

---

# Hadoop

---

2015

---

# Content

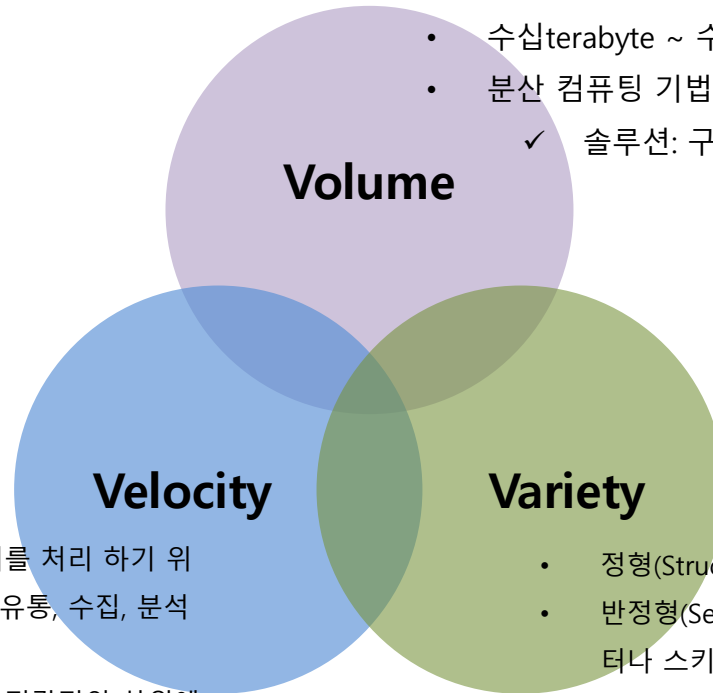
---

- Introduction
    - Big Data
    - Hadoop
  - Hadoop 기본 개념
  - Hadoop Eco System
  - Hadoop 설치
  - HDFS(Hadoop Distributed File System)
  - Hadoop 2.0
-

# Big Data

---

- 빅 데이터의 3대 요소: 크기(Volume), 속도(Velocity), 다양성(Variety)



- 수십terabyte ~ 수십petabyte
  - 분산 컴퓨팅 기법으로 데이터를 저장, 분석해야 함
    - ✓ 솔루션: 구글의 GFS, 아파치의 하둡
- 빠른 속도로 생성되는 디지털 데이터를 처리 하기 위해 실시간으로 데이터를 생산, 저장, 유통, 수집, 분석 처리해야 함
  - 수집된 대량의 데이터를 장기적이고 전략적인 차원에서 접근하여 분석해야 함
    - ✓ 데이터 마이닝, 기계 학습, 자연어 처리, 패턴 인식 등을 활용
- 정형(Structured) 데이터: 고정된 필드에 저장되는 데이터
  - 반정형(Semi-structured) 데이터: XML, HTML 같이 메타데이터나 스키마 등을 포함하는 데이터
  - 비정형(Unstructured) 데이터: 인터넷 상에서 발생하는 SNS 데이터, 동영상, 위치정보, 통화 내용 등

# Hadoop

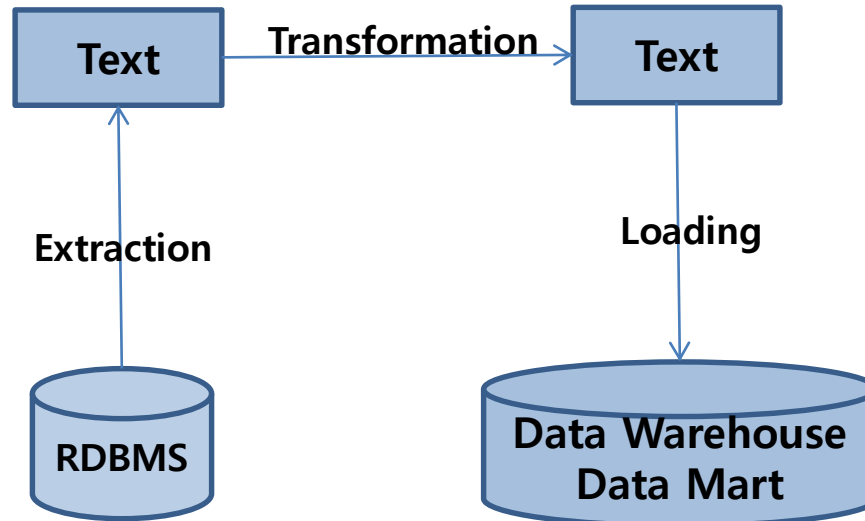
---

- 하둡이란
    - 대용량 데이터를 분산 처리할 수 있는 자바 기반의 오픈소스 프레임워크
    - 분산 파일 시스템인 **HDFS(Hadoop Distributed Files System)**에 데이터를 저장하고 분산 처리 시스템인 **맵리듀스**를 이용해 데이터를 처리
    - 2005년에 더그 커팅(Doug Cutting)이 구글이 논문으로 발표한 GFS(Google File System)와 MapReduce를 구현한 결과물
    - 데이터의 복제본을 저장하기 때문에 데이터의 유실이나 장애가 발생했을 때도 데이터의 복구가 가능함
  - ✓ 정형 데이터 → RDBMS저장 (RDBMS는 라이선스 비용이 비쌈)
    - RDBMS: 데이터가 저장된 서버에서 데이터를 처리
  - ✓ 비정형 데이터(사이즈 큼) → Hadoop 저장
    - 하둡: 여러 대의 서버에 데이터를 저장하고, 데이터가 저장된 각 서버에서 동시에 데이터를 처리
-

# Hadoop

---

- Q1. 하둡이 MS-SQL, Sybase, MySQL같은 RDBMS를 대체한다?
  - 하둡은 기존 RDBMS를 대체하지 않음. 오히려 RDBMS와 상호 보완적인 특성을 가짐
  - 하둡은 ETL과정에 도움을 줌(ETL; Extraction, Transformation, Loading)
    - ETL은 자체적으로 셸 스크립트나 SQL문을 이용해 진행하거나, Data Stage같은 상용 솔루션을 이용해 진행



# Hadoop

---

- Q2. 하둡은 NoSQL? (RDBMS가 아니니까 NoSQL이다?)
    - 하둡은 SQL언어를 사용할 수 있음 (Hive에서 HiveQL이라는 쿼리 언어 제공)
  - NoSQL
    - 관계형 데이터 모델과 SQL문을 사용하지 않는 데이터베이스 시스템 혹은 데이터 저장소
    - 단순히 키와 값의 쌍으로만 이뤄져 있음
    - 인덱스와 데이터가 분리되어 별도로 운영됨
    - 조인이 없음
    - 데이터를 하나의 집합된 형태로 저장
    - Sharding : 데이터를 분할해서 다른 서버에 나누어 저장
  - RDBMS
    - 엔티티 간의 관계에 중점을 두고 테이블 구조를 설계하는 방식
    - 데이터가 여러 행(row)으로 존재함
    - 핵심데이터 관리 (데이터 무결성과 정합성 제공)
-

# Hadoop

---

YAHOO!

amazon.com®

ebay

facebook®

myspace.com

1-1-1-1

allego kt

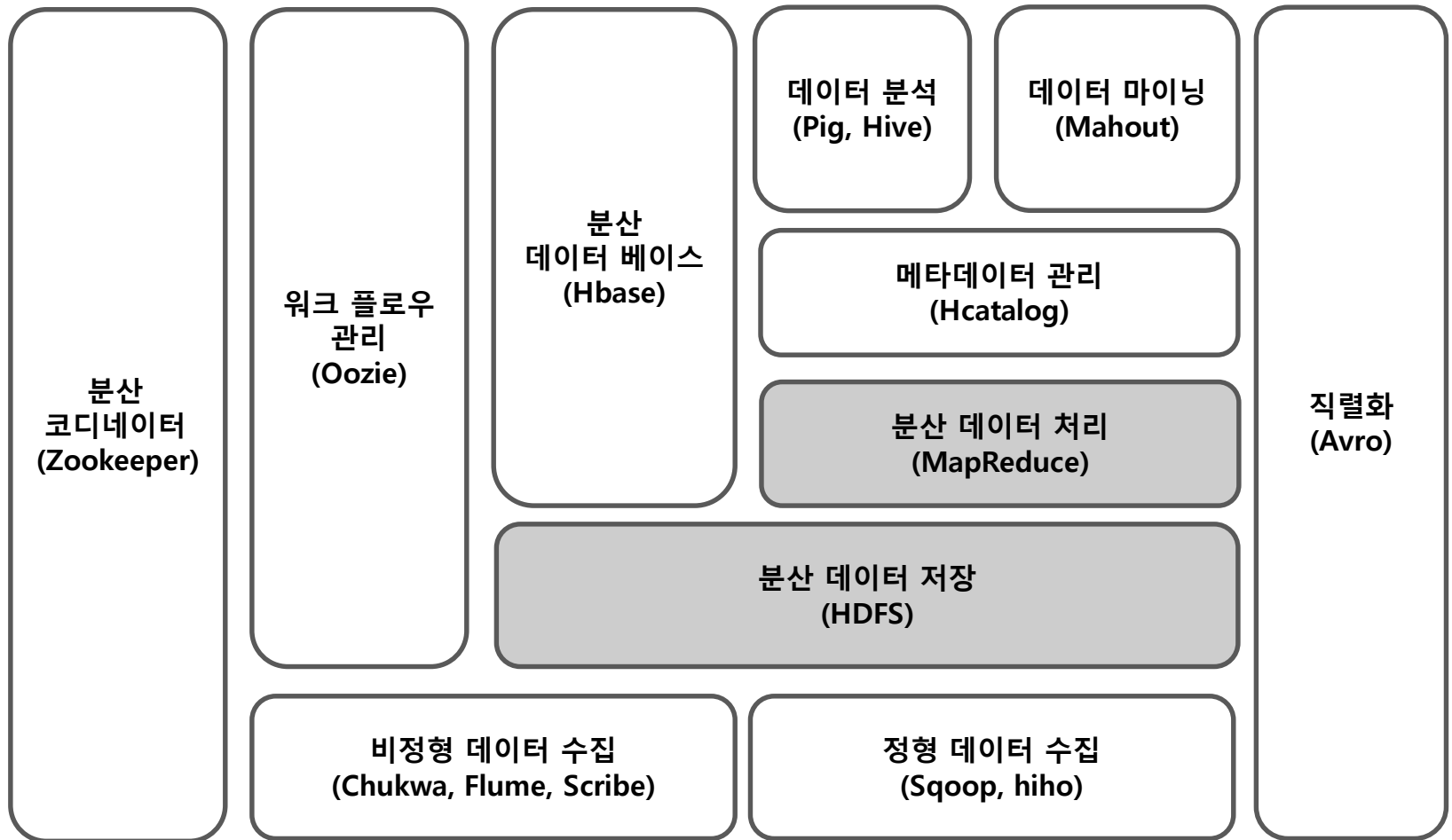
Dum

SK telecom

---

# Hadoop Ecosystem

- 비즈니스에 효율적으로 적용하기 위한 다양한 서브 프로젝트





# Hadoop Ecosystem

---

분산  
코디네이터  
(Zookeeper)

## Zookeeper

분산 환경에서 서버 간의 상호 조정이 필요한 다양한 서비스를 제공하는 시스템

1. 하나의 서버에만 서비스가 집중되지 않게 서비스를 알맞게 분산해 동시에 처리하게 해줌
2. 하나의 서버에서 처리한 결과를 다른 서버와도 동기화해서 데이터의 안정성을 보장해줌
3. 운영(active) 서버가 문제가 발생해서 서비스를 제공할 수 없을 경우, 다른 대기 중인 서버를 운영서버로 바꿔서 서비스가 중지 없이 제공되게 함
4. 분산 환경을 구성하는 서버들의 환경설정을 통합적으로 관리

# Hadoop Ecosystem

---

워크 플로우  
관리  
(Oozie)

분산  
데이터 베이스  
(Hbase)

## Hbase

- HDFS 기반의 칼럼 기반 데이터베이스
- 실시간 랜덤 조회 및 업데이트 가능
- 각 프로세스는 개인의 데이터를 비동기적으로 업데이트
- 단, 맵리듀스는 일괄 처리 방식으로 수행됨
- 트위터, 야후!, 어도비, 국내 NHN(모바일 메신저 Line)

## Oozie

- 하둡 작업을 관리하는 워크플로우 및 코디네이터 시스템
- 자바 서블릿 컨테이너에서 실행되는 자바 웹 애플리케이션 서버
- 맵리듀스 작업이나 피그(데이터 분석) 작업 같은 특화된 액션으로 구성된 워크 플로우를 제어

# Hadoop Ecosystem

---

## Pig

- 야후!에서 개발, 현재 아파치 프로젝트에 속함
- 복잡한 맵리듀스 프로그래밍을 대체할 Pig Latin이라는 자체 언어 제공
- 맵리듀스 API를 크게 단순화함
- SQL과 유사한 형태. 단, SQL 활용이 어려운 편임

## Hive

- 데이터웨어하우징용 솔루션
- 페이스북에서 개발, 현재 아파치 프로젝트에 속함
- SQL과 매우 유사한 **HiveQL** 쿼리 제공(내부적으로 맵리듀스 잡으로 변환되어 실행됨)
- 자바를 모르는 데이터 분석가들도 쉽게 하둡 데이터를 분석할 수 있게 도와줌
- 짧은 임시쿼리보다는 일괄적인 MapReduce처리에 이상적임

데이터 분석  
(Pig, Hive)

데이터 마이닝  
(Mahout)

## Mahout

- 하둡 기반으로 데이터 마이닝 알고리즘을 구현하였음
  - 분류(Classification)
  - 클러스터링(Clustering)
  - 추천 및 협업 필터링(Recommenders/Collaborative filtering)
  - 패턴 마이닝(Pattern Mining)
  - 회귀 분석(Regression)
  - 차원 리덕션(Dimension reduction)
  - 진화 알고리즘(Evolutionary Algorithms)

# Hadoop Ecosystem

---

## Hcatalog

- 하둡으로 생성한 데이터를 위한 테이블 및 스토리지 관리 서비스
- 하둡 에코시스템 간의 상호운용성 향상에 큰 영향
- Hcatalog의 이용으로 Hive에서 생성한 테이블이나 데이터 모델을 Pig나 맵리듀스에서 손쉽게 이용할 수 있음 (이 전엔 모델 공유는 가능했으나, 상당한 백엔드 작업이 필요했었음)

메타데이터 관리  
(Hcatalog)

직렬화  
(Avro)

## Avro

- RPC(Remote Procedure Call)과 데이터 직렬화를 지원
- JSON을 이용해 데이터 형식과 프로토콜을 정의
- 작고 빠른 바이너리 포맷으로 데이터를 직렬화

# Hadoop Ecosystem

---

## Chukwa

- 분산 환경에서 생성되는 데이터를 HDFS에 안정적으로 저장하는 플랫폼(야후!에서 개발)
- 분산된 각 서버에서 에이전트(agent)를 실행하고, 콜렉터(collector)가 에이전트로부터 데이터를 받아 HDFS에 저장
- 콜렉터는 100개의 에이전트당 하나씩 구동
- 데이터 중복 제거 등 작업은 맵리듀스로 처리

## Flume

- Chukwa처럼 분산된 서버에 에이전트가 설치되고, 에이전트로부터 데이터를 전달받는 콜렉터로 구성
- 전체 데이터의 흐름을 관리하는 마스터 서버가 있음
- 즉, 데이터를 어디서 수집, 어떤 방식으로 전송, 어디에 저장할지를 동적으로 변경할 수 있음(클라우드라 개발)

## Scribe

- 데이터 수집 플랫폼(페이스북에서 개발)
- 데이터를 중앙 집중 서버로 전송
- 최종 데이터는 HDFS 외 다양한 저장소를 활용할 수 있음
- 설치와 구성이 쉽도록 다양한 프로그래밍 언어를 지원
- HDFS에 데이터 저장을 위해 JNI(Java Native Interface) 이용해야 함

## Sqoop

- 대용량 데이터 전송 솔루션
- HDFS, RDBMS, DW, NoSQL 등 다양한 저장소에 대용량 데이터를 신속하게 전송할 수 있는 방법 제공
- 상용RDBMS도 지원하고, MySQL, PostgreSQL 오픈소스 RDBMS도 지원함

## Hiho

- 대용량 데이터 전송 솔루션
- 하둡에서 데이터를 가져오기 위한 SQL을 지정할 수 있음
- JDBC 인터페이스 지원
- 오라클과 MySQL의 데이터 전송만 지원함

비정형 데이터 수집  
(Chukwa, Flume, Scribe)

정형 데이터 수집  
(Sqoop, hiho)

# Hadoop 설치

---

- Virtual Box에 하둡 설치
- 실습 순서
  - Virtual Box 설치 및 설정
  - 1개의 가상머신을 생성 : 호스트명(hadoop01)
  - Linux 설치 -> 하둡 설치

# Hadoop 설치

- **Virtual Box 설치**

- Virtual Box 설치(

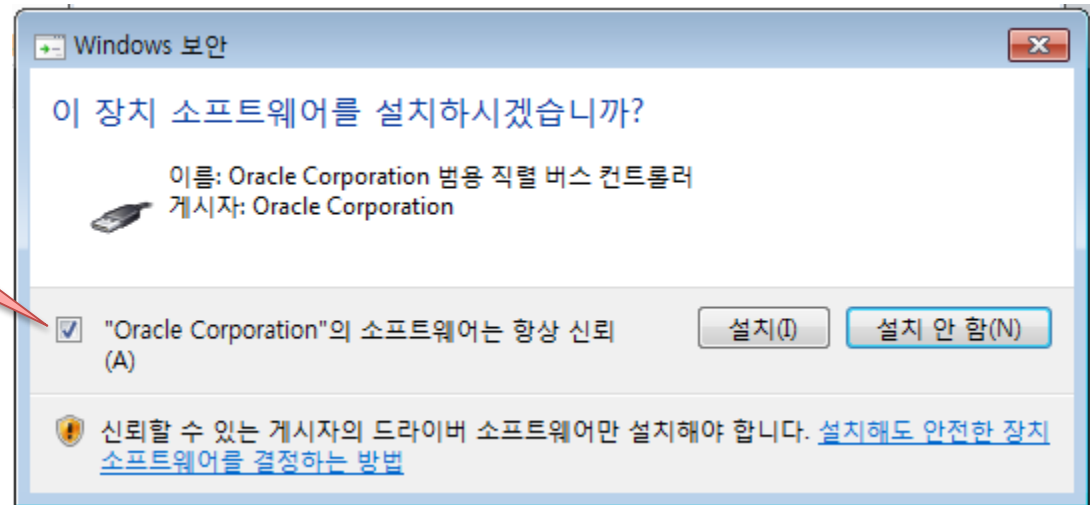
- 다운로드 : <http://download.virtualbox.org/virtualbox/4.3.26/VirtualBox-4.3.26-98988-Win.exe>
    - 파일 : VirtualBox-4.3.26-98988-Win.exe

- **[실행]**

- 설치 시 유의사항

- 네트워크에 문제가 발생할 경우 Virtual Box를 삭제한 후 PC를 리부팅하고 다시 설치하세요.
    - [Install] 과정에서 다음과 같은 창이 뜹니다.

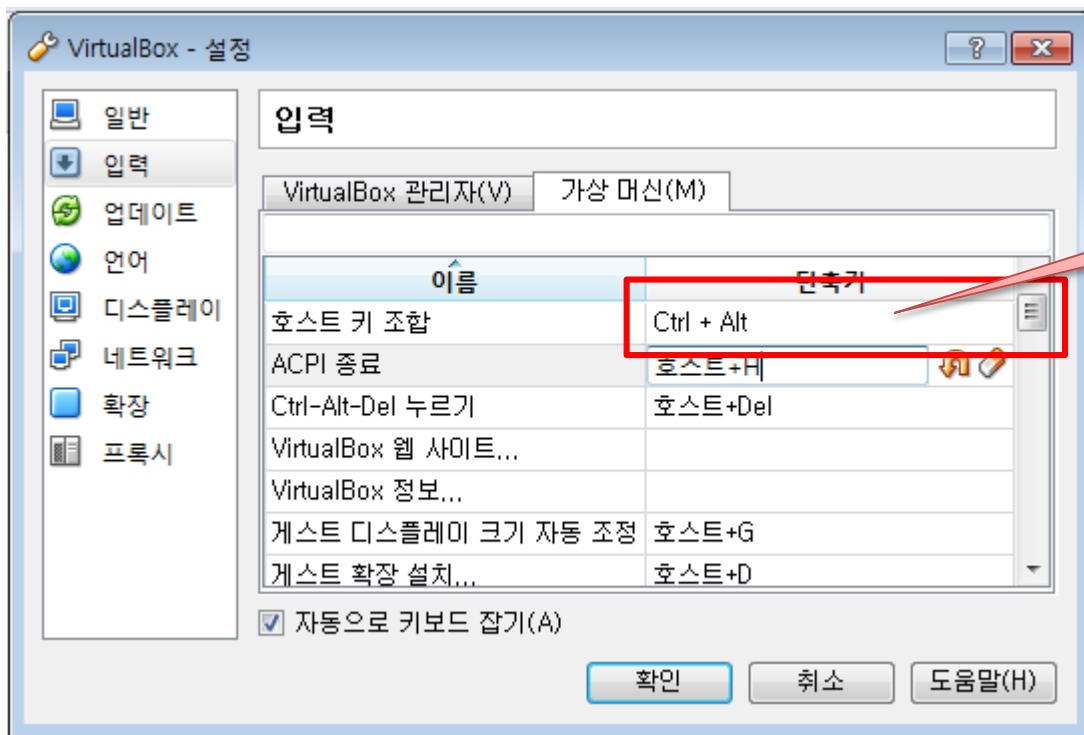
[v] 체크하세요



# Hadoop 설치

- **Virtual Box 설정**

- 가상머신 윈도우에서 쉽게 빠져나올 수 있도록 [호스트 키 조합]을 설정
- 파일(F) -> [환경설정] -> 입력 -> 두번째 TAB [가상 머신(M)]
- 호스트 키 조합의 단축키를 [ Ctrl + Alt ]로 변경 후 키보드 [아래 화살표] 누른 후 [확인]



Ctrl + Alt



# Hadoop 설치

- 가상머신 생성

- 가상머신 생성

- Virtual Box 어플리케이션에서 [새로 만들기]

- 설치 시 유의사항

- 가상 머신의 이름과 종류 및 버전

- [이름] **hadoop01** [NOTE] hadoop02,

- [종류] **Linux**

- [버전] **Red Hat 또는 Red Hat(32비트)**

세요.

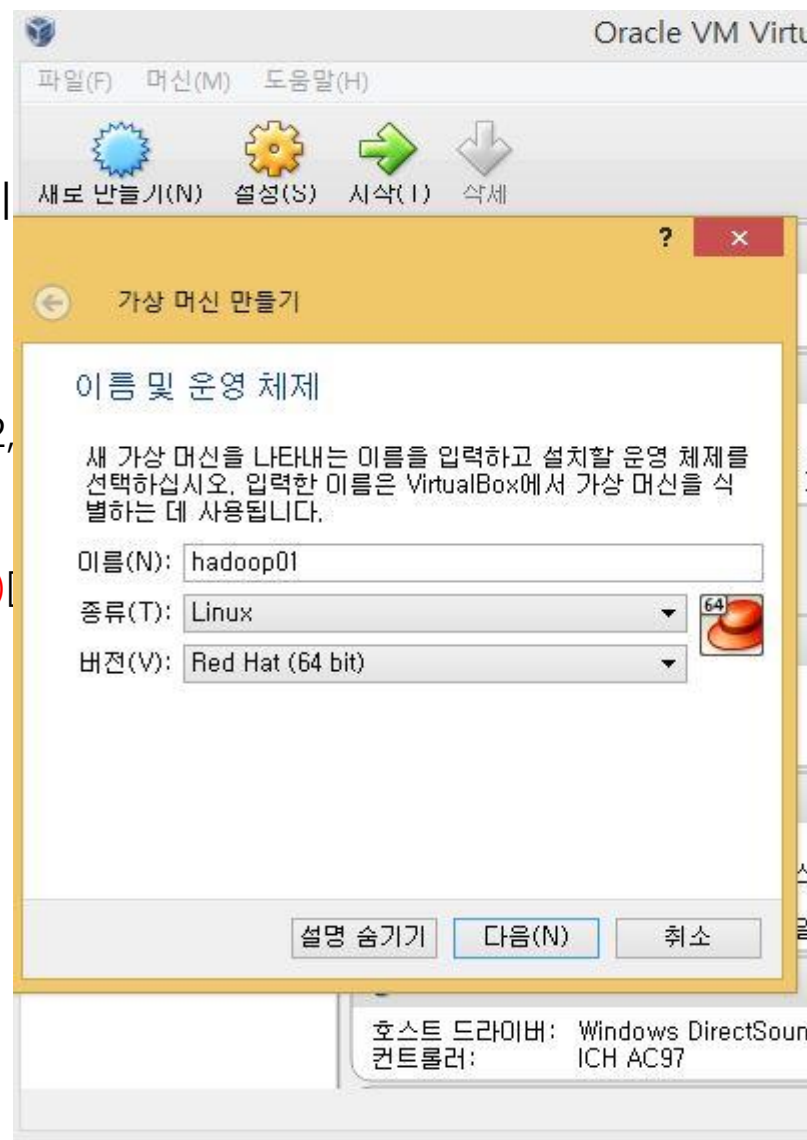
- [메모리 크기] **권장 2GB** / 최대 2.8GB

- [하드 드라이브]

- [하드 드라이브 종류] **VDI**

- » [동적/고정] 고정 크기

- » [파일 크기] 10GB



# Hadoop 설치

- 가상머신 설정

- 실습 관련

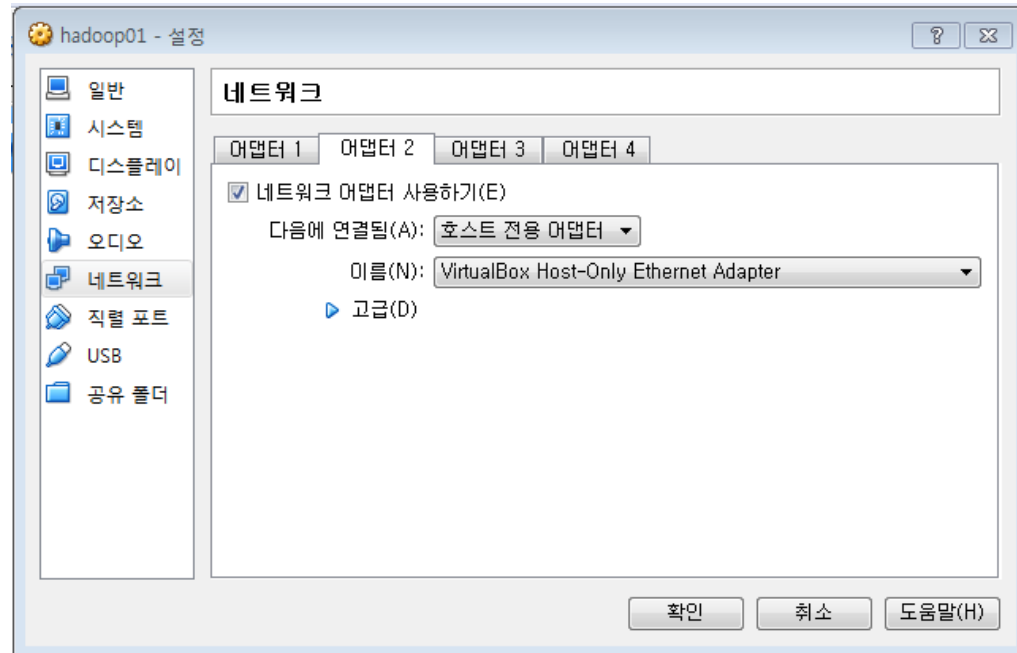
- Case 1 : 리눅스 부터 하둡 설치까지 직접 설치
    - Case 2 : 리눅스와 하둡이 설치된 VDI 파일을 사용

- 설정 시 유의사항

- [네트워크] 두번째 TAB [어댑터2]

[V] 네트워크 어댑터 사용하기 체크

연결 : 호스트 전용 어댑터



# Hadoop 설치

---

- 리눅스 설치

- 설치 개요

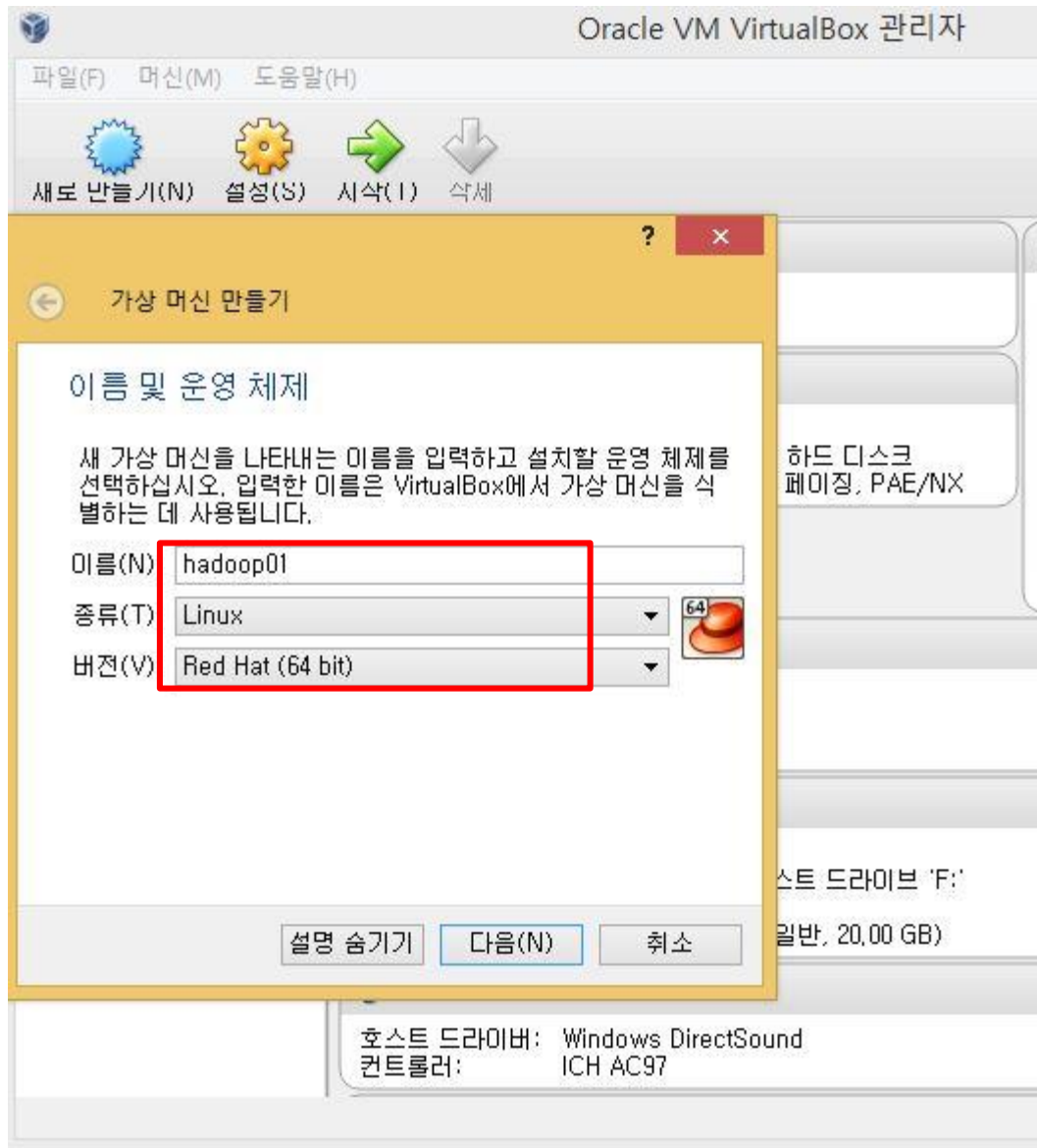
- CentOS 6.X 32비트 설치 [NOTE] 64비트는 설치 안되는 경우가 많음
    - 파티션 설정이나 사용자 정의 패키지 설치는 학습 범위를 벗어남

- 설치 시 유의사항

- [호스트명] **hadoop01**
      - [네트워크 설정]  
eth0, eth1 : **[V] 자동으로 연결 선택**
      - [root 비밀번호] **hadoop**
      - [소프트웨어 설치] **Desktop** 선택
-

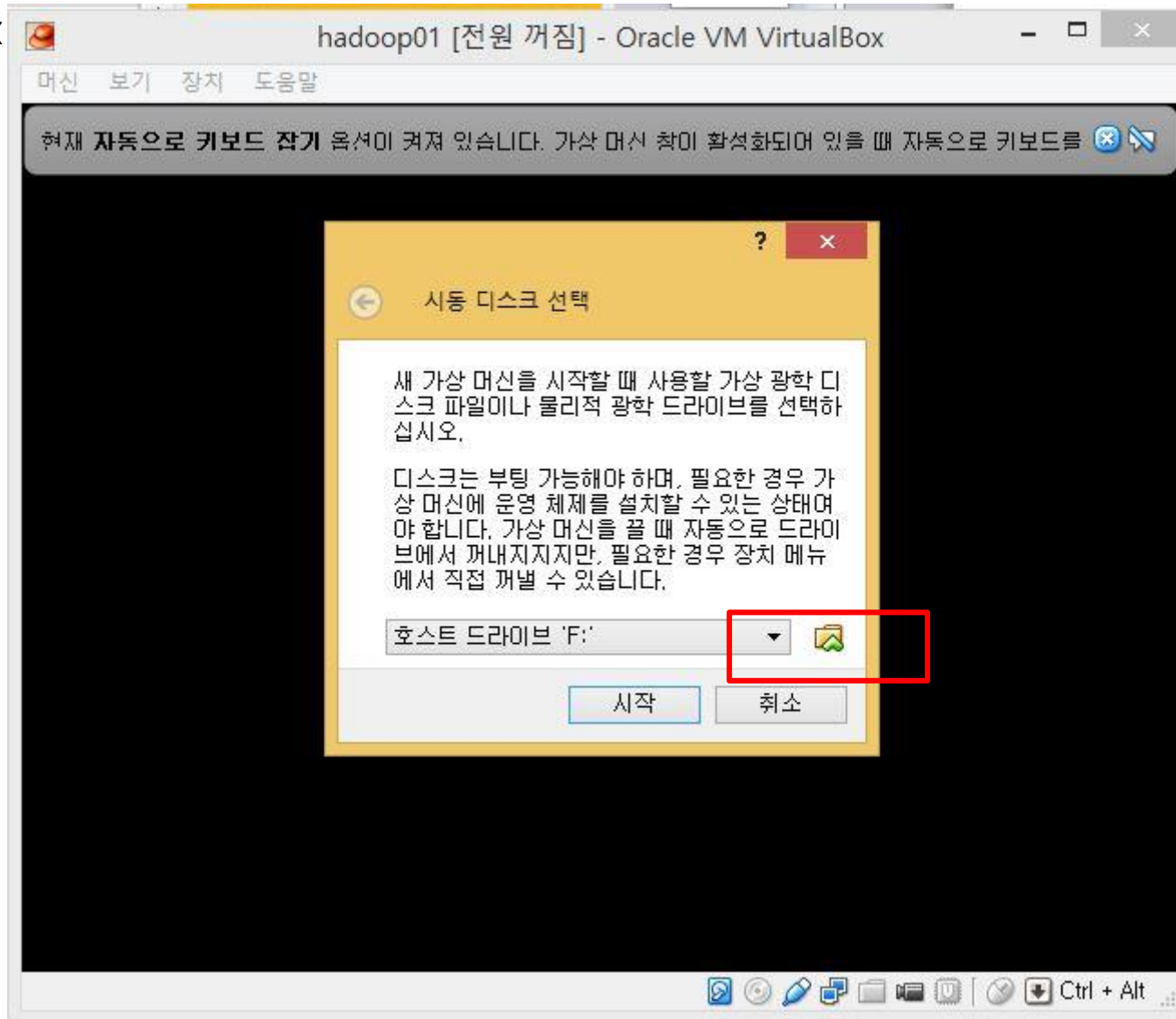
# Hadoop 설치

- 



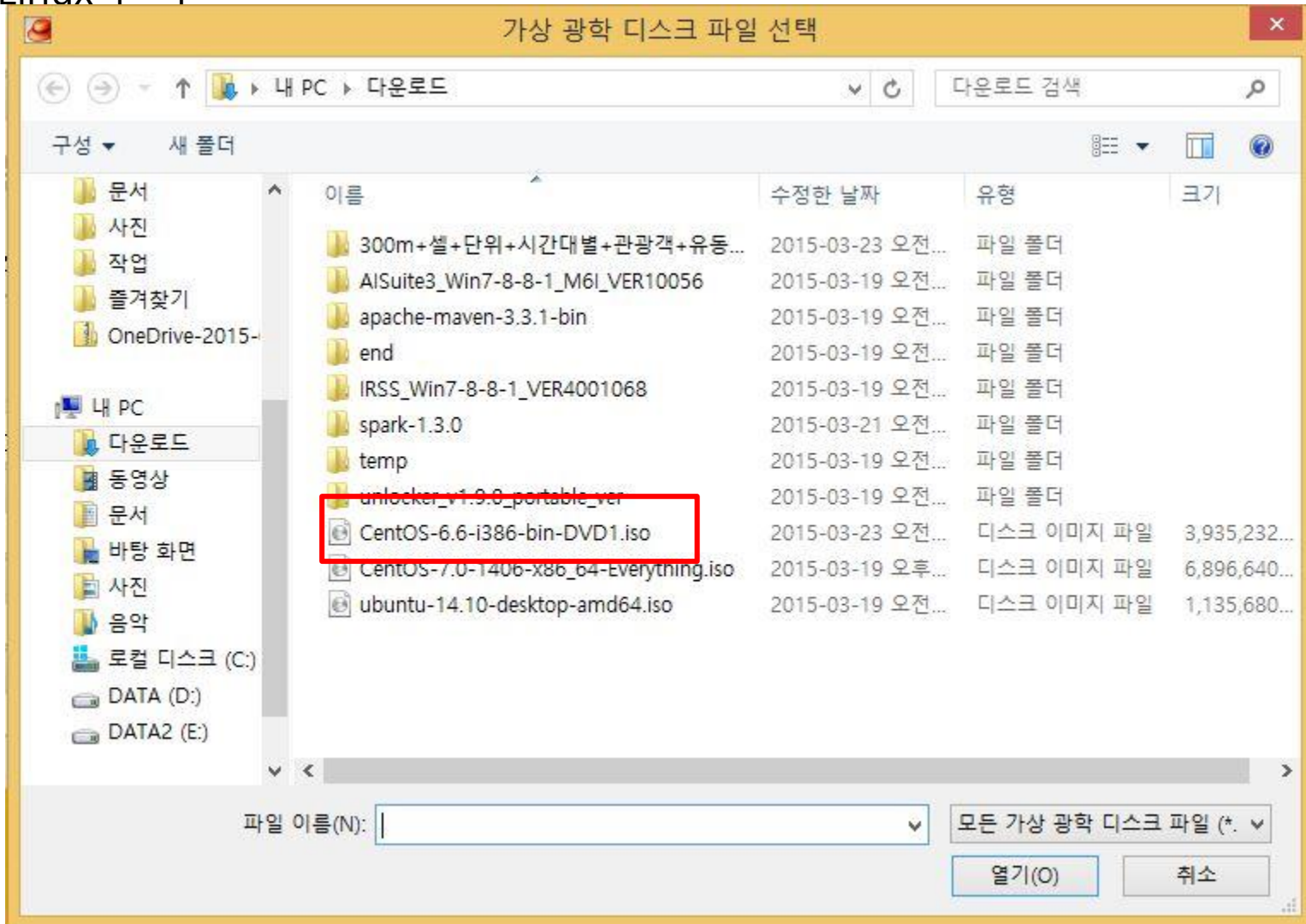
# Hadoop 설치

- Linux



# Hadoop 설치

- Linux 설치



Welcome to CentOS 6.6!

- 1** Install or upgrade an existing system
- 2 Install system with basic video driver
- 3 Rescue installed system
- 4 Boot from local drive
- 5 Memory test

Press [Tab] to edit options

Automatic boot in 59 seconds...

**CentOS 6**  
Community ENTERprise Operating System



# Hadoop 설치

---

- **Hadoop 설치**

- 설치 개요

- **root 계정 :**

- 네트워크 설정
      - hadoop 계정 생성
      - 방화벽 설정
      - JDK 설치
      - 환경 설정
      - 저장 디렉토리 생성

- **hadoop 계정 :**

- 하둡 설치
        - 하둡 설정
        - HDFS 포맷
        - 하둡 서비스 시작/중단
        - 하둡 서비스 확인
        - ( 개인 환경 설정 )
-



# Hadoop 설치

---

## ●Hadoop 하둡 설치 – root 계정

### ■네트워크 설정

- IP 확인하기( 192.168.56.101 혹은 102, 103 ... )

```
[root@hadoop01 ~]# ifconfig
```

- 호스트 설정( 기존에 있는 2줄 아래에 추가 )

```
[root@hadoop01 ~]# vi /etc/hosts
```

```
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
```

```
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

### **10.0.2.15 hadoop01**

- 방화벽 중단하기

```
[root@hadoop01 ~]# ntsysv
```

```
....
```

```
[*] ip6tables -> [ ] ip6tables
```

```
[*] iptables -> [ ] iptables
```

```
....
```

```
[root@hadoop01 ~]# iptables -F
```

---

# Hadoop 설치

---

## ●root 계정(계속)

### ■하둡 계정 추가 / JDK 설치 / 환경 설정

- 리눅스 하둡 계정 추가(이미 존재한다면 넘어감)

```
[root@hadoop01 ~]# adduser hadoop
```

- JDK설치

- 다운로드 <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>

```
[root@hadoop01 ~]# cp ./jdk-7u75-linux-i586.tar.gz /usr/local
```

```
[root@hadoop01 ~]# cd /usr/local
```

```
[root@hadoop01 local]# tar xvfz jdk-7u75-linux-i586.tar.gz
```

```
[root@hadoop01 local]# ln -s jdk1.7.0_75 java
```

- 리눅스 전역 환경 설정

```
[root@hadoop01 ~]# vim /etc/profile
```

```
export JAVA_HOME=/usr/local/java
```

```
export PATH=$PATH:$JAVA_HOME/bin
```

```
export CLASSPATH=""
```

```
export BASEHOME=/home/hadoop
```

```
export HADOOP_HOME=$BASEHOME/hadoop
```

```
[root@hadoop01 ~]# source /etc/profile
```

- 확인하기( jps 앞에 나오는 7786은 프로세스 ID로 계속 달라짐 )

```
[root@hadoop01 ~]# javac -version
```

```
javac 1.8.0_31
```

```
[root@hadoop01 ~]# jps
```

```
7786 Jps
```

---

# Hadoop 설치

---

## ●계정(계속)

- 실제 데이터 저장 디렉토리 생성

```
[root@hadoop01 ~]# mkdir -p /data/hadoop/tmp
```

```
[root@hadoop01 ~]# mkdir -p /data/hadoop/dfs/name
```

```
[root@hadoop01 ~]# mkdir -p /data/hadoop/dfs/data
```

- /data/hadoop owner 변경

```
[root@hadoop01 ~]# chown -R hadoop:hadoop /data/hadoop
```

# Hadoop 설치

---

## ●하둡 설치 – hadoop 계정

- hadoop 계정으로 작업

```
[root@hadoop01 ~]# su - hadoop
```

```
[hadoop@hadoop01 ~]$
```

- SSH 설정

-ssh 디렉토리 만들기( 비밀번호 입력하지 말고 그냥 ENTER 키를 여러번 입력하세요 )

```
[hadoop@hadoop01 ~]$ ssh localhost
```

The authenticity of host 'localhost (::1)' can't be established.

RSA key fingerprint is 78:28:9d:9e:0e:4f:03:9c:fd:8d:60:d1:78:13:8d:33.

Are you sure you want to continue connecting (yes/no)? **yes**

-SSH 설정 순서

- 1) 열쇠+자물쇠 생성하기
- 2) 인증키박스(현관)에 자물쇠 등록
- 3) 확인하기

# Hadoop 설치

---

- (계속) hadoop 계정

- 환경 변수 등록

```
[hadoop@hadoop01 ~]$ vim ~/.bash_profile
```

```
#JAVA
```

```
export JAVA_HOME=/usr/local/java
```

```
export PATH=$PATH:$JAVA_HOME/bin
```

```
export HADOOP_HOME=/home/hadoop/hadoop
```

```
export PATH=$PATH:$HADOOP_HOME/bin
```

```
[hadoop@hadoop01 ~]$ source ~/.bash_profile
```

---

# Hadoop 설치

---

- (계속) **hadoop** 계정

- SSH 설정(계속)

- 열쇠+자물쇠 생성하기

- ```
[hadoop@hadoop01 ~]$ ssh-keygen -t dsa
```

- < 비밀번호 입력 하지 말고 엔터 3번 >

- 인증키박스에 자물쇠 등록

- ```
[hadoop@hadoop01 ~]$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

- ```
[hadoop@hadoop01 ~]$ chmod 400 ~/.ssh/authorized_keys
```

- 확인하기( localhost, hadoop01 모두 확인해야 함 )

- ```
[hadoop@hadoop01 ~]$ ssh localhost
```

- ```
[hadoop@hadoop01 ~]$ exit
```

- logout

- Connection to localhost closed.

- ```
[hadoop@hadoop01 ~]$ ssh hadoop01
```

- The authenticity of host 'hadoop01 (10.0.2.15)' can't be established.

- RSA key fingerprint is 78:28:9d:9e:0e:4f:03:9c:fd:8d:60:d1:78:13:8d:33.

- Are you sure you want to continue connecting (yes/no)? **yes**

- ```
[hadoop@hadoop01 ~]$ exit
```

- ```
[hadoop@hadoop01 ~]$ ll .ssh
```

# Hadoop 설치

---

## ●(계속) hadoop 계정

### ■하둡 설치

[ 하둡 버전 ] 2.6.0

[ 하둡 설정 ] : 개인 배포 설정(Free)

-하둡 설치( 다운로드, 압축 풀기, 심벌릭 링크 )

```
[hadoop@hadoop01 ~]$
```

```
wget http://apache.mirror.cdnetworks.com/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
```

```
[hadoop@hadoop01 ~]$ tar xvzf hadoop-2.6.0.tar.gz
```

```
[hadoop@hadoop01 ~]$ ln -s /home/hadoop/hadoop-2.6.0 /home/hadoop/hadoop
```

-마스터와 슬레이브 추가

- \$HADOOP\_HOME/etc/hadoop/masters

localhost

- \$HADOOP\_HOME/etc/hadoop/slaves

localhost

-\$HADOOP\_HOME/etc/hadoop/hadoop-env.sh 수정

export JAVA\_HOME=/usr/local/java

---

# Hadoop 설치

---

- (계속) hadoop 계정

- 환경변수 추가

- \$HADOOP\_HOME/etc/hadoop/hadoop-env.sh

- ```
export HADOOP_COMMON_LIB_NATIVE_DIR=/home/hadoop/hadoop/lib/native
```

- ```
Export HADOOP_OPTS="-Djava.library.path=/home/hadoop/hadoop/lib"
```

- \$HADOOP\_HOME/etc/hadoop/yarn-env.sh

- ```
export HADOOP_COMMON_LIB_NATIVE_DIR=/home/hadoop/hadoop/lib/native
```

- ```
Export HADOOP_OPTS="-Djava.library.path=/home/hadoop/hadoop/lib"
```

---



# Hadoop 설치

---

- Core-site.xml

```
<configuration>  
  <property>  
    <name>fs.defaultFS</name>  
    <value>hdfs://localhost:9000</value>  
  </property>  
  <property>  
    <name>hadoop.tmp.dir</name>  
    <value>/data/hadoop/tmp</value>  
  </property>  
</configuration>
```

# Hadoop 설치

---

- **Hdfs-site.xml**

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/data/hadoop/dfs/name</value>
    <final>true</final>
  </property>
  <property>
    <name>dfs.namenode.checkpoint.dir</name>
    <value>/data/hadoop/dfs/namesecondary</value>
    <final>true</final>
  </property>
  <property>
    <name>dfs.http.address</name>
    <value>hadoop01:50070</value>
  </property>
  <property>
    <name>dfs.secondary.http.address</name>
    <value>hadoop01:50090</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/data/hadoop/dfs/data</value>
    <final>true</final>
  </property>
  <property>
    <name>dfs.permissions</name>
    <value>>false</value>
  </property>
</configuration>
```

---

# Hadoop 설치

---

- **Mapred-site.xml**

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapred.local.dir</name>
    <value>/data/hadoop/hdfs/mapred</value>
  </property>
  <property>
    <name>mapred.system.dir</name>
    <value>/data/hadoop/hdfs/mapred</value>
  </property>
</configuration>
```

---

# Hadoop 설치

---

- **Yarn-site.xml**

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
  <property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>hadoop01:8025</value>
  </property>
  <property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>hadoop01:8030</value> </property>
  <property>
    <name>yarn.resourcemanager.address</name>
    <value>hadoop01:8035</value>
  </property>
</configuration>
```

---

# Hadoop 설치

---

## ● 계속) hadoop 계정

### ▪ HDFS Format

```
[hadoop@hadoop01 ~]$ hadoop namenode -format
```

### ▪ Hadoop 실행

```
[hadoop@hadoop01 ~]$ ./hadoop/sbin/start-dfs.sh
```

```
[hadoop@hadoop01 ~]$ ./hadoop/sbin/start-yarn.sh
```

```
[hadoop@hadoop01 ~]$ ./hadoop/sbin/mr-jobhistory-daemon.sh start historyserver
```

### ▪ 실행 확인

```
[hadoop@hadoop01 ~]$ jps
```

웹에서 실행 확인하기

<http://localhost:50070>

-하둡 서비스 중단하기

```
[hadoop@hadoop01 ~]$ ./hadoop/sbin/stop-yarn.sh
```

```
[hadoop@hadoop01 ~]$ ./hadoop/sbin/stop-dfs.sh
```

---

# HDFS(Hadoop Distributed File System)

---

- HDFS(Hadoop Distributed File System)
  - 수십 테라바이트 또는 페타바이트 이상의 대용량 파일을 분산된 서버에 저장하고, 많은 클라이언트가 저장된 데이터를 빠르게 처리할 수 있게 설계된 파일 시스템
  - 저사양 서버를 이용해 스토리지를 구성할 수 있음
  - 대규모 데이터를 저장, 배치로 처리하는 경우

대용량 파일 시스템	특징
DAS(Direct-Attached Storage)	<ul style="list-style-type: none"><li>• 서버에 직접 연결된 스토리지</li><li>• 여러 개의 하드디스크를 장착할 수 있는 외장형 하드디스크</li></ul>
NAS(Network-Attached Storage)	<ul style="list-style-type: none"><li>• 파일 시스템을 안정적으로 공유할 수 있음</li><li>• 별도의 운영체제 사용</li></ul>
SAN(Storage Area Network)	<ul style="list-style-type: none"><li>• 수십-수백대의 SAN스토리지를 데이터 서버에 연결해 총괄적으로 관리해주는 네트워크</li><li>• DBMS와 같이 안정적으로 빠른 접근이 필요한 데이터를 저장하는데 사용(고성능, 고가용성)</li></ul>

# HDFS 설계 목표

---

## 1. 장애 복구

- 디스크 오류로 인한 데이터 저장 실패 및 유실과 같은 장애를 빠른 시간에 감지하고 대처
- 데이터를 저장하면, 복제 데이터도 함께 저장해서 데이터 유실을 방지함
- 분산 서버 간 주기적인 상태 체크

## 2. 스트리밍 방식의 데이터 접근

- HDFS에 파일 저자 및 조회를 위해 스트리밍 방식으로 데이터에 접근해야 함
- 배치 작업과 높은 데이터 처리량을 위해 스트리밍 방식을 사용

## 3. 대용량 데이터 저장

- 하나의 파일이 기가바이트에서 테라바이트 이상의 사이즈로 저장될 수 있게 설계
- 높은 데이터 전송 대역폭, 하나의 클러스터에서 수백 대의 노드를 지원
- 하나의 인스턴스에서 수백만 개 이상의 파일을 지원

## 4. 데이터 무결성

- 한번 저장한 데이터를 수정할 수 없음(읽기만 가능)
  - 파일 이동, 삭제, 복사할 수 있는 인터페이스 제공
-

# HDFS 기본

---

- 하나의 HDFS에 하나의 네임스페이스 제공
  - 파일은 블록들로 저장 됨
    - 보통 64MB ~ 128MB
    - 큰 파일을 다루는데 적합한 시스템
  - 하부 운영체제의 파일 시스템을 그대로 사용
  - 복제를 통한 안정성
    - 각 블록은 3대의 DataNode에 복제
  - **Write Once Read Many**
    - File Overwrite not Append
  - 마스터가 모든 메타데이터 관리
    - 간단한 중앙 집중 관리
  - **No 데이터 캐시**
    - 데이터 사이즈가 커서 캐시로 인한 혜택 없음
-



# HDFS(Hadoop Distributed File System)

---

- 네임노드
  - 파일시스템의 NameSpace 관리
    - 파일명과 블록들의 집합을 관리
    - 블록과 블록이 위치한 데이터 노드들의 정보 관리
    - 블록에 대한 복제 스케줄링
  - 메타데이터
    - Fimage : 네임스페이스/Inode 정보
      - 파일들의 리스트, 파일 별 속성 정보
      - 각 블록 별 DataNode 들의 리스트
    - Edit Log : 트랜잭션 로그
      - 파일 생성, 삭제 기록 들

# HDFS(Hadoop Distributed File System)

---

- Secondary NameNode
  - NameNode 의 fsimage와 트랜잭션 로그를 복사함
  - 복사한 fsimage와 트랜잭션 로그(Edit log) 병합
  - 새로운 fsimage를 NameNode에게 보냄
  - Secondary NameNode
    - NameNode의 단순 백업 데몬이 아님
    - Fsimage 관리를 공동 담당

# HDFS(Hadoop Distributed File System)

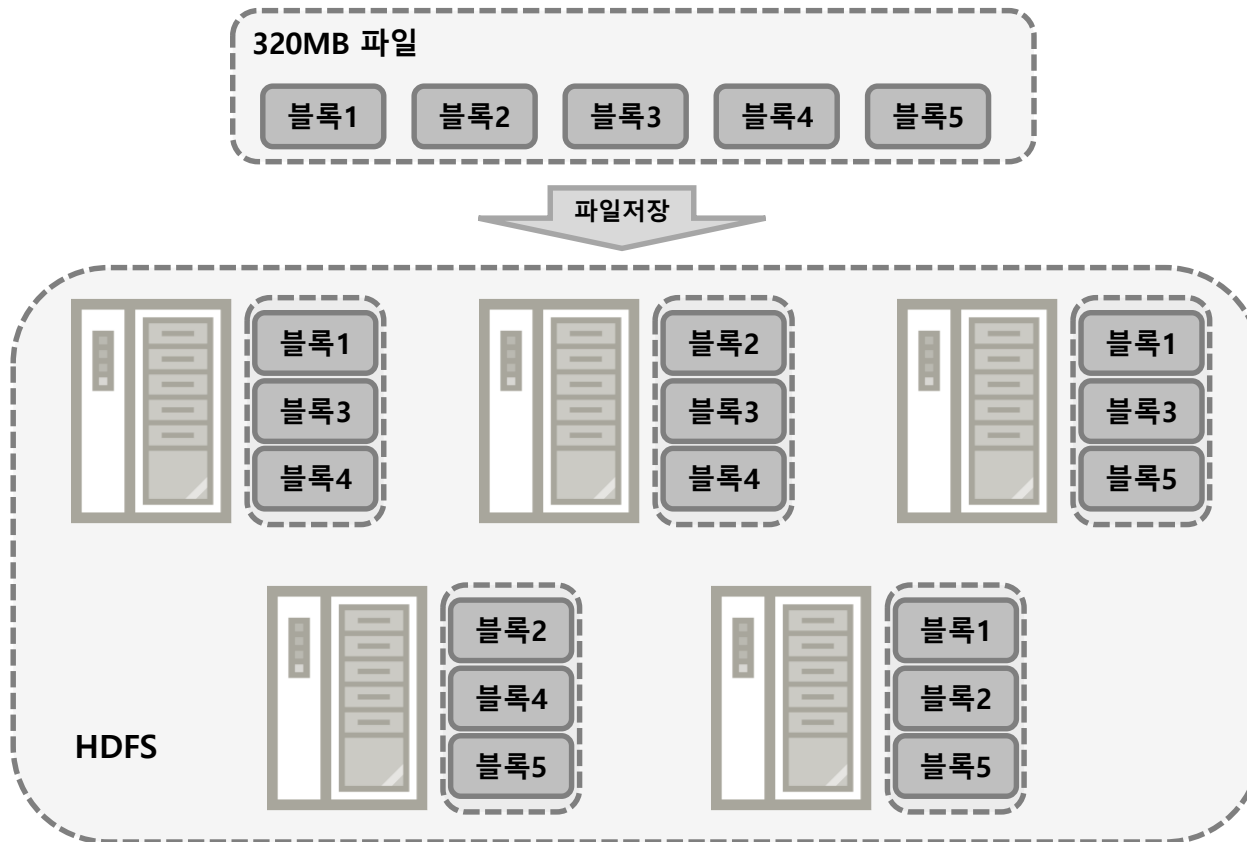
---

- DataNode
    - **블록 서버**
      - 블록을 메타데이터를 로컬에 저장(ext3)
      - 클라이언트에게 블록&메타데이터 제공
    - **블록 보고**
      - 주기적으로(Heart Beat) 존재하는 모든 블록들의 리스트를 NameNode에 보고
      - 기본 3초마다 (dfs.heartbeat.interval in hdfs-site.xml)
    - **Pipelining 데이터 저장**
      - 데이터를 다른 DataNode에 전달
    - **블록 배치 순서**
      - 첫 번째 복제본은 로컬 노드에 저장
      - 두 번째 복제본은 원격 Rack에 저장
      - 세 번째 복제본은 같은 Rack에 저장
    - **클라이언트 읽기**
      - 로컬 노드 -> 같은 Rack -> 원격 Rack 순서
-

# HDFS 아키텍처

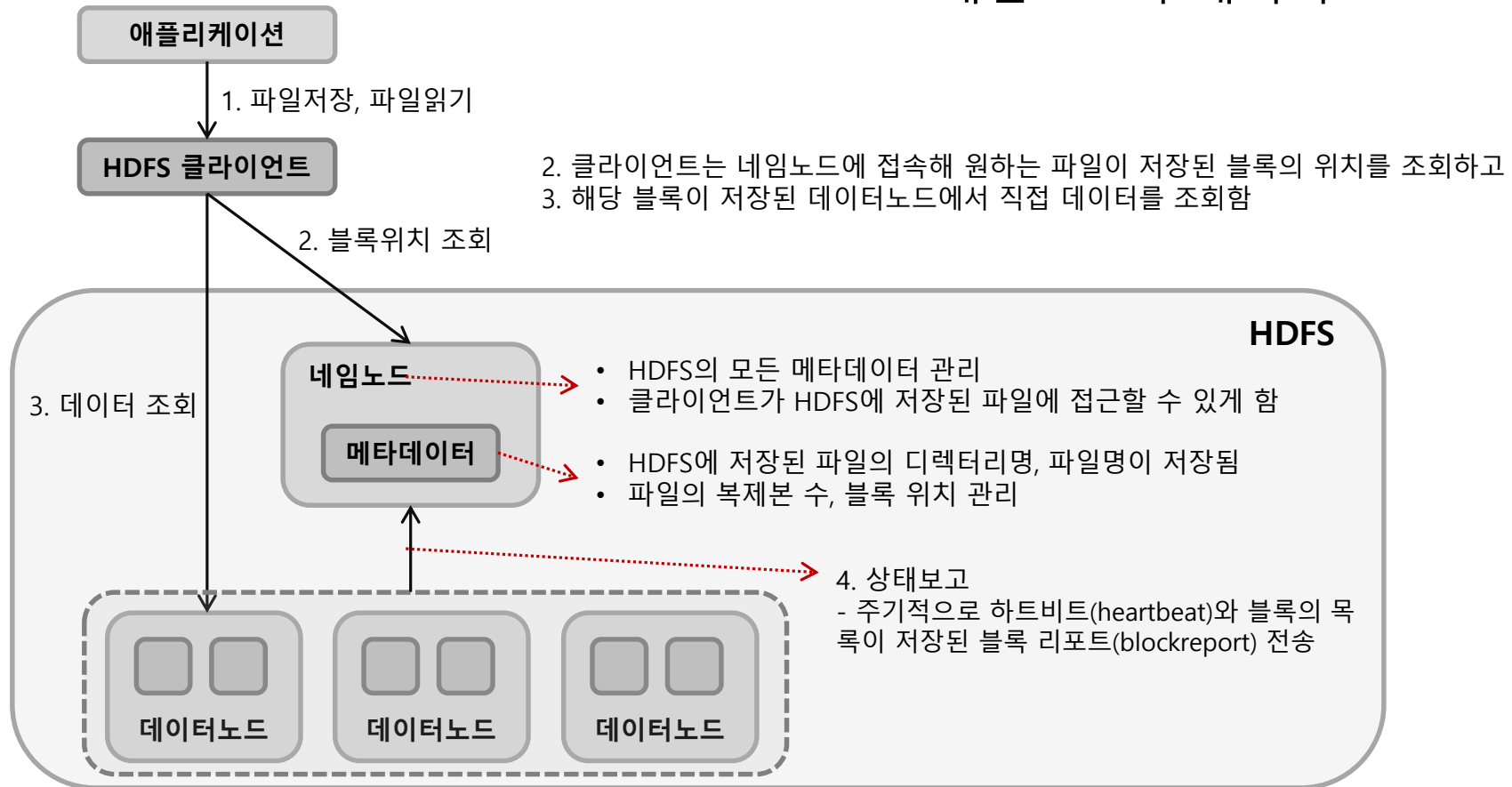
- 블록 구조 파일 시스템

- 블록 사이즈가 기본적으로 64MB로 설정돼 있음(설정파일에서 수정 가능)
- 파일 사이즈가 기본 블록 설정보다 작을 경우 파일사이즈로 저장
- 블록을 저장할 때 기본적으로 3개씩 블록의 복제본을 저장(수정 가능)



# HDFS 아키텍처

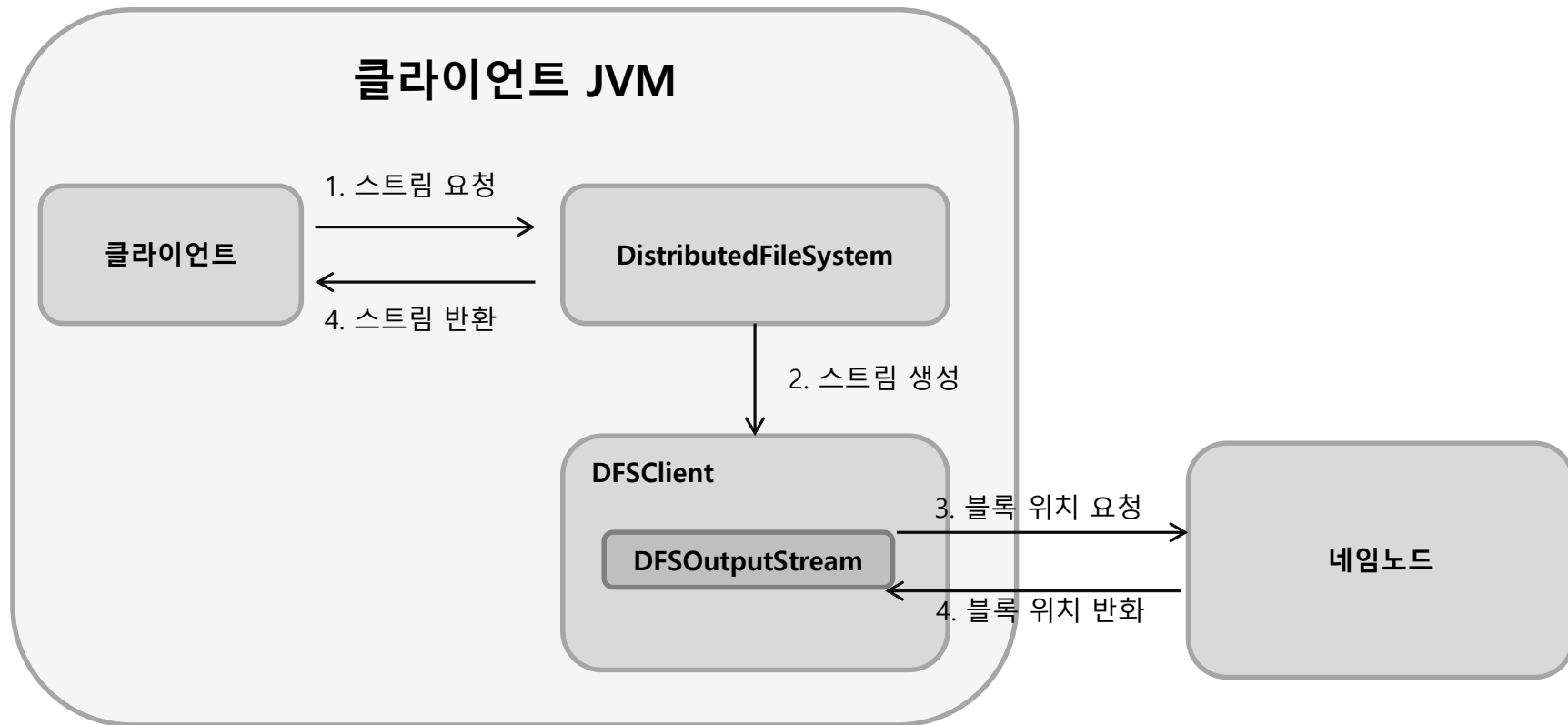
## • 네임노드와 데이터노드



4. 네임노드는 하트비트를 통해 데이터노드가 정상작동하는지 확인하고 블록 리포트를 통해 데이터노드의 모든 블록을 확인, 파일 복제본의 위치 결정

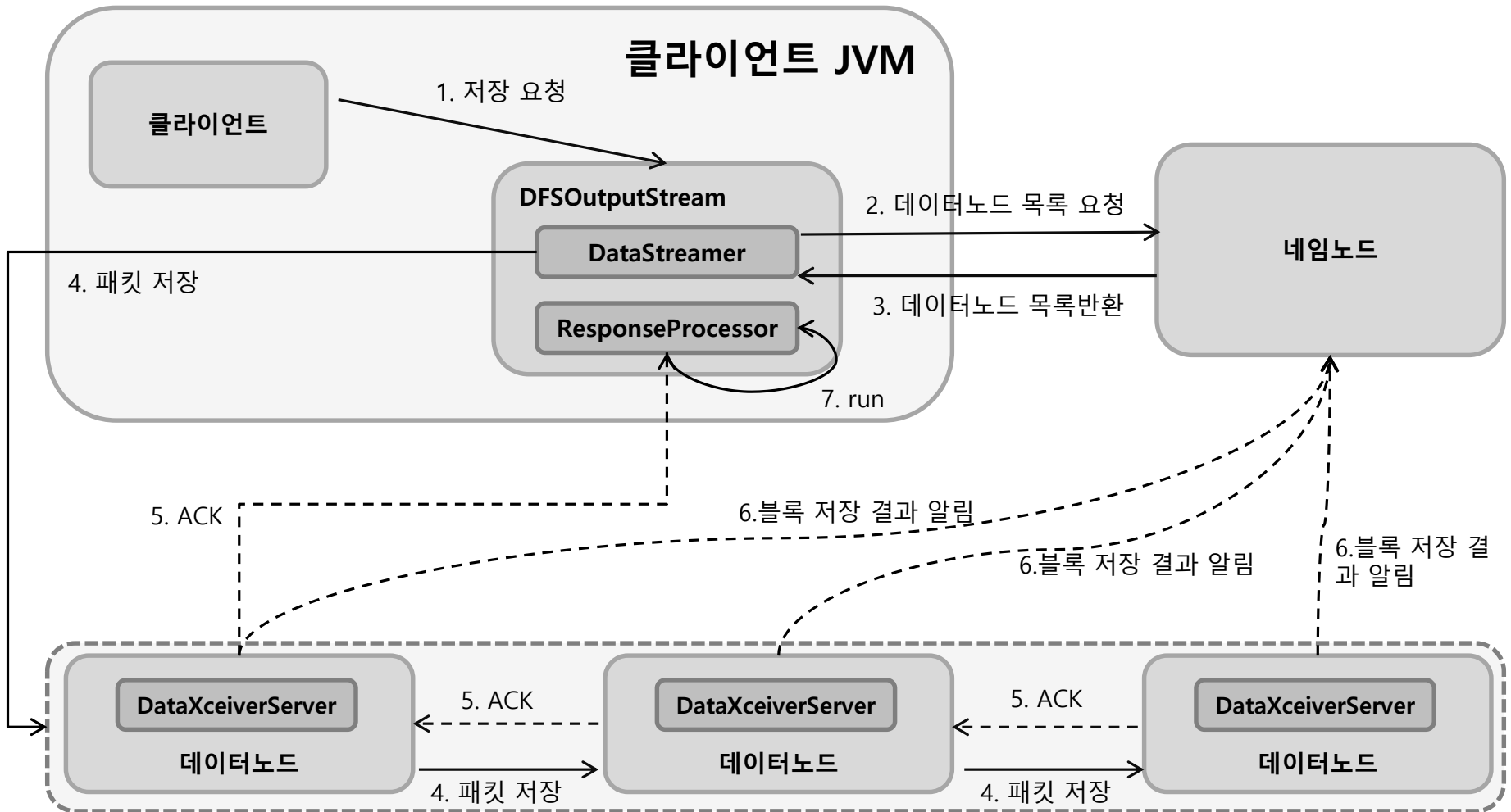
# HDFS의 파일 저장

- 파일 저장 요청



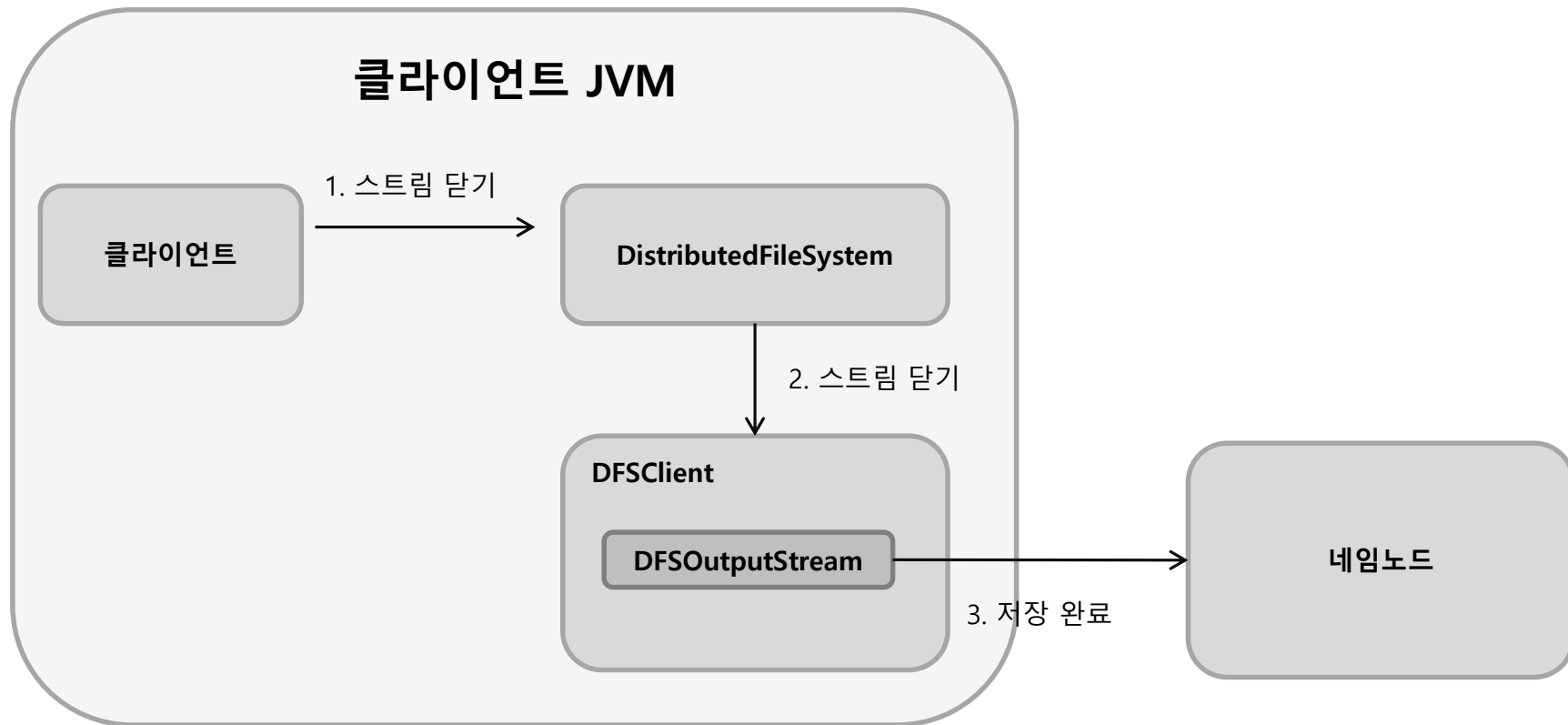
# HDFS의 파일 저장

- 패킷 전송



# HDFS의 파일 저장

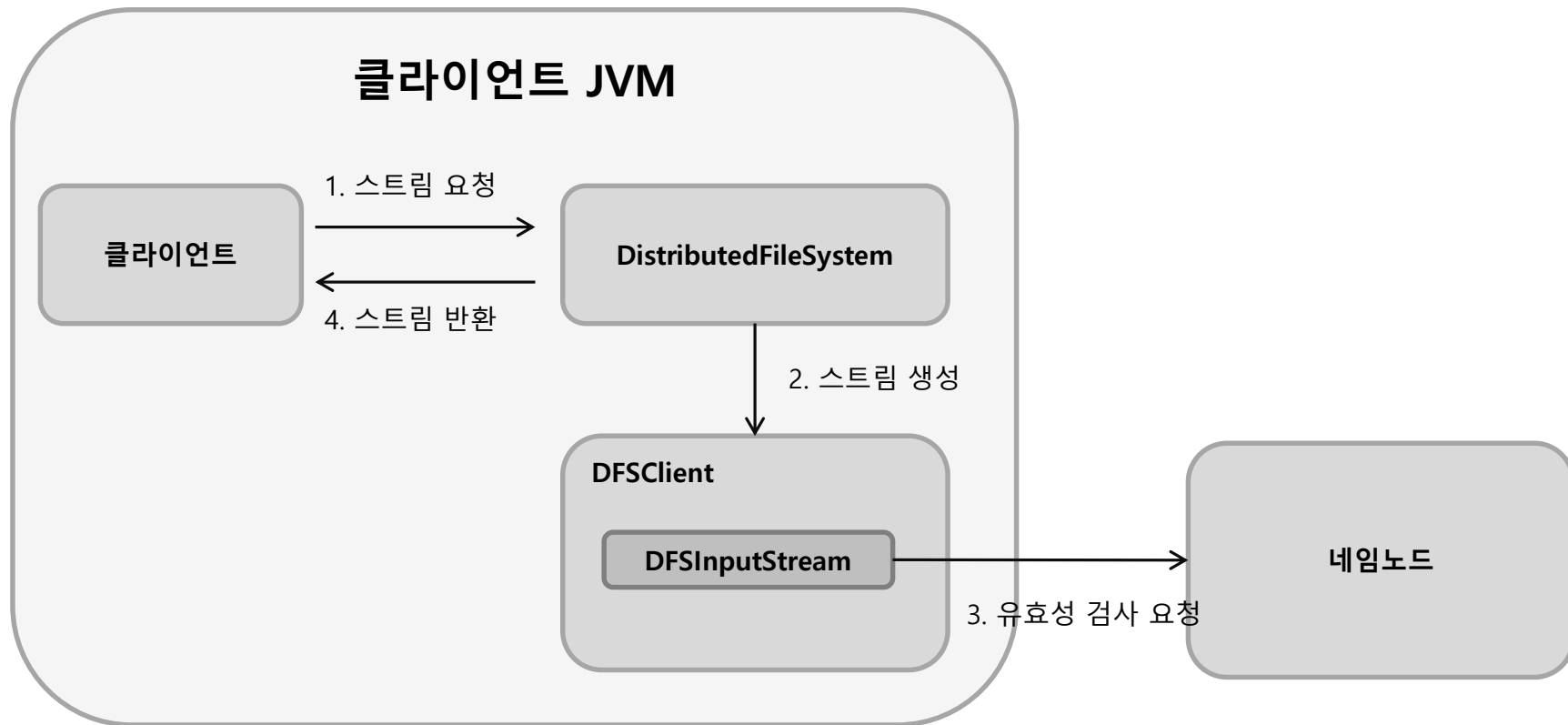
- 파일 닫기





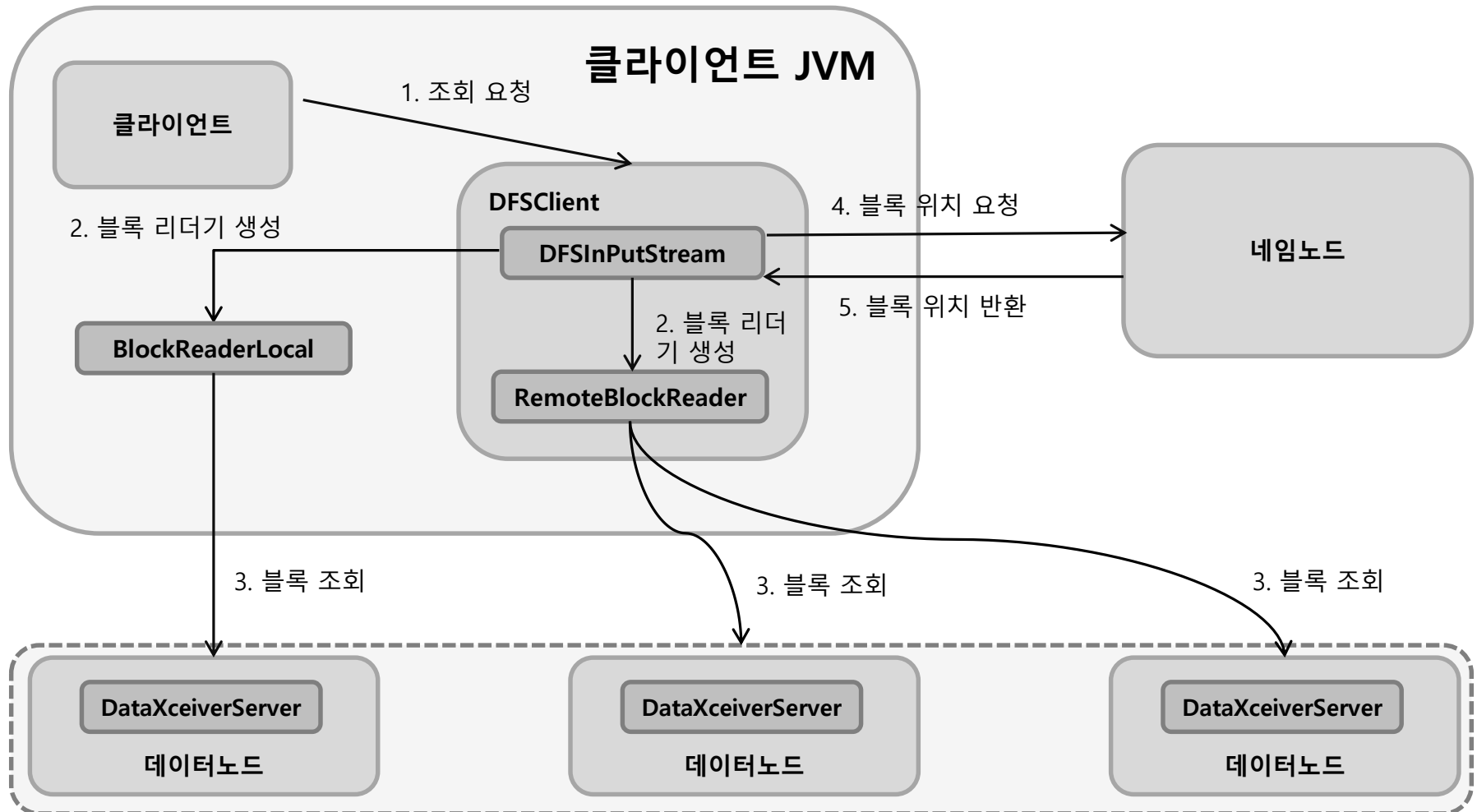
# HDFS의 파일 읽기

- 파일 조회 요청



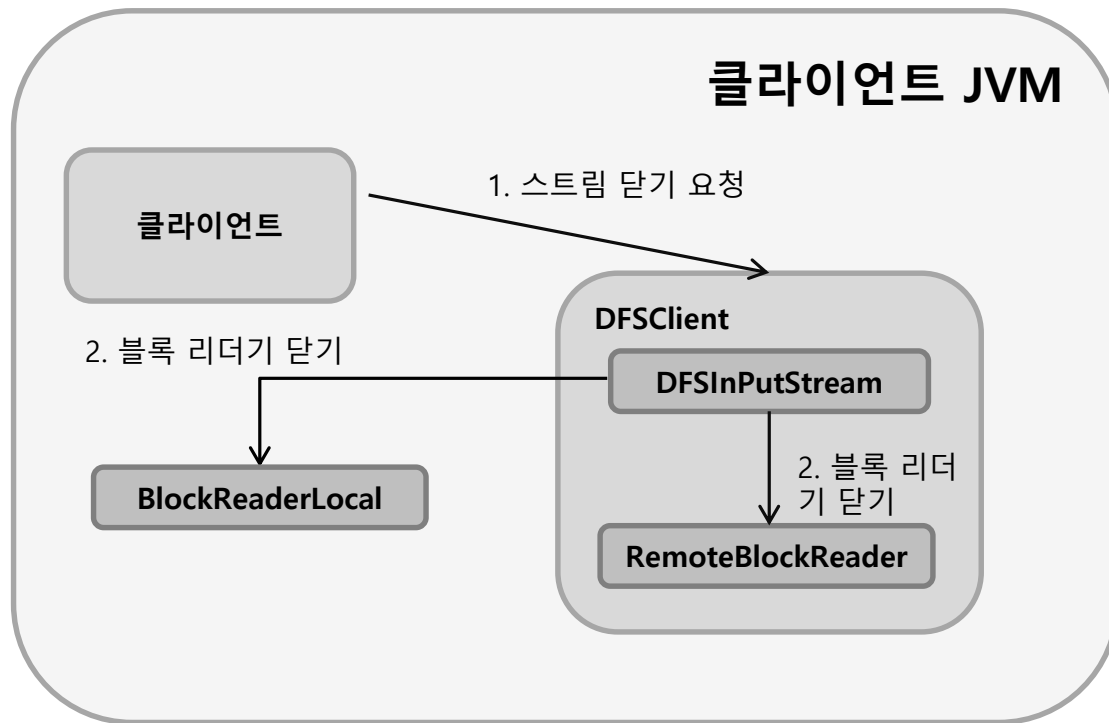
# HDFS의 파일 읽기

- 블록 조회



# HDFS의 파일 읽기

- 입력 스트림 닫기



# HDFS CLI

---

- **hdfs (d)fs -cmd <args>**
    - [appendToFile <localsrc> <dst>]**
    - [-ls <path>]**
    - [-du <path>]**
    - [-cp <src> <dst>]**
    - [-rm <path>]**
    - [-put <localsrc> <dst>]**
    - [-copyFromLocal <localsrc> <dst>]**
    - [-moveFromLocal <localsrc> <dst>]**
    - [-get [-crc] <src> <localdst>]**
    - [-cat <src>]**
    - [-copyToLocal [-crc] <src> <localdst>]**
    - [-moveToLocal [-crc] <src> <localdst>]**
    - [-mkdir <path>]**
    - [-touchz <path>]**
    - [-text [-ezd] <path>]**
    - [-stat [format] <path>]**
    - [-help [cmd]]**
-

# HDFS CLI

---

- **appendToFile**
  - `hdfs dfs -appendToFile <localsrc> ... <dst>`
  - HDFS에 존재하는 파일에 한 개 또는 다수의 파일을 추가함
  - STDIN 입력을 읽어서도 추가 가능

# HDFS CLI

---

- **cat**
  - `hdfs dfs -cat URI [URI ...]`
  - 파일 내용을 나타냄
  - 압축된 파일을 읽기 위해서는 `fs` 대신 `text` 명령어 사용

# HDFS CLI

---

- **chgrp**
  - `hdfs dfs -chgrp [-R] GROUP URI [URI ...]`
  - 파일과 디렉터리의 그룹을 변경함
  - -R 옵션은 변경을 재귀적으로 적용
  - 사용자는 해당 파일의 소유자이거나 슈퍼유저여야 함

# HDFS CLI

---

- **chmod**

- `hdfs dfs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI [URI ...]`
- 유닉스와 유사하게 권한 모드는 세자리 8진수 모드 또는 +/-{rwxX}
- -R 옵션은 변경을 재귀적으로 적용
- 사용자는 해당 파일의 소유자이거나 슈퍼유저여야 함

- **chwon**

- `hdfs dfs -chown [-R] [OWNER][:[GROUP]] URI [URI ]`
  - 파일과 디렉터리의 소유자를 변경
  - -R 옵션은 변경을 재귀적으로 적용
  - 사용자는 슈퍼유저여야 함
-



# HDFS CLI

---

- **copyFromLocal**

- `hdfs dfs -copyFromLocal <localsrc> URI`
- put과 동일함
- 로컬 파일 시스템으로부터 파일들을 복사

- **copyToLocal**

- `hdfs dfs -copyToLocal [-ignorecrc] [-crc] URI <localdst>`
  - get과 동일함
  - 파일들을 로컬 파일 시스템으로 복사
-

# HDFS CLI

---

- **count**
  - `hdfs dfs -count [-q] [-h] <paths>`
  - PATH에 있는 모든 파일과 디렉터리에 대한 이름, 사용된 바이트 수, 파일 개수
  - `-q` 옵션은 할당 정보를 나타냄

# HDFS CLI

---

- **cp**
  - `hdfs dfs -cp [-f] [-p | -p[topax]] URI [URI ...] <dest>`
  - 소스에 있는 파일들을 목적지로 복사함
  - 만약 다수의 소수 파일들이 지정되면, 목적지는 디렉터리여야 함

# HDFS CLI

---

- **du**

- `hdfs dfs -du [-s] [-h] URI [URI ...]`
- 파일 크기를 나타냄
- 만약 PATH가 디렉터리이면, 그 디렉터리에 있는 각 파일의 크기가 리포트 됨
- 파일명 앞에 전체 URI 프로토콜이 붙음
- 비록 du는 디스크 사용량을 나타내지만 있는 그대로 받아들여서는 안됨
- 디스크 사용량은 블록 크기와 복제 요소들에 따라 다르기 때문

- **dus**

- `hdfs dfs -dus <args>`
  - du와 비슷함
  - 그런데 디렉터리에 대해서 dus는 개별적으로 나타내기보다는 합계 (파일 크기)를 리포트 함
-

# HDFS CLI

---

- **expunge**
    - `hdfs dfs -expunge`
    - 휴지통을 비움
    - 만약 휴지통 기능이 활성화 되어 있으면, 파일이 삭제되었을 때 우선 임시 `.Trash/` 폴더로 이동하게 됨
    - 사용자가 설정한 시간 후에 `.Trash/` 폴더에 있는 파일들을 강제로 삭제
    - `.Trash/` 폴더에 파일이 존재하는 한 그것을 원래 위치로 이동시켜 해당 파일을 복구 할 수 있음
-

# HDFS CLI

---

- **get**
  - `hdfs dfs -get [-ignorecrc] [-crc] <src> <localdst>`
  - 파일들을 로컬 파일 시스템으로 복사할 때, 만약 다수의 소스 파일들이 지정되면 로컬 목적지는 디렉터리여야 함
  - 만약 LOCALDST가 -이면, 그 파일들은 `stdout` 으로 복사됨
  - HDFS는 파일에 대한 각 블록의 체크섬을 계산함
  - 파일에 대한 체크섬은 숨김 파일에 저장되고, 해당 파일이 HDFS에 읽힐 때 숨김 파일에 있는 체크섬들은 해당 파일의 무결성을 확인하는데 사용
  - Get 명령어에서 `-crc` 옵션을 사용하면, 숨김 파일도 복사하고, `-ignorecrc` 옵션은 복사할 때 체크섬을 확인하는 과정을 건너뛴다

# HDFS CLI

---

- **getmerge**
  - `hdfs dfs -getmerge <src> <localdst> [addnl]`
  - SRC에서 확인된 모든 파일을 가져와 합치고, 로컬 파일 시스템에 존재하는 하나의 LOCALDST 파일에 기록함
  - ddnl 옵션은 각 파일의 끝을 나타내는 개행 문자를 LOCALDST 파일에 추가함

# HDFS CLI

---

- **ls**

- `hdfs dfs -ls [-R] <args>`
- 파일과 디렉터리를 조회함
- 각 엔트리는 명칭, 권한, 소유자, 그룹, 크기, 수정 일을 나타냄
- 파일 엔트리는 복제 요소도 포함해서 보여줌

- **lsr**

- `hdfs dfs -lsr <args>`
  - ls의 재귀적 버전
-



# HDFS CLI

---

- **mv**

- `hdfs dfs -mv URI [URI ...] <dest>`
- SRC에 있는 파일들을 DST로 옮김
- 만약 다수의 소스파일들이 지정되면, 목적지는 디렉터리여야 함
- 파일 시스템들 간 이동은 허가되지 않음

- **moveToLocal**

- `hdfs dfs -moveToLocal [-crc] <src> <dst>`
- 아직 구현되지 않음

# HDFS CLI

---

- **put**
  - `hdfs dfs -put <localsrc> ... <dst>`
  - 로컬 시스템으로부터 파일과 디렉터리를 목적지 파일 시스템으로 복사
  - 만약 LOCALSRC가 - 로 설정되어 있으면 입력은 STDIN으로 지정되고, DST는 파일이어야 함

# HDFS CLI

---

- **rm**
  - `hdfs dfs -rm [-f] [-r|-R] [-skipTrash] URI [URI ...]`
  - 파일과 빈 디렉터리를 삭제함
- **rmr**
  - `hdfs dfs -rmr [-skipTrash] URI [URI ...]`
  - `rm` 의 재귀적 버전

# HDFS CLI

---

- **setrep**

- `hdfs dfs -setrep [-R] [-w] <numReplicas> <path>`
- 주어진 파일들에 대한 대상 복제 갯수를 REP로 설정함
- -R 옵션은 PATH에 의해 확인된 디렉터리들에 대한 파일들의 대상 복제 갯수를 재귀적으로 적용함

- **stat**

- `hdfs dfs -stat [FORMAT] URI [URI ...]`
  - 파일의 통계 정보를 보여줌
  - **FORMAT** : %d (파일크기), %F(파일형식), %n(파일이름), %r(복제), %o(블록크기)
-

# HDFS CLI

---

- **tail**
  - `hdfs dfs -tail [-f] URI`
  - 파일의 마지막 1K byte의 내용을 출력
- **test**
  - `hdfs dfs -test [-ezd] URI`
  - PATH에 다음의 형식 점검을 수행함
  - `-e` PATH 존재 유무, PATH가 존재하면 0을 반환함
  - `-z` 빈 파일 여부, 파일 길이가 0이면 0을 반환함
  - `-d` PATH가 디렉터리이면 0을 반환함

# HDFS CLI

---

- **text**

- `hdfs dfs -text <src>`
- 파일의 텍스트 내용을 나타냄
- 만약 파일이 텍스트 파일이라면 `cat` 명령과 동일함
- 압축형식(gzip과 하둡의 바이너리 시퀀스 파일 포맷)으로 알려진 파일들은 우선 압축을 해제 함

- **touchz**

- `hdfs dfs -touchz URI [URI ...]`
  - 길이가 0인 파일을 생성함
  - 만약 길이가 0이 아닌 파일이 이미 존재하면 에러가 발생함
-

# HDFS CLI

---

- **Balancer 실행**

- `$hadoop balancer`
- 데이터가 분산이 잘 되는지 확인

- **distcp 실행**

- 원격 DFS의 `/user/hadoop/distcp`의 모든 내용을 자신의 DFS로 복사
- `$hadoop distcp -p -update <src> <dst>`

---

# Hadoop

---

2015

---



# Content

---

- hadoop 운영

# 하둡운영

---

- 파일 시스템 상태 확인
  - \$hadoop fsck

상태	내용
Over-replicated blocks	복제본이 과도하게 생성된 경우를 의미 (3 개의 복제본이 저장돼야하는데 , 3 개를 초과해서 복제본이 저장된 경우)
Under-replicated blocks	부족하게 복제된 블록 개수를 의미 (3 개의 복제본이 저장돼야 하는데 , 2개 혹은 1 개의 복제본만 저장돼 있다면 부족하게 저장된 복제본으로 인식합니다 주로 네임노드가 다운되 기 직 전이나 직후에 데이터를 저장할 경우 발생)
Mis-replicated blocks	복제된 블록이 유실된 상태
Corrupt blocks	블록에 오류가 발생한 경우

- \$hadoop balancer -threshold [threshold]
  - \$sbin/hadoop-daemon.sh start balancer
-

# 하둡운영

---

- HDFS 어드민 명령어
  - \$hadoop dfsadmin [option]
    - -report
      - Hdfs의 기본적인 정보와 상태 출력
    - -safemode [enter|leave]
      - 파일쓰기 변경 작업 금지 읽기만 허용
    - -saveNamespace
      - 로컬 파일 시스템에 저장돼 있는 파일 시스템 이미지 파일과 에디트 로그를 현재 버전으로 갱신
      - 안전모드 상태에서 실행 가능

# 하둡 운영

---

- 파일 관리

- \$hadoop dfsadmin [option]

- -setQuota

- 파일과 하위 디렉토리 개수 설정

- » \$hadoop fs -mkdir quota\_test

- » \$hadoop dfsadmin -setQuota 2 quota\_test

- » \$hadoop fs -put ../data/2008.csv quota\_test/2008 . Csv

- » \$hadoop fs -put ../data/2008.csv quota\_test/2008 . Csv

- -clrQuota

- 쿼터해제 또는 재설정

- » \$hadoop dfsadmin -clrQuota quota\_test

- -setSpaceQuota

- 용량 제한

- » \$hadoop dfsadmin -setSpaceQuota 1500m quota\_test

- -clrSpaceQuota

- 용량 제한 해제

- » \$hadoop dfsadmin -clrSpaceQuota quota\_test

---

# 하둡 운영

---

- 데이터노드제거
  - hdfs-site.xml
    - <property>
      - <name>dfs.hosts.exclude</name>
      - <value>/home/hadoop/hadoop/conf/exclude\_server</value>
    - </property>
  - exclude\_server 파일에 제거할 데이터노드의 호스트명 작성
    - ex> hadoop02
  - \$hadoop dfsadmin -refreshNodes
    - hdfs-site.xml 의 dsf-hosts.exclude 와 dfs.hosts 를 리프레쉬
  - 데이터노드를 제거했을 때 남게 되는 데이터노드의 수가 dfs.replication(HDFS 데이터 복제본 수) 옵션으로 설정한 값보다 크거나 같아야함

# 하둡운영

---

- 데이터노드추가
  - 네임노드의 slaves 파일에 데이터노드를 추가한다.
  - 데이터노드의 환경 설정 파일(hdfs-site.xml , core-site.xml, mapred-site.xml)에 네임노드 접속주소를 설정한다.
  - 데이터노드용 서버에서 데이터노드와 태스크트래커 를 구동한다.
  - hdfs-site.xml
    - <property>  
    <name>dfs.hosts</name>  
    <value>/home/hadoop/hadoop/conf/include\_server</value>  
  </property>
    - include\_server 파일에 추가할 데이터노드의 호스트명 작성
      - ex>  
    hadoop01  
    hadoop02  
    hadoop03
    - \$hadoop dfsadmin -refreshNodes
      - hdfs-site.xml 의 dfs.hosts.exclude 와 dfs.hosts 를 리프레쉬
    - \$hadoop-daemon.sh start balancer

# 하둡운영

- 네임노드 장애 복구
  - 네임노드의 디렉토리 구조

디렉토리	내용
<code>\${dfs.name.dir}/current/VERSION</code>	파일 시스템 레이아웃
<code>\${dfs.name.dir}/current/fsimage</code>	체크포인팅한 시점의 파일 시스템 레이아웃
<code>\${dfs.name.dir}/current/fstime</code>	체크포인팅을 실행한 시간
<code>\${dfs.name.dir}/current/edits</code>	체크포인팅 이후의 HDFS 트랜잭션 로그
<code>\${dfs.name.dir}/image/fsimage</code>	체크포인팅을 시작하기 직전의 파일 시스템 이미지
<code>\${dfs.name.dir}/previous.checkpoint/</code>	마지막으로 체크포인팅

- 보조 네임노드의 디렉토리 구조

디렉토리	내용
<code>\${fs.checkpoint.dir}/current/VERSION</code>	파일 시스템 레이아웃
<code>\${fs.checkpoint.dir}/current/fsimage</code>	체크포인팅한 시점의 파일 시스템 레이아웃
<code>\${fs.checkpoint.dir}/current/fstime</code>	체크포인팅을 실행한 시간
<code>\${fs.checkpoint.dir}/current/edits</code>	체크포인팅 이후의 HDFS 트랜잭션 로그
<code>\${fs.checkpoint.dir}/image/fsimage</code>	체크포인팅을 시작하기 직전의 파일 시스템 이미지

# 하둡운영

---

- 네임노드 장애 복구

- 보조 네임노드를 이용한 장애 복구

```
[hadoop@hadoop01 hadoop]$ ./bin/stop-all .sh
```

```
[hadoop@hadoop01 hadoop]$ rm -rf /data/dfs/name
```

```
[hadoop@hadoop01 hadoop]$ hadoop namenode
```

```
[hadoop@hadoop01 hadoop]$ mkdir /data/dfs/name
```

```
[hadoop@hadoop01 hadoop]$ hadoop namenode -importCheckpoint
```

```
[hadoop@hadoop01 hadoop]$ $HADOOP_HOME/sbin/start-all.sh
```

```
[hadoop@hadoop01 hadoop]$ ls -l /data/dfs/name
```

- 보조 파일시스템을 사용하여 복구

```
<property>
```

```
<name>dfs.name.dir</name>
```

```
<value>로컬 파일 시스템 디렉터리 , NFS 디렉터리2.1</value>
```

```
</ property>
```

---



# 하둡운영

---

- 데이터노드 장애 복구
  - 해당 데이터노드를 중지
  - 손상된 디스크를 언마운트하고 새로운 디스크를 마운트
  - 새로운 디스크에 dfs.data.dir 속성으로 설정한 디렉터리를 생성
  - 데이터 노드를 재실행
  - 네임 노드서버에서 밸런서를 실행
  - HDFS 웹 UI에서 해당 데이터 노드가 조회되는지 확인

---

# Hadoop 2.x

---

2015

---

# Content

---

- Hdfs 페더레이션 ( Federation )
- YARN
- Namenode HA

# HDFS 페더레이션

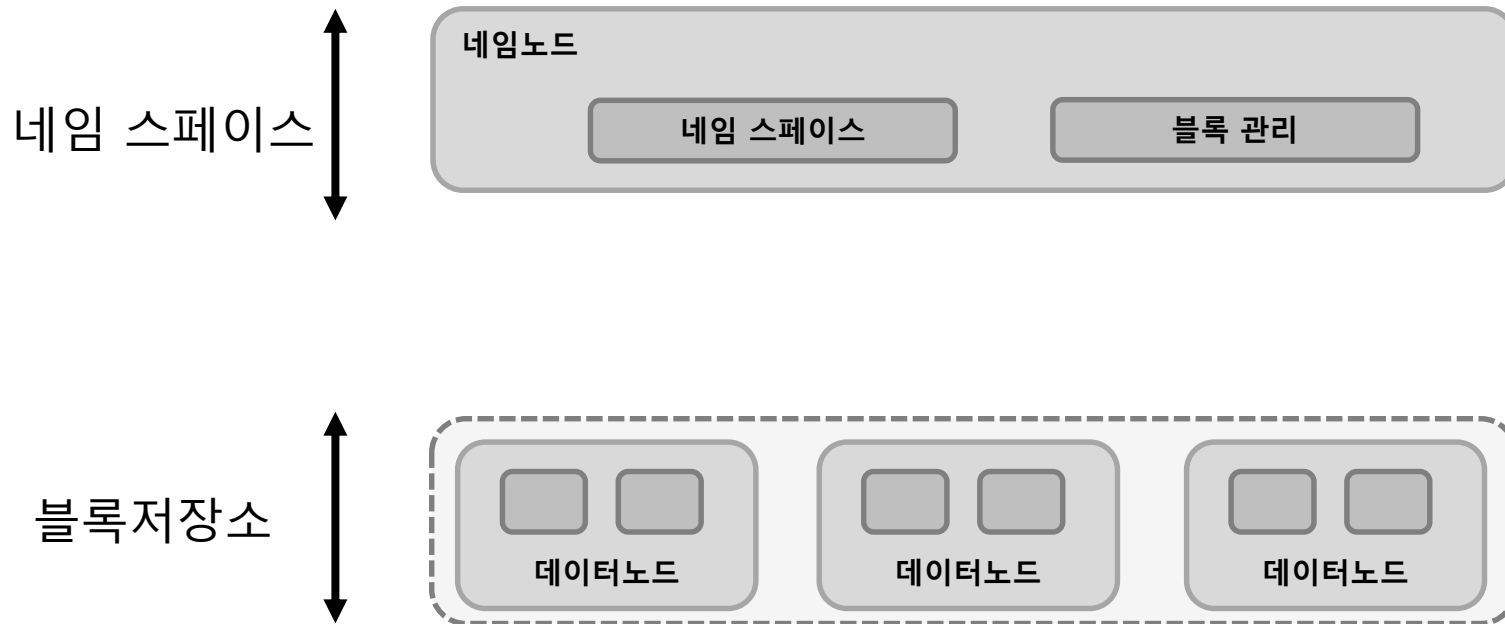
---

- HDFS 페더레이션
  - HDFS가 네임노드에 기능이 집중되는 것을 막기 위함

# HDFS 페더레이션

---

- 기존 HDFS 문제점



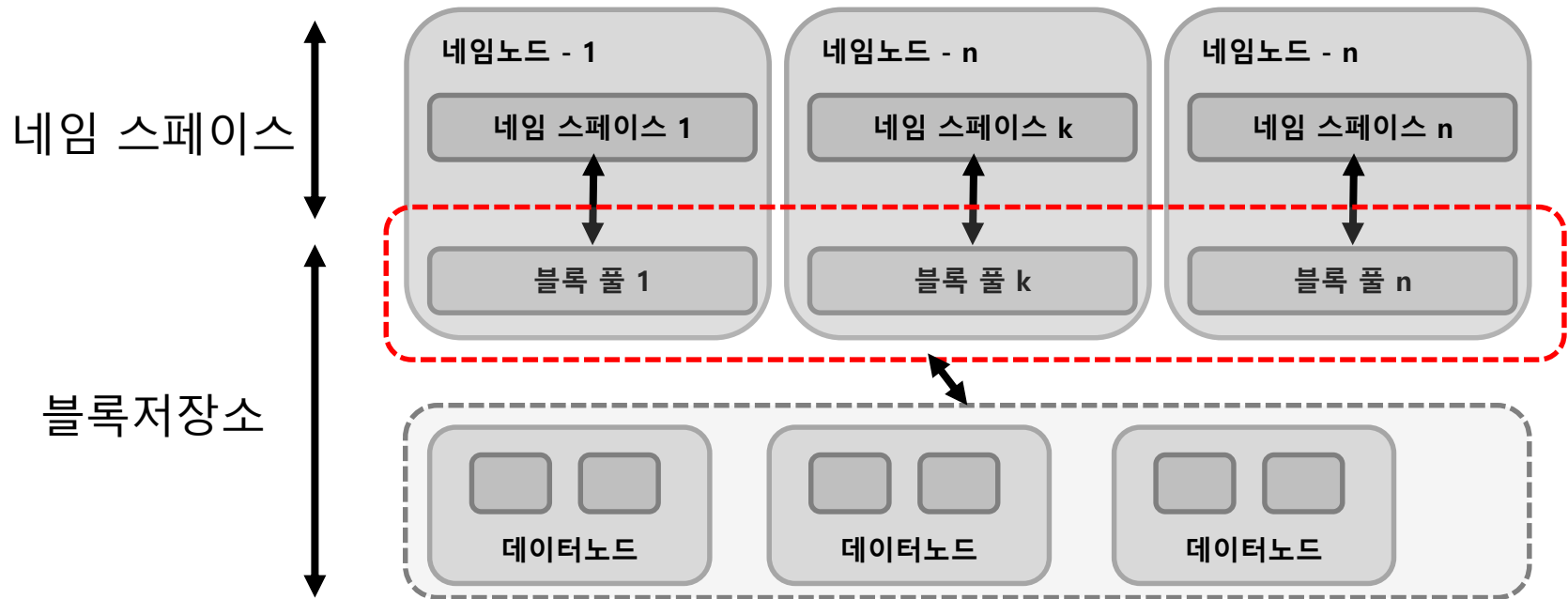
# HDFS 페더레이션

---

- 기존 HDFS 문제점
  - 네임 노드 확장
    - 수평정 확장 불가능
    - 네임노드 구동시 전체 파일 시스템 이미지를 메모리에 생성
      - 64GB ram – 약 2억5천 만개 파일과 블록 저장 가능
  - 성능
    - HDFS에 대한 모든 요청이 네임노드에 요청됨
    - 쓰기시 트랜잭션에 대한 에디트 로그 생성
  - 동일 네임노드 사용
  - 네임스페이스와 블록 관리의 지나친 결합

# HDFS 페더레이션

- HDFS 페더레이션 아키텍처



# HDFS 페더레이션

---

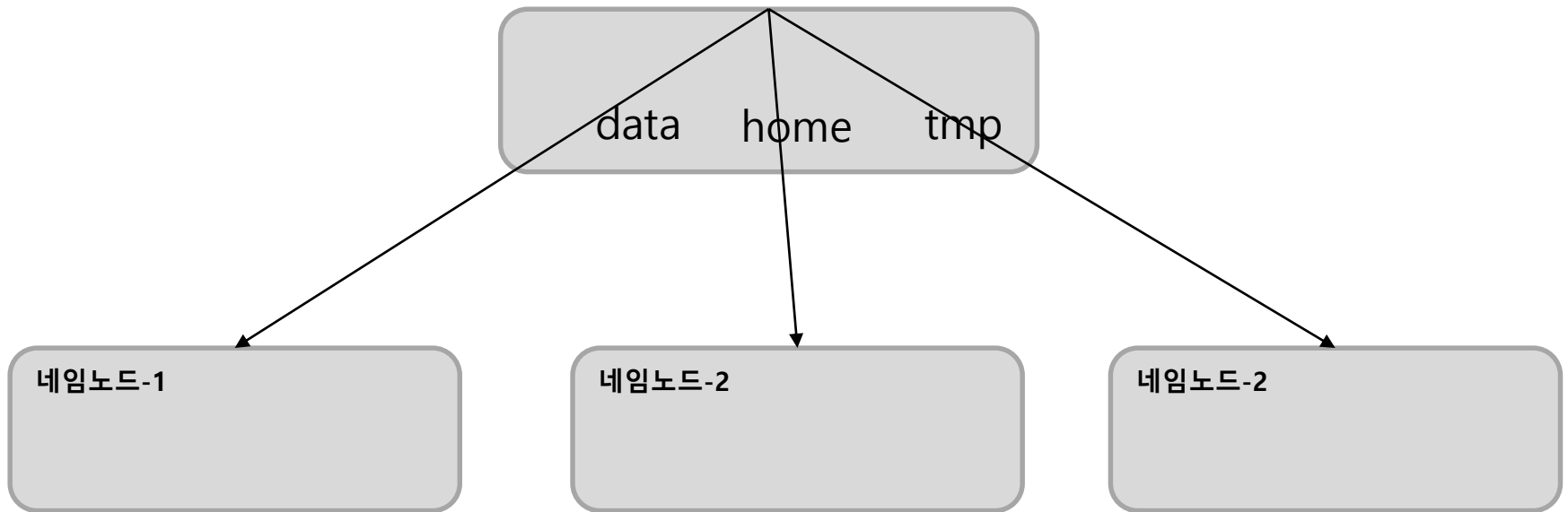
- HDFS 페더레이션의 장점
  - 네임 노드 분리
  - 기존 구성 변경 불필요
  - 네임노드 등록 삭제시 재시작 불필요
  - 밸런서 디터미션 네임노드 개별적 수행 가능



# HDFS 페더레이션

---

- 네임스페이스 관리
  - 마운트 테이블



# HDFS 페더레이션

---

- 기존

```
<property>  
<name>fs.default.name</name>  
<value>hdfs://namenodeOfClusterX:port</value>  
</property>
```

- Mount table

```
<property>  
    <name>fs.default.name</name>  
    <value>viewfs://clusterX</value>  
</property>  
<property>  
    <name>fs.viewfs.mounttable.default.link./NN1Home</name>  
    <value>hdfs://namenode1:9001/home</value>  
</property>  
<property>  
    <name>fs.viewfs.mounttable.default.link./NN2Home</name>  
    <value>hdfs://namenode2:9001/home</value>  
</property>
```

---

# YARN

---

- YARN 등장 배경
  - 잡 트래커가 모든 잡의 스케줄링과 자원 관리
  - 잡 트래커 구조 확장의 필요성
  - 메모리 소비 개선
  - 스레드 모델 개선
  - 등

# YARN

---

## ***Single Use System***

*Batch Apps*

### **HADOOP 1.0**

#### **MapReduce**

(cluster resource management  
& data processing)

#### **HDFS**

(redundant, reliable storage)



## ***Multi Purpose Platform***

*Batch, Interactive, Online, Streaming, ...*

### **HADOOP 2.0**

#### **MapReduce**

(data processing)

#### **Others**

#### **YARN**

(cluster resource management)

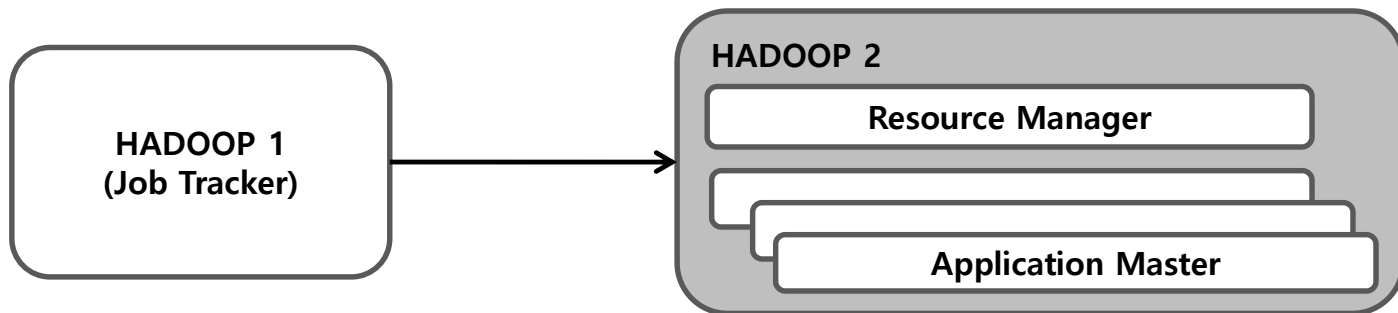
#### **HDFS2**

(redundant, highly-available & reliable storage)

# YARN

---

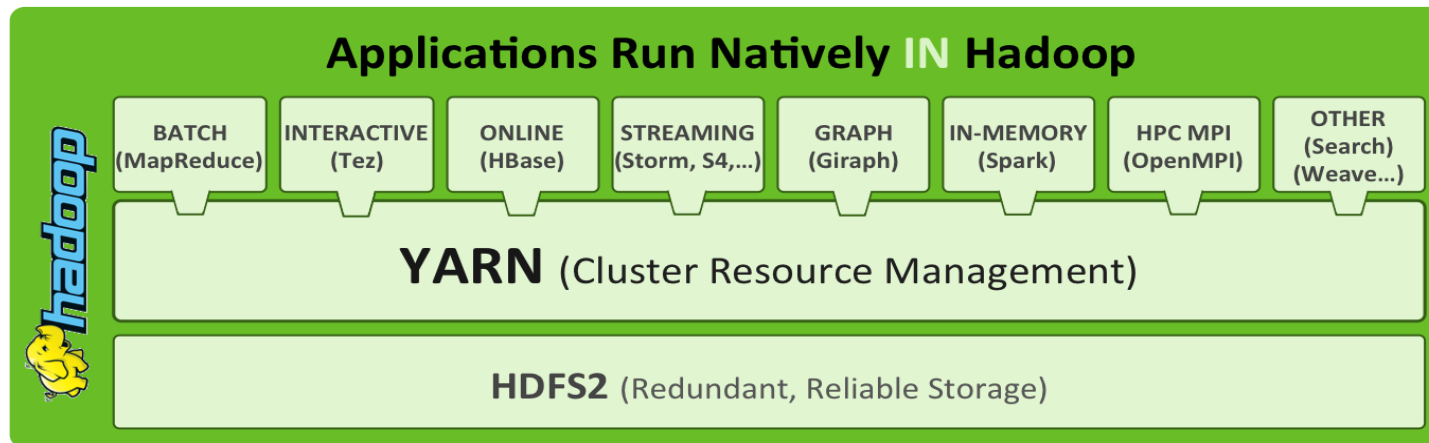
- Hadoop 1.0의 단점
  - 한노드에서 실행할 수 있는 Map과 Reduce용 작업숫자가 제한되어, 노드에 여유 자원이 있어도 그 자원을 활용하지 못하는 상황이 발생
    - 클러스터의 규모와는 상관없이 Job Tracker의 개수는 1개 (병목 지점)
  - 자원 분배 및 작업 관리의 비효율성
- YARN (Yet Another Resource Negotiator)
  - 자원 관리, Job 상태 관리를 ResourceManager와 ApplicationMaster로 분리하여, 기존 Job Tracker에 몰리던 병목을 제거
  - MapReduce 외에 다양한 어플리케이션을 실행할 수 있으며, 어플리케이션마다 자원(CPU, 메모리)을 할당 받음
    - 클러스터에는 여러개의 Application이 동작 가능
    - 각 Application은 Application Master가 모든 task를 관리



# YARN

---

- Application 호환성
  - HADOOP 1
    - MapReduce외에 다른 Application은 클러스터의 자원을 공유할 수 없는 문제
  - HADOOP 2 YARN
    - 같은 클러스터 내에서 M/R와 다른 application을 실행할 수 있도록 지원



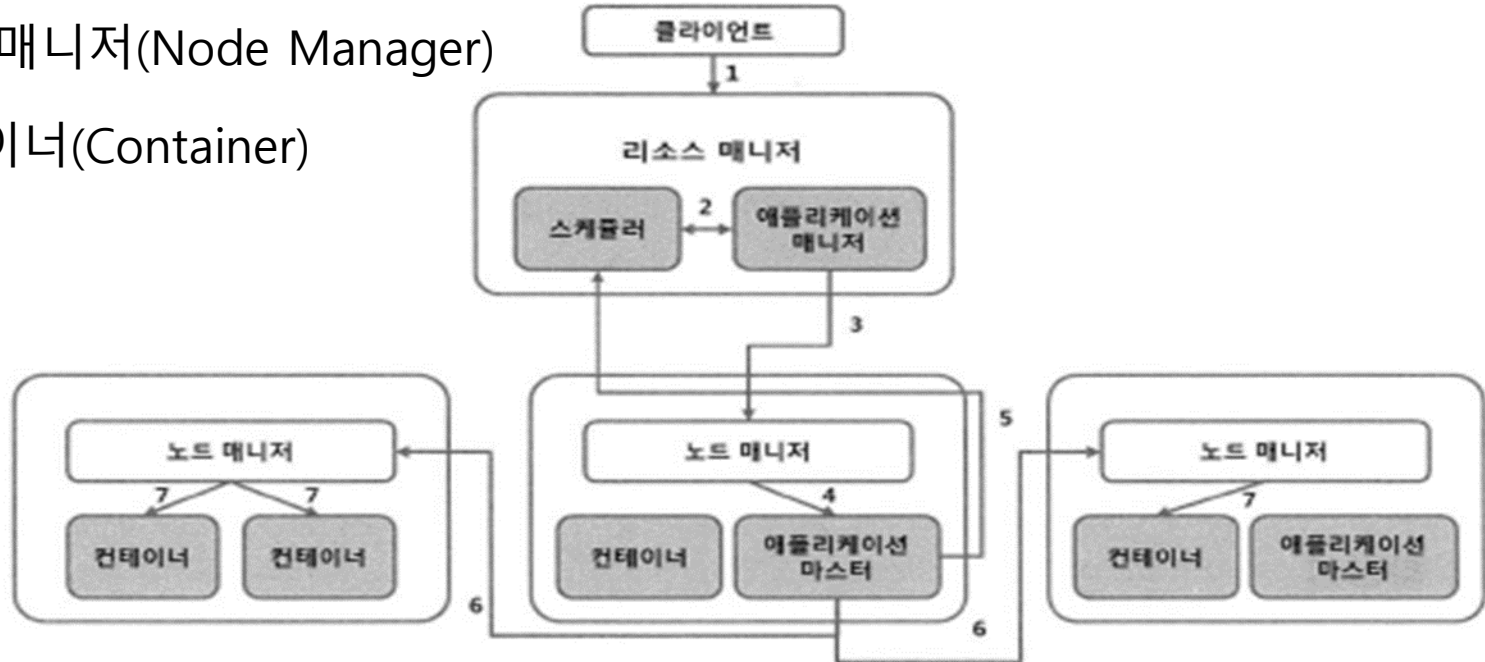
# YARN

---

- 자원 할당의 유연성
    - HADOOP 1
      - Slot에 미리 자원(memory, cpu cores)를 할당 후 미리 정해진 설정에 따라서 slot을 job에 할당
      - Job이 모두 끝나기 전까지는 자원이 반납되지 않는 문제
    - HADOOP 2 YARN
      - 요청이 있을 때마다 요구하는 자원의 spec에 맞게 자원을 container의 개념으로 할당
      - container마다 다른 spec의 자원을 갖을 수 있음
      - 모든 task는 container에서 수행되고 task가 끝나는 즉시 자원을 반납
-

# YARN

- YARN (Yet Another Resource Negotiator)
  - 리소스 매니저(Resource Manager)
    - 어플리케이션 매니저(Applications Manager)
    - 스케줄러(Scheduler)
  - 어플리케이션 마스터(Application Master)
  - 노드 매니저(Node Manager)
  - 컨테이너(Container)





# YARN

---

- YARN (Yet Another Resource Negotiator)
  - 리소스 매니저(Resource Manager)
    - master node에서 동작
    - global resource scheduler
    - application들의 자원 요구의 할당 및 관리

# YARN

---

- YARN (Yet Another Resource Negotiator)
  - 노드 매니저(Node Manager)
    - slave node에서 동작
    - node의 자원을 관리
    - container에 node의 자원을 할당

# YARN

---

- YARN (Yet Another Resource Negotiator)
    - 컨테이너(Container)
      - RM(Resource Manager)의 요청에 의해 NM(Node Manager)에서 할당
      - slave node의 CPU core, memory의 자원을 할당
      - applications은 다수의 container로 동작
-

# YARN

---

- YARN (Yet Another Resource Negotiator)
  - 어플리케이션 마스터(Application Master)
    - application당 한 개씩 존재
    - application의 spec을 정의
    - container에서 동작
    - application task를 위해 container 할당을 RM에게 요청

# YARN

---

- YARN (Yet Another Resource Negotiator)
  - 어플리케이션 마스터(Application Master)
    - application당 한 개씩 존재
    - application의 spec을 정의
    - container에서 동작
    - application task를 위해 container 할당을 RM에게 요청

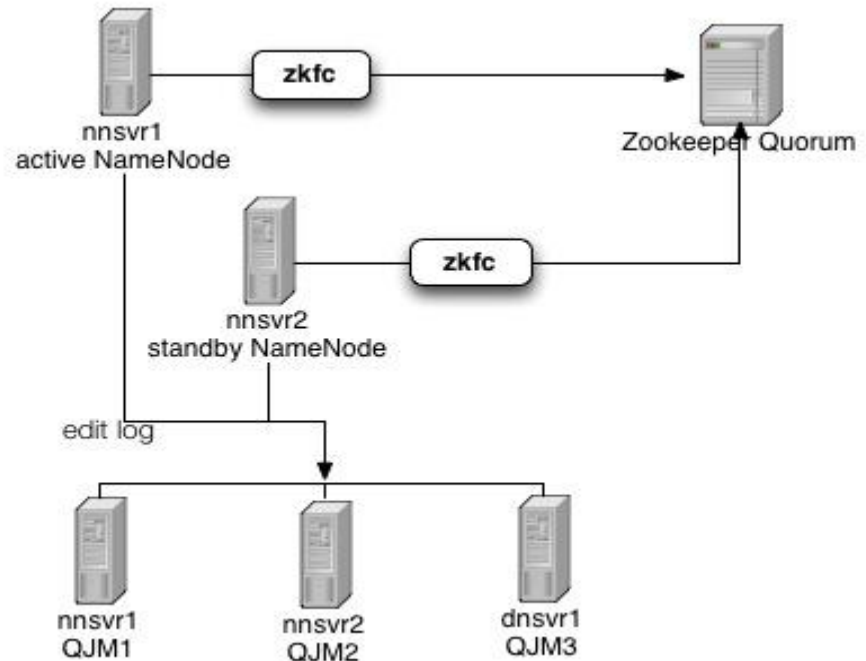
# YARN

---

- YARN 장점
    - 확장성
    - 가용성
      - Zookeeper 이용한 HA
    - 호환성
    - 리소스 이용 확대
      - YARN은 CPU, 메모리, 디스크, 네트워크와 같은 리소스를 기준으로 애플리케이션 수행을 예약하고 , 실행함
    - 알고리즘 지원 확대
    - 성능 개선 ( x2 )
      - SSE4.2
      - fadvise 지원 – linux native
      - 로컬 데이터 읽기 개선
      - 셔플 성능 개선 ( x2 )
      - 소규모 잡의 실행 환경 개선
      - 네임노드의 에디트 로그의 단순화
-

# 네임노드 HA

- Hadoop 2.0.0
  - Manual Failover
  - NFS를 통한 edit log 공유
- Hadoop 2.0.2
  - Automatic Failover 지원
  - NFS를 통한 edit log 공유
- Hadoop 2.0.3
  - Automatic Failover 지원
  - QJM 지원



# 네임노드 HA

---

- HDFS HA using QJM

- core-site.xml

- ```
<property>
```

- ```
<name>fs.defaultFS</name>
```

- ```
<value>hdfs://mycluster</value>
```

- ```
</property>
```

- ```
<property>
```

- ```
<name>ha.zookeeper.quorum</name>
```

- ```
<value>zookeeper1:2181,zookeeper2:2181,zookeeper3:2181</value>
```

- ```
</property>
```

---



# 네임노드 HA

---

- HDFS HA using QJM
  - Hdfs-site.xml

```
<!-- Automatic failover configuration -->
```

```
<property>
```

```
<name>dfs.ha.automatic-failover.enabled</name>
```

```
<value>true</value>
```

```
</property>
```

```
<!-- Storage for edits' files -->
```

```
<property>
```

```
<name>dfs.namenode.shared.edits.dir</name>
```

```
<value>qjournal://namenode01:8485;namenode02:8485;namenode03:8485/hadoop-cluster</value>
```

```
</property>
```

---

# 네임노드 HA

---

- HDFS HA using QJM

```
<!-- common server name -->
<property>
  <name>dfs.nameservices</name>
  <value>hadoop-cluster</value>
</property>
<property>
  <name>dfs.name.dir</name>
  <value>/home/name</value>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/data1/hdfs,/data2/hdfs</value>
</property>
<property>
  <name>dfs.journalnode.edits.dir</name>
  <value>/home/hadoop/hdfs/jn</value>
</property>
```

---

# 네임노드 HA

---

- HDFS HA using QJM

```
<!-- HA configuration -->
<property>
  <name>dfs.ha.namenodes.hadoop-cluster</name>
  <value>nn1,nn2</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.hadoop-cluster.nn1</name>
  <value>namenode01:8020</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.hadoop-cluster.nn2</name>
  <value>namenode02:8020</value>
</property>
<property>
  <name>dfs.namenode.http-address.hadoop-cluster.nn1</name>
  <value>namenode01:50070</value>
</property>
<property>
  <name>dfs.namenode.http-address.hadoop-cluster.nn2</name>
  <value>namenode02:50070</value>
</property>
```

---

# 네임노드 HA

---

- HDFS HA using QJM
  - `$HADOOP_HOME/sbin/hadoop-daemon.sh start journalnode` //각노드에서
  - `$HADOOP_HOME/sbin/hadoop-daemon.sh start zkfc` // namenode 에서

---

# Hadoop – MapReduce

---

2015

---

# Content

---

- MapReduce
  - WordCount
  - DepartureDelayCount

# MapReduce

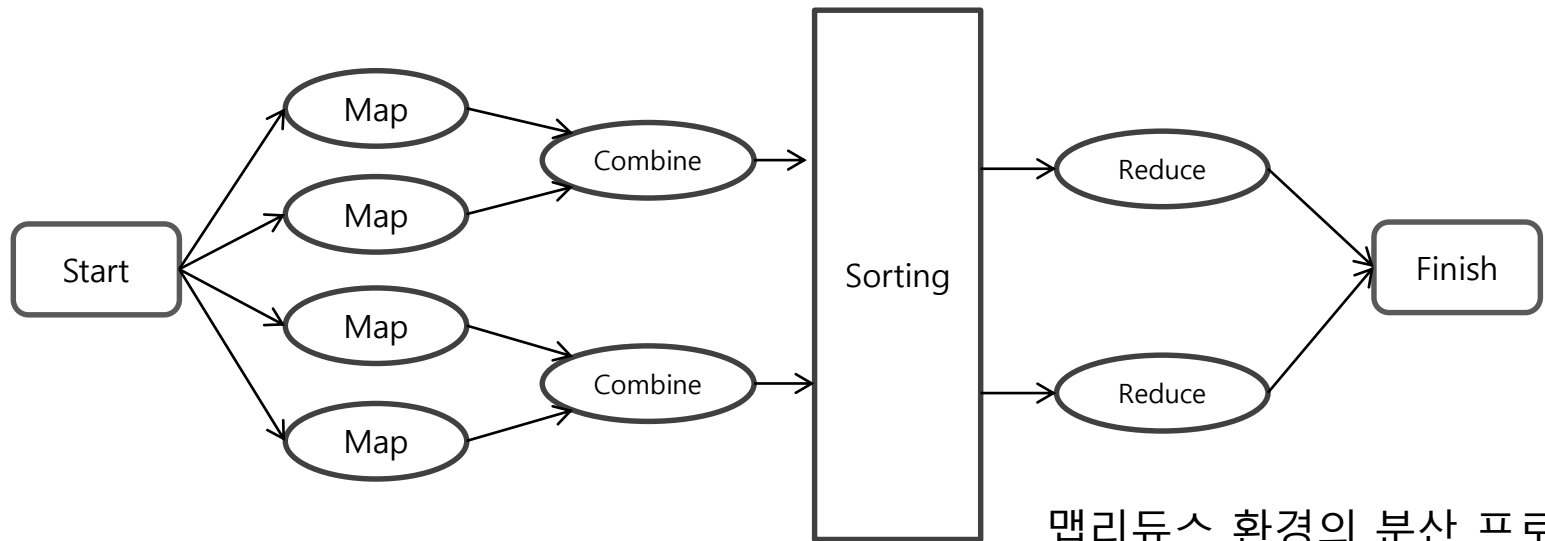
---

- 맵리듀스
  - HDFS에 분산 저장된 데이터에 스트리밍 접근을 요청하며 빠르게 분산 처리하도록 고안된 프로그래밍 모델, 이를 지원하는 시스템
  - 대규모 분산 컴퓨팅 혹은 단일 컴퓨팅 환경에서 개발자가 **대량의 데이터를 병렬로 분석**할 수 있음
  - 개발자는 맵리듀스 알고리즘에 맞게 **분석 프로그램을 개발**하고, 데이터의 입출력과 병렬 처리 등 기반 작업은 프레임워크가 알아서 처리해줌

# MapReduce

---

- 맵(Map) :  $(k1, v1) \rightarrow \text{list}(k2, v2)$ 
  - 데이터를 가공해서 분류(연산 가공자)
- 리듀스(Reduce) :  $(k2, \text{list}(v2)) \rightarrow \text{list}(k3, v3)$ 
  - 분류된 데이터를 통합(집계 연산자)



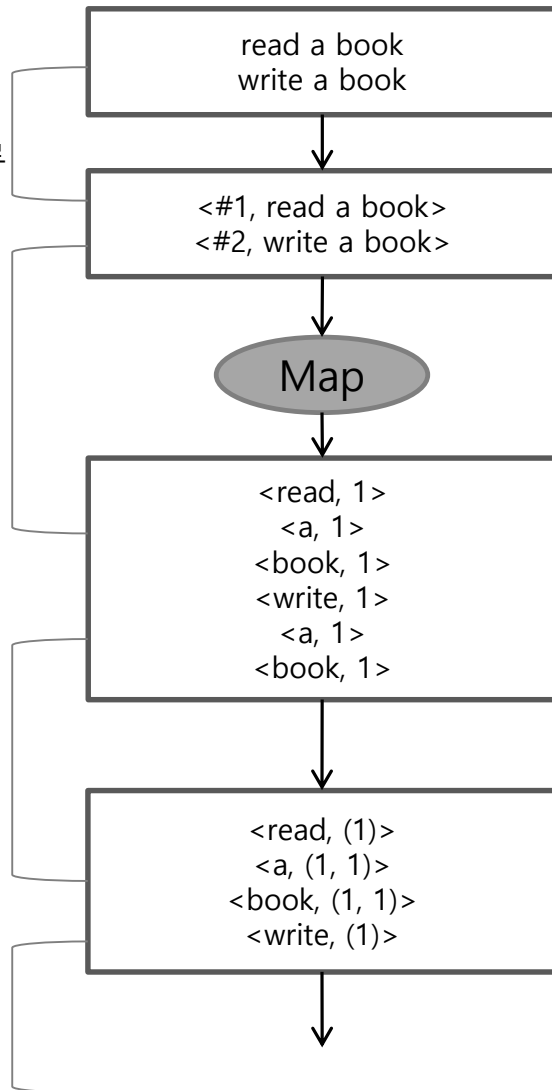


# MapReduce

**입력 데이터 분리**  
: 키와 값으로 분류

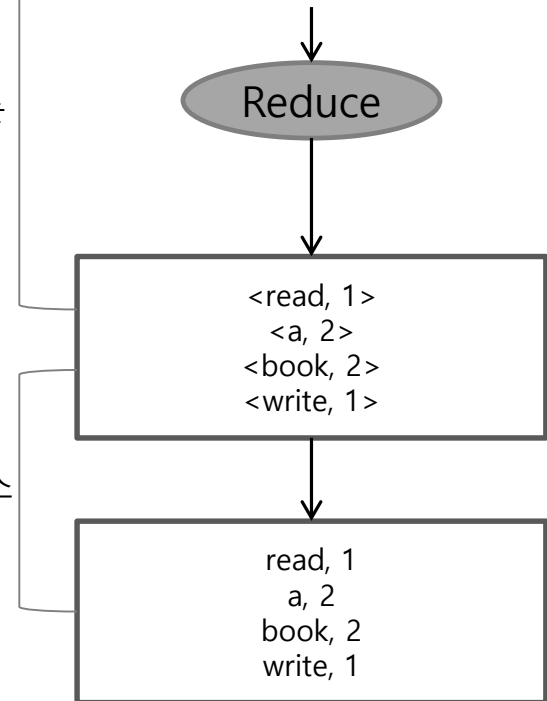
**맵 메서드**  
: 라인별로 문장을 체크, 키에 해당하는 글자별로 글자 수 출력

**정렬과 병합**  
: 맵 메서드의 출력 데이터를 정렬, 병합



**리듀스 메서드**  
: 키에 해당하는 글자별로 글자 수를 합산해서 출력

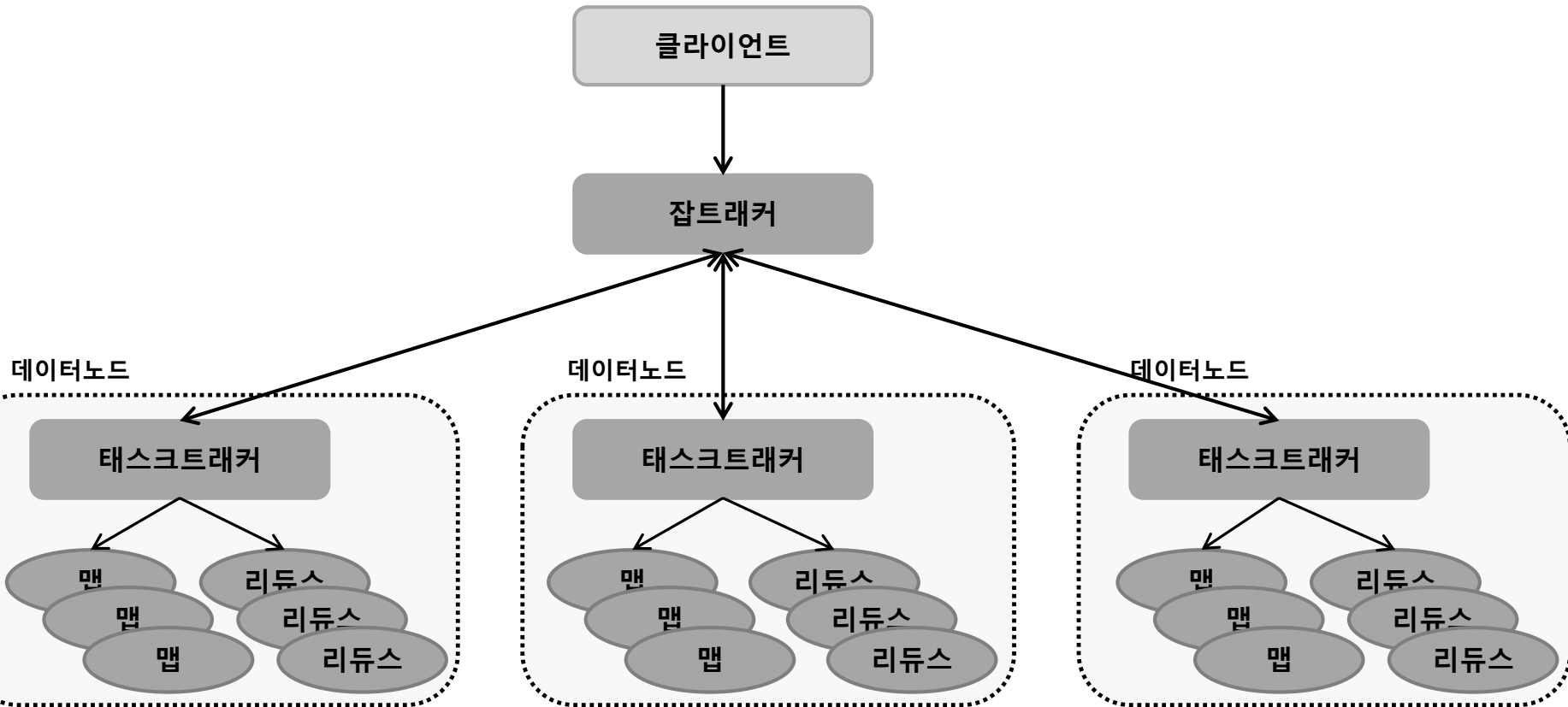
**저장**  
: 하둡 파일 시스템에 저장



Map input records=2  
Map output records=6  
Reduce input records=6  
(Reduce input groups=4)  
Reduce output records=4

# MapReduce Architecture

---



# MapReduce Architecture

- 클라이언트

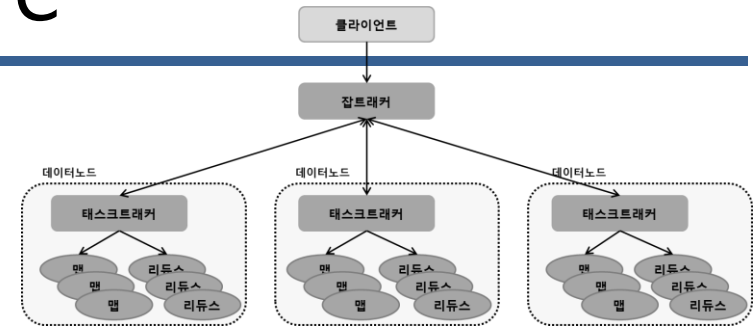
- 맵리듀스 프로그램 & API

- 잡트래커(JobTracker) (네임노드 서버에서 실행)

- 맵리듀스 프로그램은 잡(Job) 이라는 하나의 작업 단위로 관리됨
- 하둡 클러스터에 등록된 **전체 잡의 스케줄링을 관리하고 모니터링**
  - 사용자가 새로운 잡을 요청하면 잡트래커는 잡을 처리하기 위해 몇 개의 맵과 리듀스를 실행할지 계산 → 어떤 태스크트래커에서 실행할지 결정 → 잡 할당
- 전체 클러스터에서 하나의 잡트래커가 실행됨

- 태스크트래커(TaskTracker) (데이터노드에서 실행)

- 잡트래커의 작업 수행 요청을 받아 맵리듀스 프로그램 실행
- 잡트래커가 **요청한 맵과 리듀스 개수만큼 맵, 리듀스 태스크를 생성**
- 새로운 JVM을 구동해 맵, 리듀스 태스크 실행
  - 이 때, JVM은 재사용할 수 있게 설정 가능. 데이터노드가 하나라도 여러 개의 JVM을 실행해서 데이터를 동시에 분석함(병렬처리 가능)

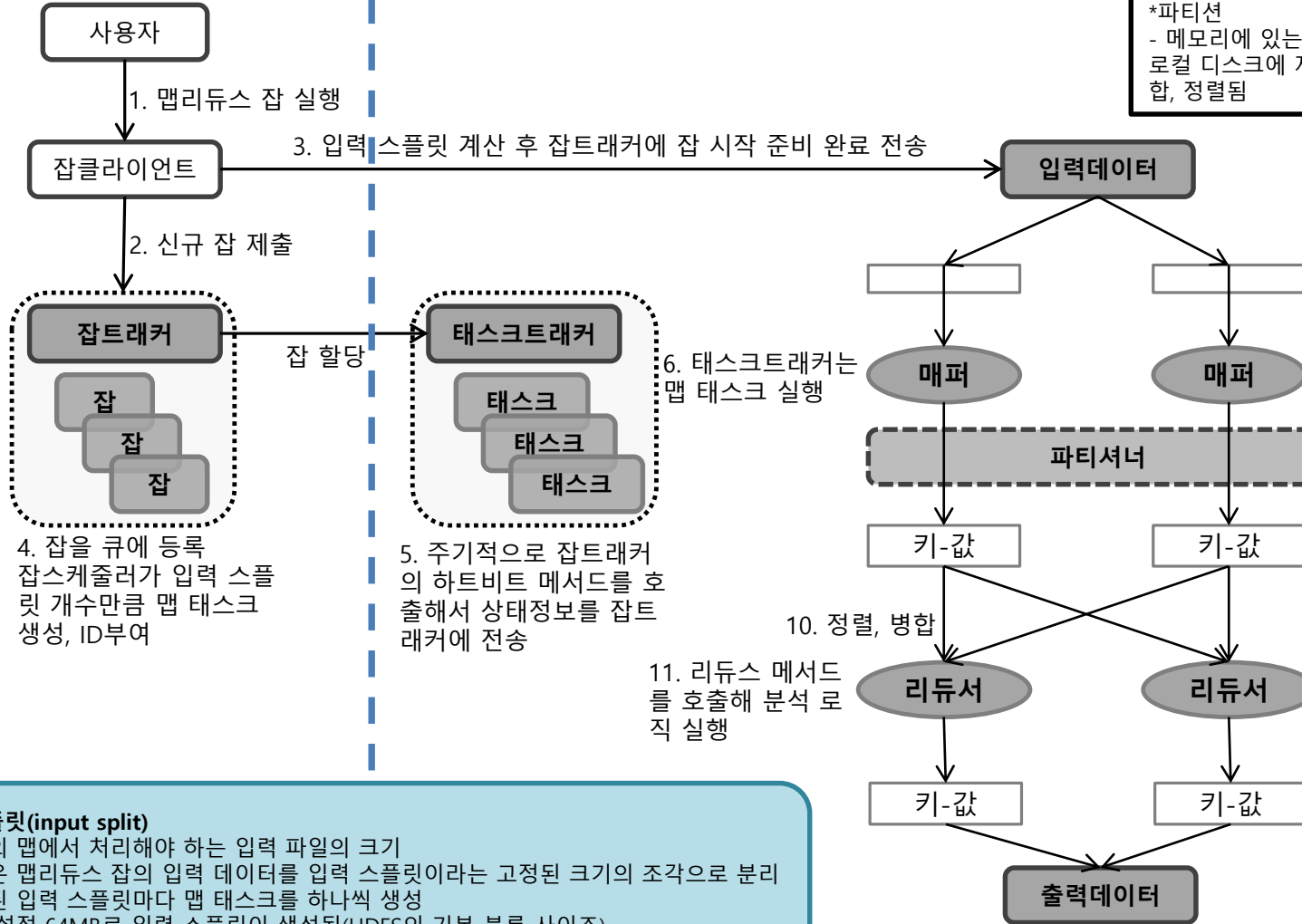


# MapReduce 작동방식

네임노드

데이터노드

- \*입력스플릿 결과
  - 입력 스플릿 정보, 설정 파일, 맵리듀스 JAR 파일을 HDFS에 저장
- \*하트비트 메서드를 통해 상태정보 전송
  - CPU, 메모리, 서버가용 리소스정보, 현재 실행 중 태스크 개수, 실행 가능한 최대 태스크 개수, 신규 태스크 실행 가능 여부 등
- \*파티션
  - 메모리에 있는 데이터를 키에 따라 정렬 후, 로컬 디스크에 저장, 하나의 출력 파일로 병합, 정렬됨



## 입력 스플릿(input split)

- 하나의 맵에서 처리해야 하는 입력 파일의 크기
- 하둡은 맵리듀스 잡의 입력 데이터를 입력 스플릿이라는 고정된 크기의 조각으로 분리
- 생성된 입력 스플릿마다 맵 태스크를 하나씩 생성
- 기본 설정 64MB로 입력 스플릿이 생성됨(HDFS의 기본 블록 사이즈)

# 응용분야

---

- Google Source Tree에 등록된 응용 프로그램
  - distributed Grep
  - distributed Sort
  - web access log stats
  - inverted index construction
  - document clustering
  - machine learning
  - statistical machine translation

---

# 하둡 프로그래밍 요소

---

# 하둡 프로그래밍 요소

---

- 1) 데이터 타입
- 2) InputFormat
- 3) 매퍼(Mapper)
- 4) 파티셔너(Partitioner)
- 5) 리듀서(Reducer)
- 6) 콤바이너
- 7) OutputFormat

# 1) 데이터 타입

- 맵리듀스 API는 자주 쓰는 데이터타입에 대한 WritableComparable 인터페이스를 구현한 Wrapper 클래스 제공

클래스명	대상 데이터 타입
BooleanWritable	Boolean
ByteWritable	단일 byte
DoubleWritable	Double
FloatWritable	Float
IntWritable	Integer
LongWritable	Long
Text Wrapper	UTF8 형식의 문자열
NullWritable	데이터값이 필요 없을 경우에 사용

- WritableComparable 인터페이스
  - Writable & Comparable 인터페이스를 다중 상속하고 있음
  - Comparable 인터페이스
    - Java.lang 패키지의 인터페이스로, 정렬을 처리하기 위해 compareTo 메서드 제공
  - Writable 인터페이스
    - write 메서드 → 데이터값을 직렬화함
    - readFields 메서드 → 직렬화된 데이터값을 해제해서 읽는 역할



## 2) InputFormat

---

- 입력 스플릿(Input Split)을 맵 메서드의 입력 파라미터로 사용할 수 있게 InputFormat 추상화 클래스 제공
- InputFormat 클래스
  - getSplits 메서드 → 입력 스플릿을 맵 메서드가 사용할 수 있도록 함
  - createRecordReader 메서드 → 맵 메서드가 입력 스플릿을 키와 목록의 형태로 사용할 수 있게 RecordReader 객체 생성함
    - 맵 메서드는 RecordReader 객체에 담겨 있는 키와 값을 읽어들이 분석 로직을 수행함

InputFormat	기능
TextInputFormat	텍스트 파일을 분석할 때 사용. 키는 LongWritable 타입, 값은 Text타입을 사용
KeyValueTextInputFormat	텍스트 파일을 입력 파일로 사용할 때 라인 번호가 아닌 임의의 키값을 지정해서 키와 값의 목록으로 읽음
NLineInputFormat	맵 태스크가 입력 받은 텍스트 파일의 라인 수를 제한하고 싶을 때 사용
DelegatingInputFormat	여러 개의 서로 다른 입력 포맷을 사용하는 경우에 각 경로에 대한 작업을 위임
CombineFileInputFormat	여러 개의 파일을 스플릿으로 묶어서 아용(다른 InputFormat은 파일당 스플릿을 생성)
SequenceFileInputFormat	시퀀스 파일(바이너리 형태의 키와 값의 목록으로 구성된 텍스트 파일)을 입력 데이터로 쓸 때 사용.
SequenceFileAsBinaryInputFormat	시퀀스 파일의 키와 값을 임의의 바이너리 객체로 변환해서 사용
SequenceFileAsTextInputFormat	시퀀스 파일의 키와 값을 Text 객체로 변환해서 사용

### 3) 매퍼(Mapper)

---

[ Mapper class ] 매퍼는 입력 스플릿마다 하나의 맵 태스크(매퍼 클래스)를 생성함.

- 대부분 이 매퍼 클래스를 상속바다 매퍼 클래스를 새롭게 구현하여 사용함
- Context 객체를 이용해 job에 대한 정보를 얻어오고, 입력 스플릿을 레코드 단위로 읽을 수 있음
  - RecordReader로 맵 메서드가 키와 값의 형태로 데이터를 읽을 수 있음
- map 메서드
- run 메서드
  - Context 객체에 있는 키를 순회하면서 맵 메서드를 호출함

```
Public class Context extends MapContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT>{  
    public Context(Configuration conf, TaskAttemptID taskid,  
                    RecordReader<KEYIN, VALUEIN> reader,  
                    RecordWriter<KEYOUT, VALUEOUT> writer,  
                    OutputCommitter committer, StatusReporter reporter,  
                    InputSplit split) throws IOException, InterruptedException {  
        super(conf, taskid, reader, writer, committer, reporter, split);  
    }  
}
```

## 4) 파티셔너 (Partitioner)

---

[ Partitioner ] 맵 태스크의 출력 데이터가 어떤 리듀스 태스크로 전달될지 결정함

- getPartition 메서드
  - 맵 태스크의 출력 키, 값과 전체 리듀스 태스크 개수를 파라미터로 받아 파티션을 계산

## 5) 리듀서(Reducer)

---

- 맵 태스크의 출력 데이터를 입력 데이터로 전달받아 집계 연산을 수행

## 6) 콤바이너 클래스(Combiner)

---

- 셔플(Shuffle)
  - 맵 태스크와 리듀스 태스크 사이의 데이터 전달 과정
  - 맵 태스크의 출력 데이터는 네트워크를 통해 리듀스 태스크로 전달됨
- 콤바이너 클래스
  - 셔플할 데이터의 크기를 줄이는 데 도움을 줌
    - 매퍼의 출력 데이터를 입력 데이터로 전달받아 연산을 수행
    - 로컬 노드에서 로컬에 생성된 매퍼의 출력 데이터를 이용하기 때문에 네트워크 비용이 발생하지 않음

## 7) OutputFormat

---

OutputFormat	기능
TextOutputFormat	텍스트 파일에 레코드를 출력할 때 사용. 키, 값의 구분자는 탭을 사용
SequenceFileOutputFormat	시퀀스 파일을 출력물로 쓸 때 사용
SequenceFileAsBinaryOutputFormat	위에 있는 것을 상속받아 구현되었음. 바이너리 포맷의 키와값을 SequenceFile 컨테이너에 씀
FilterOutputFormat	OutputFormat 클래스를 편리하게 사용할 수 있는 메서드 제공 (OutputFormat 클래스의 Wrapper 클래스)
LazyOutputFormat	FileOutputFormat을 상속받은 클래스는 출력할 내용이 없을때도 part-nnnnn을 생성함. 이 포맷을 사용하면 첫 번째 레코드가 해당 파티션(part-nnnnn)으로 보내질 때만 출력 파일을 생성함
NullOutputFormat	출력 데이터가 없을 때 사용

# WordCountMapper.java

---

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends
    Mapper<LongWritable, Text, Text, IntWritable> { //입력키, 입력값, 출력키, 출력값 타입

    private final static IntWritable one = new IntWritable(1); // 출력값 one의 초기값 '1'로 설정
    private Text word = new Text();
    public void map(LongWritable key, Text value, Context context) // 입력키, 입력값타입, Context 객체
        throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString()); //Tokenizer 선언
        while (itr.hasMoreTokens()) { // 공백 단위로 구분된 string값을 순회함
            word.set(itr.nextToken());
            // context객체의 write 메서드 → 매퍼의 출력 데이터에 레코드를 추가
            // 키가 word, 값이 one임
            context.write(word, one);}}}



---


```

# WordCountReducer.java

---

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends
    Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();
    // a<1,1> 과 같이 입력되는 값이 group으로 되어있을때 Iterator 선언.
    // Iterator를 이용해 입력 값 데이터를 탐색하고 계산할 수 있음
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) { // 입력값을 순회하면서 값을 더함
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);}}
```

---



# WordCount.java

---

```
package wikibooks.hadoop.chapter04;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        if (args.length != 2) {
            System.err.println("Usage: WordCount <input> <output>");
            System.exit(2);
        }
        Job job = new Job(conf, "WordCount");

        job.setJarByClass(WordCount.class);
        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
}
```

---

---

# 미국항공데이터

---

- 기본출력
  - Output value column 수 늘리기
  - MultipleOutput
-

# ASA(American Standards Association: 미국 규격 협회) 에서 2009년 공개한 미국 항공편 운항 통계 데이터

<http://stat-computing.org/dataexpo/2009>

1987-2008년까지의 미국 내 모든 상업 항공편에 대한 항공편 도착과 출발 세부 사항에 대한 정보 제공

The screenshot shows a web browser window titled "Data expo 09. ASA Statistics Computing and Graphics - Chrome". The address bar shows the URL "stat-computing.org/dataexpo/2009/". The page content includes the ASA logo, navigation links for "Statistical Computing" and "Statistical Graphics", and a search bar. The main section is titled "Airline on-time performance" and contains text about the dataset, its size, and the challenge of analyzing it. A sidebar on the right lists links for "Posters & results", "Competition description", "Download the data", "Supplemental data sources", "Using a database", and "Intro to command line tools". At the bottom, there are three download buttons for "1991.csv.bz2", "1990.csv.bz2", and "1989.csv.bz2", each showing the file size and time to download.

ASA Sections on:

- [Statistical Computing](#)
- [Statistical Graphics](#)

[ [Computing](#), [Graphics](#) ]  
[ [Awards](#), [Data expo](#), [Video library](#) ]  
[ [Events](#), [News](#), [Newsletter](#) ]

Data expo

## Airline on-time performance

Have you ever been stuck in an airport because your flight was delayed or cancelled and wondered if you could have predicted it if you'd had more data? This is your chance to find out.

**The results**

We had a total of [nine entries](#), and turn out at the poster session at the JSM was great, with plenty of people stopping by to find out why their flights were delayed.

**The data**

[The data](#) consists of flight arrival and departure details for all commercial flights within the USA, from October 1987 to April 2008. This is a large dataset: there are nearly 120 million records in total, and takes up 1.6 gigabytes of space compressed and 12 gigabytes when uncompressed. To make sure that you're not overwhelmed by the size of the data, we've provide two brief introductions to some useful tools: [linux command line tools](#) and [sqlite](#), a simple sql database.

**The challenge**

The aim of the data expo is to provide a **graphical** summary of important features of the data set. This is intentionally vague in order to allow different entries to focus on different aspects of the data, but here are a few ideas to get you started:

- When is the best time of day/day of week/time of year to fly to minimise delays?
- Do older planes suffer more delays?
- How does the number of people flying between different locations change over time?
- How well does weather predict plane delays?
- Can you detect cascading failures as delays in one airport create delays in others? Are there critical links in the system?

You are also welcome to work with interesting subsets: you might want to compare flight patterns before and after 9/11, or between the pair of cities that you fly between most often, or all flights to and from a major airport like Chicago (ORD). Smaller subsets may also help you to match up the data to [other interesting datasets](#).

Data expo 09

- [Posters & results](#)
- [Competition description](#)
- [Download the data](#)
- [Supplemental data sources](#)
- [Using a database](#)
- [Intro to command line tools](#)

1991.csv.bz2  
0.4/47.6MB, 1시간 남음

1990.csv.bz2  
0.5/49.6MB, 1시간 남음

1989.csv.bz2  
0.5/46.9MB, 1시간 남음

다운로드 항목 모두 표시...

# Download the data

The data. Data expo 09. ASA Statistics Computing and Graphics - Chrome

HDFS/user/medinfo/path Job (Hadoop 1.0.4 API) Context (Java Platform SE The data. Data expo 09.

stat-computing.org/dataexpo/2009/the-data.html

## Data expo '09

### Get the data

The data comes originally from [RITA](#) where it is [described in detail](#). You can download the data there, or from the bziped csv files listed below. These files have derivable variables removed, are packaged in yearly chunks and have been more heavily compressed than the originals.

Download individual years:

[1987](#), [1988](#), [1989](#), [1990](#), [1991](#), [1992](#), [1993](#), [1994](#), [1995](#), [1996](#), [1997](#), [1998](#), [1999](#), [2000](#), [2001](#), [2002](#), [2003](#), [2004](#), [2005](#), [2006](#), [2007](#), [2008](#)

### Keep in touch

If you download the data, please also subscribe to the data expo mailing list, so we can keep you up to date with any changes to the data:

Email:

### Variable descriptions

Name	Description
1 Year	1987-2008
2 Month	1-12
3 DayOfMonth	1-31
4 DayOfWeek	1 (Monday) - 7 (Sunday)
5 DepTime	actual departure time (local, hhmm)
6 CRSDepTime	scheduled departure time (local, hhmm)
7 ArrTime	actual arrival time (local, hhmm)
8 CRSArrTime	scheduled arrival time (local, hhmm)
9 UniqueCarrier	<a href="#">unique carrier code</a>
10 FlightNum	flight number
11 TailNum	plane tail number
12 ActualElapsedTime	in minutes
13 CRSElapsedTime	in minutes
14 AirTime	in minutes
15 ArrDelay	arrival delay, in minutes
16 DepDelay	departure delay, in minutes
17 Origin	origin <a href="#">IATA airport code</a>
18 Dest	destination <a href="#">IATA airport code</a>
19 Distance	in miles

1991.csv.bz2  
0.2/47.6MB, 52분 남음

1990.csv.bz2  
0.1/49.6MB, 1시간 남음

1989.csv.bz2  
0.2/46.9MB, 57분 남음

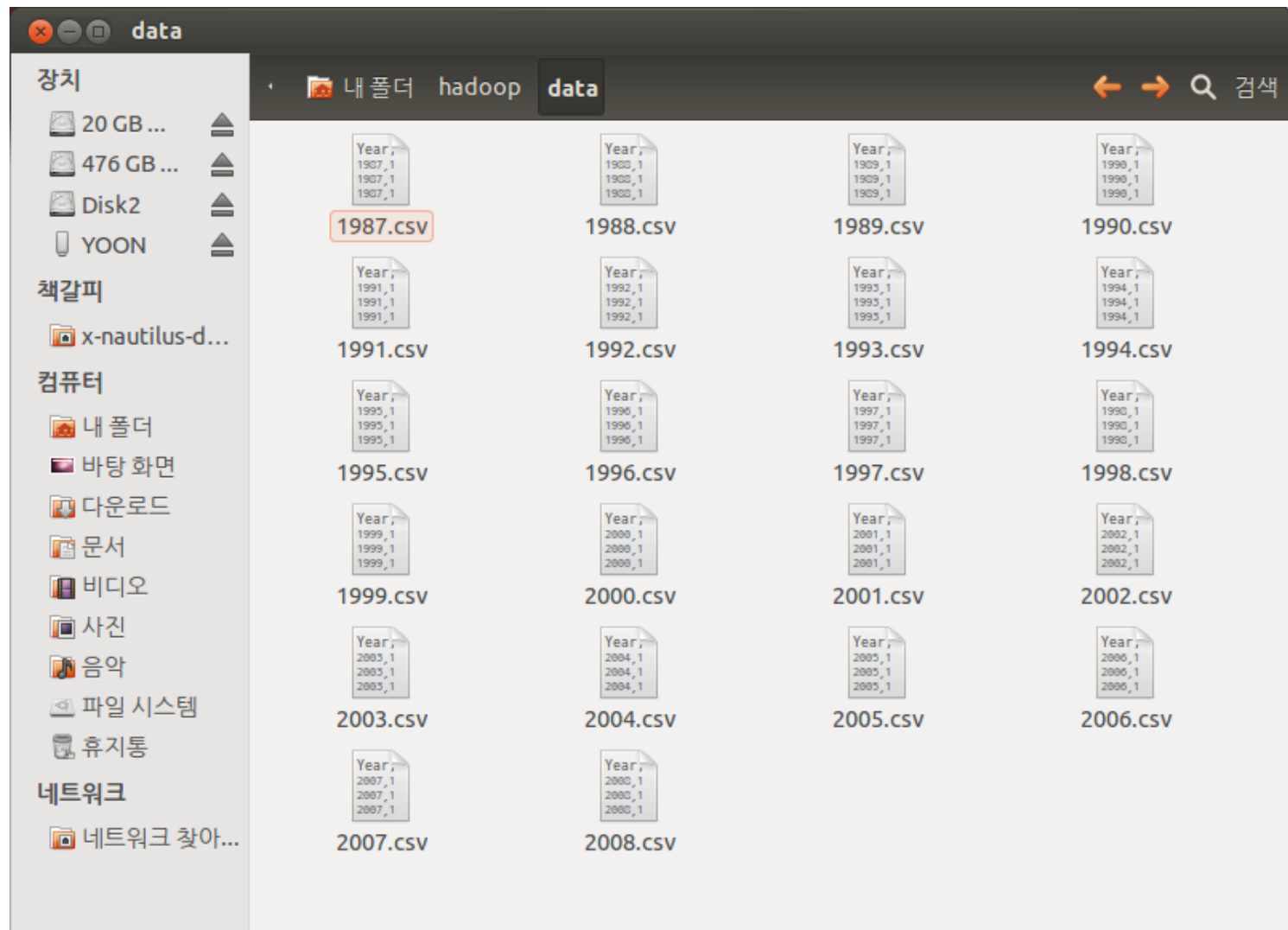
다운로드 항목 모두 표시

# 1987.csv 파일 상세 (29개의 칼럼)

\* 숫자 형식으로 데이터가 존재함

파일(F) 편집(E) 보기(V) 삽입(I) 서식(O) 도구(T) 데이터(D) 창(W) 도움말(H)																												
나눔고덕 10																												
A1 f(x) Σ = Year																												
1	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	
2	Year	Month	DayOfMonth	DayOfWeek	DepTime	CBSDepTime	ArrTime	CBSArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CBSElapsedTime	AirTime	ArrDelay	DepDelay	Origin	Dest	Distance	TaxiIn	TaxiOut	Cancelled	CancellationCode	Diverted	CarrierDelay	WeatherDelay	NASDe	
3	1987	10	14	3	741	730	912	849	PS	1451	NA	91	79	NA	23	11	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
4	1987	10	15	4	729	730	903	849	PS	1451	NA	94	79	NA	14	-1	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
5	1987	10	17	6	741	730	918	849	PS	1451	NA	97	79	NA	29	11	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
6	1987	10	18	7	729	730	847	849	PS	1451	NA	78	79	NA	-2	-1	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
7	1987	10	19	1	749	730	922	849	PS	1451	NA	93	79	NA	33	19	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
8	1987	10	21	3	728	730	848	849	PS	1451	NA	80	79	NA	-1	-2	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
9	1987	10	22	4	728	730	852	849	PS	1451	NA	84	79	NA	3	-2	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
10	1987	10	23	5	731	730	902	849	PS	1451	NA	91	79	NA	13	1	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
11	1987	10	24	6	744	730	908	849	PS	1451	NA	84	79	NA	19	14	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
12	1987	10	25	7	729	730	851	849	PS	1451	NA	82	79	NA	2	-1	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
13	1987	10	26	1	735	730	904	849	PS	1451	NA	89	79	NA	15	5	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
14	1987	10	28	3	741	725	919	855	PS	1451	NA	98	90	NA	24	16	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
15	1987	10	29	4	742	725	906	855	PS	1451	NA	84	90	NA	11	17	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
16	1987	10	31	6	726	725	848	855	PS	1451	NA	82	90	NA	-7	1	SAN	SFO	447	NA	NA	0	NA	0	NA	NA	NA	
17	1987	10	1	4	936	915	1035	1001	PS	1451	NA	59	46	NA	34	21	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
18	1987	10	2	5	918	915	1017	1001	PS	1451	NA	59	46	NA	16	3	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
19	1987	10	3	6	928	915	1037	1001	PS	1451	NA	69	46	NA	36	13	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
20	1987	10	4	7	914	915	1003	1001	PS	1451	NA	49	46	NA	2	-1	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
21	1987	10	5	8	1042	915	1129	1001	PS	1451	NA	47	46	NA	88	87	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
22	1987	10	6	9	934	915	1024	1001	PS	1451	NA	50	46	NA	23	19	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
23	1987	10	7	10	946	915	1037	1001	PS	1451	NA	51	46	NA	36	31	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
24	1987	10	8	11	932	915	1033	1001	PS	1451	NA	61	46	NA	32	17	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
25	1987	10	9	12	947	915	1036	1001	PS	1451	NA	49	46	NA	35	32	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
26	1987	10	10	13	915	915	1022	1001	PS	1451	NA	67	46	NA	21	0	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
27	1987	10	11	14	916	915	1006	1001	PS	1451	NA	50	46	NA	5	1	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
28	1987	10	12	15	944	915	1027	1001	PS	1451	NA	43	46	NA	26	29	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
29	1987	10	13	16	941	915	1036	1001	PS	1451	NA	55	46	NA	35	26	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
30	1987	10	14	17	930	915	1029	1001	PS	1451	NA	59	46	NA	28	15	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
31	1987	10	15	18	920	915	1023	1001	PS	1451	NA	63	46	NA	22	5	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
32	1987	10	16	19	1009	915	1104	1001	PS	1451	NA	55	46	NA	63	54	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
33	1987	10	17	20	915	915	1008	1001	PS	1451	NA	53	46	NA	7	0	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
34	1987	10	18	21	940	915	1032	1001	PS	1451	NA	52	46	NA	31	25	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
35	1987	10	19	22	913	915	1003	1001	PS	1451	NA	50	46	NA	2	-2	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
36	1987	10	20	23	915	915	1017	1001	PS	1451	NA	62	46	NA	16	0	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
37	1987	10	21	24	927	915	1022	1001	PS	1451	NA	55	46	NA	21	12	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
38	1987	10	22	25	929	915	1052	1001	PS	1451	NA	83	46	NA	51	14	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
39	1987	10	23	26	914	915	1011	1001	PS	1451	NA	57	46	NA	10	-1	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
40	1987	10	24	27	917	915	1023	1001	PS	1451	NA	66	46	NA	22	2	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
41	1987	10	25	28	916	915	1014	1001	PS	1451	NA	58	46	NA	13	1	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
42	1987	10	26	29	941	925	1106	1015	PS	1451	NA	85	50	NA	51	16	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
43	1987	10	27	30	940	925	1044	1015	PS	1451	NA	64	50	NA	29	15	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
44	1987	10	28	31	1009	925	1105	1015	PS	1451	NA	56	50	NA	50	44	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
45	1987	10	29	1	945	925	1040	1015	PS	1451	NA	55	50	NA	25	20	SFO	RNO	192	NA	NA	0	NA	0	NA	NA	NA	
46	1987	10	30	2	1520	1505	1624	1608	PS	1453	NA	64	63	NA	16	15	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
47	1987	10	31	3	1508	1505	1615	1608	PS	1453	NA	67	63	NA	7	3	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
48	1987	10	1	4	1526	1505	1625	1608	PS	1453	NA	59	63	NA	17	21	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
49	1987	10	2	5	1504	1505	1602	1608	PS	1453	NA	58	63	NA	-6	-1	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
50	1987	10	3	6	1505	1505	1607	1608	PS	1453	NA	62	63	NA	-1	0	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
51	1987	10	4	7	1512	1505	1614	1608	PS	1453	NA	62	63	NA	6	7	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
52	1987	10	5	8	1528	1505	1636	1608	PS	1453	NA	68	63	NA	28	23	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
53	1987	10	6	9	1606	1505	1708	1608	PS	1453	NA	62	63	NA	60	61	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
54	1987	10	7	10	1505	1505	1602	1608	PS	1453	NA	57	63	NA	-6	0	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
55	1987	10	8	11	NA	1505	NA	1608	PS	1453	NA	67	63	NA	NA	NA	BUR	OAK	325	NA	NA	1	NA	0	NA	NA	NA	
56	1987	10	9	12	1533	1505	1640	1608	PS	1453	NA	67	63	NA	32	28	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
57	1987	10	10	13	1528	1505	1639	1608	PS	1453	NA	71	63	NA	31	23	BUR	OAK	325	NA	NA	0	NA	0	NA	NA	NA	
58	1987	10	11	14	1543	1505	1645	1608	PS	1453	NA	62	63	NA	37	38	BUR	OAK	325	NA								

# 총 22개의 파일 → 11.4GB



# HDFS에 적재 (소요시간 6분)

---

```
medinfo@localhost: ~/hadoop
Command 'csv2' from package 'xml2' (universe)
Command 'cs' from package 'csound' (universe)
Command 'cv' from package 'radiance' (universe)
Command 'csi' from package 'chicken-bin' (universe)
Command 'csc' from package 'chicken-bin' (universe)
Command '2csv' from package 'xml2' (universe)
Command 'csr' from package 'rheolef' (universe)
Command 'csh' from package 'csh' (universe)
Command 'csh' from package 'tcsh' (universe)
Command 'sv' from package 'runit' (universe)
csv: command not found
medinfo@localhost:~/hadoop$ ./bin/hadoop fs -put /home/hadoop/data/ input
put: File /home/hadoop/data does not exist.
medinfo@localhost:~/hadoop$ ./bin/hadoop fs -put /home/medinfo/hadoop/data/ input
t
```

# ./bin/hadoop fs -ls input 으로 적재된 데이터 목록 확인

```
medinfo@localhost: ~/hadoop
put: File /home/hadoop/data does not exist.
medinfo@localhost:~/hadoop$ ./bin/hadoop fs -put /home/medinfo/hadoop/data/ input
medinfo@localhost:~/hadoop$ ./bin/hadoop fs -ls input
Found 22 items
-rw-r--r-- 1 medinfo supergroup 127162942 2013-01-29 20:24 /user/medinfo/input/1987.csv
-rw-r--r-- 1 medinfo supergroup 501039472 2013-01-29 20:21 /user/medinfo/input/1988.csv
-rw-r--r-- 1 medinfo supergroup 486518821 2013-01-29 20:19 /user/medinfo/input/1989.csv
-rw-r--r-- 1 medinfo supergroup 509194687 2013-01-29 20:24 /user/medinfo/input/1990.csv
-rw-r--r-- 1 medinfo supergroup 491210093 2013-01-29 20:19 /user/medinfo/input/1991.csv
-rw-r--r-- 1 medinfo supergroup 492313731 2013-01-29 20:23 /user/medinfo/input/1992.csv
-rw-r--r-- 1 medinfo supergroup 490753652 2013-01-29 20:21 /user/medinfo/input/1993.csv
-rw-r--r-- 1 medinfo supergroup 501558665 2013-01-29 20:24 /user/medinfo/input/1994.csv
-rw-r--r-- 1 medinfo supergroup 530751568 2013-01-29 20:20 /user/medinfo/input/1995.csv
-rw-r--r-- 1 medinfo supergroup 533922363 2013-01-29 20:25 /user/medinfo/input/1996.csv
-rw-r--r-- 1 medinfo supergroup 540347861 2013-01-29 20:22 /user/medinfo/input/1997.csv
-rw-r--r-- 1 medinfo supergroup 538432875 2013-01-29 20:23 /user/medinfo/input/1998.csv
-rw-r--r-- 1 medinfo supergroup 476064515 2013-01-29 20:25 /user/medinfo/input/1999.csv
-rw-r--r-- 1 medinfo supergroup 570151613 2013-01-29 20:21 /user/medinfo/input/2000.csv
-rw-r--r-- 1 medinfo supergroup 600411462 2013-01-29 20:26 /user/medinfo/input/2001.csv
-rw-r--r-- 1 medinfo supergroup 530507013 2013-01-29 20:25 /user/medinfo/input/2002.csv
-rw-r--r-- 1 medinfo supergroup 626745242 2013-01-29 20:26 /user/medinfo/input/2003.csv
-rw-r--r-- 1 medinfo supergroup 669879113 2013-01-29 20:22 /user/medinfo/input/2004.csv
-rw-r--r-- 1 medinfo supergroup 390569790 2013-01-29 20:20 /user/medinfo/input/2005.csv
-rw-r--r-- 1 medinfo supergroup 672068096 2013-01-29 20:22 /user/medinfo/input/2006.csv
-rw-r--r-- 1 medinfo supergroup 702878193 2013-01-29 20:20 /user/medinfo/input/2007.csv
-rw-r--r-- 1 medinfo supergroup 428362658 2013-01-29 20:23 /user/medinfo/input/2008.csv
medinfo@localhost:~/hadoop$
```



# DepartureDelayCount 실행 소요시간(19분)

---

```
medinfo@localhost: ~/hadoop
glmedinfo@localhost's password:
localhost: starting tasktracker, logging to /home/medinfo/hadoop-1.0.4/libexec/.
./logs/hadoop-medinfo-tasktracker-localhost.out
medinfo@localhost:~/hadoop$ jps
2217
2423 NameNode
3316 TaskTracker
3379 Jps
3058 JobTracker
2963 SecondaryNameNode
2671 DataNode
medinfo@localhost:~/hadoop$ ./bin/hadoop jar wikibooks-hadoop-examples.jar wikib
ooks.hadoop.chapter05.DepartureDelayCount input dep_delay_count
13/01/29 20:35:16 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
13/01/29 20:35:16 INFO input.FileInputFormat: Total input paths to process : 22
13/01/29 20:35:16 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/01/29 20:35:16 WARN snappy.LoadSnappy: Snappy native library not loaded
13/01/29 20:35:16 INFO mapred.JobClient: Running job: job_201301292033_0001
13/01/29 20:35:17 INFO mapred.JobClient:  map 0% reduce 0%
13/01/29 20:35:39 INFO mapred.JobClient:  map 1% reduce 0%
13/01/29 20:35:54 INFO mapred.JobClient:  map 2% reduce 0%
13/01/29 20:36:06 INFO mapred.JobClient:  map 3% reduce 0%
```

# 출력 결과

---

- 맵(117개 태스크)에서 30초, 리듀서(1개 태스크)에서 18분 30초 소요.
- Map input records = 117,161,410 건
- Map output & Reduce input records = 47,765,560
- Reduce output records = 245 건

# Input&output Data Type

---

**Task:** 연도별로 얼마나 많은 항공기이 출발이 지연됐는지 계산

Class	Input or Output	Key	Value
Mapper	Input	오프셋	항공 운항 통계 데이터
	Output	운항연도, 운항월	출발 지연 건수
Reducer	Input	운항연도, 운항월	출발 지연 건수
	Output	운항연도, 운항월	출발 지연 건수 합계

# DepartureDelayCountMapper.java

---

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class DepartureDelayCountMapper extends
    Mapper<LongWritable, Text, Text, IntWritable> {

    // map 출력값
    private final static IntWritable outputValue = new IntWritable(1);
    // map 출력키
    private Text outputKey = new Text();

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        if (key.get() > 0) {
            // 콤마 구분자 분리
            String[] columns = value.toString().split(",");
            if (columns != null && columns.length > 0) {
                try {
                    // 출력키 설정
                    outputKey.set(columns[0] + "," + columns[1]);
                    if (!columns[15].equals("NA")) {
                        int depDelayTime = Integer.parseInt(columns[15]);
                        if (depDelayTime > 0) {
                            // 출력 데이터 생성
                            context.write(outputKey, outputValue);
                        }
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

---

# DelayCountReducer.java

---

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class DelayCountReducer extends
    Reducer<Text, IntWritable, Text, IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable value : values)
            sum += value.get();
        result.set(sum);
        context.write(key, result);
    }
}
```

---

# DepartureDelayCount.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

```
public class DepartureDelayCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        // 입력출 데이터 경로 확인
        if (args.length != 2) {
            System.err.println("Usage: DepartureDelayCount <input> <output>");
            System.exit(2);
        }
        // Job 이름 설정
        Job job = new Job(conf, "DepartureDelayCount");

        // 입출력 데이터 경로 설정
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Job 클래스 설정
        job.setJarByClass(DepartureDelayCount.class);
        // Mapper 클래스 설정
        job.setMapperClass(DepartureDelayCountMapper.class);
        // Reducer 클래스 설정
        job.setReducerClass(DelayCountReducer.class);

        // 입출력 데이터 포맷 설정
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        // 출력키 및 출력값 유형 설정
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.waitForCompletion(true);
    }
}
```

# 130304\_OutputKey<[0],[1],[8]>

---

```
outputKey.set(columns[0] + "," + columns[1] + "," + columns[8]);  
if (!columns[15].equals("NA")) {  
    int depDelayTime = Integer.parseInt(columns[15]);  
    if (depDelayTime > 0) {
```

```
medinfo@localhost:~/hadoop$ ./bin/hadoop jar wikibooks-hadoop-examples.jar wikib  
ooks.hadoop.chapter05.CopyOfDepartureDelayCount input copy_chap05  
Exception in thread "main" java.lang.ClassNotFoundException: wikibooks.hadoop.ch  
apter05.CopyOfDepartureDelayCount  
    at java.net.URLClassLoader$1.run(URLClassLoader.java:202)  
    at java.security.AccessController.doPrivileged(Native Method)  
    at java.net.URLClassLoader.findClass(URLClassLoader.java:190)  
    at java.lang.ClassLoader.loadClass(ClassLoader.java:306)  
    at java.lang.ClassLoader.loadClass(ClassLoader.java:247)  
    at java.lang.Class.forName0(Native Method)  
    at java.lang.Class.forName(Class.java:247)  
    at org.apache.hadoop.util.RunJar.main(RunJar.java:149)  
medinfo@localhost:~/hadoop$ ./bin/hadoop jar wikibooks-hadoop-examples.jar wikib  
ooks.hadoop.chapter05.CopyOfDepartureDelayCount input copy_chap05  
13/03/04 13:52:40 WARN mapred.JobClient: Use GenericOptionsParser for parsing th  
e arguments. Applications should implement Tool for the same.  
13/03/04 13:52:41 INFO input.FileInputFormat: Total input paths to process : 22  
13/03/04 13:52:41 INFO util.NativeCodeLoader: Loaded the native-hadoop library  
13/03/04 13:52:41 WARN snappy.LoadSnappy: Snappy native library not loaded  
13/03/04 13:52:41 INFO mapred.JobClient: Running job: job_201303041346_0001  
13/03/04 13:52:42 INFO mapred.JobClient: map 0% reduce 0%
```

# 130304\_OutputKey<[0],[1],[8]>

---

1	1987,10,AA	26795
2	1987,10,AS	2230
3	1987,10,CO	12924
4	1987,10,DL	21240
5	1987,10,EA	7026
6	1987,10,HP	5671
7	1987,10,NW	10138
8	1987,10,PA (1)	843
9	1987,10,PI	22674
10	1987,10,PS	9273
11	1987,10,TW	9513
12	1987,10,UA	20307
13	1987,10,US	18524
14	1987,10,WN	8410
15	1987,11,AA	26005
16	1987,11,AS	2077
17	1987,11,CO	14975
18	1987,11,DL	23379
19	1987,11,EA	8504
20	1987,11,HP	5399
21	1987,11,NW	10956
22	1987,11,PA (1)	1310
23	1987,11,PI	23797
24	1987,11,PS	7396
25	1987,11,TW	9109
26	1987,11,UA	18692
27	1987,11,US	17892
28	1987,11,WN	7727
29	1987,12,AA	30507
30	1987,12,AS	3096
31	1987,12,CO	19683
32	1987,12,DL	29856
33	1987,12,EA	11521
34	1987,12,HP	5739
35	1987,12,NW	15437
36	1987,12,PA (1)	1478
37	1987,12,PI	26594
38	1987,12,PS	8931
39	1987,12,TW	11873
40	1987,12,UA	26270
41	1987,12,US	18970
42	1987,12,WN	8903
43	1988,1,AA	28894
44	1988,1,AS	1990
45	1988,1,CO	18176
46	1988,1,DL	23281
47	1988,1,EA	12929
48	1988,1,HP	4681
49	1988,1,NW	15180
50	1988,1,PA (1)	1685
51	1988,1,PI	24730
52	1988,1,PS	6960
53	1988,1,TW	11701
54	1988,1,UA	24714
55	1988,1,US	16847
56	1988,1,WN	6842



# MultipleOutput

- org.apache.hadoop.mapreduce.lib.output.**MultipleOutputs**
- 다수의 파일 출력
- MultipleOutputs는 여러 개의 OutputCollectors를 만들고 각 OutputCollectors에 대한 출력 경로, 출력 포맷, 키와 값 유형을 설정함
- TASK: 기본으로 MULTIPLEOUTPUT 실행 후 코드 변경하여 3개의 파일이 출력되도록 해보기 → if문 추가하여 변수 추가

```
medinfo@localhost:~/hadoop$ ./bin/hadoop jar wikibooks-hadoop-examples.jar wikibooks.hadoop.chapter05.DelayCountWithMultipleOutputs input delay_count_mos
```

Name	Type	Size	Replication	Block Size
<a href="#">_SUCCESS</a>	file	0 KB	1	64 MB
<a href="#">_logs</a>	dir			
<a href="#">arrival-r-00000</a>	file	3.41 KB	1	64 MB
<a href="#">departure-r-00000</a>	file	3.4 KB	1	64 MB
<a href="#">part-r-00000</a>	file	0 KB	1	64 MB

Name	Type	Size	Replication	Block Size
<a href="#">_SUCCESS</a>	file	0 KB	1	64 MB
<a href="#">_logs</a>	dir			
<a href="#">arrival-r-00000</a>	file	5.62 KB	1	64 MB
<a href="#">cancelled-r-00000</a>	file	3 KB	1	64 MB
<a href="#">departure-r-00000</a>	file	3.4 KB	1	64 MB
<a href="#">part-r-00000</a>	file	0 KB	1	64 MB

---

# GenericOptionsParser, Tool, ToolRunner

131p(노란하둑), 135p(The Definitive guide)

---

**2013.03.08**

# GenericOptionsParser, Tool, ToolRunner(사용자 정의 옵션)

---

- org.apache.hadoop.util 패키지에 개발자가 편리하게 맵리듀스 프로그램을 개발할 수 있도록하는 다양한 헬퍼 클래스가 있음
- GenericOptionsParser, Tool, ToolRunner 클래스를 이용해 Job을 실행할 때 환경설정 정보 확인, 잡 드라이버 클래스에서 환경정보를 수정할 수 있음
- 현재 예제에서 세가지 옵션을 이용하여 구현한 사항은 다음과 같음
  - DelayCount에서 입력이 'A'일 땐 도착지연건수를, 'D'는 출발지연건수를 출력하는 옵션

# GenericOptionsParser

---

- 하둡 콘솔 명령어에서 입력한 옵션을 분석하는 클래스
- 사용자가 하둡 콘솔 명령어에서 네임노드, 잡트래커 추가 구성 자원 등을 설정할 수 있는 여러 가지 옵션을 제공
- GenericOptionsParser 클래스는 내부적으로 Configuration 객체를 만들어 생성자에게 전달받은 환경설정 정보를 설정함

옵션	기능
-conf<파일명>	명시한 파일을 환경설정에 있는 리소스 정보에 추가
-D<옵션=값>	하둡 환경설정 파일에 있는 옵션에 새로운 값을 설정
-fs<네임노드 호스트: 네임노드 포트>	네임노드를 새롭게 설정
-jt<잡트래커 호스트: 잡트래커 포트>	잡트래커를 새롭게 설정
-files<파일1, 파일2, ..., 파일n>	로컬에 있는 파일을 HDFS에서 사용하는 공유 파일 시스템으로 복사
-libjars<JAR파일1, JAR파일2,...,JAR파일n>	로컬에 있는 JAR파일을 HDFS에서 사용하는 공유 파일시스템으로 복사하고, 맵리듀스의 태스크 클래스패스에 추가
-archives<아카이브파일1, 아카이브파일2,...,아카이브파일n>	로컬에 있는 아카이브 파일을 HDFS에서 사용하는 공유 파일 시스템으로 복사한 후 압축을 풉니다.

# Tool

---

- GenericOptionsParser의 콘솔 설정 옵션을 지원하기 위한 인터페이스
- GenericOptionsParser가 사용하는 Configuration 객체를 상속받음

# ToolRunner

---

- GenericOptionsParser를 내부적으로 선언한 ToolRunner 클래스
- Tool인터페이스의 실행을 도와주는 헬퍼 클래스
- 즉, Configuration 객체를 Tool 인터페이스에 전달한 후, Tool 인터페이스의 run메서드를 실행함

# 체인

---

- 하나의 맵리듀스 잡에서 여러 개의 매퍼와 리듀서를 실행할 수 있게 체인맵퍼(ChainMapper)와 체인리듀서(ChainReducer) 제공
- 체인을 이용하면 입출력이 감소할 것임
- 매퍼1 → 매퍼2 → 리듀서 → 매퍼3 → 매퍼4의 경우 각 job에서 나오는 출력 key, value값이 그 다음 매퍼 혹은 리듀서의 입력 key, value가 됨

---

# 정렬(sorting)

---

- 보조 정렬(Secondary Sort)
  - 부분 정렬(Partial Sort)
  - 전체 정렬(Total Sort)
-



# 보조 정렬(Secondary Sort)

- 항공 운항 데이터 분석 결과를 보면 '월'의 순서가 제대로 처리되지 않았음
  - 1 → 10 → 11 → 12 → 2 → 3 → 의 순서로 처리되어있음
  - Key값이 연도와 월이 합쳐진 하나의 문자열로 인식되었기 때문
- 보조정렬
  - Key의 값들을 그룹핑하고, 그룹핑된 레코드에 순서를 부여하는 방식
  - 1. 기존 key의 값들을 조합한 복합키(composit key)를 정의. 이 때 키 값 중 어떤 키를 그룹핑 키로 사용할지 결정
  - 2. 복합키의 레코드를 정렬하기 위한 비교기(Comparator)를 정의
  - 3. 그룹핑 키를 파티셔닝할 파티셔너(Partitioner)를 정의
  - 4. 그룹핑 키를 정렬하기 위한 비교기(Comparator)를 정의

1987,10	175568
1987,11	177218
1987,12	218858
1988,1	198610
1988,10	162211
1988,11	175123
1988,12	189137
1988,2	177939
1988,3	187141
1988,4	159216
1988,5	164107
1988,6	165596
1988,7	174844
1988,8	175591
1988,9	138322
1989,1	178161
1989,10	173312
1989,11	176805
1989,12	213745
1989,2	181324
1989,3	204720
1989,4	157890

# 보조 정렬(Secondary Sort)

---

- Task: 항공 운항 지연 데이터의 결과를 월 순서로 정렬하기
  - 복합키의 연도를 그룹키로 사용 → 연도별로 그룹핑된 데이터들은 월의 순서대로 정렬

## 1. 복합키 구현

- 복합키란 기존 키값을 조합한 일종의 키 집합 클래스
- 현재까지의 출력key는 단순한 하나의 문자열
- 복합키를 적용하면 연도와 월이 각각 멤버 변수로 정의됨
- WritableComparable 인터페이스
  - 파라미터는 바로 자신인 DateKey로 설정
  - 여기엔 세가지 메서드가 구현되어 있어야 함
  - readFields : 입력스트림에서 연도와 월을 조회
  - write: 출력스트림에 연도와 월을 출력
  - compareTo : 복합키와 복합키를 비교해서 순서 정할 때 사용
- 연도는 string타입, 월은 integer타입 & setter와 getter 메서드를 함께 구현해야 함
- toString 메서드를 호출하여 출력 양식을 구현하여 값을 출력
  - 정의하지 않으면 자바에서 사용하는 클래스 식별자가 출력됨

# 보조 정렬(Secondary Sort)

---

## 2. 복합키 비교기 구현

- 복합키의 정렬 순서를 부여하기 위한 클래스
- 두 개의 복합키를 비교하며 각 멤버변수를 비교해 정렬순서를 정함
- WritableComparator를 상속받아 비교기 구현
  - compare 메서드 : 기본으로 객체를 스트림에서 조회한 값을 비교하게 되므로 정확한 정렬 순서 반영하지 못함. 그래서 이 메서드를 재정의해서 멤버 변수를 비교하는 로직을 구현해야 함
- compare 메서드에서 파라미터로 전달받은 객체 w1, w2를 DateKey에서 선언한 멤버변수를 조회하기 위해 DateKey클래스로 변환함 ( `DateKey k1 = (DateKey) w1;` )
- 연도 값 비교
  - String 클래스에서 제공하는 compareTo 메서드 사용하여
  - 'k1 == k2'일 때 '0', 'k1 > k2'일 때 '1', 'k1 < k2'일 때 '-1'임. 이건 compareTo 메서드에 정의되어 있는것
  - Year에 대해선 같은 연도 안에서 월의 순서가 정렬되어야 하니까, 일치하기만 하면 월의 정렬을 하는 코드로 이동!
  - 실제 연도 값을 정렬하는건 '그룹키 비교기'에서 함
- 월 값 비교
  - 'k1 == k2'일 때 '0', 'k1 > k2'일 때 '1', 'k1 < k2'일 때 '-1' 을 똑같이 이용하여 오름차순으로 정렬
  - 월 값은 integer이므로 compareTo 메서드를 사용하는게 아니라 수동으로 규칙을 입력하여 정렬

# 보조 정렬(Secondary Sort)

---

## 3. 그룹키 파티셔너 구현

- 파티셔너는 맵 태스크의 출력데이터를 리듀스 태스크의 입력 데이터로 보낼지 결정함
- 이렇게 파티셔닝된 데이터는 맵태스크의 출력 데이터의 key, value값에 따라 정렬됨
- 그룹핑 키로 사용하는 연도에 대한 파티셔닝을 수행함
- Partitioner를 상속받아 구현해야함
  - 두 개의 파라미터는 mapper 출력데이터의 key와 value
  - getPartition 메서드로 파티셔닝 번호를 조회하고, 재정의해서 구현 후 연도에 대한 해시코드를 조회해 파티션 번호생성

## 4. 그룹키 비교기 구현

- 같은 연도에 해당하는 모든 데이터를 하나의 REDUCER 그룹에서 처리할 수 있음
- 복합키 비교기처럼 WritableComparator를 상속받아 클래스 선언

# 보조 정렬(Secondary Sort)

---

- 위의 네 가지의 job을 적용하여 매퍼와 리듀서를 작성함
  - 문제사항: 리듀서에서 count를 합산할 경우 데이터에 오류 발생(171p)
    - 2008년도 데이터를 처리할 때, 2008년 12월만 출력되고, 카운트는 2008년 모든 카운트가 합산되어 출력됨
    - 그 이유는 year정보인 그룹키를 기준으로 연산을 수행하기 때문
    - 이를 해결하기 위해 복합키를 구분해서 처리해야 하고, 그를 위한 것으로 백업month(bMonth)를 이용해서 데이터를 나누어야 함
    - bMonth와 현재 들어온 key.getMonth()를 비교해서 일치하지 않으면 bMonth의 카운트를 출력하는 방식
    - 다음 순서의 월의 카운트를 합산하기 위해 sum변수를 0으로 초기화함
-

# 보조 정렬(Secondary Sort)

```
medinfo@localhost:~/hadoop$ ./bin/hadoop jar wikibooks-hadoop-examples.jar wikibooks.hadoop.chapter06.DelayCountWithDateKey input delay_count_sorting
```

Name	Type	Size	Replication	Block Size
<a href="#">_SUCCESS</a>	file	0 KB	1	64 MB
<a href="#">_logs</a>	dir			
<a href="#">arrival-r-00000</a>	file	3.41 KB	1	64 MB
<a href="#">departure-r-00000</a>	file	3.4 KB	1	64 MB
<a href="#">part-r-00000</a>	file	0 KB	1	64 MB

Ex. arrival\_delay\_count

```
1987,10 265658
1987,11 255127
1987,12 287408
1988,1 261810
1988,2 242219
1988,3 255083
1988,4 219288
1988,5 221071
1988,6 215385
1988,7 224274
1988,8 227943
1988,9 204558
1988,10 230876
1988,11 237343
1988,12 249340
1989,1 239136
1989,2 231073
1989,3 251148
1989,4 207190
1989,5 227174
1989,6 249363
1989,7 241827
1989,8 260291
1989,9 216720
1989,10 235073
```

```
medinfo@localhost:~/hadoop$ ./bin/hadoop jar wikibooks-hadoop-examples.jar wikibooks.hadoop.chapter06.DelayCountWithDateKey input delay_count_sorting_three
```

Name	Type	Size	Replication	Block Size
<a href="#">_SUCCESS</a>	file	0 KB	1	64 MB
<a href="#">_logs</a>	dir			
<a href="#">arrival-r-00000</a>	file	3.72 KB	1	64 MB
<a href="#">cancelled-r-00000</a>	file	2.72 KB	1	64 MB
<a href="#">departure-r-00000</a>	file	3.4 KB	1	64 MB
<a href="#">part-r-00000</a>	file	0 KB	1	64 MB

# 부분 정렬(Partial Sort)

---

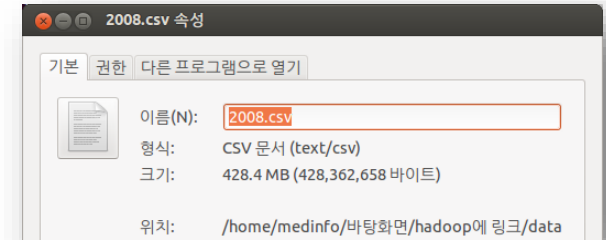
- 매퍼의 출력 데이터(파티셔닝 된 것)를 맵파일(MapFile)로 변경해서 데이터를 검색하는 방법
- 특정 키에 대한 데이터를 검색할 경우, 해당 키에 대한 데이터가 저장된 맵파일에 접근해서 데이터를 조회함
- Task: 미국 항공 지연 통계 데이터를 항공 운항 거리 순서대로 정렬하는 부분 정렬 프로그램
  1. 입력 데이터를 시퀀스 파일로 생성
  2. 시퀀스 파일을 맵파일로 변경
  3. 맵파일에서 데이터를 검색

# 부분 정렬(Partial Sort)

## 1. 시퀀스 파일 생성

- 미국 항공 운항 데이터를 시퀀스 파일로 출력
- 이 시퀀스 파일은 다음 단계에서 맵파일로 변환됨
- 2008년도 데이터를 대상으로 시퀀스 파일 생성
- 첫 행은 컬럼명이 써져 있기 때문에 출력하지 않음
- 키를 운항거리, 값은 심표로 구분된 데이터들이 출력됨

클래스	입출력 구분	키	값
매퍼	입력	오프셋(LongWritable)	항공 운항 통계 데이터(Text)
	출력	운항거리(IntWritable)	항공 운항 통계 데이터(Text)



Name	Type	Size	Replication	Block Size
<a href="#">_SUCCESS</a>	file	0 KB	1	64 MB
<a href="#">_logs</a>	dir			
<a href="#">part-00000</a>	file	17.92 MB	1	64 MB
<a href="#">part-00001</a>	file	18.16 MB	1	64 MB
<a href="#">part-00002</a>	file	17.5 MB	1	64 MB
<a href="#">part-00003</a>	file	17.7 MB	1	64 MB
<a href="#">part-00004</a>	file	17.54 MB	1	64 MB
<a href="#">part-00005</a>	file	17.42 MB	1	64 MB
<a href="#">part-00006</a>	file	7.14 MB	1	64 MB

```
medinfo@localhost:~/hadoop$ ./bin/hadoop jar wikibooks-hadoop-examples.jar wikibooks.hadoop.chapter06.SequenceFileCreator input/2008.csv 2008_seq_file
```

```
810 2008,1,3,4,2003,1955,2211,2225,WN,335,N712SW,128,150,116,-14,8,IAD,TPA,810,4,8,0,,0,NA,NA,NA,NA,NA,
810 2008,1,3,4,754,735,1002,1000,WN,3231,N772SW,128,145,113,2,19,IAD,TPA,810,5,10,0,,0,NA,NA,NA,NA,NA,
515 2008,1,3,4,628,620,804,750,WN,448,N428WN,96,90,76,14,8,IND,BWI,515,3,17,0,,0,NA,NA,NA,NA,NA,
515 2008,1,3,4,926,930,1054,1100,WN,1746,N612SW,88,90,78,-6,-4,IND,BWI,515,3,7,0,,0,NA,NA,NA,NA,NA,
515 2008,1,3,4,1829,1755,1959,1925,WN,3920,N464WN,90,90,77,34,34,IND,BWI,515,3,10,0,,0,2,0,0,0,32
688 2008,1,3,4,1940,1915,2121,2110,WN,378,N726SW,101,115,87,11,25,IND,JAX,688,4,10,0,,0,NA,NA,NA,NA,NA,
1591 2008,1,3,4,1937,1830,2037,1940,WN,509,N763SW,240,250,230,57,67,IND,LAS,1591,3,7,0,,0,10,0,0,0,47
1591 2008,1,3,4,1039,1040,1132,1150,WN,535,N428WN,233,250,219,-18,-1,IND,LAS,1591,7,7,0,,0,NA,NA,NA,NA,NA,
```



# 부분 정렬(Partial Sort)

## 2. 맵파일 생성

- 맵파일 : 키값을 검색할 수 있게 색인과 함께 정렬된 시퀀스 파일
- 물리적으로 색인이 저장된 **index**파일과 데이터 내용이 저장돼 있는 **data** 파일로 구성됨
- 앞에서 생성된 시퀀스 파일을 변환해 맵파일로 생성할 수 있음
- 운항 거리를 기준으로 정렬되어 맵파일이 출력됨

```
medinfo@localhost:~/hadoop$ ./bin/hadoop jar wikibooks-hadoop-examples.jar wikibooks.hadoop.chapter06.MapFileCreator 2008_seq_file2 2008_map_file
```

Name	Type	Size	Replication	Block Size
<a href="#">data</a>	file	96.44 MB	1	64 MB
<a href="#">index</a>	file	6.02 KB	1	64 MB

```
0      2008,8,12,2,1847,1857,1922,1932,YV,1962,N37342,35,35,22,
11     2008,5,15,4,2037,1800,2125,1900,OH,4988,N806CA,48,60,31,145,157,JFK,LGA,11,10,7,0,,0,145,0,0,0,0
17     2008,3,8,6,NA,1105,NA,1128,AA,1368,,NA,23,NA,NA,NA,EWR,LGA,17,NA,NA,1,B,0,NA,NA,NA,NA,NA
21     2008,5,9,5,48,100,117,130,AA,588,N061AA,29,30,11,-13,-12,MIA,FLL,21,6,12,0,,0,NA,NA,NA,NA,NA
21     2008,2,8,5,NA,1910,NA,1931,AA,1668,,NA,21,NA,NA,NA,FLL,MIA,21,NA,NA,1,A,0,NA,NA,NA,NA,NA
24     2008,3,12,3,955,931,1021,948,9E,2009,91619E,26,17,10,33,24,IAH,HOU,24,7,9,0,,0,0,0,9,0,24
24     2008,1,2,3,1245,1025,1340,1125,OH,5610,N806CA,55,60,11,135,140,IAD,DCA,24,5,39,0,,0,135,0,0,0,0
28     2008,2,22,5,2046,2050,NA,2156,00,3698,N298SW,NA,66,NA,NA,-4,SLC,OGD,28,NA,12,0,,1,NA,NA,NA,NA,NA
30     2008,1,8,2,816,805,907,855,B6,9002,N236JB,51,50,19,12,11,JFK,HPN,30,5,27,0,,0,NA,NA,NA,NA,NA
30     2008,1,6,7,2226,2200,2301,2240,CO,348,N56859,35,40,11,21,26,SJC,SFO,30,7,17,0,,0,0,0,0,0,21
```

# 부분 정렬(Partial Sort)

---

## 3. 검색 프로그램 구현

- 맵파일에서 우리가 원하는 키에 해당하는 값을 검색하는 방법
- 검색의 '키'는 '파티셔너'임
- 검색하고자 하는 키가 속하는 파티션 번호를 조회한 후, 파티션 번호로 맵파일에 접근해 데이터를 검색

```
medinfo@localhost:~/hadoop$ ./bin/hadoop jar wikibooks-hadoop-examples.jar wikibooks.hadoop.chapter06.SearchValueList 2008_map_file2 100 | head -10
```

```
2008,1,30,3,823,825,855,857,YV,1002,N654BR,32,32,21,-2,-2,OGG,HNL,100,5,6,0,,0,NA,NA,NA,NA,NA,NA
2008,1,30,3,623,625,700,657,YV,1001,N654BR,37,32,25,3,-2,OGG,HNL,100,5,7,0,,0,NA,NA,NA,NA,NA,NA
2008,1,8,2,725,730,800,805,YV,1010,N693BR,35,35,21,-5,-5,HNL,OGG,100,4,10,0,,0,NA,NA,NA,NA,NA,NA
2008,1,8,2,932,935,1015,1010,YV,1011,N651BR,43,35,21,5,-3,HNL,OGG,100,7,15,0,,0,NA,NA,NA,NA,NA,NA
2008,1,8,2,1735,1735,1812,1810,YV,1017,N651BR,37,35,22,2,0,HNL,OGG,100,5,10,0,,0,NA,NA,NA,NA,NA,NA
2008,1,8,2,1945,1935,2024,2010,YV,1018,N646BR,39,35,22,14,10,HNL,OGG,100,4,13,0,,0,NA,NA,NA,NA,NA,NA
2008,1,8,2,2149,2200,2232,2235,YV,1019,N693BR,43,35,29,-3,-11,HNL,OGG,100,2,12,0,,0,NA,NA,NA,NA,NA,NA
2008,1,8,2,1552,1545,1631,1620,YV,1032,N651BR,39,35,21,11,7,HNL,OGG,100,5,13,0,,0,NA,NA,NA,NA,NA,NA
2008,1,8,2,1350,1350,1430,1425,YV,1037,N693BR,40,35,23,5,0,HNL,OGG,100,5,12,0,,0,NA,NA,NA,NA,NA,NA
2008,1,8,2,623,625,700,657,YV,1001,N693BR,37,32,22,3,-2,OGG,HNL,100,6,9,0,,0,NA,NA,NA,NA,NA,NA
```

# 전체 정렬(Total Sort)

---

- 하나의 파티션만 사용해서 쉽게 해결 가능
    - 모든 맵리듀스 잡은 입력데이터의 키를 기준으로 정렬하므로 단일 파티션에 모든 데이터가 정렬되어 출력 됨
    - 데이터가 클 때 하나의 파티션만 사용해서 정렬하면 리듀스 태스크가 실행되는 데이터노드만 부하가 집중 될 것임
  - 전체 정렬 (분산 처리의 장점을 살리는 방법)
    - 입력 데이터를 샘플링해서 데이터의 분포도를 조사
    - 데이터의 분포도에 맞게 파티션 정보를 미리 생성
    - 미리 생성한 파티션 정보에 맞게 출력 데이터를 생성
    - 각 출력 데이터를 병합
  - org.apache.hadoop.mapred.lib.TotalOrderPartitioner, InputSampler 제공
    - TotalOrderPartitioner로 파티션 개수와 파티션에 저장할 데이터 범위 설정 가능
    - InputSampler로 파티션에 키를 고르게 배분, 입력 데이터에서 특정 개수의 데이터를 추출해서 키와 데이터 건수를 샘플링 함 (즉, 데이터의 분포도를 작성)
    - TotalOrderPartitioner는 이러한 샘플링 결과를 기준으로 파티션 생성
    - 맵리듀스 잡은 생성된 파티션에 출력 데이터를 생성
-

# reference

---

[1] 김철연, 맵리듀스 환경에서 웨이블릿 시놉시스 생성 알고리즘, 정보과학회논문지, 제39권 26호, 2012.

<http://www.slideshare.net/KeeyongHan>

[http://book.naver.com/bookdb/book\\_detail.nhn?bid=7141828](http://book.naver.com/bookdb/book_detail.nhn?bid=7141828)

<http://cafe.naver.com/cloudbigdata>

---

---

# 데이터 분석 Query 언어

---

2015

---

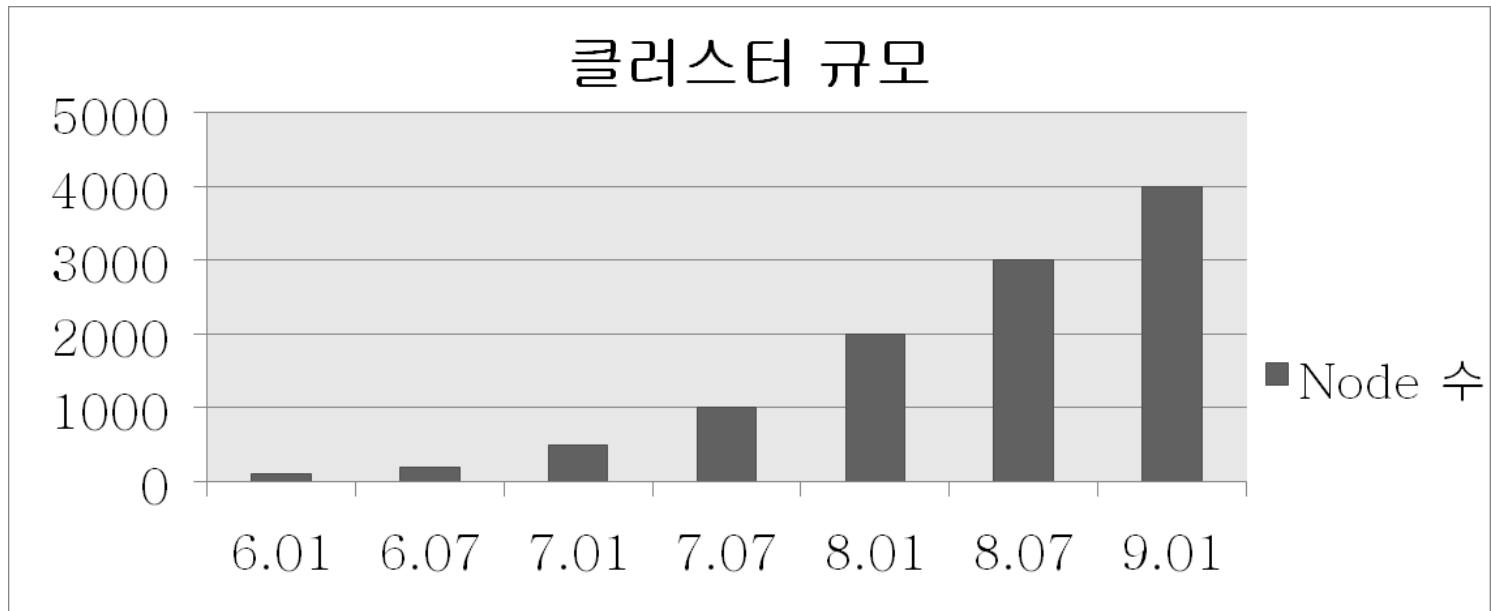
# Content

---

- 성능과 생산성
- 프레임워크별 동향
- HIVE
- 실습

# Hadoop 성능과 생산성

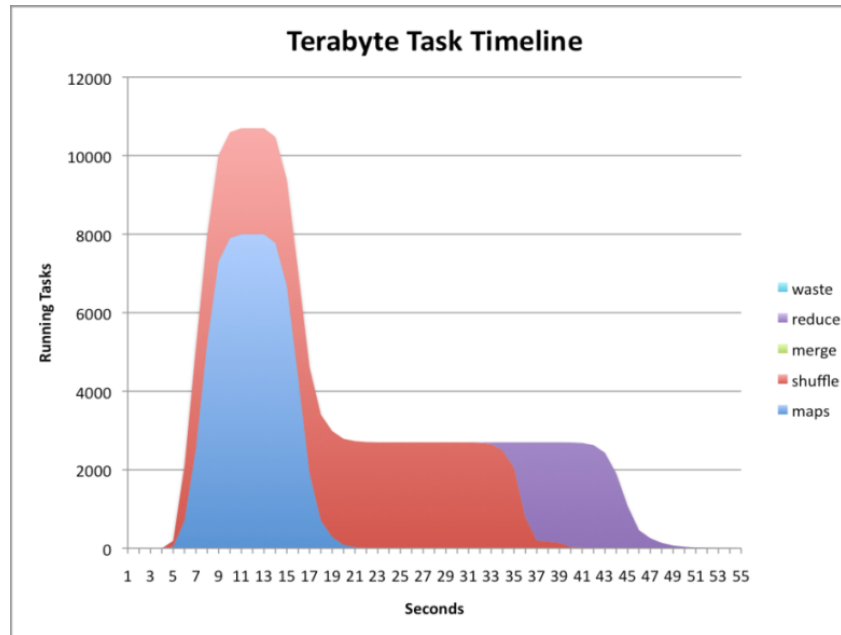
- Yahoo 단일 클러스터
  - 하둡은 SQL언어를 사용할 수 있음 (Hive에서 HiveQL이라는 쿼리 언어 제공)
- 4000 Nodes, 2Quad Cores
  - 4 x 1TB SATA Disk per Node
  - 8GB RAM, 1GB Link(8GB Up), 40Nodes Rack



# 성능과 생산성

---

- Hadoop 성능
  - Jim Grey's Sort 벤치 마크 테스트
    - 2008년: 1TB 209초 신기록
    - 2009년: 1TB 62초, 1PB 16.25시간
    - 3800대 클러스터, 서버별 8Cores & 8GB RAM

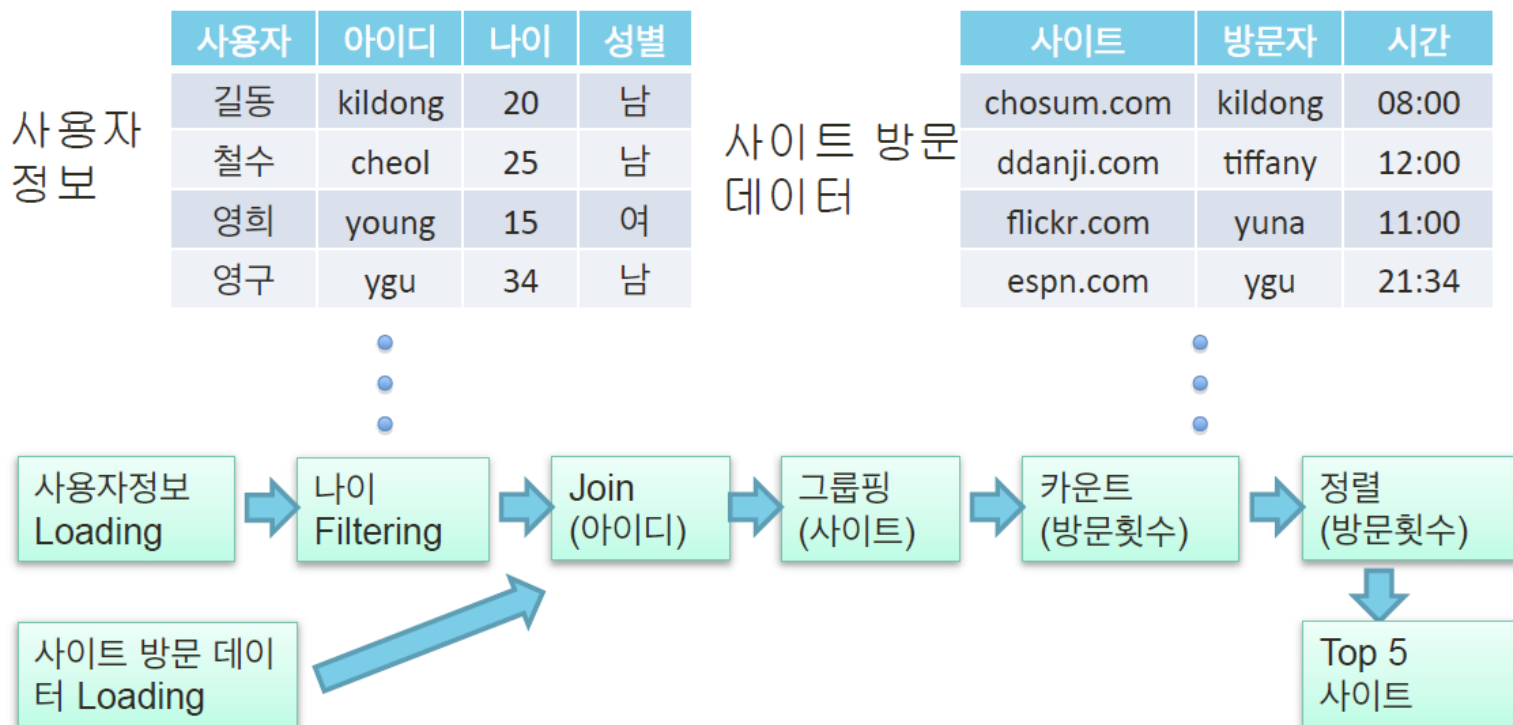




# 성능과 생산성

- Hadoop 생산성

- 18 ~ 25세 연령대의 사용자가 가장 많이 방문하는 사이트 5개를 찾아라



- 직접 MapReduce 프로그램을 코딩 할 경우

[illegible]

# 성과와 생산성

---

- Hadoop 생산성
  - 사용자 보다는 개발자 중심
  - 중복된 코딩 및 노력의 반복
  - 실행을 위한 버전 관리, 환경 설정의 복잡성
- 개발 생산성 개선 필요

# 성능과 생산성

---

- 고차원 병렬 처리 언어
  - 쉬운 MapReduce 를 위한 병렬 처리 언어
  - Pig by Yahoo
  - Hive by FaceBook
- Top 5 사이트 찾기를 병렬처리 언어로

```
Users = load 'users' as (name, age);  
Fltrd = filter Users by age >= 18 and age <= 25;  
Pages = load 'pages' as (user, url);  
Jnd = join Fltrd by name, Pages by user;  
Grpd = group Jnd by url;  
Smmd = foreach Grpd generate group, COUNT(Jnd) as clicks;  
Srtd = order Smmd by clicks desc;  
Top5 = limit Srtd 5;  
store Top5 into 'top5sites';
```

# 프레임워크별 동향

---

- Google Sawzall
  - 절차형 프로그래밍 언어
  - Google 에서 가장 많이 사용되는 언어
  - 단일 클러스터에서만 월간 3PB 데이터 처리(2005년), 일 1000개 이상의 작업(220대)
  - Pig, Hive 등의 Role Model
- 아파치 Pig
  - 데이터 처리를 위한 고차원 언어
  - 아파치 Top-Level 프로젝트
  - Yahoo 내 Hadoop 작업의 30%
  - 2007년 배포 이후 2~10배 성능 개선
  - Native 대비 70 ~ 80 % 성능

# 프레임워크별 동향

---

- 아파치 Hive
  - 데이터 웨어하우징&분석 인프라
  - 아파치 Top-Level 프로젝트
  - 분석을 위한 SQL 기반 Query
  - 저장은 Hadoop DFS 사용
  - Query 내 Hadoop Streaming 연동
  - JDBC 지원
  - FaceBook 주도로 개발
  - Hive 클러스터 at FaceBook
    - 약 5,000여대의 하둡 클러스터
    - 분석 작업에 Hive 사용
    - 수 PB 데이터 압축 관리
    - 매일 수백TB 이상 데이터 처리

# 프레임워크별 동향

---

- 아파치 Hive
  - Hive 개발 동기
    - 벤더 데이터 웨어하우스 시스템 교체
      - 데이터 확장성 문제(최초 10GB -> 수십TB)
      - 라이선스 등 운영 비용 절감
      - 벤더 DBMS 에서 Hadoop 으로 교체 결정
    - 교체 과정에서 나타난 필요 기능을 개발
      - 사용자를 위한 CLI
      - 코딩 없이 Ad-hoc 질의를 할 수 있는 기능
      - 스키마 정보들의 관리

---

# HIVE

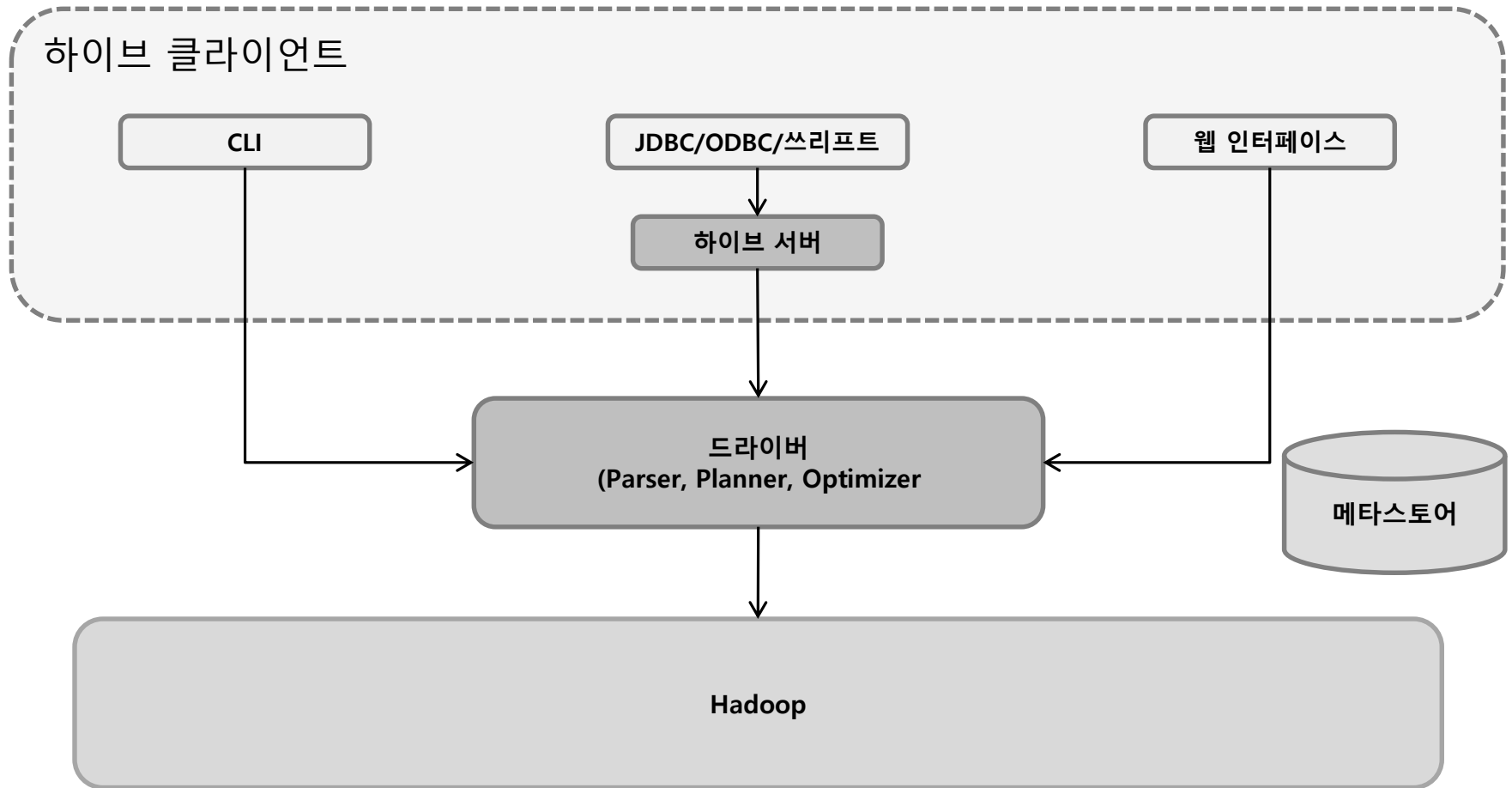
---

2015.03.09



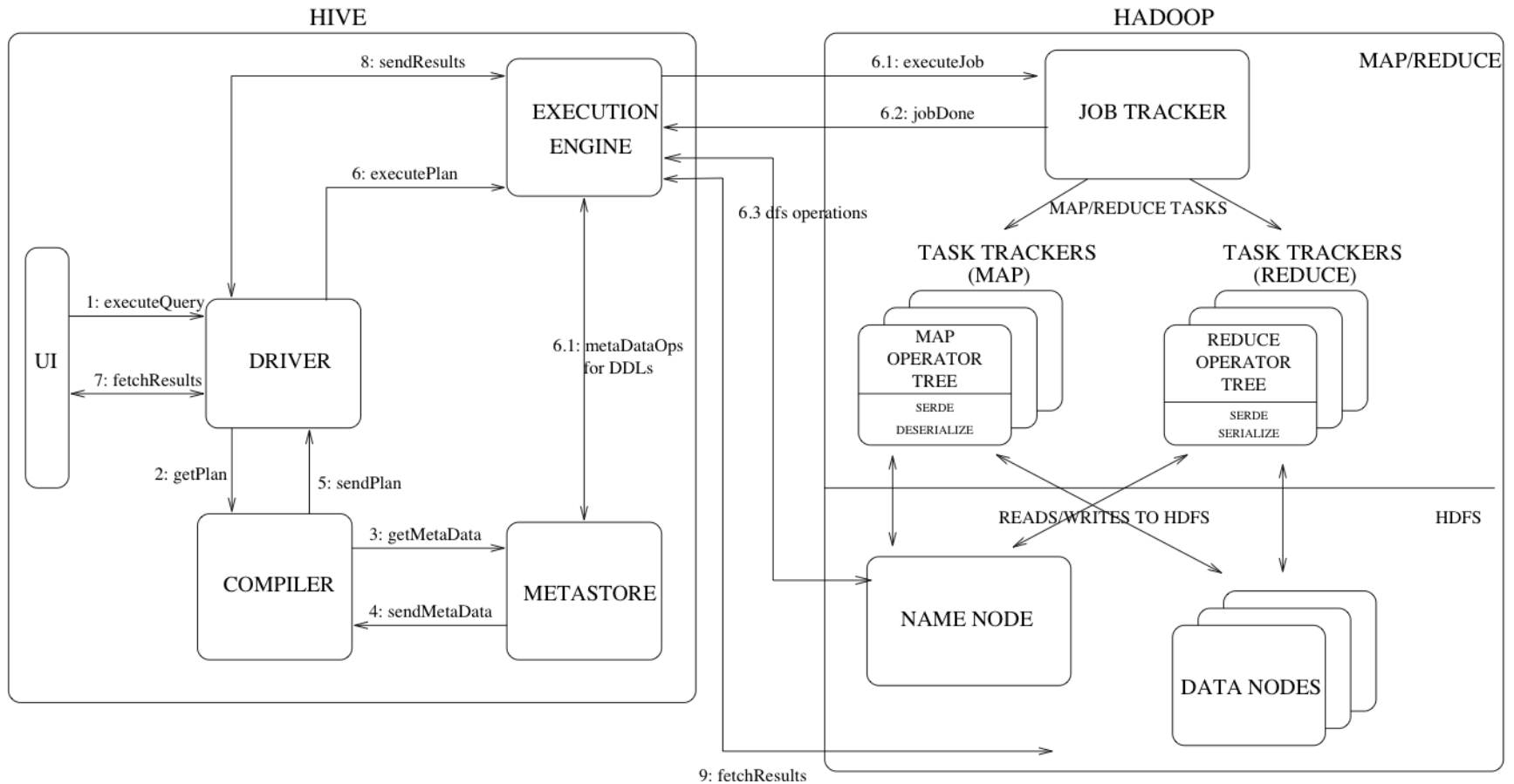
# HIVE

- Hive 구조



# HIVE

- Hive 내부



<https://cwiki.apache.org/confluence/display/Hive/Design>

# HIVE

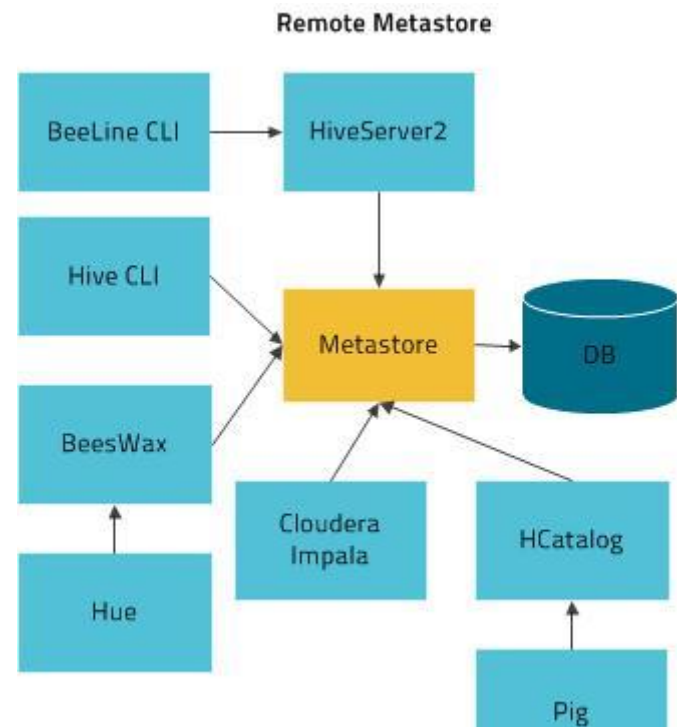
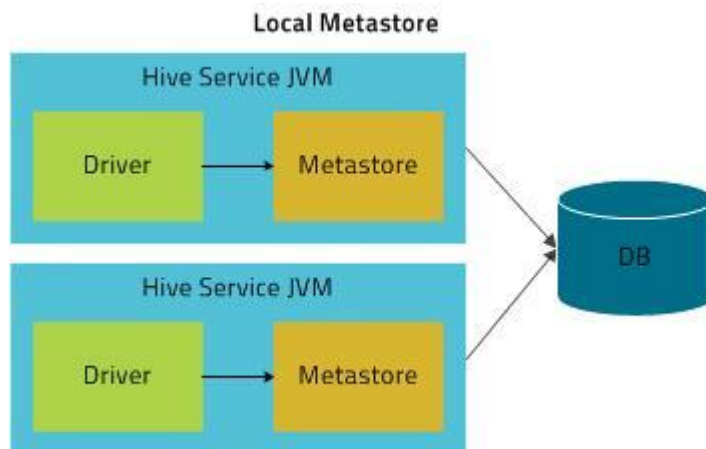
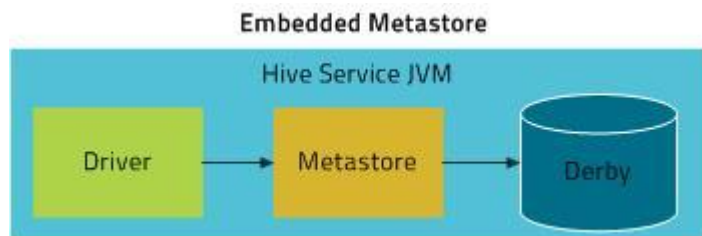
---

- Metastore
  - 데이터베이스: 테이블들의 네임스페이스
  - 테이블 속성 보관 (타입, 물리적인 배치)
  - 데이터 파티셔닝
  - JPOX 를 지원하는 Derby, MySQL 등의 다른 일반 RDMBS 를 사용 가능

# HIVE

- Metastore

- 데이터베이스: 테이블들의 네임스페이스
- 테이블 속성 보관 (타입, 물리적인 배치)
- 데이터 파티셔닝
- JPOX 를 지원하는 Derby, MySQL 등의 다른 일반 RDMBS 를 사용 가능



# HIVE

---

- Hive 언어 모델
    - DDL (Data Definition Language)
      - 테이블 생성, 삭제, 변경
      - 테이블 및 스키마 조회
    - DML (Data Manipulation Language)
      - 로컬 to DFS 업로드
      - Query 결과 to 테이블, 로컬, DFS
    - Query
      - Select, Group By, Sort By
      - Join, Union, Sub Queries, Sampling, Transform
-

# HIVE

---

- Hive 데이터 모델
  - 테이블
    - 컬럼 타입 지원(정수, 실수, 문자열, 날짜 등)
    - 리스트나 Map 같은 Collection 타입도 지원
  - 파티션
    - 예) 날짜 기간에 의한 파티션 등
  - Buckets
    - 범위 내에서 해쉬 파티션 지원(Sampling 및 최적화된 Join 가능)

# HIVE

---

- 물리적인 배치
  - HDFS 내의 warehouse 디렉토리
    - 예) **/user/hive/warehouse**
  - 테이블들은 warehouse 의 서브디렉토리
    - Partitions 과 buckets 은 테이블들의 서브디렉토리
  - 실제 데이터는 Flat File 들로 저장
    - 구분자로 분리된 텍스트 형식
    - SerDe 를 통해 임의의 포맷 지원 가능

# HIVE

---

- 물리적인 배치
  - HDFS 내의 warehouse 디렉토리
    - 예) **/user/hive/warehouse**
  - 테이블들은 warehouse 의 서브디렉토리
    - Partitions 과 buckets 은 테이블들의 서브디렉토리
  - 실제 데이터는 Flat File 들로 저장
    - 구분자로 분리된 텍스트 형식
    - SerDe 를 통해 임의의 포맷 지원 가능
      - SEQUENCEFILE, RCFILE, ORC, PARQUET



# HIVE

---

- 파일 포맷

항목	텍스트 파일	시퀀스파일	RC파일	ORC파일	파케이
저장기반	로우 기반	로우 기반	칼럼 기반	칼럼 기반	칼럼 기반
압축	파일 압축	레코드/ 블록 압축	블록 압축	블록 압축	블록 압축
스플릿 지원	지원	지원	지원	지원	지원
압축 적용 시 스플릿 지원	미지원	지원	지원	지원	지원
하이브 키워드	TEXTFILE	SEQUENCEFILE	RCFILE	ORCFIELD	PARQUET

# HIVE

---

- 파일 포맷 사용
  - 시퀀스파일

```
CREATE TABLE table_ex()
```

```
ROW FORMAT SERDE
```

```
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerde'
```

```
STORED AS SEQUENCEFILE;
```

```
CREATE TABLE table_ex()
```

```
ROW FORMAT SERDE
```

```
'org.apache.hadoop.hive.serde2.lazybinary.LazyBinarySerde'
```

```
STORED AS SEQUENCEFILE;
```

---

# HIVE

---

- 파일 포맷 사용

- RC파일

```
CREATE TABLE table_ex()
```

```
ROW FORMAT SERDE
```

```
'org.apache.hadoop.hive.serde2.columnar.ColumnarSerde'
```

```
STORED AS RCFILE;
```

```
CREATE TABLE table_ex()
```

```
ROW FORMAT SERDE
```

```
'org.apache.hadoop.hive.serde2.columnar.LazybinaryColumnarSerde'
```

```
STORED AS RCFILE;
```

---

# HIVE

---

- 파일 포맷 사용

- ORC 파일

```
CREATE TABLE table_ex()
```

```
STORED AS ORC tblproperties() ;
```

# HIVE

---

- 파일 포맷 사용
  - 파케이

```
CREATE TABLE table_parquet(  
    id int,  
    str string,  
    mp MAP<STRING,STRING>,  
    lst ARRAY<STRING>,  
    strct STRUCT<A:STRING,B:STRING>)  
PARTITIONED BY (part string)  
STORED AS PARQUET;
```

---

---

# HIVE 실습

---

2015

---

# HIVE 실습

---

- Hive 환경 구성

- Hive 다운로드 <http://hive.apache.org> 접속 후 다운로드
- 압축을 풀고 hadoop 심볼릭 링크를 생성
  - \$ tar -zxvf apache-hive
  - \$ ln -s apache-hive-1.0.0 hive
- .bash\_profile에 HIVE\_HOME과 PATH 추가

```
#JAVA
export JAVA_HOME=/usr/local/java
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_HOME=/home/hadoop/hadoop
export HIVE_HOME=/home/hadoop/hive
export PATH=$PATH:$HADOOP_HOME/bin:$HIVE_HOME/bin
```

# HIVE 실습

---

- Hive metastore

- MySQL 설치

```
[hadoop@hadoop01 ~]$ su - root
```

```
[root@hadoop01 ~]$yum install mysql-server
```

```
[root@hadoop01 ~]$ mysqladmin -u root password hadoop
```

```
[root@hadoop01 ~]$ service mysqld start
```

```
[root@hadoop01 ~]$ chkconfig mysqld on
```

```
[hadoop@hadoop01 ~]$ wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.35.tar.gz
```

```
[hadoop@hadoop01 ~]$ tar -zxvf ./mysql-connector-java-5.1.35.tar.gz
```

```
[hadoop@hadoop01 ~]$ cp ./mysql-connector-java-5.1.35/mysql-connector-java-5.1.35-bin.jar  
/home/hadoop/hive/lib/
```

```
[hadoop@hadoop01 ~]$mysql -u root -p
```

```
mysql> grant all privileges on *.* to hive@localhost identified by 'hive' with grant option;
```

```
mysql> flush privileges;
```

```
mysql> grant all privileges on *.* to hive@'%' identified by 'hive' with grant option;
```

```
mysql> flush privileges;
```

---



# HIVE 실습

---

- Hive-site

```
<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://hadoop01/hive?createDatabaseIfNotExist=true</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hive</value>
  </property>
  <property>
    <name>hive.metastore.uris</name>
    <value>thrift://hadoop01:9083</value>
  </property>
  <property>
    <name>hive.hwi.listen.host</name>
    <value>0.0.0.0</value>
  </property>
  <property>
    <name>hive.hwi.listen.port</name>
    <value>9999</value>
  </property>
  <property>
    <name>hive.hwi.war.file</name>
    <value>lib/hive-hwi-1.0.0.war</value>
  </property>
</configuration>
```

---

# HIVE 실습

---

- Hive 실행

```
[hadoop@hadoop01 hive]$ ./bin/hive --service metastore
```

```
[hadoop@hadoop01 hive]$ ./bin/hive --service hiveserver2
```

- hive 접속

```
[hadoop@hadoop01 hive]$ beeline
```

```
Beeline version 1.0.0 by Apache Hive
```

```
beeline>!connect jdbc:hive2://hadoop01:10000
```

```
0: jdbc:hive2://hadoop01:10000>
```

---

# HIVE 실습

---

- 테이블 생성

```
0: jdbc:hive2://hadoop01:10000>CREATE TABLE IF NOT EXISTS  
stocks(`exchange` STRING,  
symbol STRING,  
ymd STRING,  
price_open FLOAT,  
price_high FLOAT,  
price_low FLOAT,  
price_close FLOAT,  
volume INT,  
price_adj_close FLOAT)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';  
0: jdbc:hive2://hadoop01:10000>desc stocks ;
```

---

# HIVE 실습

---

- 데이터 업로드

```
0: jdbc:hive2://hadoop01:10000>
```

```
load data local inpath
```

```
'/home/hadoop/sample_data/NASDAQ/NASDAQ_daily_price*' overwrite  
into table stocks;
```

```
stocks;0: jdbc:hive2://hadoop01:10000>
```

```
load data local inpath
```

```
'/home/hadoop/sample_data/NYSE/NYSE_daily_prices*' into table stocks;
```

---

# HIVE 실습

---

- Simple query / Group by

- 데이터 갯수 확인

- 0: jdbc:hive2://hadoop01:10000>SELECT count(\*) FROM stocks;

- 뉴욕 거래소 및 나스닥 전체 업종 수

- 0: jdbc:hive2://hadoop01:10000>SELECT count(DISTINCT symbol) FROM stocks;

- 애플의 각 연도별 종가 평균

- 0: jdbc:hive2://hadoop01:10000>SELECT year(ymd), avg(price\_close)  
FROM stocks WHERE `exchange` = 'NASDAQ' AND symbol =  
'AAPL' GROUP BY year(ymd);

- 2010 장종가 평균 상위 5개 업체

- 0: jdbc:hive2://hadoop01:10000>select symbol, avg(price\_open) as  
openprice from stocks where year(ymd) = '2010' group by symbol sort  
by openprice DESC limit 5

# HIVE 실습

---

- View Table

- View Table은 자주 사용되는 쿼리를 일종의 로지컬 테이블처럼 사용하는것.
- 복잡한 쿼리를 단순하게 함
- RAW table로 직접 접근하는것이 아니라 제한을 두는 목적으로도 사용함
- 테이블의 모듈화에 적합

# HIVE 실습

---

- View Table

```
0: jdbc:hive2://hadoop01:10000>SELECT s2.year, s2.avg FROM (SELECT  
year(ymd) AS year, avg(price_close) AS avg FROM stocks WHERE  
`exchange` = 'NASDAQ' AND symbol = 'AAPL' GROUP BY year(ymd)) s2  
WHERE s2.avg > 50.0;
```

```
0: jdbc:hive2://hadoop01:10000>create view stockview AS SELECT  
year(ymd) AS year, avg(price_close) AS avg FROM stocks WHERE  
`exchange` = 'NASDAQ' AND symbol = 'AAPL' GROUP BY year(ymd);
```

```
0: jdbc:hive2://hadoop01:10000>select stockview.year, stockview.avg  
from stockview where stockview.avg > 50;
```

```
0: jdbc:hive2://hadoop01:10000>
```

---

# HIVE 실습

---

- Partition

- partition 테이블 만들기

```
0: jdbc:hive2://hadoop01:10000>CREATE TABLE IF NOT EXISTS  
stocks_part (  
ymd STRING,  
price_open FLOAT,  
price_high FLOAT,  
price_low FLOAT,  
price_close FLOAT,  
volume INT,  
price_adj_close FLOAT)  
PARTITIONED by ( exchange STRING, symbol STRING)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

---



# HIVE 실습

- Partition
  - partition 테이블과 일반 테이블

col_name	data_type	comment
exchange	string	
symbol	string	
ymd	string	
price_open	float	
price_high	float	
price_low	float	
price_close	float	
volume	int	
price_adj_close	float	

```
0: jdbc:hive2://hadoop01:10000> desc stocks_part;
```

col_name	data_type	comment
ymd	string	
price_open	float	
price_high	float	
price_low	float	
price_close	float	
volume	int	
price_adj_close	float	
exchange	string	
symbol	string	
	NULL	NULL
# Partition Information	NULL	NULL
# col_name	data_type	comment
	NULL	NULL
exchange	string	
symbol	string	

15 rows selected (0.064 seconds)

# HIVE 실습

---

- Partition

- partition 테이블에 데이터 로드

```
0: jdbc:hive2://hadoop01:10000>INSERT OVERWRITE TABLE
stocks_part PARTITION(`exchange`='NASDAQ', symbol
='AAPL') SELECT
ymd,price_open,price_high,price_low,price_close,volume
int,price_adj_close
FROM stocks WHERE `exchange` = 'NASDAQ' and
symbol='AAPL';
```

- 확인

```
0: jdbc:hive2://hadoop01:10000>show partitions
stocks_part;
```

```
0: jdbc:hive2://hadoop01:10000> show partitions stocks_part;
+-----+
|          partition          |
+-----+-----+
| exchange=NASDAQ/symbol=AAPL |
+-----+-----+
```

# HIVE 실습

---

- 다이나믹 Partition

- 설정(기본값)

- hive.exec.dynamic.partition.mode=strict;

- hive.exec.dynamic.partition = false;

- hive.exec.max.dynamic.partitions.pernode=100;

- hive.exec.max.dynamic.partitions=1000;

- hive.exec.max.created.files=100000;

- partition 테이블에 데이터 로드

- 0: jdbc:hive2://hadoop01:10000>INSERT OVERWRITE TABLE stocks\_part  
PARTITION(`exchange`, symbol) SELECT  
ymd,price\_open,price\_high,price\_low,price\_close,volume  
int,price\_adj\_close,`exchange`,symbol  
FROM stocks;

---

# HIVE 실습

---

- 색인
  - 데이터 베이스의 특정 컬럼에 대해 검색가능한 형태인 자료구조를 만드는것
  - I/O의 범위를 전체적으로 줄여주는 역할을 한다
  - 색인 자체가 기본적인 정렬이 되어 있기 때문에 검색 시간도 줄어든다

# HIVE 실습

---

- 색인

- table에 색인 생성

- symbol컬럼에 색인 생성

0: jdbc:hive2://hadoop01:10000>CREATE INDEX simple\_index ON TABLE stocks (symbol) AS

'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;

- index 확인

0: jdbc:hive2://hadoop01:10000>SHOW FORMATTED INDEX ON stocks;

- query수행

0: jdbc:hive2://hadoop01:10000>SELECT s2.year, s2.avg FROM (SELECT year(ymd) AS year, avg(price\_close) AS avg FROM stocks WHERE symbol = 'AAPL' GROUP BY year(ymd)) s2 WHERE s2.avg > 50.0;

---

# HIVE 실습

---

- 색인

- 인덱스 생성

0: jdbc:hive2://hadoop01:10000>alter index simple\_index on stocks rebuild;

- 필요한 파일만 다시 로드

0: jdbc:hive2://hadoop01:10000>INSERT OVERWRITE DIRECTORY  
"/tmp/index\_result" SELECT `\_bucketname` , `\_offsets` FROM  
default\_stocks\_simple\_index\_\_ where symbol='AAPL';

- 기존 테이블과 연결

0: jdbc:hive2://hadoop01:10000>SET hive.index.compact.file=/tmp/index\_result;

0: jdbc:hive2://hadoop01:10000>SET

hive.input.format=org.apache.hadoop.hive.ql.index.compact.HiveCompactIndexInputFormat;

- query수행

0: jdbc:hive2://hadoop01:10000>SELECT s2.year, s2.avg FROM (SELECT  
year(ymd) AS year, avg(price\_close) AS avg FROM stocks WHERE symbol =  
'AAPL' GROUP BY year(ymd)) s2 WHERE s2.avg > 50.0;

# HIVE 실습

---

- UDF

- Before Writing UDF, internal Function is also UDF

0: jdbc:hive2://hadoop01:10000> show functions;

- 함수의 정의를 보려면 describe 사용

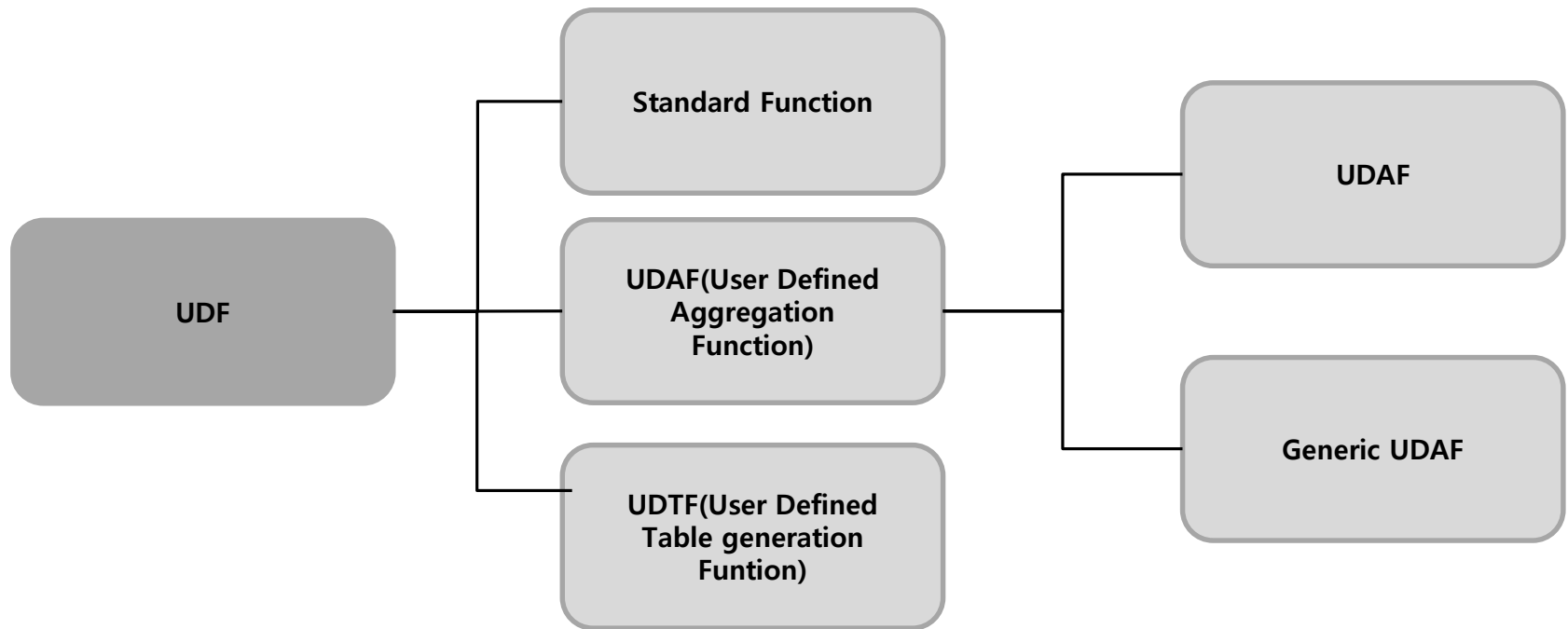
0: jdbc:hive2://hadoop01:10000> describe function abs;

```
0: jdbc:hive2://hadoop01:10000> describe function abs;
+-----+-----+
|                tab_name                |
+-----+-----+
| abs(x) - returns the absolute value of x |
+-----+-----+
1 row selected (0.021 seconds)
```

# HIVE 실습

---

- UDF 종류





# HIVE 실습

---

- UDF 종류
  - standard function (표준함수)
    - 하나의 로우 (row) 나 다수의 컬럼으로 부터 데이터를 받아서 처리
    - floor(), ucase(), concat() 과 같은 간단한 함수

# HIVE 실습

---

- UDF 종류
  - UDAF(User Defined Aggregation Function)
    - 사용자 집계(aggregation)함수
    - 하나이상의 row와 column으로 부터 데이터를 받아와서 계산을 수행하는 함수
    - `SELECT year(ymd), avg(price_close) FROM stocks WHERE exchange = 'NASDAQ' AND symbol = 'AAPL' GROUP BY year(ymd);`

# HIVE 실습

---

- UDF 종류
  - UDTF(User Defined TablegenerationFunction)
    - 하나의 변수나 컬럼을 입력받아 다수개의 테이블로(row)를 만드는 모든 함수
    - `SELECT explode(array(1,2,3)) AS element FROM src;`

# HIVE 실습

---

- UDF 예제

- 'APPL'을 'APPLE'로 변환하는 예제.

```
package hive.udf.sample;
import java.text.SimpleDateFormat;
import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
@Description(name="fullname",
value = "_FUNC_(name)- from the input string"+ " returns Fullname ",
extended = "Example _Func_('aaple') from src;\n")
public class Fullname extends UDF{
    public Fullname (){
        SimpleDateFormat df = new SimpleDateFormat("MM-dd-yyyy");
    }
    public String evaluate(String shortname)
    {
        if( "aapl".equals(shortname) || "AAPL".equals(shortname))
        {
            return "AAPLE";
        }
        else
            return shortname;
    }
}
```

---

# HIVE 실습

---

- UDF 예제

- UDF 등록

```
0: jdbc:hive2://hadoop01:10000> ADD JAR  
/home/hadoop/share/Fullname.jar;
```

```
0: jdbc:hive2://hadoop01:10000> CREATE TEMPORARY FUNCTION  
fullname AS 'hive.udf.sample.Fullname';
```

- UDF 사용

```
0: jdbc:hive2://hadoop01:10000> describe function fullname;
```

```
0: jdbc:hive2://hadoop01:10000> DESCRIBE FUNCTION EXTENDED  
fullname;
```

```
0: jdbc:hive2://hadoop01:10000> SELECT ymd,fullname(symbol) FROM  
Stocks where symbol='AAPL';
```

---

---

# Flume

---

2015

---

# Content

---

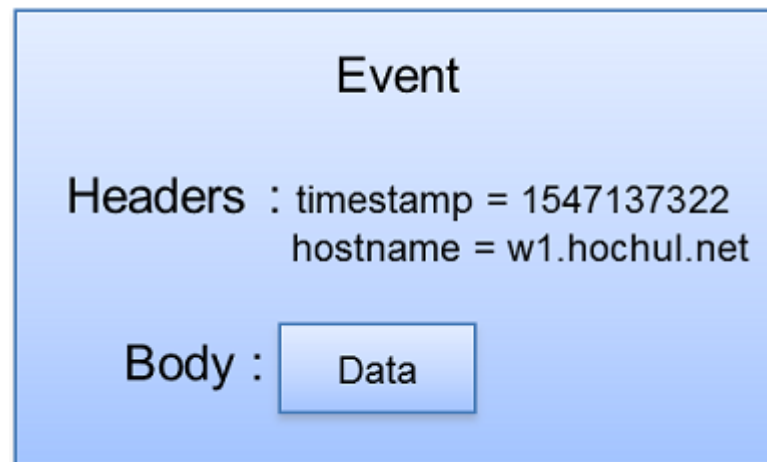
- Introduction
- 실습

# Flume

---

- Event

- Flume을 통해 전달되어지는 데이터의 기본 payload를 event라 부른다.
- Event는 0이상의 Header와 body영역으로 나뉜다. (Byte payload + set of string headers)
- Header는 key/value 형태이며, 라우팅을 결정하거나, 구조화된 정보(예를 들어, 이벤트가 발생한 서버의 호스트명, timestamp 등 추가 가능)를 운반할 때 활용된다





# Flume

---

- Agent



# Flume

---

- Agent
    - Source
      - avro : Avro 클라이언트에서 전송하는 이벤트를 입력으로 사용, Agent와 Agent를 연결해줄 때 유용
      - netcat : TCP로 라인 단위 수집
      - seq : 0부터 1씩 증가하는 EVENT 생성
      - exec : System Command를 수행하고 출력 내용을 수집
      - syslogtcp : System 로그를 입력으로 사용
      - spooldir : Spooling Directory 디렉토리에 새롭게 추가되는 파일을 데이터로 사용
      - thrift : Thrift 클라이언트에서 전송하는 이벤트를 입력으로 사용
      - jms : JMS 메시지 수집
      - Customize
-

# Flume

---

- Agent
    - Channel
      - memory : Source에서 받은 이벤트를 Memory에 가지고 있는 구조로, 간편하고 빠른 고성능(High Throughput)을 제공하지만 이벤트 유실 가능성이 있다. 즉, 프로세스가 비정상적으로 죽을 경우 데이터가 유실될 수 있다.
      - jdbc : JDBC로 저장
      - file : JDBC와 마찬가지로 속도는 Memory기반에 비해 느리지만, 프로세스가 비정상적으로 죽더라도 transactional하게 프로세스를 재시작하여 재처리하여 이벤트 유실이 없는 것이 장점이 있다.
      - Customize
-

# Flume

---

- Agent
    - Sink
      - null : 이벤트를 버림
      - logger : 테스트 또는 디버깅을 위한 로깅
      - avro : 다른 Avro 서버(Avro Source)로 이벤트 전달
      - hdfs : HDFS에 저장
      - hbase : HBase에 저장
      - elasticsearch : 이벤트를 변환해서 Elasticsearch에 저장
      - file\_roll : 로컬 파일에 저장
      - thrift : 다른 Thrift 서버(Thrift Source)로 이벤트 전달
      - Customize
-

# Flume

---

- Flow 설정

- To Define flow, link 'source' and 'sink' via 'channel'

- # list the sources, sinks and channels for the agent*

- `<Agent>.sources = <Source>`

- `<Agent>.sinks = <Sink>`

- `<Agent>.channels = <Channel1> <Channel2>`

- # set channel for source*

- `<Agent>.sources.<Source>.channels = <Channel1> <Channel2> ...`

- # set channel for sink*

- `<Agent>.sinks.<Sink>.channel = <Channel1>`

---

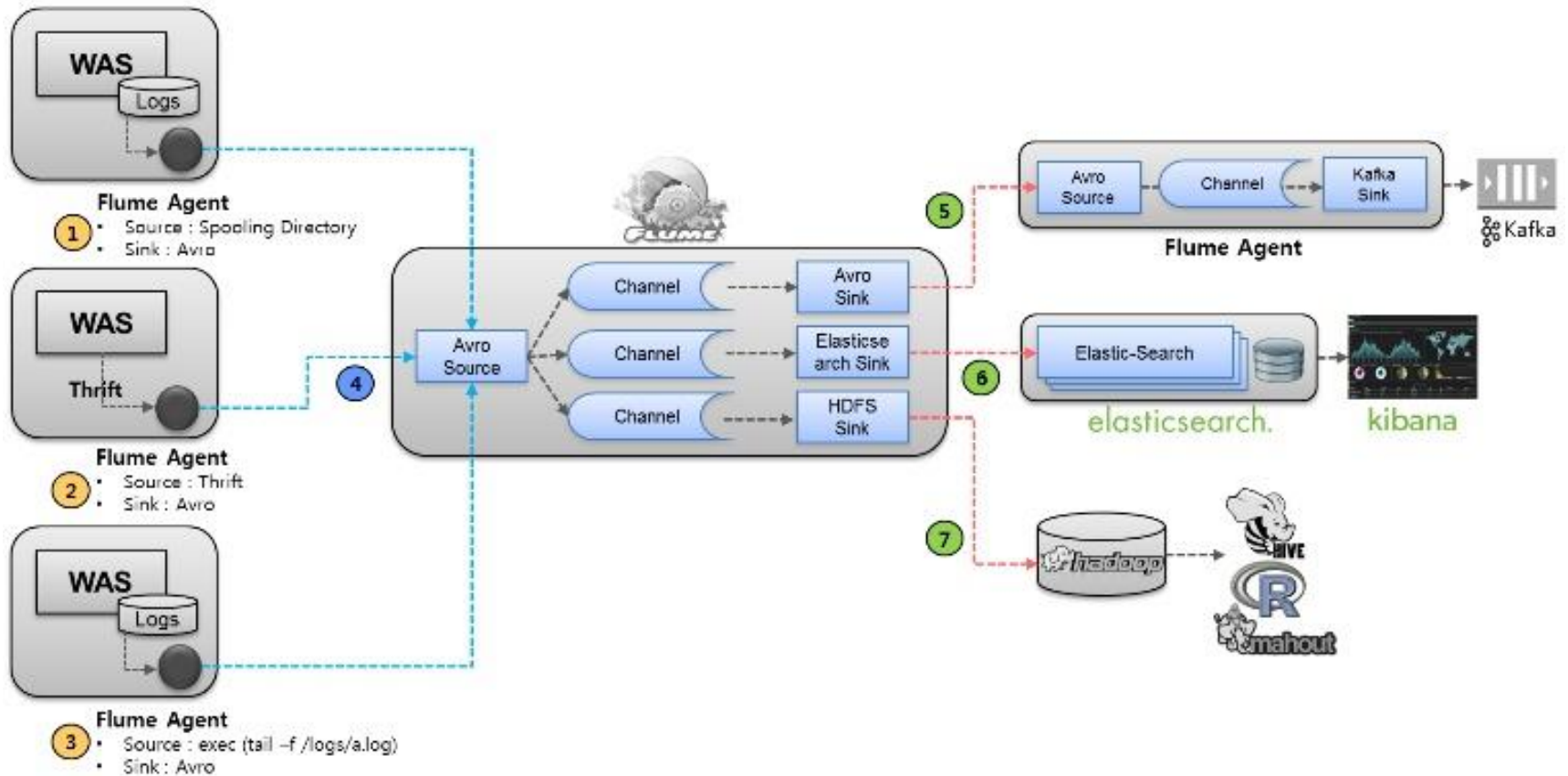
# Flume

---

- 활용
  - 하둡 클러스터 내 노드들의 로그 수집
  - 웹 서버, 메일 서버 같은 기존 시스템들의 로그 수집
  - 광고 네트워크 애플리케이션들의 노출 정보 수집
  - 시스템들의 성능 정보 수집
  - 기본적인 온라인 스트리밍 분석

# Flume

- 활용



# Flume

---

- 실습



---

# sqoop

---

2015

---

# Content

---

- Introduction
- 실습

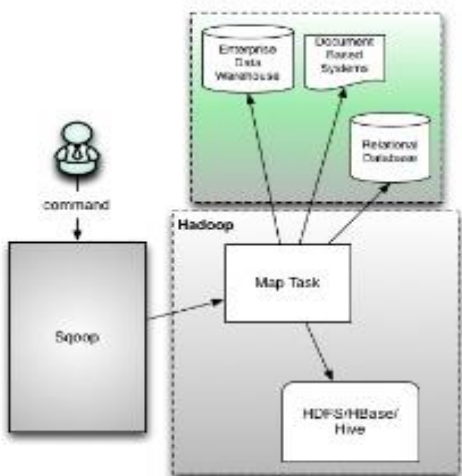
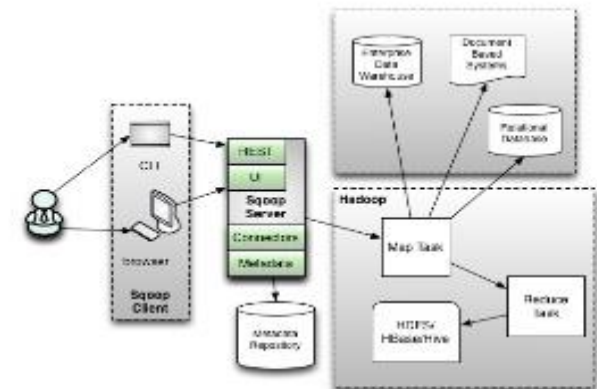
# sqoop

---

- Sql to Hadoop
  - RDBMS로 부터 수집/저장
    - 별도의 로그 수집 시스템 및 데이터 저장소가 마련되지 않아 Oracle, MySQL 등의 RDBMS에 로그를 저장하는 경우.
    - 로그 뿐 아니라, 메타성 데이터는 대부분 RDBMS에 저장되어 있는데, 이 RDBMS의 메타 데이터를 Hadoop, Hive 등으로 옮겨야 하는 경우
    - 분산 환경의 Hadoop, Hive 등에서 분석된 결과를 API 형태가 아닌 원격의 RDBMS로 전송할 경우
-

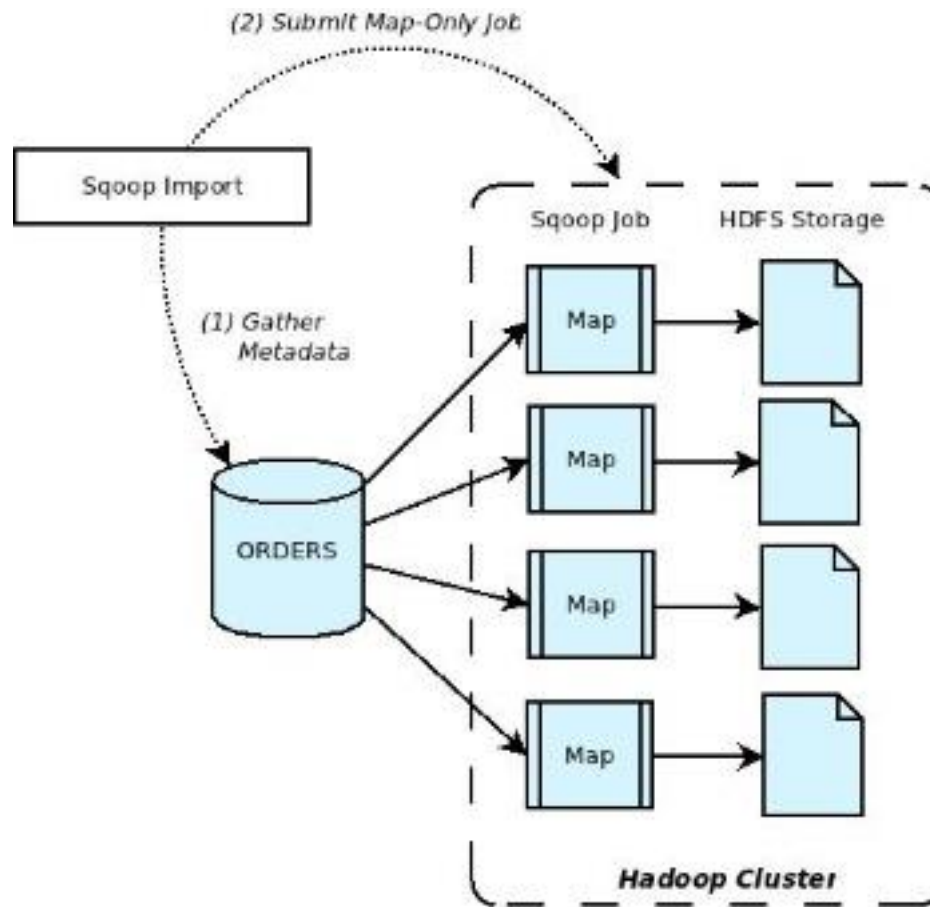
# sqoop

## • Sqoop 1 / Sqoop 2

	Sqoop1	Sqoop2
Overview	<ul style="list-style-type: none"> <li>Client-side Install</li> <li>Connectors 가 Local에 설치되어야 함</li> <li>JDBC Driver는 접속하는 Local마다 설치 필요</li> <li>CLI(Command Line Interface) 제공</li> </ul>	<ul style="list-style-type: none"> <li>Server-side install</li> <li>Connector가 필요한 서버 한 곳에만 설치하여 연결 가능</li> <li>즉, JDBC Driver가 한 곳만 설치하면 됨</li> <li>CLI 접속 외에도 Web 및 REST API를 통한 접속 가능</li> <li>Workflow Manager인 Apache Oozie와 Rest API를 활용하여 결합이 용이함</li> </ul>
Outline	 <p>The diagram for Sqoop 1 shows a user issuing a 'command' to a 'Sqoop' client. The 'Sqoop' client connects to a 'Map Task' within a 'Hadoop' cluster. The 'Map Task' interacts with 'Enterprise Data Warehouse', 'Document Based Systems', and 'Relational Database' on the left, and 'HDFS/HBase/Hive' on the right.</p>	 <p>The diagram for Sqoop 2 shows a user interacting with a 'Sqoop Client' which connects to a 'REST UI' and 'Sqoop Server'. The 'Sqoop Server' connects to a 'Hadoop' cluster. The 'Hadoop' cluster includes a 'Map Task' and a 'Reduce Task'. The 'Map Task' interacts with 'Enterprise Data Warehouse', 'Document Based Systems', and 'Relational Database' on the left, and 'HDFS/HBase/Hive' on the right. A 'Metadata Repository' is also shown connected to the 'Sqoop Server'.</p>
Last Version	1.4.4	1.99.3

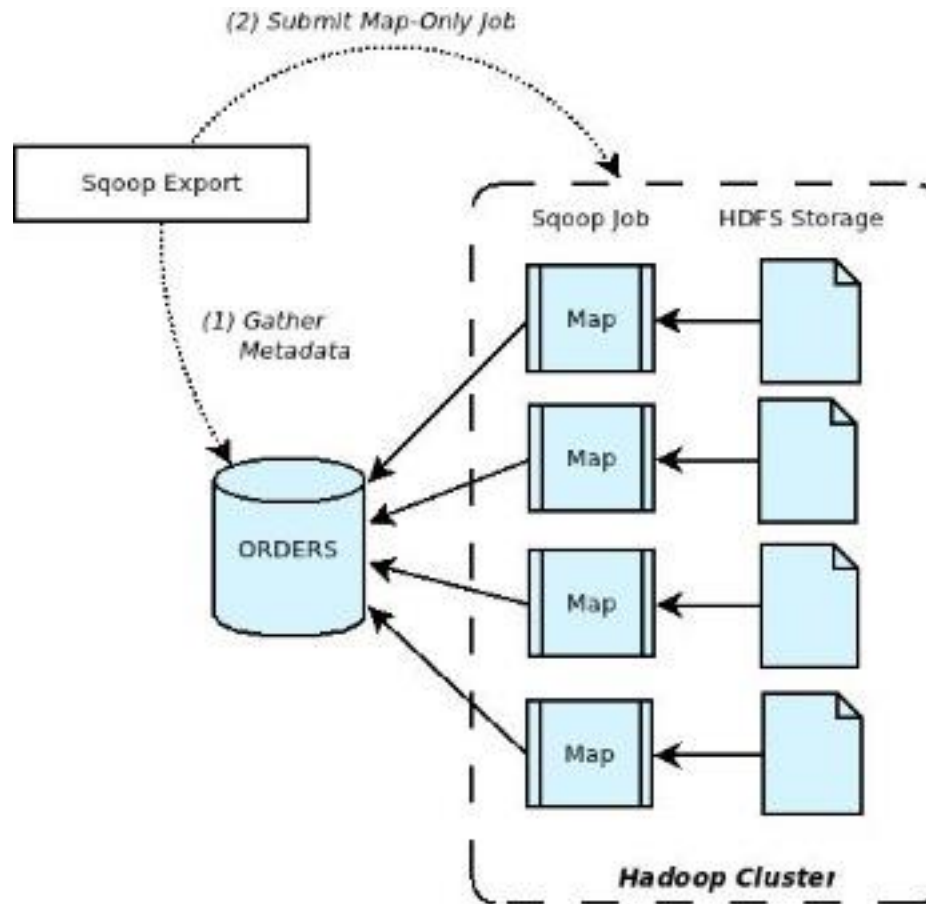
# sqoop

- import



# sqoop

- Export



# sqoop

---

- 실습

---

# Spark

---

2015

---



# Content

---

- Introduction

# Spark

---

- 빅데이터분석의 시초
    - GFS(Google File System) 논문 (2003)
      - 여러 컴퓨터를 연결하여 저장용량과 I/O성능을 scale
      - 이를 구현한 오픈소스 프로젝트인 Hadoop HDFS
    - MapReduce논문 (2004)
      - Map과 Reduce연산을 조합하여 클러스터에서 실행, 큰 데이터를 처리
      - 이를 구현한 오픈소스 프로젝트인 Hadoop MapReduce
    - Hive
      - MapReduce 코드를 짜는건 괴롭다
      - 쿼리로 MapReduce의 거의 모든 기능을 표현할 수 있다!
      - HDFS등에 있는 파일을 읽어들이어 쿼리로 분석 수행
      - HiveQL 을 작성하면 MapReduce 코드로 변환되어 실행
-

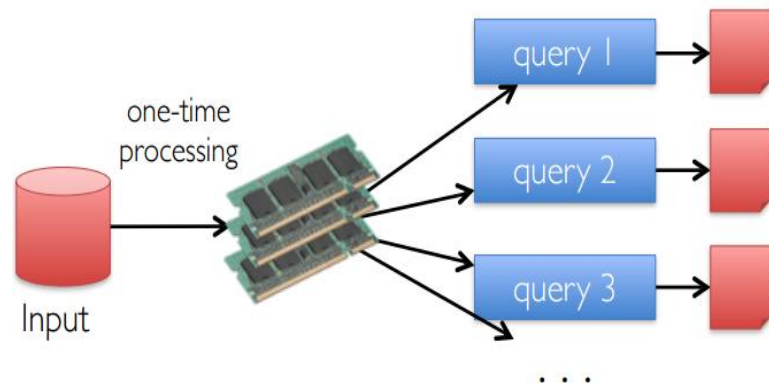
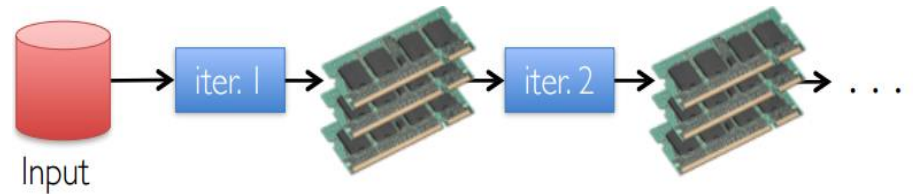
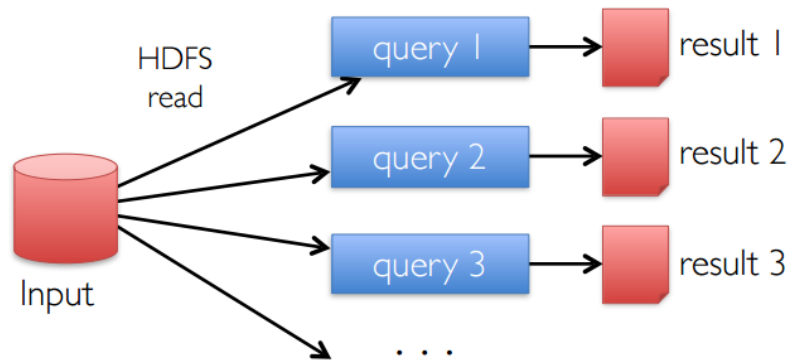
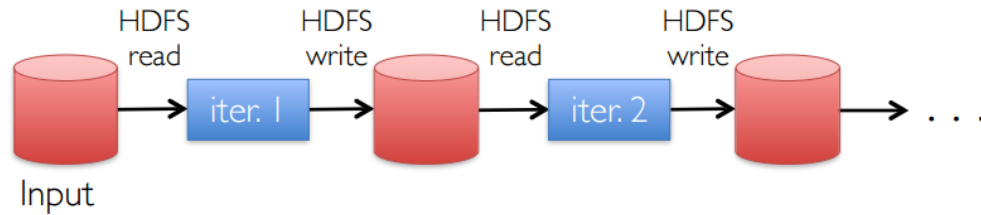
# Spark

---

- MapReduce / Hive 장단점
  - 장점
    - 빅데이터 시대를 열어준 선구적인 기술
    - 거대한 데이터를 안정적으로 처리
    - 많은 사람들이 사용 중
  - 단점
    - 오래된 기술이다 보니, 발전이 느리다
    - 불편한점이 많다
- MapReduce의 문제점
  - MapReduce는 Map의 입출력 및 Reduce의 입출력을 매번 HDFS에 쓰고, 읽는다
  - MapReduce코드는 작성하기 불편하다

# Spark

- M/R vs Spark



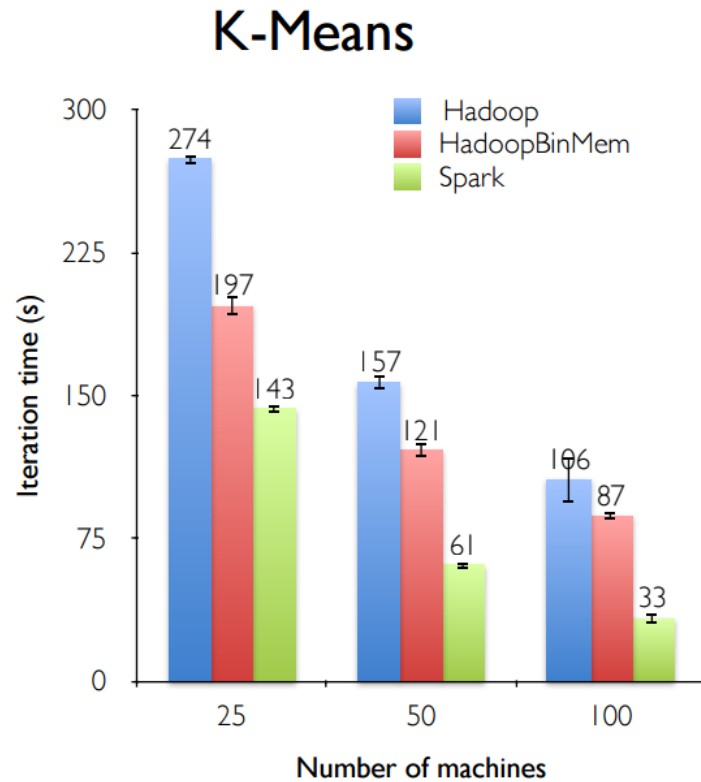
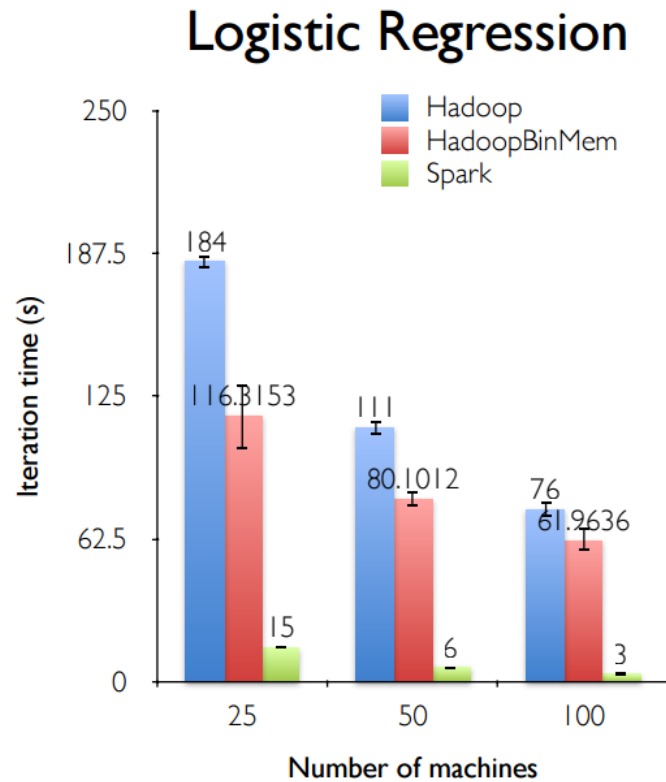
# Spark

---

- RDD
  - Resilient Distributed Dataset
    - 탄력적으로 분산된 데이터셋
    - 클러스터에 분산된 메모리를 활용하여 계산되는 List
    - 데이터를 어떻게 구해낼지를 표현하는 Transformation 을 기술한 Lineage(계보)를 interactive하게 만들어 낸 후, Action을 통해 lazy하게 값을 구해냄
    - 클러스터 중 일부의 고장 등으로 작업이 중간에 실패하더라도, Lineage를 통해 데이터를 복구

# Spark

- 성능



# Spark

---

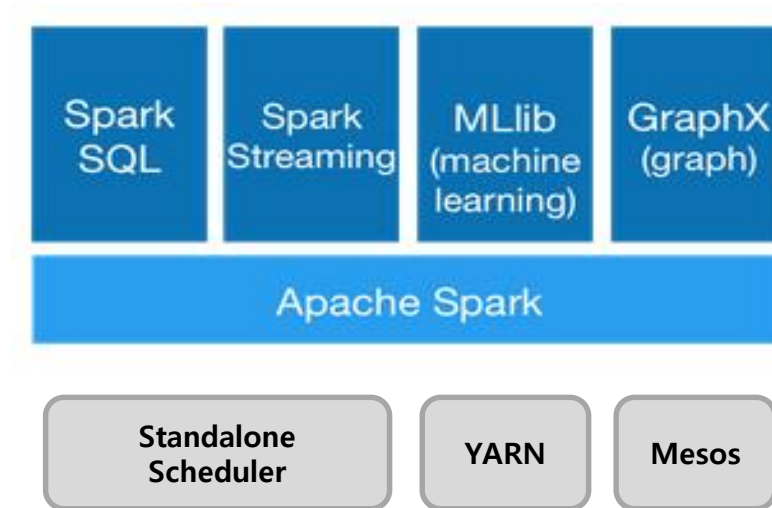
- Fast

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
<b>Sort rate</b>	<b>1.42 TB/min</b>	<b>4.27 TB/min</b>	<b>4.27 TB/min</b>
<b>Sort rate/node</b>	<b>0.67 GB/min</b>	<b>20.7 GB/min</b>	<b>22.5 GB/min</b>

# Spark

---

- Spark stack





# Spark

- Interface

- Scala

- 매우 간결한 표현이 가능한 언어
    - REPL(aka Shell) 제공, interactive하게 데이터를 다루는 것이 가능
    - Functional Programming이 가능하므로 MapReduce와 같은 functional한 개념을 표현하기에 적합함

```
public class WordCount {  
    public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {  
        private final static IntWritable one = new IntWritable(1);  
        private Text word = new Text();  
  
        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {  
            String line = value.toString();  
            StringTokenizer tokenizer = new StringTokenizer(line);  
            while (tokenizer.hasMoreTokens()) {  
                word.set(tokenizer.nextToken());  
                output.collect(word, one);  
            }  
        }  
    }  
  
    public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {  
        public void reduce(Text key, Iterable<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws  
        IOException {  
            int sum = 0;  
            while (values.hasNext()) {  
                sum += values.next().get();  
            }  
            output.collect(key, new IntWritable(sum));  
        }  
    }  
  
    public static void main(String[] args) throws Exception {  
        JobConf conf = new JobConf(WordCount.class);  
        conf.setJobName("wordcount");  
  
        conf.setOutputKeyClass(Text.class);  
        conf.setOutputValueClass(IntWritable.class);  
  
        conf.setMapperClass(Map.class);  
        conf.setCombinerClass(Reduce.class);  
        conf.setReducerClass(Reduce.class);  
  
        conf.setInputFormat(TextInputFormat.class);  
        conf.setOutputFormat(TextOutputFormat.class);  
  
        FileInputFormat.setInputPaths(conf, new Path(args[0]));  
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));  
  
        JobClient.runJob(conf);  
    }  
}
```

```
val file = spark.textFile("hdfs://...")  
val counts = file.flatMap(line => line.split(" "))  
                  .map(word => (word, 1))  
                  .reduceByKey(_ + _)  
counts.saveAsTextFile("hdfs://...")
```

# Spark

---

- 확장 프로젝트
  - Spark SQL
  - Spark Streaming
  - MLlib
  - GraphX
  - SparkR
  - Zeppelin
  - 등 ...

# Spark

---

- 장점
  - 시간과 비용을 아껴준다
    - 수십대의 Hadoop Cluster를 10대 이하의 Cluster로 대체할 수 있다
    - 수십분 기다려야 하던 작업이 1분만에 완료된다
  - 작업 능률 향상
    - MR 작업 코드 만들고, 패키징하고, submit하고 하던 복잡한 과정이, shell에서 코드 한줄 치는것으로 대체된다
    - 처음 접하는 사람도 배우기 쉽다
  - 다양한 제품을 조합해야 했던 작업이 Spark으로 다 가능하다