까지로 제한되므로 2GB이상으로 설정하지 않도록 유의한다.

### 3.1.2. PHP library 설치

php 솔루션은 복잡한 프로토콜이나 보안 솔루션에 대해서 추가 확장 라이브러리를 활용하는 경우가 많다. 예제의 동작을 위해서는 다음의 공개 라이브러리들이 필요하다.

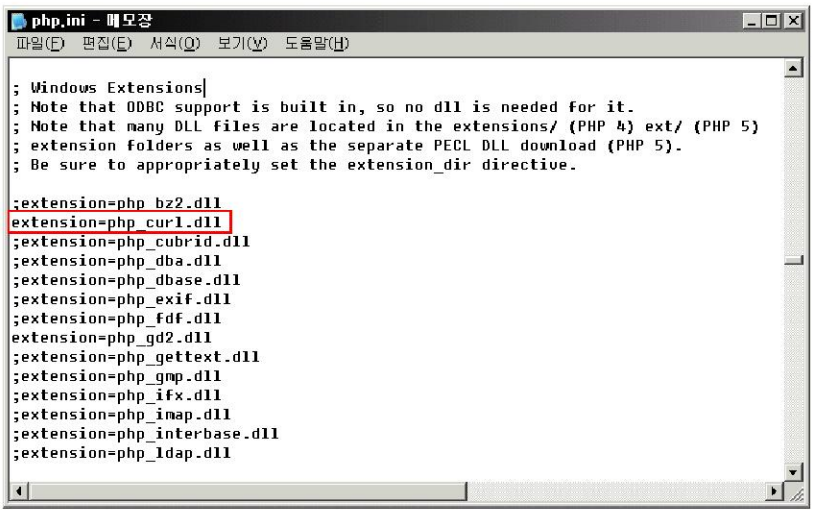
- cURL library
- pear Crypt HMAC2 package
- pear HTTP Request2 package

● cURL 설치



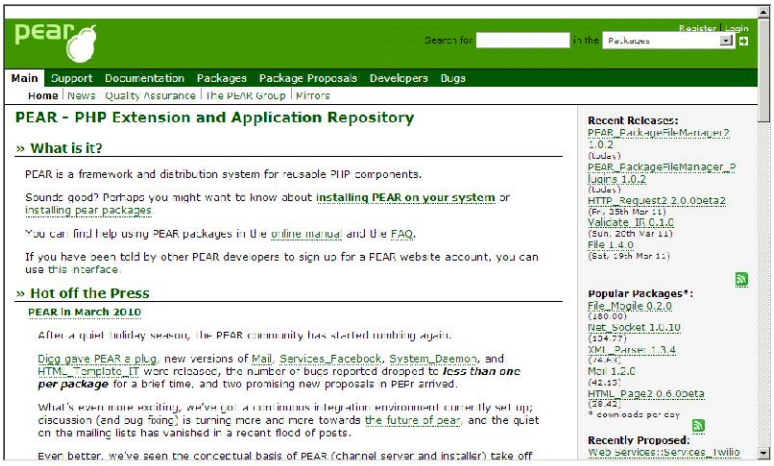

[예제 그림 4 :curl 홈페이지]

cURL은 PHP에서 HTTP 통신을 위해 널리 사용되는 라이브러리이며, cURL 홈페이지(http://curl.haxx.se)에서 다운로드 및 기술 지원을 받을 수 있다. APMSETUP을 설치하면 자동으로 내장되어 있으며, php.ini 파일에서 다음 부분을 수정한 후 APM 서버를 재실행시킴으로써 활성화시킬 수 있다.




[예제 그림 5 : php.ini - cURL library 활성화]

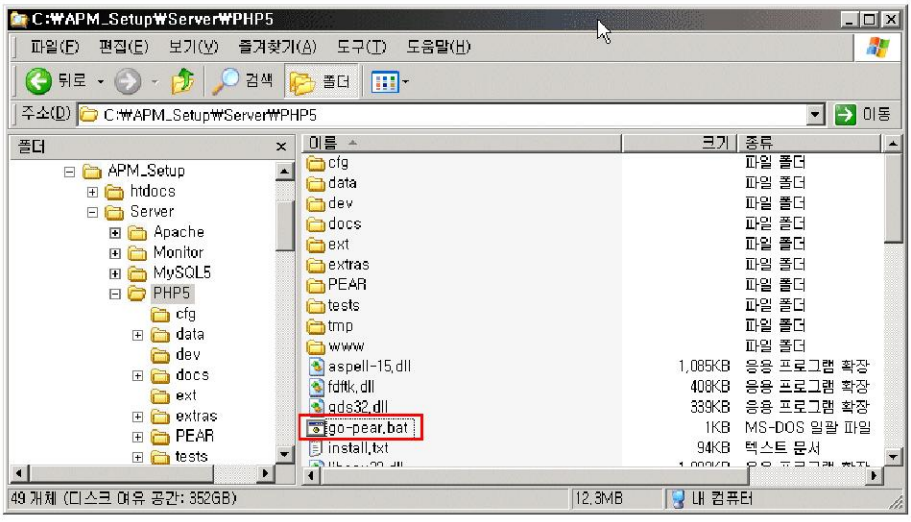
● pear 패키지 Crypt_HMAC2 와 HTTP_Request2 설치




[예제 그림 6 : pear 커뮤니티 홈페이지]

Crypt_HMAC2와 HTTP_Request2는 PHP에서 http 프로토콜로 안전한 통신을 하기 위해 필요한 확장 패키지로, pear 커뮤니티 홈페이지(http://pear.php.net/)에서 다운로드 및 기술 지원을 받을 수 있으며, APMSETUP을 설치하면 자동으로 내장되어 있다.

pear의 설치는 APMSETUP이 설치된 경로에서 "go-pear.bat" 파일을 실행시키면 된다.



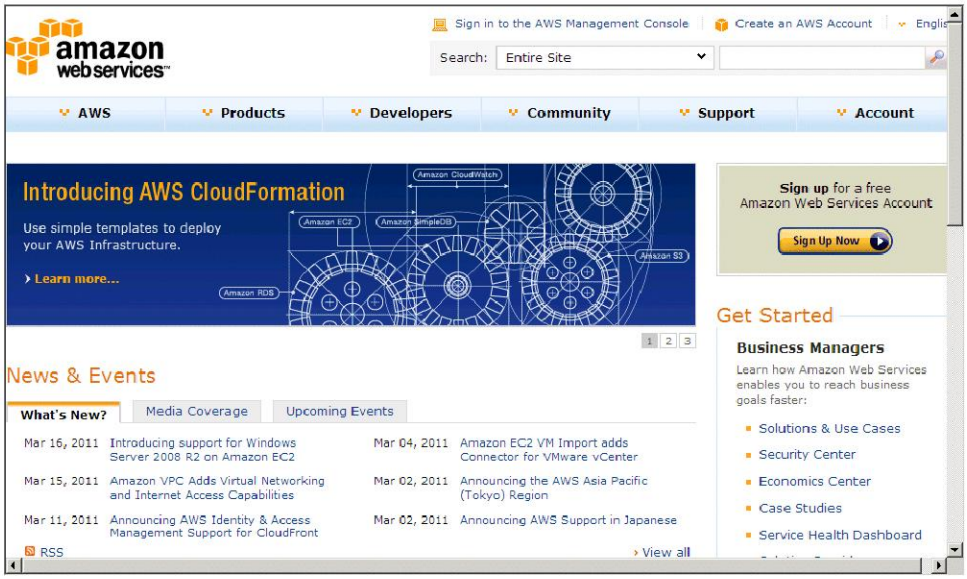

[예제 그림 7 : APMSETUP에 내장되어있는 pear설치파일 go-pear.bat]

추가적인 pear 패키지를 설치하는 방법은 다음과 같다.

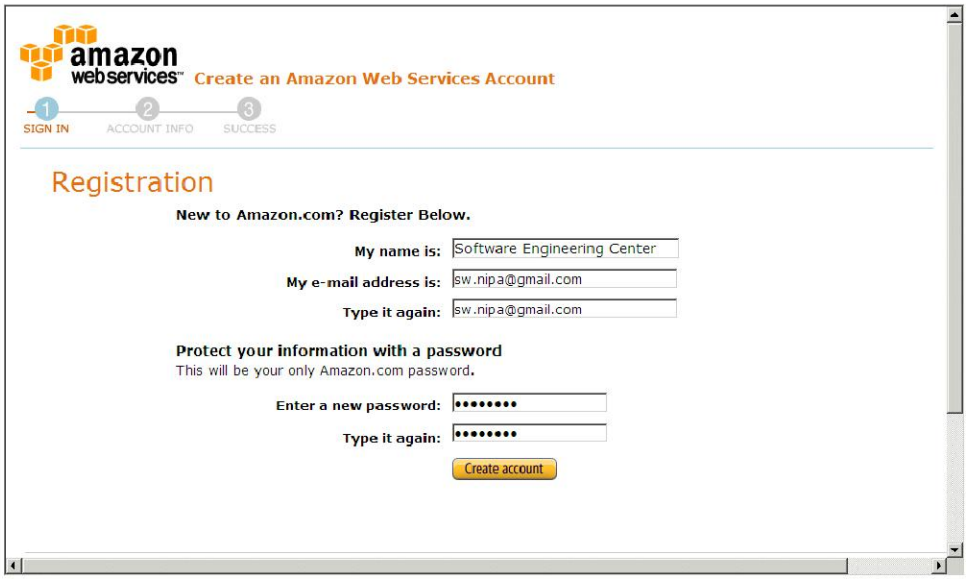
```
pear.bat install [패키지명]
```

### 3.1.3. Amazon S3 (Simple Storage Service) 계정 등록 및 결제

Amazon S3[11]를 이용하기 위해서는 AWS(Amazon Web Service)의 계정이 필요하며, AWS 홈페이지에서 다음과 같이 등록할 수 있다.[7]




[예제 그림 11 : AWS 홈페이지]




[예제 그림 12 : AWS 계정 생성]

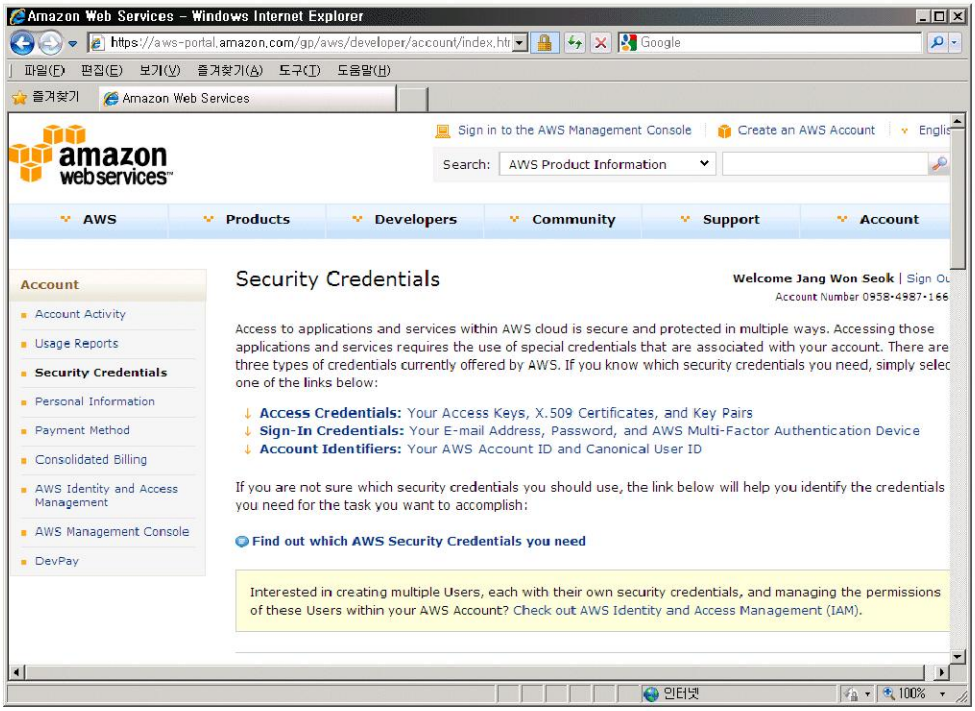
AWS 계정을 활성화시키기 위해서는 결재가 필요하다. 결재는 인터넷을 통해 할 수 있으며, 일정 사용량 기준으로 과금이 되기 때문에 (1000번의 요청에 대해서 $0.01 과금) 예제의 동작에 대해서는 과금을 걱정하지 않아도 된다.

US – Standard | US – N. California | EU – Ireland | **APAC – Singapore**

| Storage (Designed for 99.999999999% Durability) | | Reduced Redundancy Storage (Designed for 99.99% Durability) | | Data Transfer** | | Requests | |
|---|---|---|---|---|---|---|---|
| **Tier** | **Pricing** | **Tier** | **Pricing** | **Tier** | **Pricing** | **Type** | **Pricing** |
| First 1 TB / month of Storage Used | $0.140 per GB | First 1 TB / month of Storage Used | $0.093 per GB | All data transfer in | $0.100 per GB | PUT, COPY, POST, or LIST Requests | $0.01 per 1,000 Requests |
| Next 49 TB / month of Storage Used | $0.125 per GB | Next 49 TB / month of Storage Used | $0.083 per GB | First 1 GB / month data transfer out | $0.000 per GB | GET and all other Requests*** | $0.01 per 10,000 Requests |
| Next 450 TB / month of Storage Used | $0.110 per GB | Next 450 TB / month of Storage Used | $0.073 per GB | Up to 10 TB / month data transfer out | $0.190 per GB | | |
| Next 500 TB / month of Storage Used | $0.095 per GB | Next 500 TB / month of Storage Used | $0.063 per GB | Next 40 TB / month data transfer out | $0.150 per GB | | |
| Next 4000 TB / month of Storage Used | $0.080 per GB | Next 4000 TB / month of Storage Used | $0.053 per GB | Next 100 TB / month data transfer out | $0.130 per GB | | |
| Storage Used / month Over 5000 TB | $0.055 per GB | Storage Used / month Over 5000 TB | $0.037 per GB | Greater than 150 TB / month data transfer out | $0.120 per GB | | |

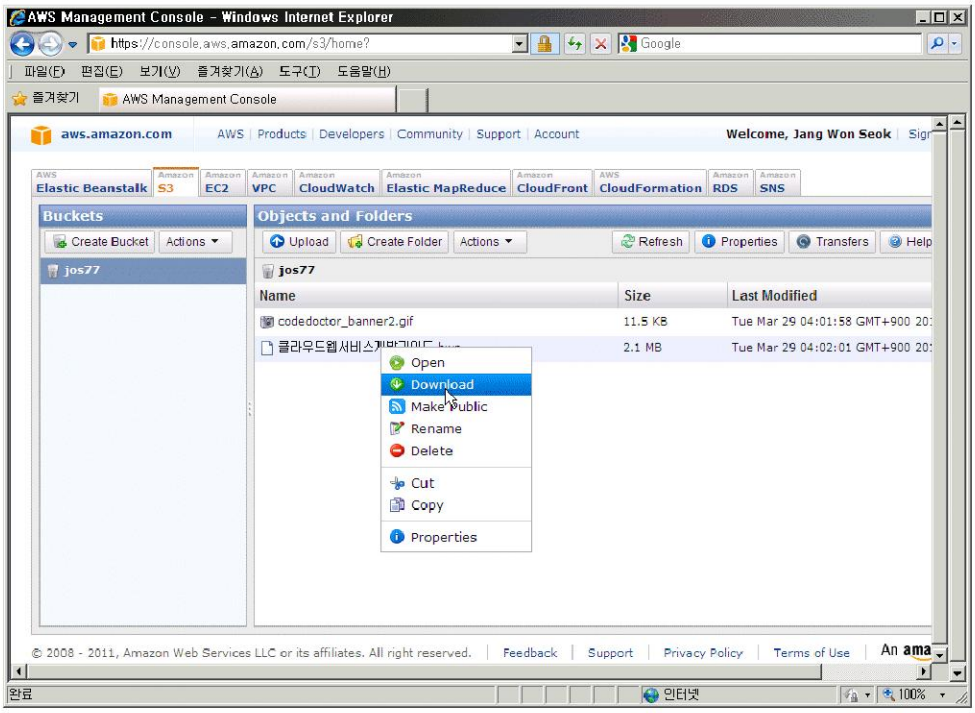
[예제 그림 13 : AWS의 과금 정책]

결재를 진행하면 Account 페이지에서 AWS에서 제공하는 key 인 Access Key ID 와 Secret Access Key 를 발급받을 수 있다



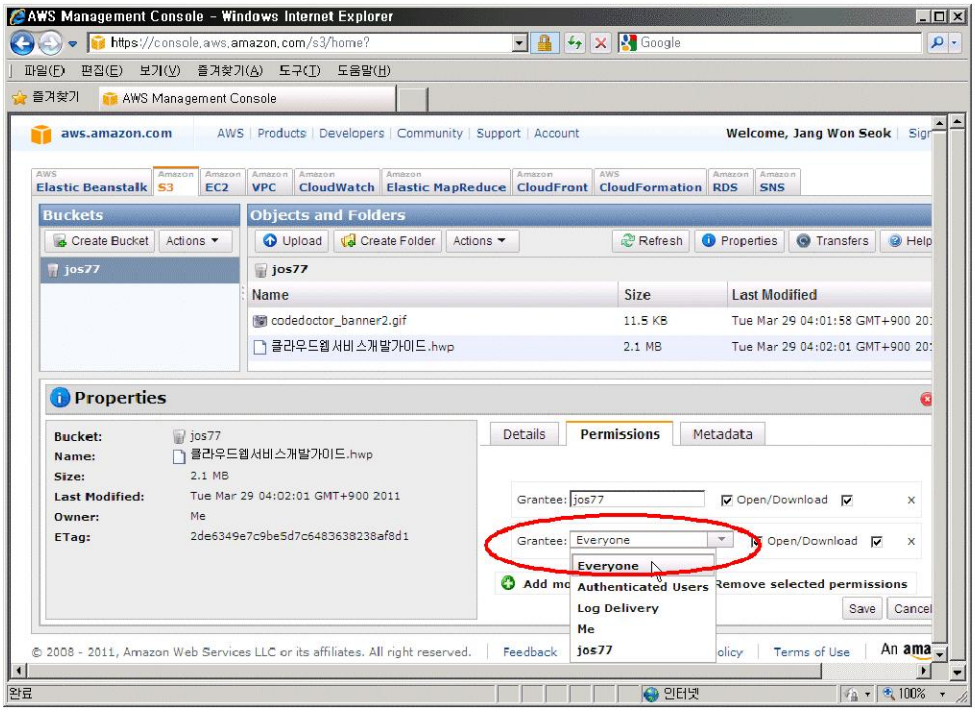

[예제 그림 14 : AWS Access Key 확인 페이지]

AWS 계정을 활성화시킨 후에는 해당 계정에 주어진 공간인 bucket을 생성하여 object(파일)의 업로드, 다운로드의 동작을 확인할 수 있다.




[예제 그림 15 : AWS 콘솔 화면 조작]

object가 보안을 요구하지 않을 경우에는 다음과 같이 권한을 설정한 후 object에 대한 직접 링크를 제공받을 수도 있다.




[예제 그림 16 : AWS object 의 권한 설정]

권한이 Everyone 으로 설정될 경우 직접 링크는 다음과 같이 제공된다.

```
http://bucketname.s3.amazonaws.com/filename
```

그러나, 보안을 위해서 bucket을 생성한 key를 가진 계정만으로 권한을 제한하고, key를 통해서만 업로드, 다운로드 하도록 하는 것이 바람직하다.

## 3.2. 적용 가이드

### 3.2.1. 기본 PHP 동작 예제

설치가 끝나면, 웹브라우저를 통해 웹 서버에 설치된 예제 페이지를 확인해본다. 웹브라우저는 IE8 혹은 Google Chrome을 추천한다.

http://localhost/sec_s3_ex/index.php


[예제 그림 17 : 기본 PHP S3 동작 예제]

이 예제는 php 언어로 되어있으며 Amazon S3 API를 이용하여 S3로 object를 업로드, 다운로드 동작이 구현되어 있다.

### 3.2.2. S3 API

Amazon S3 에서 제공되는 주요 API는 다음과 같으며, AWS 홈페이지 (http://aws.amazon.com)에서 매뉴얼을 다운받을 수 있다.

- ▶ PutBucket - bucket의 생성을 요청한다.
- ▶ GetBucket - bucket의 데이터 object 목록을 요청한다.
- ▶ DeleteBucket - bucket의 삭제를 요청한다.
- ▶ PutObject - bucket에 데이터 object 추가를 요청한다.
- ▶ GetObject - bucket에서 데이터 object 다운을 요청한다.
- ▶ DeleteObject - bucket에서 데이터 object 삭제를 요청한다.

아울러, S3 API를 사용하기 편리하게끔 래핑 레이어(wrapping layer) API를 구성하여 인터넷 상에서 배포하는 공개 라이브러리에는 여러 가지가 있다. 여기서는 undesigned.org.za에서 배포하는 S3 PHP class를 이용해 보도록 한다.

Amazon S3 API는 http 프로토콜을 통하여 동작하며, 세부 내용은 다음과 같다.

● PUT Bucket

PUT 동작은 새로운 bucket을 생성하고 Access 권한을 설정한다. 만약 기존에 동일한 이름의 bucket이 생성되어있다면 아무 동작을 하지 않는다.

요청 Request 형식:

```
PUT / HTTP/1.1
Host: BucketName.s3.amazonaws.com
Content-Length: length
Date: date
Authorization: signatureValue
```

요청, 응답 예시 :

```
PUT / HTTP/1.1
Host: colorpictures.s3.amazonaws.com
Content-Length: 0
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

```
HTTP/1.1 200 OK
x-amz-id-2:
YgIPIfBiKa2bj0KMg95r/0zo3emzU4dzsD4rcKCHQUAdQkf3ShJTOOpXUueF6QKo
x-amz-request-id: 236A8905248E5A01
Date: Wed, 01 Mar 2009 12:00:00 GMT
Location: /colorpictures
Content-Length: 0
Connection: close
Server: AmazonS3
```

● GET Bucket

GET 동작은 일부 혹은 전체(최대 1000개)의 bucket에 있는 object들을 반환한다. 응답은 xml 형식이며, 반환되는 객체의 수나 조건을 header 를 통해 지정할 수 있다.

요청 Request 형식:

```
GET / HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: signatureValue
```

요청, 응답 예시 :

```
GET / HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01">
    <Name>bucket</Name>
    <Prefix/>
    <Marker/>
    <MaxKeys>1000</MaxKeys>
    <IsTruncated>false</IsTruncated>
    <Contents>
        <Key>my-image.jpg</Key>
        <LastModified>2009-10-12T17:50:30.000Z</LastModified>
        <ETag>&quot;fba9dede5f27731c9771645a39863328&quot;</ETag>
        <Size>434234</Size>
        <StorageClass>STANDARD</StorageClass>
        <Owner>
            <ID>8a6925ce4a7f21c32aa379004fef</ID>
            <DisplayName>mtd@amazon.com</DisplayName>
        </Owner>
    </Contents>
    <Contents>
        <Key>my-third-image.jpg</Key>
        <LastModified>2009-10-12T17:50:30.000Z</LastModified>
        <ETag>&quot;1b2cf535f27731c974343645a3985328&quot;</ETag>
        <Size>64994</Size>
        <StorageClass>STANDARD</StorageClass>
        <Owner>
            <ID>8a69b1ddee97f21c32aa379004fef</ID>
            <DisplayName>mtd@amazon.com</DisplayName>
        </Owner>
    </Contents>
</ListBucketResult>
```

● DELETE Bucket

bucket을 삭제한다. bucket 내부에 있는 모든 object가 함께 삭제되므로 주의한다.

요청 Request 형식:

```
DELETE / HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: signatureValue
```

요청, 응답 예시 :

```
DELETE / HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

```
HTTP/1.1 204 No Content
x-amz-id-2:
JuKZqmXuiwFeDQxhD7M8KtsKobSzWA1QEjLbTMTagkKdBX2z7Il/jGhDeJ3j6s80
x-amz-request-id: 32FE2CEB32F5EE25
Date: Wed, 01 Mar 2009 12:00:00 GMT
Connection: close
Server: AmazonS3
```

● PUT Object

PUT Object 동작은 bucket에 object(파일)를 추가할 수 있으며 반드시 write 권한을 가진 계정으로만 동작한다.
PUT Object의 경우 HTTP Request 의 header 인수로 Content-Length 가 필수로 포함되어있어야 한다.

| Name | Description |
|---|---|
| *Content-Length* | Object 의 크기<br>자세한 정보는 http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.13. 참조<br>Type : 문자열(String)<br>Default : 없음(None)<br>Constraint : 없음(None) |

header 인수로 포함되는 추가 정보는 여러 가지가 있으며, 이에 대해서는 S3 홈페이지에서 제공되는 API Reference를 참조하도록 한다.

요청 Request 형식:

```
PUT /ObjectName HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: signatureValue
Content-Length : Object size
[Upload Object data]
```

요청, 응답 예시 :

```
PUT /my-image.jpg HTTP/1.1
Host: myBucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 11434
Expect: 100-continue
[11434 bytes of object data]
```

```
HTTP/1.1 100 Continue
HTTP/1.1 200 OK
x-amz-id-2: LriYPLdmOdAiIfgSm/F1YsViT1LW94/xUQxMsF7xiEb1a0wiIOIxl+zbwZ163pt7
x-amz-request-id: 0A49CE4060975EAC
Date: Wed, 12 Oct 2009 17:50:00 GMT
ETag: "1b2cf535f27731c974343645a3985328"
Content-Length: 0
Connection: close
Server: AmazonS3
```

● GET Object

GET Object 동작은 Amazon S3로부터 object를 다운받을 수 있으며, 반드시 read 권한이 있는 계정으로만 동작한다.

요청 Request 형식:

```
GET /ObjectName HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: signatureValue
Range:bytes=byte_range
```

요청, 응답 예시 :

```
GET /my-image.jpg HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS 02236Q3V0WHVSRW0EXG2:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

```
HTTP/1.1 200 OK
x-amz-id-2: eftixk72aD6Ap51TnqcoF8eFidJG9Z/2mkiDFu8yU9AS1ed4OpIszj7UDNEHGran
x-amz-request-id: 318BC8BC148832E5
Date: Wed, 28 Oct 2009 22:32:00 GMT
Last-Modified: Wed, 12 Oct 2009 17:50:00 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 434234
Content-Type: text/plain
Connection: close
Server: AmazonS3
[434234 bytes of object data]
```

● DELETE Object

DELETE Object 동작은 bucket에서 object를 삭제한다.

요청 Request 형식:

```
DELETE /ObjectName HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Content-Length: length
Authorization: signatureValue
```

요청, 응답 예시 :

```
DELETE /my-second-image.jpg HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
```

```
HTTP/1.1 204 NoContent
x-amz-id-2: LriYPLdmOdAiIfgSm/F1YsViT1LW94/xUQxMsF7xiEb1a0wiIOIxl+zbwZ163pt7
x-amz-request-id: 0A49CE4060975EAC
Date: Wed, 12 Oct 2009 17:50:00 GMT
Content-Length: 0
Connection: close
Server: AmazonS3
```

### 3.2.3. 예제 소스

S3 클래스를 이용한 PUT Object 구현 (/sec_s3_ex/index.php)

```
<?php
    //include the S3 class
    if (!class_exists('S3'))require_once('S3.php');

    //AWS access info
    if (!defined('awsAccessKey')) define('awsAccessKey','xxxxxxxxxxxxxxxx');
    if (!defined('awsSecretKey')) define('awsSecretKey', 'xxxxxxxxxxxxxxxx');
    if (!defined('awsBucketName')) define('awsBucketName', 'xxx');

    //instantiate the class
    $s3 = new S3(awsAccessKey, awsSecretKey);

    //retreive post variables
    $fileName = $_FILES['theFile']['name'];
    $fileTempName = $_FILES['theFile']['tmp_name'];

    //create a new bucket
    $s3->putBucket(awsBucketName, S3::ACL_PUBLIC_READ);

    //move the file
    if ($s3->putObjectFile($fileTempName, awsBucketName, $fileName, S3::ACL_PUBLIC_READ)) {
        echo "<strong>We successfully uploaded your file.</strong>";
    }else{
        echo "<strong>Something went wrong while uploading your file... sorry.</strong>";
    }
?>
```

S3 클래스 생성자를 통해 Access Key와 Secret Key를 설정해준 후, S3 클래스의 멤버 함수 putBucket, putObjectFile로 구현된 S3 API를 통해서 데이터를 주고받을 수 있다.

## 3.3. 질문과 답변

| Question 1. 클라우드 서비스란 무엇입니까? |
| --- |
| 클라우드 컴퓨팅은 네트워크로 연결된 인프라스트럭처 소프트웨어를 이용하여 사용자에게 온디멘드 환경의 자원을 제공하는 것입니다. 유틸리티 컴퓨팅이라고도 알려져 있습니다. 클라우드는 보편적으로 가상화된 컴퓨터들의 클러스터를 제공하여 사용자들로 하여금 서버를 시작하거나 종료할 수 있게 하고 필요할 때에만 전산 처리를 사용하게 하며 오직 그 서비스의 사용에 대해서만 과금을 할 수 있게 합니다. |

| Question 2. 클라우드 서비스에는 어떤 것들이 있습니까? |
| --- |
| 아마존의 AWS (Amazon Web Service) 홈페이지에서 제공하는 S3, Simple DB 서비스 등이 Hadoop 기술을 기반으로 클라우드 데이터 서비스를 제공하고 하고 있으며, 세일즈포스 닷컴에서 SaaS 형식의 온디멘드 서비스로서 다양한 컴포넌트를 기반으로 클라우드 환경의 그룹웨어를 제공하고 있습니다. 아울러, 국내에서는 KT에서 2010년 11월 아마존보다 저렴한 '유클라우드'라는 클라우드 데이터 서비스를 시작했습니다. |

| Question 3. Amazon S3를 일반 파일 시스템처럼 사용할 수 있습니까? |
| --- |
| bucket의 목록화할때 Prefix 와 Delimiter 인수를 사용하여 파일 시스템 구조처럼 사용할 수 있습니다. 객체를 저장할 때에는 일반 파일 시스템 경로를 key 값으로 지정하여 사용할 수 있습니다. 예를 들면, 다음의 형태로 key를 저장할 수 있습니다.<br>john/setting/conf.txt<br>jane/setting/conf.txt<br>인수를 사용하여 일반 파일 시스템 구조를 보여줄 수 있다고 해도, Amazon S3 를 일반 파일 시스템이 아니라 분산 시스템으로서 동작 제한이 있습니다. 예를들면, 객체 업데이트가 전파되기 위해서는 몇 초가 걸릴 수 있습니다. |

| Question 4. Amazon S3의 bucket을 rename, copy, move 할 수 있습니까? |
| --- |
| 현재로서는 기존의 bucket에 대하여 해당 명령이 지원되지 않습니다. 기존의 bucket 의 데이터를 다운로드한 후 새로 생성된 bucket으로 다시 업로드해야 합니다. |

| Question 5. Amazon S3에 요청이 실패한 통신에 대해서도 과금되나요? |
| --- |
| 시스템 자원의 사용량에 대해서 균등하게 과금을 하는 것이 목적이므로 403 에러와 404에러에 대해서도 (다른 요청들과 마찬가지로) 과금을 합니다. 단, Amazon S3 의 내부 시스템 에러로 인한 요청 실패에 대해서는 과금하지 않습니다. |

※ 기타 궁금하신 부분에 대해서는 소프트웨어 공학센터 홈페이지나 담당자 이메일로 문의해주시기 바랍니다.

## 4. 참고자료

[1] Gartner Homepage (http://www.gartner.com/technology/research/cloud-computing/index.jsp)에서 인용
[2] 출처 http://www.jansipke.nl/top-cloud-computing-providers
[3] 출처 http://infolayan.blogspot.com/2010/05/cloud-computing.html
[4] Google Research http://research.google.com
[5] Google File System http://labs.google.com/papers/gfs.html
[6] MapReduce http://labs.google.com/papers/mapreduce.html
[7] BigTable http://labs.google.com/papers/bigtable.html
[8] Chubby http://labs.google.com/papers/chubby.html
[9] 『클라우드 컴퓨팅 구현 기술』 김형준/조준호/안성화/김병준 지음 , 에이콘 에서 인용
[10] Casandra http://cassandra.apache.org/
[11] Amazon Simple Storage Service http://aws.amazon.com/s3