

# II. Hadoop



- Author: JinKyoung Heo
- Issue Date: 2014.08.14
- Revision #: Rev 31
- Homepage: [www.javaspecialist.co.kr](http://www.javaspecialist.co.kr)
- Email: [hjk7902@gmail.com](mailto:hjk7902@gmail.com)

이 페이지는 여백 페이지입니다.

# 1 Hadoop 개요

---

## Objectives

- 하둡의 구성요소와 하둡이 왜 필요한지에 대하여 배웁니다.
- HDFS와 MapReduce의 개념과 구조에 대하여 배웁니다.
- 하둡과 관련된 다른 프로젝트들은 어떤 것이 있는지에 대하여 배웁니다.

하둡 에코 시스템

What is Apache Hadoop?

GFS 개발 배경, 구성 및 데이터 흐름

Hadoop 모듈

Hadoop 파일 시스템의 장/단점

Hadoop 설계 목표

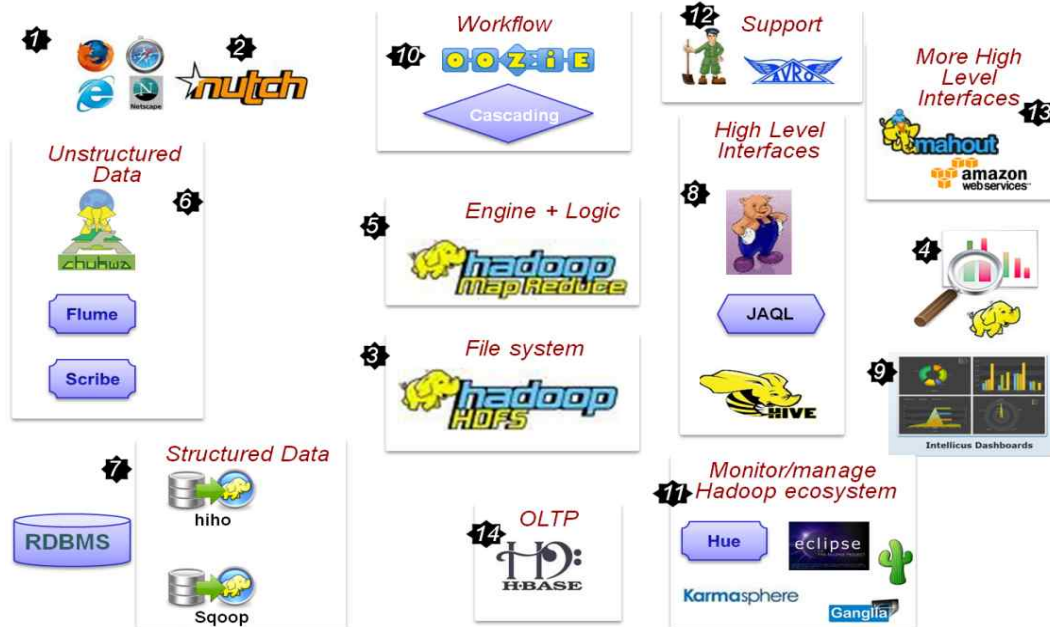
Hadoop Architecture, Stacks

Apache Hadoop NextGen MapReduce (YARN)

HDFS(Hadoop Distributed File System)

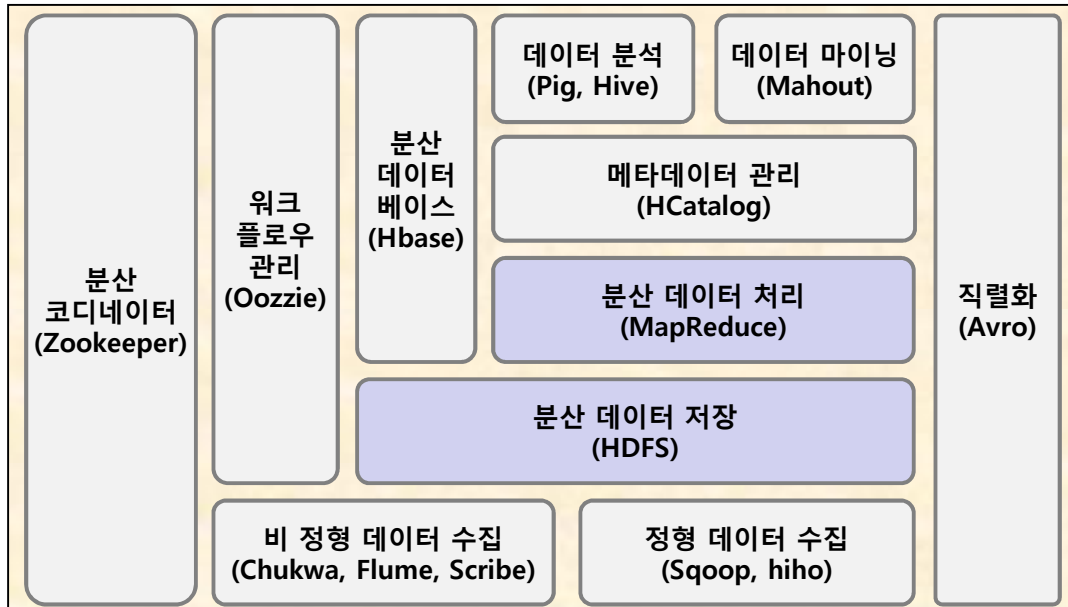
이 페이지는 여백 페이지입니다.

## Hadoop Eco System



- 어떻게 시작되었나 : 웹상의 엄청난 데이터를 어떻게 수집하고 처리할 것인가?
- 너치(Nutch)는 이런 웹상의 데이터를 수집하기 위한 오픈 소스 검색엔진입니다.
- 빅데이터가 저장되기 위해 : Hadoop분산파일시스템(HDFS)의 탄생
- 이런 데이터를 어떻게 활용하나?
- 맵리듀스 프레임워크(Map reduce framework)는 분석 코딩과 실행을 위해 존재 (스트리밍 또는 파이프를 통한 자바 또는 여러 언어들)
- 비정형 데이터를 어떻게 가져오나? : 웹로그, 클릭정보 - 퓨즈(fuse), 웹다브(webdav), 척와(chukwa), 플룸(flume), 스크라이브(Scribe)
- Hadoop분산파일시스템(HDFS)에 정형 데이터를 넣기 위한 하이호(Hiho), 스콥(sqoop) - RDBMS는 Hadoop에 편승할 수 있다
- 저수준 맵리듀스 프로그래밍 위에 있는 고수준의 인터페이스 툴 - 피그(Pig), 하이버(Hive), 자클(Jaql)
- 향상된 UI 리포팅과 함께 하는 BI 툴 - 드릴다운 등 - 인텔리커스(Intellicus)
- 맵리듀스 프로세스와 고수준 언어들 위의 워크플로우 툴 - 오지(Oozie), 캐스캐이딩(Cascading)
- Hadoop 모니터링 및 관리, 잡스와 하이버 운영, HDFS 모니터링 - 고수준의 모니터링 - 휴(Hue), 캄스피어(Karmasphere), 이클립스 플러그인(Eclipse plugin), 캐티(Cacti), 강리아(Ganglia)
- 프레임워크 및 클러스터 지원 - 에이브로(Avro, 직렬화), 주키퍼(Zookeeper, 코디네이터)
- 더욱 향상된 고수준의 인터페이스와 활용 - 마훿(Mahout, 기계학습), 일레스틱 맵리듀스(Elastic map Reduce)
- OLTP, NoSQL 데이터베이스 역시 가능 - Hbase

## 하둡 에코 시스템



### 하둡 에코 시스템 구성요소

#### 데이터 저장 및 관리 기술

하둡에서 핵심 코어에 속하는 기술로는 데이터를 저장하는 하둡 분산 파일 시스템(HDFS, Hadoop Distributed File System)과 저장된 데이터를 관리하는 맵-리듀스(Map-Reduce)등이 있습니다. 하둡 분산 파일 시스템(HDFS)은 대용량 파일을 지리적으로 분산되어 있는 수많은 서버에 저장하는 솔루션이며, 맵-리듀스는 분산되어 저장된 대용량 데이터를 병렬로 처리하는 솔루션입니다.

초기에 설계된 하둡 에코 시스템 1.0의 경우 배치 처리(Batch processing)를 목적으로 만들어진 시스템으로 맵과 리듀스간의 셔플링(Shuffling) 작업에 많은 시간을 허비하여 실시간 분석에 적합하지 않으며, 기존에 사용하고 있거나 새로 만들어진 솔루션을 확장하여 적용하는데 문제점을 가지고 있었습니다.

이러한 하둡 에코 시스템 1.0의 문제점은 관계형 데이터베이스 분석 시스템에 익숙한 많은 분석가들과 사용자들을 하둡으로 끌어들이는데 실패하는 원인이 되었으며, 이러한 문제점을 해결하기 위해서 하둡 에코 시스템 2.0에서는 맵-리듀스(Map-Reduce)를 버리고 YARN(Yet Another Resource Negotiator)을 채택하여 확장성과 데이터 처리 속도를 개선시켰습니다.

## 하둡 에코 시스템

### 데이터 수집 기술

하둡에서 데이터 수집을 담당하는 기술로는 아파치의 스쿱(Apache's Sqoop), 클라우데라의 플룸(Cloudera's Flume), 아파치의 척와(Apache's Chukwa)등이 있으며, 하둡에서 제공하는 데이터 수집 기술은 아니지만, 페이스북에서 만들어서 사용하고 있는 스크라이브(Facebook's Scribe)등이 있습니다.

정형데이터를 수집하는 기술로는 스쿱(Sqoop)이 있으며, 비정형데이터를 수집하는 기술로는 클라우데라(Cloudera)의 플룸(Flume)과 아파치(Apache) 척와(Chukwa)등이 있으며, 페이스북에서 사용하는 스크라이브(Scribe)도 비정형데이터 수집 기술입니다.

대용량 데이터는 정형데이터와 비정형데이터 두 가지 형태로 분류할 수 있습니다. 정형데이터는 저장된 형태가 규칙적인 데이터를 말하며, 비정형 데이터는 저장된 형태가 규칙적이지 않은 데이터를 말합니다. 정형데이터는 기업이나 공공기관에서 관리하고 있는 관계형 데이터베이스(RDB, Relational Database)를 말하며, 비정형데이터는 트위터(Twitter), 페이스북(Facebook)과 같은 소셜 네트워크 서비스(SNS, Social Network Service)에서 만들어지는 데이터를 말합니다.

### 분산 데이터베이스 H베이스(HBase)

하둡에서 제공하는 NoSQL(Not Only SQL) 데이터베이스입니다. H베이스는 자바 언어로 만들어졌으며, 하둡 분산 파일 시스템(HDFS)를 이용하여 분산된 컴퓨터에 데이터를 저장합니다. H베이스는 압축 기능과 자주 사용되는 데이터를 미리 메모리에 캐싱하는 인-메모리(In-Memory) 기술을 사용하여 데이터 검색 속도를 높입니다.

하둡에서 사용하는 데이터베이스는 관계형 데이터베이스가 아닌 NoSQL(Not Only SQL)입니다. 관계형 데이터베이스는 대용량 데이터를 처리하기 위해서 만들어진 시스템이 아니며, 국가나 회사에서 관리하는 정보 자원을 저장하고, 사용자가 손쉽게 SQL언어를 사용하여 저장된 정보를 다양한 방법으로 분석해 주는 시스템입니다. 현재도 많은 분석가들과 사용자들이 관계형 데이터베이스를 사용하고 있는데, 이러한 이유로는 관계형 데이터베이스를 통하여 정보(Information)를 다양한 도구를 사용하여 여러가지 측면에서 분석이 가능하며, 또한 결과를 실시간으로 조회할 수 있기 때문입니다.

## 하둡 에코 시스템

NoSQL(Not Only SQL)은 대용량 데이터를 처리하기 위해서 탄생된 데이터베이스입니다. 대용량 데이터를 저장하고 처리하기 위해서는 지리적으로 분산되어 있는 노드들에 대한 안정적인 관리가 필요하며, 속도보다는 데이터를 손실하지 않고 저장하는 방법과 더 많은 데이터를 처리하기 위해서 노드들을 쉽게 확장할 수 있는 방법에 중점을 두고 설계가 되었습니다. 이로 인하여 현장에서 데이터베이스를 운영하고 관리하는 분석가들과 사용자들이 원하는 빠른 속도와 쉬운 분석 방법 제공 그리고 관계형 데이터베이스에서 사용하고 손에 익었던 도구들을 그대로 사용하고 싶은 욕구를 충족시키지 못하는 문제점을 가지고 있습니다.

NoSQL(Not Only SQL) 데이터베이스에는 다양한 제품들이 있으며, 이 제품들을 특성에 맞게 'Key-Value store', 'Graph Database', 'Document Store', 'Wide Column Store' 4개의 제품 군으로 군집화 시킬 수 있습니다.

'Key-Value store' 제품 군으로는 맴캐시드(Mem-Cached)와 레디스(Redis, Remote Dictionary Server)등이 있으며, 키-밸류 형태로 이루어진 비교적 단순한 데이터 타입을 데이터베이스에 저장합니다. 'Graph Database' 제품에는 네오포제이(Neo4j)가 있으며, 그래프 모델에서 필요한 정점(Vertex)와 간선(Edge) 그리고 속성(Property)등과 같은 정보를 데이터베이스에 저장합니다. 'Document Store' 제품군에는 카우치DB와 몽고DB가 있으며, 문서 형태의 정보를 JSON 형식으로 데이터베이스에 저장합니다. 마지막으로 'Wide Colum Store' 제품군에는 H베이스(HBase)와 카산드라(Cassandra)가 있으며, 컬럼안에 여러 정보들을 JSON 형태로 저장할 수 있습니다.

하둡 진영에서는 관계형 데이터베이스에 호감을 가지고 있는 분석가와 사용자들의 마음을 돌리기 위해서, 빠른 속도와 손쉬운 사용 그리고 관계형 데이터베이스에서 사용하던 도구들을 이용할 수 있는 시스템을 만들기 위해서 노력하고 있습니다. 우리는 이러한 데이터베이스 제품들을 'NewSQL'이라고 부르고 있으며, 여기에는 그루터(Gruter)의 타조(Tajo), 구글의 드레멜(Dremel), 호튼웍스의 스팅어(Stinger), 클라우데라의 임팔라(Impala)등이 있습니다.



## 하둡 에코 시스템

### 하이브(Hive)

아파치의 하이브(Hive)는 'SQL 온 하둡(SQL On Hadoop)' 시스템입니다. 'SQL 온 하둡'은 하둡에서 SQL 언어를 사용하여 편리하게 정보를 조회(Query)하는 방법을 제공합니다. 하둡은 하이브(Hive)가 없었던 시절에는 맵-리듀스(Map-Reduce) 언어를 사용하여 하둡 분산 파일 시스템(HDFS)에 저장된 정보들을 조회(Query) 했습니다. 프로그램을 모르는 일반 사용자들이 맵-리듀스 언어를 사용하여 조회(Query)하는 일이 어려웠습니다. 하이브는 하이브큐엘(HiveQL)이라는 SQL과 거의 유사한 언어를 사용하여 일반 사용자들이 쉽게 데이터를 조회(Query)할 수 있도록 지원합니다.

하지만 하이브에도 몇 가지 단점이 있습니다. 하이브큐엘(HiveQL)을 통해서 조회(Query)를 실행하면 내부적으로 맵-리듀스 언어로 변환하는 작업을 거칩니다. 맵-리듀스는 맵과 리듀스간의 셔플링 작업으로 인하여 속도가 느리다는 단점이 있습니다. 하이브큐엘(HiveQL)이 일반 사용자들에게 손쉽게 조회(Query)할 수 있는 방법을 제공하고 있지만 처리 속도가 느리다는 문제점은 여전히 가지고 있습니다. 또한 하이브큐엘(HiveQL) 언어는 SQL과 비슷한 언어이지만 표준 SQL(Ansi SQL)의 규칙을 준수하지 않으며, 사용자들은 이러한 차이점을 다시 배워야하는 문제점을 가지고 있습니다.

### SQL 온 하둡(SQL On Hadoop)

하이브가 가지고 있는 문제점을 개선하기 위한 하둡 진영은 크게 두 갈래로 나뉩니다. "하이브를 완전히 대체하는 새 기술을 쓸 것인가?" 아니면 "하이브를 개선해 속도를 높일 것인가?"

하이브를 살려야 한다는 입장을 가장 강력하게 내세운 회사는 호튼웍스입니다. 호튼웍스는 하이브를 최적화하고 파일 포맷 작업을 통해 하이브 쿼리 속도를 100배 끌어올리겠다는 비전을 내 놓았습니다. 이것이 바로 스팅어(Stinger)입니다.

하이브를 버리고 새로운 엔진을 찾아야 한다는 진영은 그루터의 타조와 클라우데라의 임팔라가 있습니다. 타조는 하이브를 개선하는데 한계가 명확하기 때문에 대용량 SQL 쿼리 분석에 적합하지 않다는 입장이며, 기획 단계부터 하이브를 대체하는 새로운 엔진을 개발하고 있습니다.

클라우데라의 임팔라는 좀 특이한 경우인데, 일정 규모 이상의 데이터는 임팔라로 분석이 불가능합니다. 임팔라는 메모리 기반 처리 엔진이어서, 일정 용량 이상에서는 디스크 환경의 하이브를 사용해야 합니다. 하지만 전체 틀에서는 하이브를 버리는 쪽으로 무게를 두고 있습니다.

## What is Apache Hadoop?

- ◆ 신뢰성 있고, 확장성 있는 분산 컴퓨팅을 위한 오픈소스 프레임워크
- ◆ 간단한 프로그래밍 모델을 사용하여 대용량 데이터의 분산 처리를 할 수 있는 프레임워크
- ◆ GFS, MapReduce 소프트웨어구현체
  - 아파치 Top-Level 프로젝트
  - 코어는 Java, C/C++, Python 등 지원
- ◆ 대용량 데이터 처리를 위한 플랫폼
  - 분산파일시스템(HDFS)
  - 분산병렬처리시스템(MapReduce)
  - 기반소프트웨어프레임워크(Core)



<http://hadoop.apache.org/>

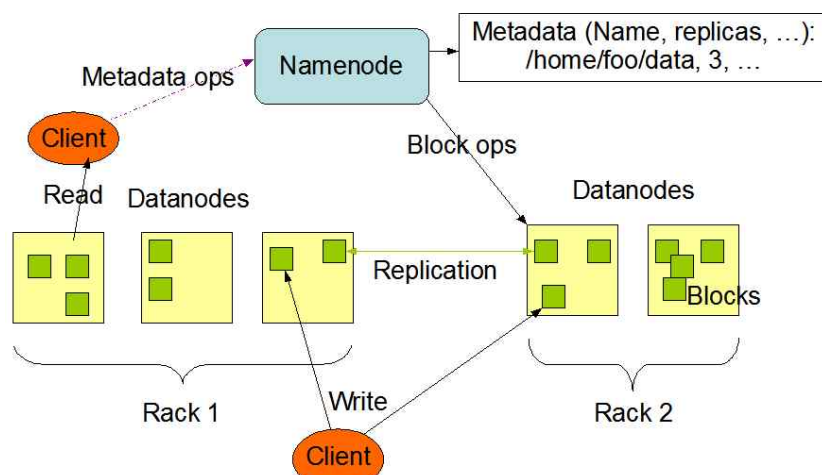
하둡은 아파치 재단에서 개발한 신뢰성 있고, 확장성 있는 분산 컴퓨팅을 위한 오픈소스 프레임워크입니다. 즉, 소프트웨어 라이브러리란 의미이기도 합니다. 하둡 프레임워크를 이용하면 간단한 프로그래밍 모델을 사용하여 대용량 데이터의 분산 처리를 할 수 있습니다.

하둡은 GFS(Google File System), MapReduce 소프트웨어구현체라고 할 수 있는데, 아파치의 Top-Level 프로젝트로 진행되고 있습니다.

- <http://hadoop.apache.org>

하둡 코어는 Java, C/C++, Python 등 지원하고 있으며, 대용량 데이터 처리를 위한 플랫폼이기 때문에 분산파일시스템(HDFS; Hadoop Distributed File System), 분산병렬처리시스템(MapReduce), 기반소프트웨어프레임워크(Core)으로 구성되어 있습니다.

HDFS Architecture



이미지 출처: <http://hadoop.apache.org/docs/r2.2.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>

## What is Apache Hadoop?

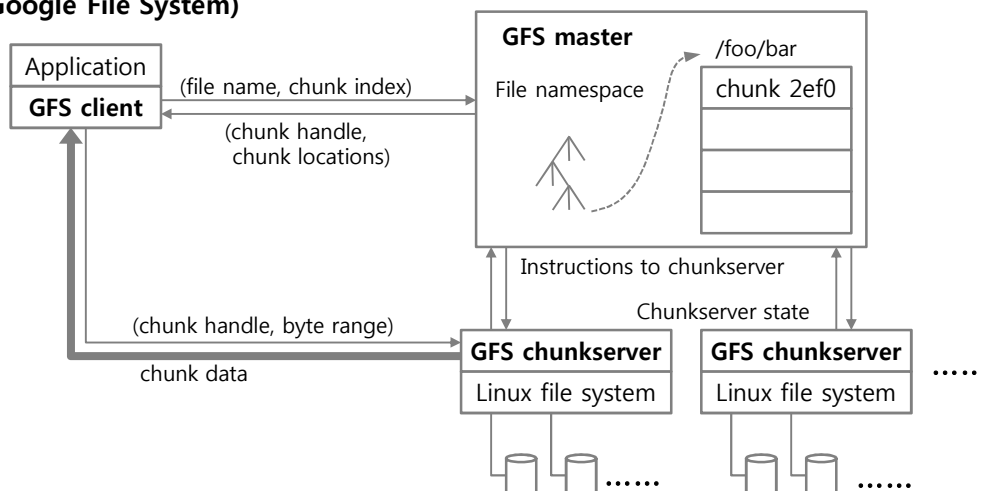
- ◆ 2004년 발표한 구글의 논문을 참고
- ◆ 분산 파일 시스템 GFS, MapReduce 소프트웨어구현체
- ◆ 분산 처리 시스템 MapReduce
- ◆ 2005년 오픈소스 검색엔진 "Nutch"를 개발 중이던 더그 커팅이 대용량 데이터 처리를 위해 구글 논문을 참조하여 구현
- ◆ 규모가 커지고 중요성이 높아지면서 별도 아파치 프로젝트로 분리



하둡은 GFS(Google File System), MapReduce 소프트웨어구현체입니다. 2005년 오픈소스 검색엔진 "Nutch"(너치)를 개발 중 이던 더그 커팅(Doug Cutting)은 2004년 구글이 발표한 분산파일 시스템 (GFS)를 참고하여 기업들이 페타바이트 수준의 체계가 없는 데이터를 저장하고 분석할 수 있도록 하는 오픈소스 하둡 프레임워크를 만들어냈습니다. 하둡은 더그 커팅의 아들이 가지고 놀던 코끼리 인형의 이름에서 유래되었다고 합니다.

분산시스템은 대용량의 데이터를 처리/관리, 복잡한 계산 처리를 위해 여러 대의 컴퓨터를 연결하여 하나의 시스템처럼 사용하는 것을 말합니다.

### GFS(Google File System)



## GFS 개발 배경

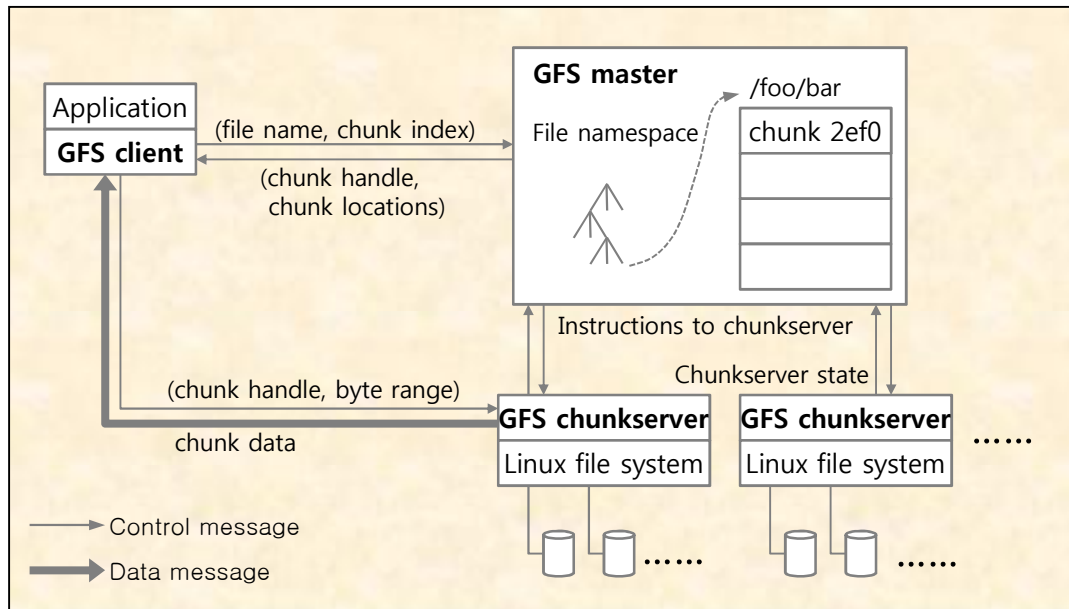
- ◆ 초기의 구글
  - 6000만개 웹페이지, 이미지 제외하고 500GB
  - 이때 신제품 HDD 4GB
- ◆ 자료를 업데이트하고 검색하기 위한 고용량 서버 필요함
  - Scale-Up 대신 Scale-out 방식 선택
  - 그래서 탄생한 것인 GFS, 맵리듀스
- ◆ 구글이 논문을 발표한 2006년 이후 하둡 개발 본격화됨
  - 구글의 논문을 참고해 500GB를 59초, 100TB를 173분만에 정렬
- ◆ 하둡을 유명하게 만든 활용사례
  - 뉴욕타임스가 1851년부터 1980년까지 누적된 기사 1100만 건을 데이터화
  - 뉴욕타임스는 이 작업에 필요한 하드웨어와 소프트웨어를 구매하는 대신 하둡을 활용해 단기간에 작업을 완료했다.

구글 파일 시스템은 구글의 대규모 클러스터 서비스 플랫폼의 기반이 되는 파일 시스템으로, 위 슬라이드에서 언급한 요구 사항을 만족시키기 위해 개발됐습니다.

다음은 GFS를 개발하게 된 배경입니다.

- 저가형 서버로 구성된 환경으로 서버의 고장이 빈번히 발생할 수 있다고 가정 합니다.
  - 대부분의 파일은 대용량 파일을 가정합니다. 그러므로 대용량 파일을 효과적으로 관리할 수 있는 방법이 요구됩니다.
  - 작업 부하는 주로 연속적으로 많은 데이터를 읽는 연산이거나 임의의 영역에서 적은 데이터를 읽는 연산으로 구성됩니다.
  - 파일에 대한 쓰기 연산은 주로 순차적으로 데이터를 추가하는 연산이며 파일에 대한 수정은 드물게 이루어집니다.
  - 여러 클라이언트에서 동시에 동일한 파일에 데이터를 추가하는 환경에서 동기화 오버헤드를 최소화할 수 있는 방법이 요구됩니다.
  - 낮은 응답 지연 시간보다 높은 처리율이 좀 더 중요합니다.
- 하둡은 대규모 데이터 처리가 필수적인 구글, 야후, 페이스북, 트위터 등 인터넷 서비스 기업에서 먼저 활용하기 시작했습니다. 국내에서도 네이버와 다음, KT가 서비스의 여러 부분에 사용하고 정부기관이 발주하는 빅데이터 프로젝트 대부분이 하둡 적용을 요구하는 것으로 알려지며 빅데이터 분석 인프라 분야에서 서서히 표준으로 자리를 잡아가는 상황입니다. 이에 따라 IT업계에서는 최근 하둡 전문가를 확보하기 위한 경쟁이 치열한 상황입니다.

## GFS 구성



GFS는 엄청나게 많은 데이터를 보유해야 하는 구글의 핵심 데이터 스토리지와 구글 검색 엔진을 위해 최적화되었습니다. 구글 초창기에 래리 페이지와 세르게이 브린에 의해 개발된 "빅파일"에서 개선된 것입니다. 파일들은 일반적인 파일 시스템에서의 클러스터들과 섹터들과 비슷하게 64MB로 고정된 크기의 청크들로 나뉩니다. 이것들은 덮어쓰거나 크기를 줄이는 경우가 극히 드물며 보통 추가되거나 읽혀지기만 합니다. 가격이 저렴한 범용 컴퓨터들로 구성되고 집적도가 높은 구글의 컴퓨팅 클러스터들에서 잘 동작하도록 최적화되었습니다. 가격이 저렴한 서버에서도 사용되도록 설계되었기 때문에 하드웨어 안정성이나 자료들의 유실에 대해서 고려하여 설계되었고 레이턴시가 조금 길더라도 데이터의 높은 처리율에 중점을 두었습니다.

좀 더 많은 정보를 원한다면 다음 사이트를 방문해 보세요.

[http://ko.wikipedia.org/wiki/구글\\_파일\\_시스템](http://ko.wikipedia.org/wiki/구글_파일_시스템)

## Hadoop 모듈

- ◆ Hadoop Common
  - 다른 하둡 모듈을 지원하는 유틸리티
- ◆ HDFS
  - Hadoop Distributed File System
  - 어플리케이션 데이터에 고성능 접근을 지원하는 분산 파일 시스템
- ◆ HBase
  - 분산 데이터베이스
- ◆ Hadoop YARN
  - 잡 스케줄링과 클러스터 리소스 관리를 위한 프레임워크
- ◆ Hadoop MapReduce
  - 대용량 데이터의 병렬처리를 위한 양 기반 시스템

HDFS와 기존 대용량 파일 시스템의 가장 큰 차이점은 저사양 서버를 이용해 스토리지를 구성할 수 있다는 것입니다. HDFS를 이용하면 수십 혹은 수백 대의 웹 서버급 서버를 묶어서 하나의 스토리지처럼 사용할 수 있습니다.

HDFS가 기존 대용량 파일 시스템을 완전히 대체하는 것은 아닙니다. DBMS처럼 고성능, 고가용성이 필요한 경우에는 SAN을 사용하며, 안정적인 파일 저장이 필요한 경우에는 NAS를 사용합니다. 또 전자상거래처럼 트랜잭션이 중요한 경우는 HDFS가 적합하지 않습니다. 그러나 대규모 데이터를 저장하거나, 배치로 처리하는 경우에 HDFS를 이용하면 됩니다.

## Hadoop 파일 시스템의 장/단점

### ◆ 특징 및 장점

- 선형적인 확장성 제공
- 글로벌 네임스페이스 제공
- 비용절감
- 데이터분석 처리에 활용

### ◆ 일반적인 파일 시스템과 다른 특징과 제약사항

- 응용프로그램 기반의 파일 시스템
- 불변 파일만 저장
- NameSpace 관리를 NameNode 메모리에 저장
- 전체 처리 용량증가
- NameNode 이중화 문제
  - Namenode(master)가 죽으면?

- 선형적인 확장성 제공 - 스토리지 용량 증설을 위해서는 스토리지로 사용할 리눅스 장비를 추가하고 Hadoop의 데이터 노드만 실행해 주면 Hadoop 파일 시스템에서 자동으로 인식합니다. 스토리지 용량 증설을 위해 응용 계층에서의 작업은 필요 없으며, 서버 측 작업도 마운트 등의 작업이 필요 없이 서버만 추가하면 됩니다.
- 글로벌 네임스페이스 제공 - Hadoop 파일 시스템을 이용할 경우 회사 전체의 시스템이나 전체 서비스에서 파일에 대한 유일한 식별 단위를 가질 수 있습니다. 글로벌 네임스페이스를 이용하면 시스템 간의 파일 데이터에 대한 중복 저장 등 파일 공유 등의 문제를 쉽게 해결할 수 있습니다.
- 비용절감 - 처음 Hadoop이 출시됐을 때에는 비용 절감이 가장 큰 장점이었고 최근에도 비용 절감이 Hadoop을 선택하게 하는 항목이지만 과거에 비해 큰 비중은 없습니다. 어떤 용도로 스토리지를 사용하느냐에 따라 NAS 같은 기존 스토리지 솔루션과 비교해 비용 절감 효과가 클 수도 있고 비슷할 수도 있습니다.
- 전체 처리 용량 증가 - Hadoop 같은 분산파일 시스템은 분산된 서버의 디스크를 이용하기 때문에 네트 워크, 디스크 I/O 등이 각 서버로 분산됩니다. 따라서 동일한 용량의 스토리지 솔루션을 구축할 경우 전체 처리 용량은 증가합니다.

## Hadoop 파일 시스템의 장/단점

- 데이터분석 처리에 활용 - 최근 Hadoop 파일 시스템이 점점 일반 스토리지 서비스에도 많이 사용되고 있기는 하지만 가장 많이 사용되는 분야는 역시 데이터 분석 분야입니다. 분석용 데이터를 Hadoop 파일 시스템에 저장하고 맵리듀스라는 분산/병렬 처리 프레임워크를 이용해 빠르게 분석하는 데 주로 활용 중입니다.
- 응용 프로그램 기반의 파일 시스템 - Hadoop 분산 파일 시스템은 응용 프로그램으로 구성된 파일 시스템이기 때문에 파일 시스템이라고는 하지만 일반적인 파일 시스템처럼 운영체제에서 제공하는 ls, copy, rm 등과 같은 파일 처리 명령을 이용할 수 없습니다. 파일 시스템으로 사용하려면 운영체제에 마운팅 해야 하는데, 기본적으로 Hadoop 분산 파일 시스템에서는 마운팅 기능을 제공하지 않고 FUSE 같은 도구를 이용해 마운팅을 할 수 있습니다.
- 불변 파일만 저장 - Hadoop 분산 파일 시스템에서 파일은 한 번 써지면 변경되지 않는다고 가정합니다. 한 번 저장된 파일을 오픈 해 특정 위치에 데이터를 수정하는 랜덤 쓰기 능은 제공하지 않습니다. 데이터를 추가하는 기능은 최근 배포된 버전에서야 지원하기 시작했습니다. 따라서 Hadoop 분산 파일 시스템은 파일을 저장하고 저장된 파일에 대해 스트리밍 방식의 읽기 요청 위주인 응용이나 배치 작업 등에 적합합니다. 이런 제약 때문에 파일 처리를 주로 하는 기존 솔루션이나 시스템을 수정 없이 Hadoop 분산 파일 시스템을 적용할 수는 없습니다.
- 네임스페이스 관리를 네임 노드 메모리에 저장 - Hadoop에 저장할 수 있는 파일 과 디렉토리의 개수는 네임 노드의 메모리 크기에 제한을 받습니다. 네임 노드 는 하나의 파일 정보를 저장하는 데 수백 바이트를 사용합니다. 네임 노드의 메모리가 수 GB라고 가정하면 대략 수 천만 개 정도의 파일을 저장할 수 있습니다. 이런 제약은 일정 크기로 여러 개의 Hadoop 클러스터를 구성하는 방식으로 해결할 수 는 있지만 본질적인 해결책은 아닙니다.
- 네임 노드 이중화 문제 - 네임노드가 SPOF(Single Point Of Failure) 입니다. 즉, 일부 데이터 노드에 장애가 발생해도 파일은 안정적으로 서비스할 수 있지만 네임 노드에 문제가 발생하면 파일 시스템 전체 클러스터에 장애가 발생합니다. 이 부분은 향후 개선 또는 보완될 예정입니다.



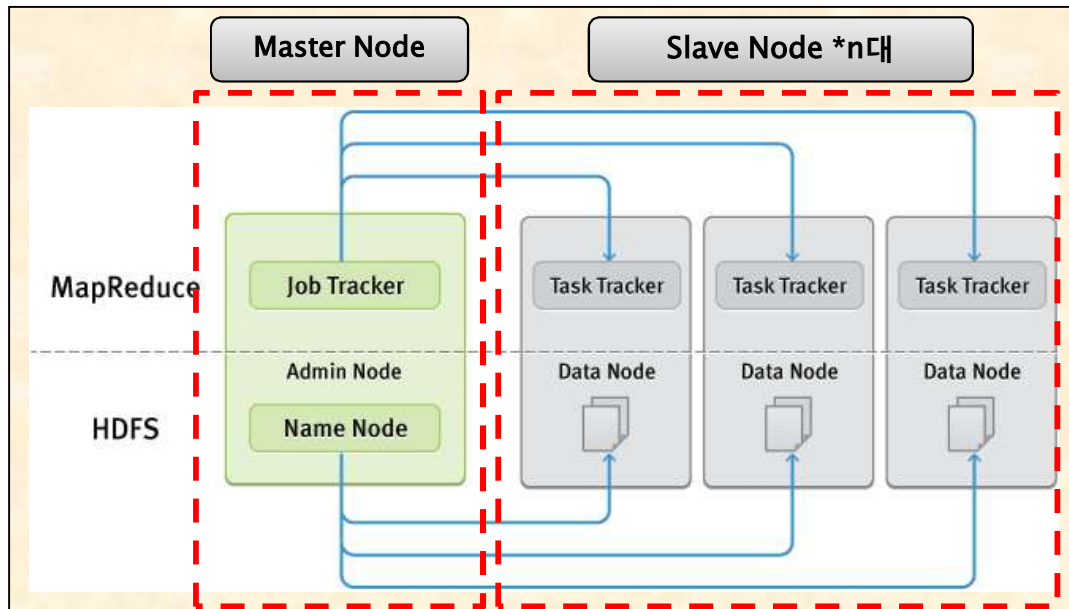
## Hadoop 설계 목표

- ◆ 장애복구
  - 복제 데이터 저장
  - 주기적인 상태 체크
- ◆ 스트리밍 방식의 데이터 접근
  - 높은 데이터 처리량에 중점을 두고 있음
- ◆ 대용량 데이터 저장
  - 하나의 파일이 테라바이트 이상 저장 가능
  - 하나의 인스턴스에서 수백만 개 이상의 파일 지원
- ◆ 데이터 무결성
  - 읽기 전용
  - 파일 이동, 삭제, 복사 인터페이스 제공

HDFS는 다음 4가지 목표를 가지고 설계되었습니다.

- 장애복구 : HDFS는 장애를 빠른 시간에 감지하고 대처할 수 있게 설계돼 있습니다. HDFS는 데이터를 저장하면, 복제 데이터도 함께 저장해서 데이터 유실을 방지합니다. 또한 분산 서버 간에는 주기적으로 상태를 체크해 빠른 시간에 장애를 인지하고, 대처할 수 있게 도와줍니다.
- 스트리밍 방식의 데이터 접근 : HDFS에 파일을 저장하거나 저장된 파일을 조회하려면 스트리밍 방식으로 데이터에 접근해야 합니다. HDFS는 기존 파일 시스템들과 달리 배치 작업에 적합하도록 설계돼 있고, 낮은 데이터 접근 지연 시간보다는 높은 데이터 처리량에 중점을 두고 있습니다.
- 대용량 데이터 저장 : HDFS는 하나의 파일이 기가바이트, 테라바이트 이상의 사이즈로 저장될 수 있게 설계 됐습니다. 높은 데이터 전송 대역폭과 하나의 클러스터에서 수백 대의 노드를 지원할 수 있어야 합니다. 또한 하나의 인스턴스에서는 수백만 개 이상의 파일을 지원합니다.
- 데이터 무결성 : HDFS는 한번 저장한 데이터는 더는 수정할 수 없고, 읽기만 가능하게 해서 데이터 무결성을 유지합니다. 데이터 수정은 불가능 하지만 파일이동, 삭제, 복사할 수 있는 인터페이스를 제공합니다. 데이터를 수정할 수 없는 문제는 하둡 0.20버전부터 검토돼 왔고, 하둡 2.0 버전부터 HDFS에 저장된 파일에 append가 제공됩니다.

## Hadoop Architecture



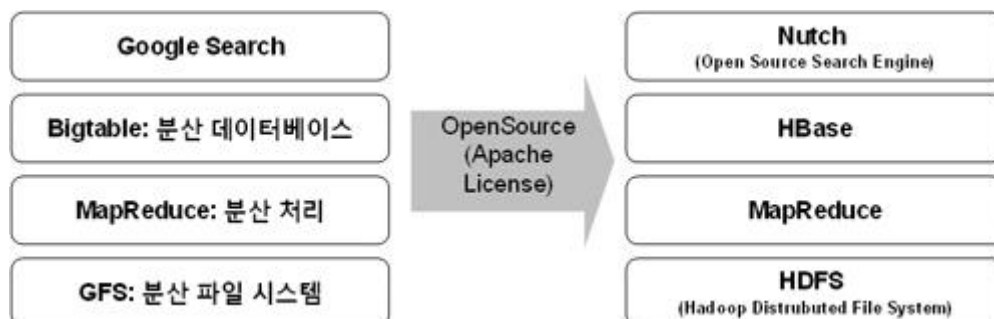
HDFS와 MapReduce로 구분 할 수 있습니다. HDFS는 파일 분산 저장 목적으로 하는 Layer이며, MapReduce는 파일 분석하기 위한 Layer입니다.

국내/외 사용업체를 보면 국외에는 Yahoo(40,000nodes), Facebook, IBM, Amazon, Twitter등이 있으며, 국내에는 Naver, Daum 등이 있습니다. 하둡과 관련 있는 업체들에 대하여 자세한 정보는 아래의 주소를 참고하세요.

<http://wiki.apache.org/hadoop/PoweredBy>

Hadoop 활용분야는 유전자 염기 서열분석, Google이나 Yahoo 등 파일 및 데이터 검색, 로그 분석, 통신사 대용량 로그 데이터 처리, 대용량 이미지 데이터 처리, 교통흐름 분석하기 위한 정보 / 파일,문서 등 비정형 데이터 처리 등이 있습니다.

다음 그림은 Google과 Hadoop을 비교한 그림입니다.



## Hadoop Stacks

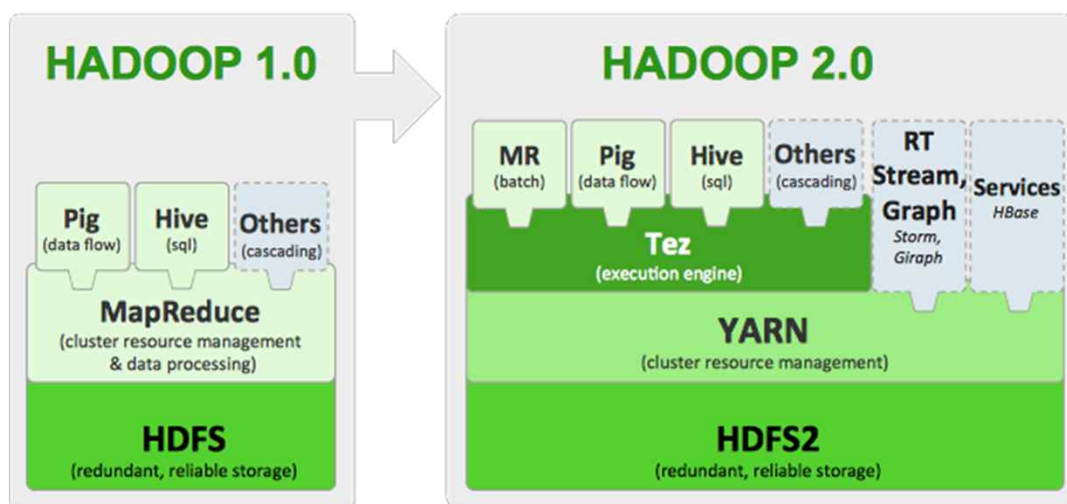


그림 출처 : <http://hortonworks.com/blog/apache-hadoop-2-is-ga/>

## Apache Hadoop NextGen MapReduce (YARN)

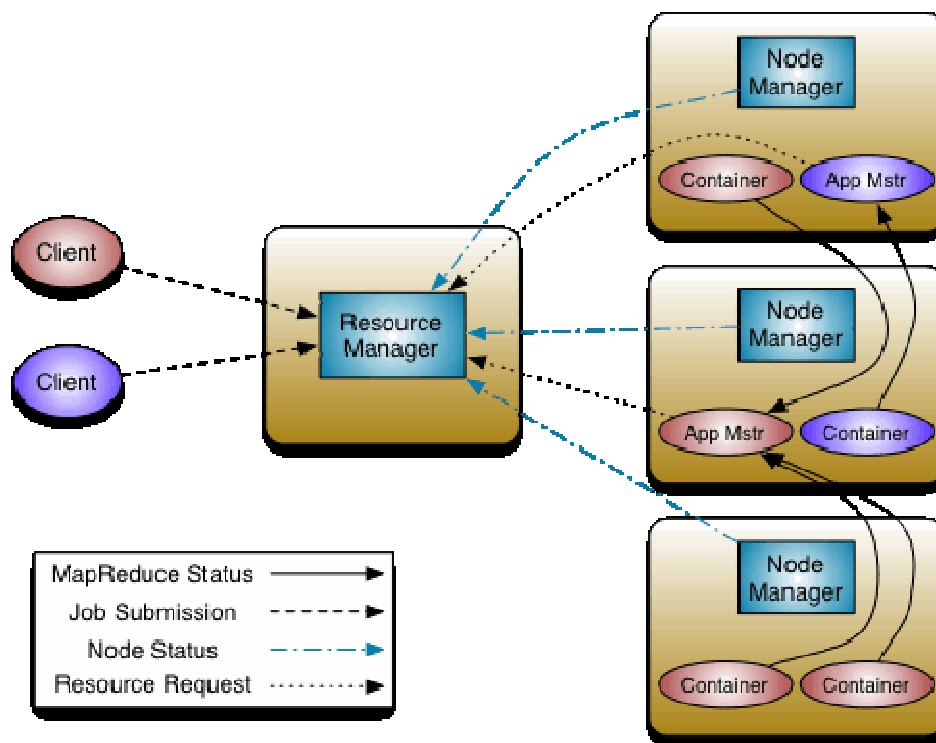


그림 출처 : <http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/hadoop-yarn-site/YARN.html>

## HDFS(Hadoop Distributed File System)

- ◆ 파일의 분산 저장이 목적
- ◆ Namenodes와 Datanodes로 구성
  - Master Namenode
  - Secondary Namenode
  - Datanode
- ◆ 저렴한 컴퓨터로 대 용량 데이터를 저장할 수 있는 시스템
  - 네트워크 Raid와 같이 연결된 것 처럼 사용하는 하드디스크
  - Scale Out
- ◆ Block(Chunk) 단위로 파일관리 (저장/복제/삭제)
  - Default Size는 128M(134217728)
- ◆ 복제기능을 통해 안전성/신뢰성을 보장
- ◆ 1대의 Master서버에 4000+이상의 Datanodes를 운영할 수 있음.
- ◆ API지원
  - 하둡 코어는 Python, Java, C/C++

HDFS는 파일의 분산 저장이 목적입니다. HDFS는 name 노드와 data 노드로 구성됩니다. name 노드는 master와 secondary 노드(보조 노드)로 구성될 수 있습니다. 보조노드는 마스터 노드의 백업 노드가 아닙니다.

HDFS를 이용하면 저가의 컴퓨터로 대 용량 데이터를 저장할 수 있는 시스템을 구축할 수 있습니다. 그것도 무종단 용량 확장(Scale out) 방식을 지원한다는 것이죠. 그러므로 노드의 추가 또는 제거 등 관리 관점에서도 유리합니다.

블록단위로 파일을 관리하며, 복제 기능을 통해 안정성 및 신뢰성을 보장합니다. 디폴트 블록 단위는 128Mbyte입니다. 블록의 크기는 dfs.blocksize 속성을 이용하면 변경할 수 있습니다.

1대의 마스터 서버에 4000개 이상의 데이터 노드를 운영할 수 있으며, 코어는 파이썬, Java, C/C++등을 지원하고 있습니다.

## MapReduce

- ◆ 정의
  - MapReduce는 구글에서 분산 컴퓨팅을 지원하기 위한 목적으로 제작하여 2004년 발표한 소프트웨어 프레임워크로 HDFS에 저장된 파일이용
- ◆ 구성
  - 프레임워크는 함수형 프로그래밍에서 일반적으로 사용되는 map()과 reduce() 함수 기반으로 주로 구성.
  - map() : (key, value) 쌍을 처리하여 또 다른 (key, value) 쌍을 생성하는 함수
  - reduce() : 맵으로부터 생성된 (key, list(value))들을 병합(merge)하여 최종적으로 list(value) 들을 생성하는 함수
- ◆ 목적
  - 프레임워크는 페타바이트(Petabyte) 이상의 대용량 데이터를 신뢰할 수 없는 컴퓨터로 구성된 클러스터 환경에서 병렬 처리를 지원하기 위해서 개발
- ◆ 단점
  - Java 언어를 습득할 필요.

MapReduce는 구글에서 분산 컴퓨팅을 지원하기 위한 목적으로 제작하여 2004년 발표한 소프트웨어 프레임워크로 HDFS에 저장된 파일을 이용합니다.

프레임워크는 함수형 프로그래밍에서 일반적으로 사용되는 Map()과 Reduce() 함수 기반으로 주로 구성됩니다.

Map()은 (key, value) 쌍을 처리하여 또 다른 (key, value) 쌍을 생성하는 함수입니다. Reduce()는 맵(map)으로부터 생성된 (key, list(value)) 들을 병합(merge)하여 최종적으로 list(value) 들을 생성하는 함수를 말합니다.

MapReduce 프레임워크는 페타바이트 이상의 대용량 데이터를 신뢰할 수 없는 컴퓨터로 구성된 클러스터 환경에서 병렬 처리를 지원하기 위해서 개발되었습니다.

그런데 MapReduce는 Java 언어를 습득해야 합니다. 자바를 모른다면 처음 진입장벽이 높을 수 밖에 없을 것입니다. 특히 자바의 컬렉션(Collection) 프레임워크와 제네릭(Generic) 기능에 대해 잘 알아둬야 합니다.

## Hadoop 관련 아파치 프로젝트

- ◆ Avro : 데이터 직렬화 시스템
- ◆ Cassandra : 싱글 포인터 실패가 없는 확장 가능한 멀티 마스터 Database
- ◆ Chukwa : 대용량 분산 시스템 관리를 위한 데이터 수집 시스템
- ◆ HBase : 대용량 테이블들을 위한 데이터 스토리지 구조를 지원하기 위한 확장 가능한 분산 데이터베이스
- ◆ Hive : 하둡 기반 데이터 웨어하우스 인프라스트럭처, 데이터 요약과 Ad hoc 쿼리를 제공
- ◆ Mahout : 확장 가능한 기계 학습과 데이터 마이닝 라이브러리
- ◆ Pig : 병렬 계산을 위한 높은 수준의 데이터 흐름 언어와 실행 프레임 워크
- ◆ ZooKeeper : 분산 어플리케이션을 위한 고성능 코디네이션 서비스

### • Cassandra(<http://cassandra.apache.org/>)



카산드라는 싱글 포인터 실패(Single Pointer of Failure)가 없는 확장 가능한 멀티 마스터 데이터베이스입니다. '싱글 포인터 실패가 없는'이라는 의미는 다른 서버의 데이터 복제본을 구성해서 특정 로드 장애 시 서비스 영향을 주지 않는다는 의미이며, '확장 가능한 멀티 마스터 데이터베이스'는 토큰링을 배경으로 키 구간 설정으로 서버의 추가 및 제거만으로 전체 저장 공간이 확장/축소 됩니다. 이때 링이 여러 개이면서 데이터가 링 노드에 교차 저장됩니다.

카산드라는 대용량 자료의 저장/처리를 목적으로 하면서 높은 가용성을 가지게 합니다.

Key/Value 기반 해시 알고리즘 사용으로 빠른 성능 제공합니다. 물론 이런 기능은 오라클, MS-SQL도 가능하지만 카산드라는 데이터 저장을 위한 NoSQL 데이터베이스라는 것입니다.

NoSql Database에는 Cassandra, HBase, MongoDB, Cloudata 등이 있습니다.

ad hoc query : 특별한 목적을 위한 쿼리

## Hadoop 관련 아파치 프로젝트

- Chukwa(<http://incubator.apache.org/chukwa/>)



척와는 아파치의 오픈소스 프로젝트로 진행되는 대용량 분산 시스템 관리를 위한 데이터 수집 시스템입니다. 이와 유사한 것으로 Scribe와 Flume(플룸)이 있습니다.

다음 표는 Scribe와 Flume을 비교했습니다.

Scribe	Flume
<ul style="list-style-type: none"> <li>• Apache 오픈소스 프레임워크</li> <li>• 대용량 로그 데이터 수집,</li> <li>• Thrift 기반,</li> <li>• RPC환경을 지원하기 위한 Cross Language Framework</li> <li>• C++ 구현체, 확장이 어려움</li> </ul>	<ul style="list-style-type: none"> <li>• 대용량 로그 데이터를 효율적으로 수집/집계 및 이동 목적으로 개발</li> <li>• 클라우데라(Cloudera)에서 공개한 오픈소스</li> <li>• 자바기반, 확장성이 좋음</li> </ul>

- HBase(<http://hbase.apache.org/>)



하둡의 구성요소로 HBase가 포함됩니다. HBase는 대용량 테이블들을 위한 데이터 스토리지 구조를 지원하기 위한 확장 가능한 분산 데이터베이스입니다.

Single master이며, 분산 파일 시스템 위에 스토리지가 구축되어 있고, 서버마다 데이터가 ad hoc하게 파티션 되어 있습니다.

카산드라를 사용할 것인가? Hbase를 사용할 것인가? 몽고 디비를 사용할 것인가? 고민 할 수 있습니다. 카산드라는 높은 가용성이 있는 반면, HBase는 관리가 용이하며, MongoDB는 Auto Sharing을 지원한다는 장점이 있습니다.

- 카산드라 : 높은 가용성
- HBase : 관리 용이
- MongoDB : Auto Sharing

## Hadoop 관련 아파치 프로젝트



- **Hive**(<http://hive.apache.org/>)

하이브는 하둡 기반 데이터 웨어하우스 인프라스트럭처입니다. 데이터 요약과 Ad hoc 쿼리 (특별한 목적을 위한 쿼리)를 제공하기 위하여 Hive QL이라는 SQL 기반 쿼리를 제공해서 대량의 데이터를 위한 동적 쿼리와 분석이 가능합니다.



- **Mahout**(<http://mahout.apache.org/>)

머하웃은 확장 가능한 기계 학습(Machine Learning)과 데이터 마이닝 라이브러리입니다.

Machine Learning이라는 의미는 새로운 정보를 학습하고, 습득한 정보를 효율적으로 사용할 수 있는 능력과 결부시키는 지식 습득한다는 것을 의미합니다. 작업을 반복적으로 수행함으로써 결과를 얻어내는 기술의 개선 과정이라고 할 수 있습니다.

머하웃을 이용한 시스템 구축 예로 몇 가지가 있습니다.

- 과거 구매 레코드를 기반으로 사용자에게 제품을 제안하는 아마존 등의 시스템
- 지정한 날짜의 유사한 뉴스 기사를 모두 찾아주는 시스템

머하웃은 3가지 특징(협업 필터링, 클러스터링, 분류)을 가지고 있습니다.

- 협업 필터링(Collaboration Filtering)

사용자와 관련 항목간의 유사성을 계산하여 이 정보를 다른 곳에 사용하도록 하는 기능입니다. 협업 필터링은 사용자기반(유사한 사용자), 항목기반(유사한 항목), Slope-One(항목 + 등급), 모델기반(사용자 + 등급)으로 나뉩니다.

- 클러스터링

높은 잠재력의 유망 고객 집단을 찾아내는 시장분석기법입니다. 협업 필터링(CF)은 컬렉션 내의 항목간의 유사성을 계산하지만 Clustering은 유사항목을 그룹화하는 작업만 수행합니다.

- 분류(categorization, classification)

처음 본 문서에 레이블을 지정하여 그룹화 하는 것입니다. 그룹화 된 문서들의 특징과 레이블을 가지고 통계 처리하며, 통계 처리된 결과를 가지고 이전에 못 본 문서를 분류 할 때 사용할 수 있습니다.



## Hadoop 관련 아파치 프로젝트

### • Pig(<http://pig.apache.org/>)



피그는 하둡을 기반으로 동작하는 병렬 데이터 처리 엔진입니다. Pig Latin 스크립트 언어를 기반으로 데이터를 처리합니다. 하둡은 데이터를 처리하기 위해 MapReduce와 같이 언어 기반 프로그래밍을 합니다.

### • Zookeeper(<http://zookeeper.apache.org/>)



주키퍼는 분산 어플리케이션을 위한 고성능 코디네이션 서비스입니다.

좀 더 쉽게 이야기 하면 동물원 사육사와 같이 Hadoop(코끼리), chukwa(거북이), pig(돼지), hive(벌) 등을 관리하며, 분산처리 환경에서 다중의 서버에 집합을 묶어서 관리해주는 시스템에서 장애, 동기화 등 문제발생시 해결을 쉽게 하기 위한 것입니다.

주키퍼는 로드밸런서의 역할을 하는데, 이를 위해 다음과 같은 기능을 제공합니다.

- 네임서비스 (서비스 룩업)를 통한 부하 분산
- 분산 락이나 동기화 문제 해결
- 클러스터 멤버십
- 높은 가용성을 위한 장애복구

분산 시스템에서는 다음과 같은 문제를 생각하지 않을 수 없습니다.

- 서버의 로드상태에 따라 동적으로 새로운 서버를 추가 할 수 있는가?
- DB에 락을 건 서버가 장애가 발생하면 어떻게 되나?
- Active/Standby 구조일 경우 액티브 서버의 장애 판단은?(만일 좀비 상태가 된다면?)

주키퍼는 다음 4가지의 노드들을 가지고 있습니다.

- Znodes
- Ephemeral Nodes (임시노드)
- Sequence Nodes (순차적노드)
- Watches

이 페이지는 여백 페이지입니다.

## 2 *Hadoop 클러스터 구성*

---

### **Objectives**

- 리눅스 시스템에 하둡 클러스터를 구성합니다.
- HDFS을 사용하는 어플리케이션을 개발하기 위한 개발 환경을 설정합니다.

Hadoop 클러스터 구성 환경  
JDK와 Eclipse 설치  
Hadoop 다운로드 및 설치  
.bash\_profile 환경설정  
/etc/hosts 파일 설정  
SSH 설정  
하둡 환경 설정 파일  
hadoop-env.sh 설정  
노드 설정  
core-site.xml 파일 수정  
hdfs-site.xml 설정  
mapred-site.xml 파일 수정  
yarn-site.xml 파일 수정  
다른 노드 설정파일 동기화  
네임노드 초기화  
실행  
방화벽 설정  
하둡 H/W 시스템 구성도

이 페이지는 여백 페이지입니다.

## Hadoop 클러스터 구성 환경

### ◆ 노드 구성

- 네임노드 1(master)
- 보조 네임노드 1(backup)
- 데이터 노드 2(slave1, slave2)
- 모든 노드에 동일하게 hadoop 계정 생성

### ◆ 설치환경

- Cent OS 6.4 x86 32bit
- Java SE 7u51(<http://java.sun.com>)
- Apache Hadoop 2.2.0(<http://hadoop.apache.org>)
- Eclipse 4.3(Kepler)(<http://eclipse.org>)



하둡과 관련 프로젝트에 대하여 살펴 보았습니다. 그럼 이제 하둡을 직접 설치해 보겠습니다.

노드 구성은 네임 노드 1대, 보조 네임노드 1대, 그리고 데이터 노드 2대로 구성하겠습니다. 만일 실제로 노드를 구성할 수 없거나, VMWare 등 가상화 서비스를 사용하더라도 시스템 사양이 따라주지 않는다면 4대의 노드를 테스트를 수행하지 못할 가능성이 있습니다. 그렇더라도 1대의 노드가 마치 4대의 노드인 것처럼 설정한다면 가상화 서비스를 이용해 클러스터 환경을 구성해 볼 수 있습니다.

본 교재에서 우선 노드들을 설치하기 위한 운영체제는 Cent OS 6.4 x86 32bit입니다. 운영체제 설치 시 SSH를 설치해야 합니다. SSH는 디폴트로 선택되어 있기 때문에 설치 시 설정 변경이 없다면 디폴트로 설치됩니다.

그리고 모든 노드에 동일하게 hadoop 계정을 생성합니다. Lab 파일에는 hadoop 계정(비밀번호 hadoop)이 생성되어 있습니다.(root 비밀번호는 hadoop입니다.)

## JDK와 Eclipse 설치

### ◆ JDK

- <http://java.sun.com>
- Java SE 1.7 다운로드(Java SE 1.6이상 가능)
- # tar -xvf jdk-7u25-linux-x64.tar.gz
- # chown -R hadoop:hadoop jdk1.7.0\_51/

### ◆ Eclipse

- 자바 개발 도구
- <http://eclipse.org>
- Eclipse IDE for Java EE Developers 다운로드
- # tar -xvf eclipse-jee-kepler-R-linux-gtk-x86\_64.tar.gz
- # chown -R hadoop:hadoop eclipse/

### JDK 설치

<http://java.sun.com> 에서 Java SE 7버전을 다운로드 받습니다.(Java8버전으로 설치해도 됩니다.)  
Lab파일에서는 /home/hadoop/Downloads/ 에 다운로드 받고 설치합니다. JDK 설치하는 압축만 풀면 됩니다.

```
[hadoop@master ~]$ su -
Password : hadoop
[root@master ~]# cd /usr/local
[root@master local]# pwd
/usr/local
[root@master local]# tar -xvf /home/hadoop/Downloads/jdk-7u51-linux-i586.tar.gz
[root@master local]# chown -R hadoop:hadoop /usr/local/jdk1.7.0_51/
```

### Eclipse

이클립스는 자바 개발 도구입니다. <http://www.eclipse.org/downloads> 에서 다운로드합니다.  
현재 버전은 Eclipse Kepler(4.3) 버전 입니다. 이 중에서 Eclipse IDE for Java EE Developers를 다운로드 받습니다.

이클립스 설치하는 압축만 풀면 됩니다. root 계정으로 압축을 풀고 hadoop 사용자에게 읽기 권한을 줘야 합니다.

```
[root@master local]# tar -xvf /home/hadoop/Downloads/eclipse-jee-kepler-SR1-linux-gtk.tar.gz
[root@master local]# chown -R hadoop:hadoop /usr/local/eclipse/
```

## Hadoop 다운로드 및 설치

### ◆ 다운로드

- <http://hadoop.apache.org>
- [hadoop-2.2.0.tar.gz](#)

### ◆ 설치(압축 해제)

- `# cd /usr/local/hadoop/`
- `# tar -xvf hadoop-2.2.0.tar.gz`
- `# chown -R hadoop:hadoop hadoop-2.2.0/`

설치 디렉토리는 다른 곳으로 해도 무관합니다. 본 교재에서는 하둡과 관련된 파일은 /home/hadoop/Downloads/ 디렉토리에 다운로드 한 다음 /usr/local/ 디렉토리에서 tar 명령을 실행하여 설치합니다.

```
[root@master ~]# cd /usr/local
[root@master local]# tar -xvf /home/hadoop/Downloads/hadoop-2.2.0.tar.gz
[root@master local]# chown -R hadoop:hadoop /usr/local/hadoop-2.2.0/
```

압축을 풀면 /usr/local/hadoop/hadoop-2.2.0 디렉토리가 생성됩니다.

```
[root@master local]# ls -l
total 48
drwxr-xr-x. 2 root root 4096 Sep 23 2011 bin
drwxrwsr-x. 9 hadoop hadoop 4096 Sep 19 02:17 eclipse
drwxr-xr-x. 2 root root 4096 Sep 23 2011 etc
drwxr-xr-x. 2 root root 4096 Sep 23 2011 games
drwxr-xr-x. 9 hadoop hadoop 4096 Oct 6 23:46 hadoop-2.2.0
drwxr-xr-x. 2 root root 4096 Sep 23 2011 include
drwxr-xr-x. 8 hadoop hadoop 4096 Dec 18 19:25 jdk1.7.0_51
drwxr-xr-x. 2 root root 4096 Sep 23 2011 lib
drwxr-xr-x. 2 root root 4096 Sep 23 2011 libexec
drwxr-xr-x. 2 root root 4096 Sep 23 2011 sbin
drwxr-xr-x. 5 root root 4096 Feb 18 2014 share
drwxr-xr-x. 2 root root 4096 Sep 23 2011 src
[root@master local]# exit
logout
```

## .bash\_profile 환경설정

```
$ vi .bash_profile
export PATH=$PATH:$HOME/bin
export JAVA_HOME=/usr/local/jdk1.7.0_51
export HADOOP_INSTALL=/usr/local/hadoop-2.2.0
export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_INSTALL/bin

$ source .bash_profile
  ➤ 환경변수 반영

$ hadoop version

$ java -version

$ javac -version
```

.bash\_profile 파일은 모든 노드에 동일하게 설정되어 있어야 합니다.

```
[hadoop@master Downloads]$ cd
```

```
[hadoop@master ~]$ vi .bash_profile
export PATH=$PATH:$HOME/bin
export JAVA_HOME=/usr/local/jdk1.7.0_51
export HADOOP_INSTALL=/usr/local/hadoop-2.2.0
export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_INSTALL/bin
```

```
[hadoop@master ~]$ source .bash_profile
[hadoop@master ~]$ hadoop version
Hadoop 2.2.0
Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768
Compiled by hortonmu on 2013-10-07T06:28Z
Compiled with protoc 2.5.0
From source with checksum 79e53ce7994d1628b240f09af91e1af4
This command was run using /usr/local/hadoop-2.2.0/share/hadoop/common/hadoop-common-2.2.0.jar
[hadoop@master ~]$ java -version
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) Server VM (build 24.51-b03, mixed mode)
[hadoop@master ~]$ javac -version
javac 1.7.0_51
[hadoop@master ~]$
```



## /etc/hosts 파일 설정(실습 준비작업)

- ◆ 두 대의 호스트로 테스트.
  - 호스트 1 : master, backup, slave2
  - 호스트 2 : slave1
- ◆ 아이피 확인
  - # ifconfig
- ◆ 호스트파일 변경
  - # vi /etc/hosts
  - 192.168.224.152            master backup slave2
  - 192.168.224.149           slave1
- ◆ 방화벽 수정(iptables 파일에서 아이피 3번째 자리 수정)
  - # vi /etc/sysconfig/iptables
  - # service iptables restart

VMWare를 여러 대 설정한다면 각 노드들의 IP로 설정하면 됩니다.

/etc/hosts 파일은 모든 노드에 동일하게 설정되어야 합니다.

127.0.0.1은 Hadoop에서 소켓 바인딩이 될 때 장애 발생하는 경우가 있습니다.

테스트할 때 노드 하나로 테스트 한다면 같은 IP로 설정합니다. 꼭 아이피가 위의 내용과 같을 필요는 없습니다. ifconfig로 네트워크 설정 정보를 확인 한 다음 해당 노드의 아이피로 설정하세요.

```
[hadoop@master Desktop]$ cat /etc/hosts
192.168.224.152 master backup slave2 node1
192.168.224.149 slave1 node2
192.168.224.155 master2 ganglia node3
```

실습 환경에서 slave1과 slave2를 동일 호스트로 설정 시 나중에 실행되는 Datanode가 실행되지 않을 수 있습니다.

## SSH 설정

<ul style="list-style-type: none"> <li>◆ master 서버에서</li> <li>\$ ssh-keygen -t rsa</li> <li>\$ cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys</li> <li>\$ chmod 755 ~/.ssh</li> <li>\$ chmod 644 ~/.ssh/authorized_keys</li> <li>◆ master 서버에서 공개 키 분배</li> <li>\$ scp ~/.ssh/authorized_keys hadoop@대상서버IP:~/.ssh/</li> <li>◆ master 서버에서 테스트</li> <li>\$ ssh hadoop@master date</li> <li>\$ ssh hadoop@backup date</li> <li>\$ ssh hadoop@slave1 date</li> <li>\$ ssh hadoop@slave2 date</li> </ul>	<ul style="list-style-type: none"> <li>◆ 다른 서버에서(slave1)</li> <li>\$ mkdir .ssh</li> <li>\$ chmod 755 ~/.ssh</li> </ul>
--	---

```
[hadoop@master ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa): [Enter]
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase): [Enter]
Enter same passphrase again: [Enter]
Your identification has been saved in /home/hadoop/.ssh/id_rsa.
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub.
The key fingerprint is:
0f:17:f6:e9:ad:1f:ef:87:a6:1f:50:09:f6:ef:1d:d4 hadoop@master
The key's randomart image is:
+--[ RSA 2048 ]-----+
|           o         |
|          . o . .    |
|           o  +.E    |
|          . o o . .   |
|         S . +  . .   |
|          + . o . o   |
|           . . +.o    |
|           .o+.      |
|           o=o.+     |
+-----+

```

## SSH 설정

```
[hadoop@master ~]$ chmod 755 .ssh
[hadoop@master ~]$ cd .ssh
[hadoop@master .ssh]$ ls
id_rsa      ←개인키
id_rsa.pub  ←공개키
[hadoop@master .ssh]$ cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
[hadoop@master .ssh]$ chmod 644 ~/.ssh/authorized_keys
```

### 다른 서버에서(slave1)

```
[hadoop@slave1 ~]$ mkdir .ssh
[hadoop@slave1 ~]$ chmod 755 ~/.ssh
```

하나의 노드로  
테스트할 경우  
는 진행하지 않  
아도 됩니다.

### master 서버에서 공개 키 분배

```
[hadoop@master .ssh]$ scp authorized_keys hadoop@slave1:~/.ssh/
The authenticity of host 'slave1 (192.168.224.136)' can't be established.
RSA key fingerprint is 06:35:fc:e2:da:dd:3d:56:44:de:00:b9:b1:99:27:31.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'slave1,192.168.224.136' (RSA) to the list of known hosts.
hadoop@slave1's password: [slave1의 hadoop 계정 비밀번호 입력]
authorized_keys                                100% 395    0.4KB/s   00:00
[hadoop@master .ssh]$ ssh hadoop@slave1
Last login: Mon May 12 20:23:06 2014 from slave1
[hadoop@slave1 ~]$ date
Mon May 12 20:29:20 PDT 2014
```

이 때 비밀번호를  
 물어보면 안됩니다.

공개 키 분배는 모든 slave 노드에 해 줘야 합니다.

## 하둡 환경 설정 파일

- ◆ **hadoop-env.sh**
  - JDK경로, 클래스 패스, 데몬 실행 옵션 등 설정
- ◆ **slaves**
  - 데이터 노드 설정
- ◆ **core-site.xml**
  - HDFS와 맵리듀스에서 공통적으로 사용할 환경정보 설정
- ◆ **hdfs-site.xml**
  - HDFS에서 사용할 환경 정보 설정
- ◆ **mapred-site.xml**
  - 맵리듀스에서 사용할 환경정보 설정
- ◆ **yarn-site.xml**
  - 맵리듀스 프레임워크에서 사용하는 셔플 서비스를 지정

- **hadoop-env.sh**

하둡을 실행하는 쉘 스크립트 파일에서 필요한 환경변수를 설정합니다. 하둡 홈 디렉토리의 아래에 있는 bin 디렉토리에 있는 쉘 스크립트 파일이 hadoop-env.sh 를 사용합니다. 이 파일에는 JDK경로, 클래스 패스, 데몬 실행 옵션 등 다양한 환경 변수를 설정할 수 있습니다.

- **slaves**

데이터 노드를 실행할 서버를 설정합니다. 데이터 노드가 여러 개라면 라인단위로 서버이름을 설정하면 됩니다.

- **core-site.xml**

HDFS와 맵리듀스에서 공통적으로 사용할 환경정보 설정합니다. hadoop-core-1.0.3.jar 파일에 포함되어 있는 core-default.xml을 오버라이드 한 파일입니다. core-site.xml에 설정 값이 없을 경우 core-default.xml에 있는 기본 값을 사용합니다.

- **hdfs-site.xml**

HDFS에서 사용할 환경 정보를 설정합니다. hadoop-core-1.0.3.jar 파일에 포함되어 있는 hdfs-default.xml을 오버라이드 한 파일입니다. hdfs-site.xml에 설정 값이 없을 경우 hdfs-default.xml에 있는 기본 값을 사용합니다.

- **mapred-site.xml**

맵리듀스에서 사용할 환경정보를 설정합니다. hadoop-core-1.0.3.jar 파일에 포함되어 있는 mapred-default.xml을 오버라이드 한 파일입니다. mapred-site.xml에 설정 값이 없을 경우 mapred-default.xml에 있는 기본 값을 사용합니다.

- **yarn-site.xml**

맵리듀스 프레임워크에서 사용하는 셔플 서비스를 지정합니다.

## hadoop-env.sh 설정

◆ JDK경로, 클래스 패스, 데몬 실행 옵션 등 다양한 환경 변수를 설정

```
# chown -R hadoop:hadoop /usr/local/hadoop-2.2.0
```

➤ 소유주 변경

```
# su - hadoop
```

```
$ cd /usr/local/hadoop-2.2.0/etc/hadoop/
```

```
$ vi hadoop-env.sh
```

➤ export JAVA\_HOME=/usr/local/jdk1.7.0\_51

➤ 자바 설치된 경로 확인 후 설정하세요

hadoop-env.sh 파일은 하둡을 실행하는 쉘 스크립트 파일들에서 필요로 하는 환경변수들을 설정합니다. 즉, 하둡 홈 디렉토리의 아래에 있는 bin 디렉토리에 있는 쉘 스크립트 파일이 hadoop-env.sh 를 사용합니다.

이 파일에는 JDK경로, 클래스 패스, 데몬 실행 옵션 등 다양한 환경 변수를 설정할 수 있습니다.

하둡을 설치하고 테스트 할 때 아래와 같이 경고메시지가 계속 뜰 수 있습니다.

"Warning: \$HADOOP\_HOME is deprecated"

이럴 때 아래의 내용을 hadoop-env.sh 파일에 추가해 주세요.

```
export HADOOP_HOME_WARN_SUPPRESS=1
```

## 노드 설정

### ◆ 데이터노드 설정

```
$ cd /usr/local/hadoop-2.2.0/etc/hadoop/
```

```
$ vi slaves
```

```
slave1
```

```
slave2
```

### ◆ 하둡 1.x에 있던 secondary namenode를 설정했던 masters 파일은 사라짐

- hdfs-site.xml 파일에 dfs.namenode.secondary.http-address 속성을 통해 설정함

```
[hadoop@master ~]$ cd /usr/local/hadoop-2.2.0/etc/hadoop/
```

```
[hadoop@master hadoop]$ ls -l
```

```
total 120
-rw-r--r--. 1 hadoop hadoop 3560 Oct  6 23:38 capacity-scheduler.xml
-rw-r--r--. 1 hadoop hadoop 1335 Oct  6 23:38 configuration.xml
-rw-r--r--. 1 hadoop hadoop  318 Oct  6 23:38 container-executor.cfg
-rw-r--r--. 1 hadoop hadoop  978 Feb 17 20:57 core-site.xml
-rw-r--r--. 1 hadoop hadoop 3589 Oct  6 23:38 hadoop-env.cmd
-rw-r--r--. 1 hadoop hadoop 3390 Oct  6 23:38 hadoop-env.sh
-rw-r--r--. 1 hadoop hadoop 1774 Oct  6 23:38 hadoop-metrics2.properties
-rw-r--r--. 1 hadoop hadoop 2490 Oct  6 23:38 hadoop-metrics.properties
-rw-r--r--. 1 hadoop hadoop 9257 Oct  6 23:38 hadoop-policy.xml
-rw-r--r--. 1 hadoop hadoop  940 Feb 17 21:21 hdfs-site.xml
-rw-r--r--. 1 hadoop hadoop 1180 Oct  6 23:38 httpfs-env.sh
-rw-r--r--. 1 hadoop hadoop 1657 Oct  6 23:38 httpfs-log4j.properties
-rw-r--r--. 1 hadoop hadoop   21 Oct  6 23:38 httpfs-signature.secret
-rw-r--r--. 1 hadoop hadoop  620 Oct  6 23:38 httpfs-site.xml
-rw-r--r--. 1 hadoop hadoop 9116 Oct  6 23:38 log4j.properties
-rw-r--r--. 1 hadoop hadoop  918 Oct  6 23:38 mapred-env.cmd
-rw-r--r--. 1 hadoop hadoop 1383 Oct  6 23:38 mapred-env.sh
-rw-r--r--. 1 hadoop hadoop 4113 Oct  6 23:38 mapred-queues.xml.template
-rw-r--r--. 1 hadoop hadoop  758 Oct  6 23:38 mapred-site.xml.template
-rw-r--r--. 1 hadoop hadoop   14 Feb 17 19:36 slaves
-rw-r--r--. 1 hadoop hadoop 2316 Oct  6 23:38 ssl-client.xml.example
-rw-r--r--. 1 hadoop hadoop 2251 Oct  6 23:38 ssl-server.xml.example
-rw-r--r--. 1 hadoop hadoop 2178 Oct  6 23:38 yarn-env.cmd
-rw-r--r--. 1 hadoop hadoop 4084 Oct  6 23:38 yarn-env.sh
-rw-r--r--. 1 hadoop hadoop  690 Oct  6 23:38 yarn-site.xml
```

## core-site.xml 파일 수정

- ◆ HDFS와 맵리듀스에 공통적으로 사용되는 IO와 같은 하둡 코어를 위한 환경을 설정하는 파일
- ◆ 로그파일, 네트워크 튜닝, I/O 튜닝, 파일 시스템 튜닝, 압축 등 하부 시스템 설정파일

# vi core-site.xml

- fs.defaultFS
  - hdfs://master:9000
  - 1.x의 속성 이름은 fs.default.name
- hadoop.tmp.dir
  - /usr/local/hadoop-2.2.0/tmp
- io.file.buffer.size
  - 버퍼의 크기, 디폴트 값은 4096(4K)
  - 성능상의 이점을 얻게 하기 위해 128KB(131,072바이트)를 일반적으로 사용

core-site.xml 파일은 HDFS와 맵리듀스에서 공통적으로 사용할 환경정보 설정합니다.

HADOOP\_INSTALL/share/hadoop/common/hadoop-common-2.2.0.jar 파일에 포함되어 있는 core-default.xml을 오버라이드 한 파일입니다. core-site.xml에 설정 값이 없을 경우 core-default.xml에 있는 기본 값을 사용합니다.

```
[hadoop@master hadoop]$ vi core-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://master:9000</value>
  </property>
  <property>
    <name>io.file.buffer.size</name>
    <value>131072</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/local/hadoop-2.2.0/tmp</value>
  </property>
</configuration>
```

공통 속성들에 대한 자세한 설명은 다음 주소를 참고하세요.

<http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/core-default.xml>

## core-site.xml 파일 - 파일 삭제와 휴지통

파일을 삭제했을 경우 휴지통에 임시 보관 시킬 수 있습니다. 이때 사용하는 속성이 fs.trash.interval 과 fs.trash.checkpoint.interval 속성 입니다.

fs.trash.interval

- 파일을 삭제할 때 휴지통에 보관할 시간(분 단위)을 지정합니다.
- 기본 값은 0입니다.
- 0이면 휴지통에 보관하지 않고 바로 영구 삭제 합니다.

fs.trash.checkpoint.interval

- 체크포인트 시간 간격을 지정합니다.
- 기본 값은 0입니다.
- fs.trash.interval 값 보다 작거나 같아야 합니다.
- 만일 0이면 fs.trash.interval 값과 같게 설정됩니다.

휴지통

- 휴지통은 .Trash 폴더입니다.
- 사용자의 폴더가 /user/hadoop 이라면 휴지통의 경로는 /user/hadoop/.Trash 입니다.
- 휴지통을 비우려면 hdfs dfs -expunge 명령을 이용합니다. 이는 휴지통에 설정된 최소 기간 보다 오래 있었던 파일을 지울 수 있습니다
- 복원은 .Trash의 하위 디렉토리에서 파일을 찾은 후 하위 디렉토리 밖으로 이동시키면 됩니다.

다음 설정은 삭제된 파일을 휴지통에 보관할 시간은 10분, 체크포인트 간격은 5분으로 설정한 예입니다.

```
<property>
  <name>fs.trash.interval</name>
  <value>10</value>
</property>
<property>
  <name>fs.trash.checkpoint.interval</name>
  <value>5</value>
</property>
```



## hdfs-site.xml 설정

### ◆ HDFS 의 설정 파일

\$ vi hdfs-site.xml

- dfs.replication
  - 데이터 복제 수를 결정함. 디폴트는 3
  - 실습환경에서 데이터 노드가 2개이므로 복제 수는 1로 설정한다.
- dfs.namenode.name.dir : fsimage 파일이 저장될 경로
- dfs.namenode.edits.dir : edits 파일이 저장될 경로
- dfs.datanode.data.dir : 데이터가 저장되는 데이터 노드의 경로
- dfs.namenode.checkpoint.dir : fsimage와 edits 사본이 저장되는 보조 네임 노드의 경로
- dfs.namenode.secondary.http-address
  - 보조 네임노드의 주소와 포트 번호 지정. 디폴트는 0.0.0.0:50090
  - backup:50090
- dfs.hosts.exclude
  - /usr/local/hadoop-2.2.0/etc/hadoop/exclude
- dfs.hosts
  - /usr/local/hadoop-2.2.0/etc/hadoop/include
- dfs.permissions.enabled
  - true로 하면 HDFS에서 권한을 체크하도록 한다. false로 하면 권한 체크를 해제 한다. 디폴트는 true이며, false로 설정한다.

exclude 파일과 include 파일을 만들어야 합니다. 절대경로로 입력 합니다.

hdfs-site.xml 파일은 HDFS에서 사용할 환경 정보를 설정합니다.

HADOOP\_INSTALL/share/hadoop/hdfs/hadoop-hdfs-2.2.0.jar 파일에 포함되어 있는 hdfs-default.xml을 오버라이드 한 파일입니다. hdfs-site.xml에 설정 값이 없을 경우 hdfs-default.xml에 있는 기본 값을 사용합니다.

[hadoop@master hadoop]\$ vi hdfs-site.xml

```
<?xml version="1.0"?>
```

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
```

```
<configuration>
```

```
  <property>
```

```
    <name>dfs.replication</name>
```

```
    <value>1</value>
```

```
  </property>
```

```
  <property>
```

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

```
    <value>file:///home/hadoop/dfs/name</value>
```

```
  </property>
```

블록의 크기 변경은 dfs.blocksize 속성을 이용합니다. 디폴트 블록의 크기는 128m입니다. 값은 단위를 사용할 수 있으며 단위는 소문자만 가능합니다.

## hdfs-site.xml 설정

```
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///home/hadoop/dfs/data</value>
</property>
<property>
  <name>dfs.namenode.checkpoint.dir</name>
  <value>file:///home/hadoop/dfs/namesecondary</value>
</property>
<property>
  <name>dfs.namenode.secondary.http-address</name>
  <value>backup:50090</value>
</property>
<property>
  <name>dfs.hosts.exclude</name>
  <value>/usr/local/hadoop-2.2.0/etc/hadoop/exclude</value>
</property>
<property>
  <name>dfs.hosts</name>
  <value>/usr/local/hadoop-2.2.0/etc/hadoop/include</value>
</property>
<property>
  <name>dfs.permissions.enabled</name>
  <value>false</value>
</property>
</configuration>
```

NameNode에 저장되는 fsimage(파일시스템이미지) 파일의 경로는 dfs.namenode.name.dir 속성을 이용해 설정할 수 있습니다. 기본 값은 file://\${hadoop.tmp.dir}/dfs/name 입니다. 콤마를 이용하여 여러 개 경로를 지정할 수 있습니다.

```
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///dev/sdb1</value>
</property>
```

NameNode에 저장되는 edits(에디트로그) 파일의 경로는 dfs.namenode.edits.dir 속성을 이용해 설정할 수 있습니다. 기본 값은 \${dfs.namenode.edits.dir} 입니다. 콤마를 이용하여 여러 개 경로를 지정할 수 있습니다.

```
<property>
  <name>dfs.namenode.edits.dir</name>
  <value>file:///dev/sdc1, file:///dev/sdd1</value>
</property>
```

## hdfs-site.xml 설정

데이터 파일이 저장되는 파일의 경로는 dfs.datanode.data.dir 속성을 이용해 설정할 수 있습니다. 기본 값은 file://\${hadoop.tmp.dir}/dfs/data 입니다. 콤마를 이용하여 여러 개 경로를 지정할 수 있습니다.

```
<property>
  <name>dfs.datanode.data.dir </name>
  <value>file:///dev/sdb1, file:///dev/sdc1, file:///dev/sdd1</value>
</property>
```

보조네임노드에 저장되는 fsimage와 edits 사본의 경로는 dfs.namenode.checkpoint.dir 속성을 이용해 설정할 수 있습니다. 기본 값은 file://\${hadoop.tmp.dir}/dfs/namesecondary 입니다. 콤마를 이용하여 여러 개 경로를 지정할 수 있습니다.

```
<property>
  <name>dfs.namenode.checkpoint.dir </name>
  <value>file:///dev/sdb1</value>
</property>
```

HDFS 속성들에 대한 자세한 설명은 다음 주소를 참고하세요.

<http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml>

## mapred-site.xml 파일 수정

- ◆ \$ cp mapred-site.xml.template mapred-site.xml
- ◆ \$ vi mapred-site.xml
  - mapreduce.framework.name
    - 하둡 2.0부터 맵리듀스 프레임워크를 지정할 수 있음.
  - mapreduce.jobtracker.hosts.exclude.filename
    - /usr/local/hadoop-2.2.0/etc/hadoop/exclude
  - mapreduce.jobtracker.hosts.filename
    - /usr/local/hadoop-2.2.0/etc/hadoop/include
  - mapred.map.child.java.opts                -Xmx200m
  - mapred.reduce.child.java.opts            -Xmx200m
  - mapreduce.job.maps                        2
  - mapreduce.job.reduces                    1

mapred-site.xml 파일은 맵리듀스에서 사용할 환경정보를 설정합니다.

HADOOP\_INSTALL/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.2.0.jar 파일에 포함되어 있는 mapred-default.xml을 오버라이드 한 파일입니다. mapred-site.xml에 설정 값이 없을 경우 mapred-default.xml에 있는 기본 값을 사용합니다.

```
[hadoop@master hadoop]$ cp mapred-site.xml.template mapred-site.xml
```

```
[hadoop@master hadoop]$ vi mapred-site.xml
```

```
<?xml version="1.0"?>
```

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
```

```
<configuration>
```

```
  <property>
```

```
    <name>mapreduce.framework.name</name>
```

```
    <value>yarn</value>
```

```
  </property>
```

```
  <property>
```

```
    <name>mapreduce.jobtracker.hosts.exclude.filename</name>
```

```
    <value>/usr/local/hadoop-2.2.0/etc/hadoop/exclude</value>
```

```
  </property>
```

```
  <property>
```

```
    <name>mapreduce.jobtracker.hosts.filename</name>
```

```
    <value>/usr/local/hadoop-2.2.0/etc/hadoop/include</value>
```

```
  </property>
```

```
</configuration>
```

속성들에 대한 자세한 설명은 다음 주소를 참고하세요.

<http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml>

위 주소 대신 <http://hadoop.apache.org/> 에서 Documentation -> Current 창에서 화면 외쪽 아래에 Configuration 메뉴 아래에 xml 문서들에 대한 속성 설명 페이지를 볼 수 있습니다

## yarn-site.xml 파일 수정

- ◆ \$ vi yarn-site.xml
- ◆ yarn.nodemanager.aux-services
  - mapreduce\_shuffle
  - 맵리듀스 프레임워크에서 사용하는 셔플 서비스를 지정한다.
- ◆ yarn.nodemanager.aux-services.mapreduce\_shuffle.class
  - org.apache.hadoop.mapred.ShuffleHandler
- ◆ yarn.resourcemanager.hostname
  - Resource Manager의 호스트 이름을 지정

yarn-site.xml 파일은 안에서 사용할 환경정보를 설정합니다.

HADOOP\_INSTALL/share/hadoop/yarn/hadoop-yarn-common-2.2.0.jar 파일에 포함되어 있는 yarn-default.xml을 오버라이드 한 파일입니다. yarn-site.xml에 설정 값이 없을 경우 yarn-default.xml에 있는 기본 값을 사용합니다.

```
[hadoop@master hadoop]$ vi 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>
</configuration>
```

YARN은 자체 환경설정 파일(yarn-site.xml)을 갖고 있고 그 위에서 동작하는 애플리케이션들 또한 자체 환경설정파일들을 갖게 됩니다. 맵리듀스 프레임워크라면 하둡 1.0에서처럼 mapred-site.xml이 바로 그 환경설정파일인 것입니다.

## 다른 노드 설정파일 동기화

```

◆ 프로그램 전체 동기화
# rsync -av /usr/local/ root@backup:/usr/local
# rsync -av /usr/local/ root@slave1:/usr/local
# rsync -av /usr/local/ root@slave2:/usr/local

◆ 하둡 설정파일만 동기화
$ cd /usr/local/hadoop-2.2.0/etc/hadoop
$ rsync -av . hadoop@backup:/usr/local/hadoop-2.2.0/etc/hadoop
$ rsync -av . hadoop@slave1:/usr/local/hadoop-2.2.0/etc/hadoop
$ rsync -av . hadoop@slave2:/usr/local/hadoop-2.2.0/etc/hadoop

◆ 사용자 환경설정 파일 동기화
$ rsync -av ~/.bash_profile hadoop@backup:./bash_profile
slave1과 slave2도 동기화 한다.

```

시스템 하나로 모든 노드를 테스트 한다면 동기화는 하지 않아도 됩니다.

### rsync

rsync는 rcp(remote file copy)의 강화형으로 "rsync 알고리즘"이라는 원격상의 파일의 전송이나 동기화를 가장 빠른 속도로 구현하는 방법을 사용하는 오픈 소스 유틸리티로 GPL에 기초하기 때문에 누구나 자유로이 이용이 가능합니다.

다음은 rsync의 특징입니다.

- 1) 디렉토리 트리나 파일시스템 전체를 갱신할 수 있습니다
- 2) 옵션에 따라 소프트 링크나 파일 소유자, 퍼미션, 디바이스, 타임스탬프 유지가 가능합니다.
- 3) 내부 파이프라인이 복잡한 파일의 레이턴시를 줄입니다.
- 4) 전송에 rsh, ssh 혹은 다이렉트 소켓을 쓰는 것이 가능합니다.
- 5) 이상적 미러링을 가능케 하는 anonymous rsync를 지원합니다.
- 6) 특별한 권한 없이도 실행이 가능합니다.

rsync의 주요 옵션은 -avz 옵션입니다.

-a는 archive mode입니다. 심볼릭 링크, 속성, 퍼미션, 소유권 등을 보존합니다.

-v는 verbose입니다. 실행 시 상세하게 보여줍니다.

-z는 전송시 압축을 수행하여 전송합니다.

## 네임노드 초기화

- ◆ 네임노드는 최초 한번만 실행하면 됨
  - \$ cd /usr/local/hadoop-2.2.0/bin
  - \$ **hdfs namenode -format**
  - 에러메시지가 있다면 환경설정파일이 잘못된 것임. 확인하고 수정한 다음 다시 실행 시킬 것
- ◆ 데이터 노드에 정보가 저장된 후 네임노드 초기화 하면 데이터 노드와 동기화가 되지 않음
  - java.io.IOException: Incompatible namespaceIDs
  - 하둡 설치하고 테스트 후 다시 포맷하면 데이터 노드들이 안 붙는 에러
  - hdfs-site.xml에 설정했던 디렉토리 안의 파일을 모두 삭제 해야 함(Data Node에만 적용)
    - rm -rf ./tmp/dfs/data/\* ← DataNode 에서 실행

네임노드 초기화 시에 에러메시지가 있다면 환경설정파일이 잘못된 것입니다. 에러 메시지를 확인하고 해당 설정파일을 수정한 다음 다시 실행 시키면 됩니다. 다음 그림은 정상 실행된 결과입니다.

```
[hadoop@master 바탕화면]$ hadoop namenode -format
13/07/06 03:59:30 INFO namenode.NameNode: STARTUP_MSG:
/*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = master/192.168.224.128
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 1.1.2
STARTUP_MSG: build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch-1.1 -r 1440782; compiled by 'hortonfo' on Thu Jan 31 02:03:24 UTC 2013
*****/
Re-format filesystem in /usr/local/hadoop/hadoop-1.1.2/dfs/name? (Y or N) Y
13/07/06 03:59:32 INFO util.GSet: VM type = 64-bit
13/07/06 03:59:32 INFO util.GSet: 2% max memory = 17.77875 MB
13/07/06 03:59:32 INFO util.GSet: capacity = 2^21 = 2097152 entries
13/07/06 03:59:32 INFO util.GSet: recommended=2097152, actual=2097152
13/07/06 03:59:33 INFO namenode.FSNamesystem: fsOwner=hadoop
13/07/06 03:59:33 INFO namenode.FSNamesystem: supergroup=supergroup
13/07/06 03:59:33 INFO namenode.FSNamesystem: isPermissionEnabled=true
13/07/06 03:59:33 INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
13/07/06 03:59:33 INFO namenode.FSNamesystem: isAccessTokenEnabled=false accessKeyUpdateInterval=0 min(s), accessTokenLifetime=0 min(s)
13/07/06 03:59:33 INFO namenode.NameNode: Caching file names occurring more than 10 times

13/07/06 03:59:33 INFO common.Storage: Image file of size 112 saved in 0 seconds.
13/07/06 03:59:33 INFO namenode.FSEditLog: closing edit log: position=4, editlog=/usr/local/hadoop/hadoop-1.1.2/dfs/name/current/edits
13/07/06 03:59:33 INFO namenode.FSEditLog: close success: truncate to 4, editlog=/usr/local/hadoop/hadoop-1.1.2/dfs/name/current/edits
13/07/06 03:59:33 INFO common.Storage: Storage directory /usr/local/hadoop/hadoop-1.1.2/dfs/name has been successfully formatted.
13/07/06 03:59:33 INFO namenode.NameNode: SHUTDOWN_MSG:
*****/
SHUTDOWN_MSG: Shutting down NameNode at master/192.168.224.128
*****/
[hadoop@master 바탕화면]$
```



## 실행

- ◆ \$ sbin/hadoop-daemon.sh start namenode
- ◆ \$ sbin/hadoop-daemons.sh start datanode
- ◆ \$ sbin/hadoop-daemon.sh start secondarynamenode
- ◆ \$ sbin/yarn-daemon.sh start resourcemanager
- ◆ \$ sbin/yarn-daemons.sh start nodemanager
- ◆ \$ sbin/mr-jobhistory-daemon.sh start historyserver
- ◆ 브라우저에서 http://master:50070/dfshealth.jsp 실행 후 파일 시스템 상태 보여야 함
- ◆ JobTracker는 http://master:8088/cluster 에서 확인할 수 있음

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

```
...
```

```
[hadoop@master sbin]$ ./mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /usr/local/hadoop-2.2.0/logs/mapred-hadoop-historyserver-master.out
```

```
[hadoop@master sbin]$ jps
4147 DataNode
12373 NameNode
12703 SecondaryNameNode
12851 ResourceManager
13451 Jps
9590 NodeManager
13392 JobHistoryServer
```

Java HotSpot(TM) 64-Bit Server VM warning: You have loaded library /usr/local/hadoop-2.2.0/lib/native/libhadoop.so.1.0.0 which might have disabled stack guard. The VM will try to fix the stack guard now.  
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.  
vi hadoop-env.sh(모든 노드에 설정되어야 합니다.)  
export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=/usr/local/hadoop-2.2.0/lib/native  
export HADOOP\_OPTS="-Djava.library.path=/usr/local/hadoop-2.2.0/lib"

### java.io.IOException: Incompatible namespaceIDs

데이터 노드에 정보가 저장된 후 네임노드 초기화 하면 데이터 노드와 동기화가 되지 않음

java.io.IOException: Incompatible namespaceIDs

하둡 설치하고 테스트 후 다시 포맷하면 데이터 노드들이 안 붙는 예러

hdfs-site.xml에 설정했던 디렉토리 안의 파일을 모두 삭제 해야 함(Data Node에만 적용)

```
rm -rt /usr/local/hadoop-2.2.0/tmp/dfs/data/*
```

### jps

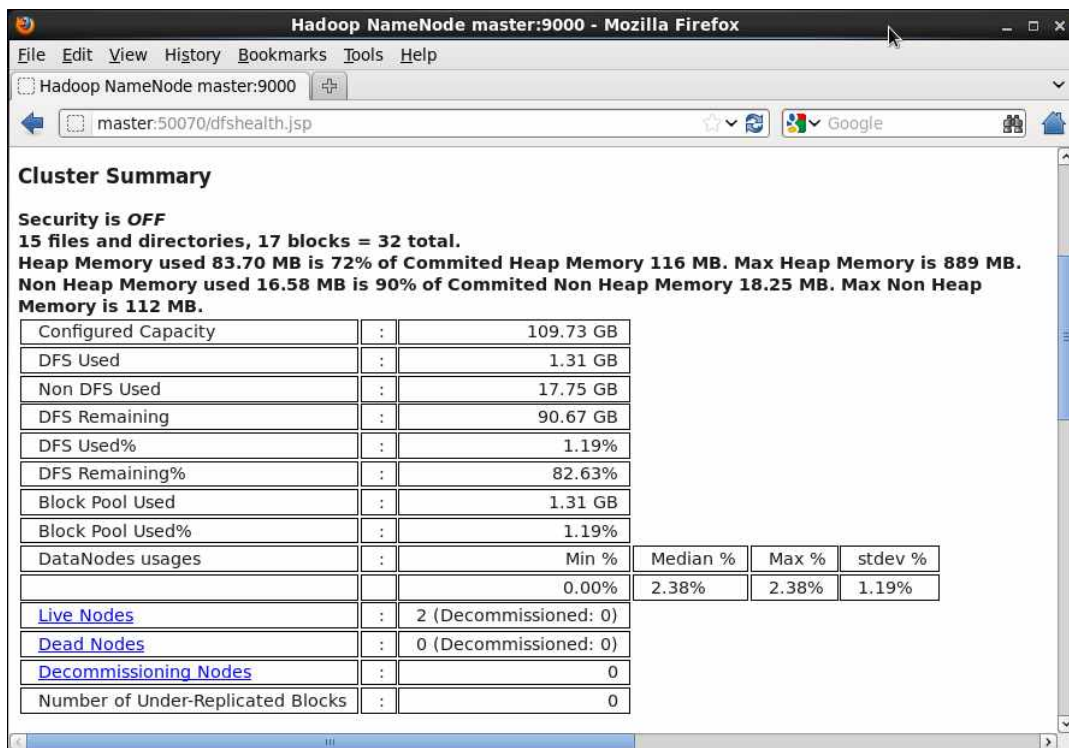
JAVA\_HOME/bin 디렉토리에 존재하는 Java Virtual Machine Process Tool 입니다.

자세한 내용은 <http://cafe.naver.com/javaspecialistgroup/303> 문서를 참고하세요.



## 방화벽 설정

- ◆ <http://master:50070/dfshealth.jsp> 로 확인한 DFS Used% 항목이 100%로 되어 있으면 데이터 노드에 더 이상 저장 공간이 없거나 방화벽에 의해 막혀 있는 경우이다.
- ◆ 방화벽 설정은 모든 노드에서 수행되어야 한다. 즉, datanode 뿐만 아니라 namenode 에도 방화벽 규칙이 추가돼 있어야 한다.
- ◆ # vi /etc/sysconfig/iptables
  - -A INPUT -s 192.168.224.0/24 -d 192.168.224.0/24 -j ACCEPT
  - -A OUTPUT -s 192.168.224.0/24 -d 192.168.224.0/24 -j ACCEPT
- ◆ # service iptables restart
- ◆ 주의 : /etc/hosts 파일 안에 localhost가 있으면 안됩니다.



## 클러스터 초기화

### 1. 클러스터 시작

`start-all.sh` 또는 `start-dfs.sh`로 시작

### 2. 클러스터 내의 모든 데이터 삭제

`hdfs dfs -rm -R /폴더이름`

namenode를 포맷하거나, 데이터노드를 제거하더라도 데이터 블록은 삭제 안 됩니다. 그러므로 클러스터를 초기화 하기 위해서는 하둡 분산파일 시스템을 시작시킨 다음, 클러스터 내의 데이터를 직접 `rm` 명령으로 삭제해야 합니다.

### 3. 클러스터 종료

`stop-dfs.sh` 또는 `stop-all.sh`

### 4. 파일시스템 디렉토리 안의 파일들 삭제(모든 노드에서)

`dfs/name` 디렉토리 내의 모든 파일 삭제

`dfs/namesecondary` 디렉토리 내의 모든 파일 삭제

`dfs/data` 디렉토리 내의 모든 파일 삭제

### 5. logs 디렉토리 내의 모든 파일 삭제(모든 노드에서)

사용자가 실수로 `root` 권한으로 클러스터를 실행했을 경우 `hadoop`계정으로 클러스터 실행 안됨

### 6. 네임노드 포맷

`hdfs namenode -format`

### 7. 클러스터 시작

`sbin/start-all.sh`

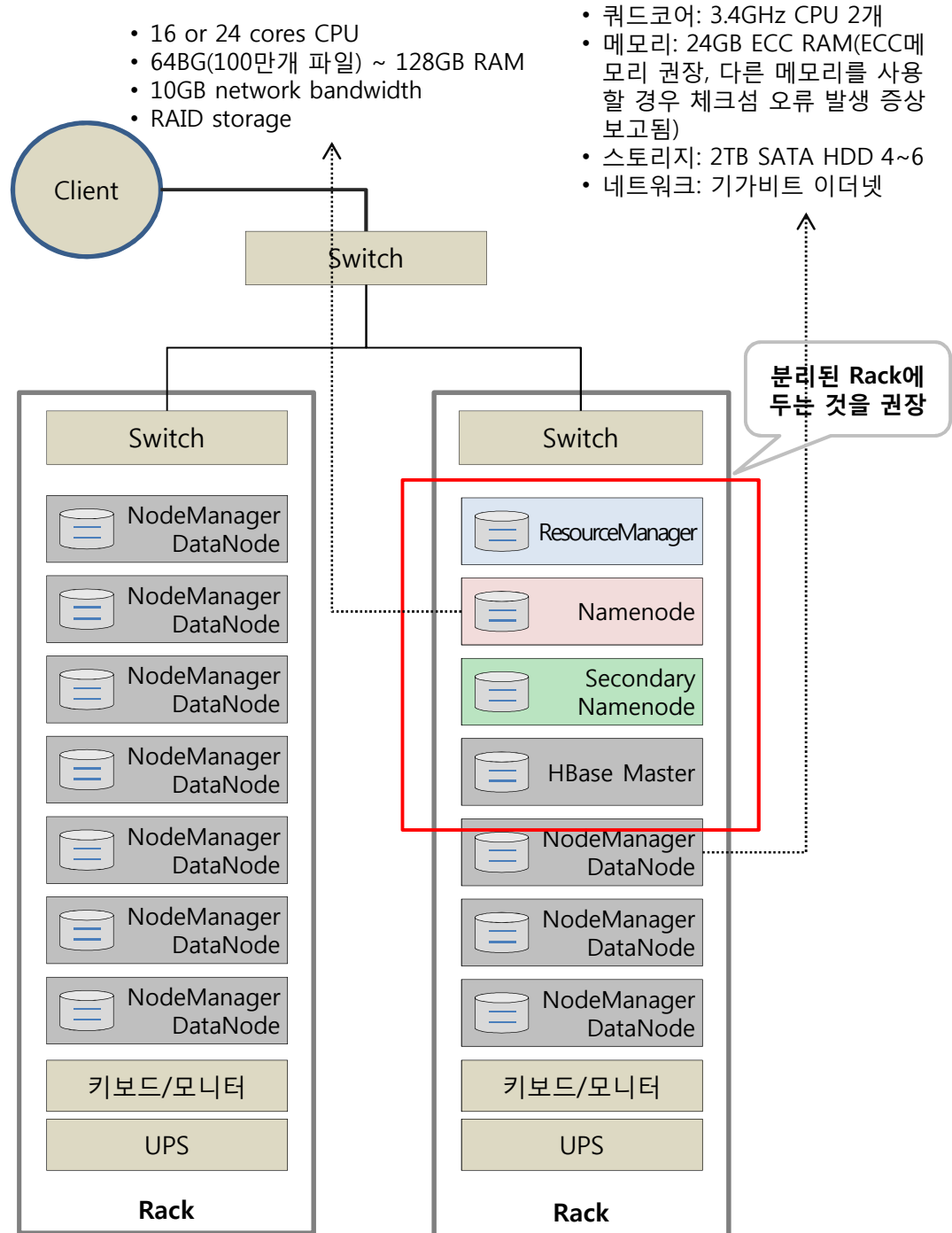
### 8. HDFS에 사용자 홈 디렉토리 생성(리눅스 사용자가 `hadoop`일 경우)

`hdfs dfs -mkdir -p /user/hadoop`

## Lab - 클러스터 구성

1. VMWare 에서 리눅스 설치, Core 2개, Ram 2G, HDD 20G
2. 리눅스 인스턴스 복사 후 디렉토리 이름 및 인스턴스 이름 변경(namenode, datanode1, datanode2, secondarynamenode 이 되도록 리눅스 인스턴스 구성)
3. 모든 노드 /etc/hosts 파일 수정
4. 모든 노드 방화벽 설정 변경 후 방화벽 재시작
5. namenode에서 ssh 설정
6. namenode에서 JDK 다운로드 및 설치
7. namenode에서 Hadoop 다운로드 및 설치
8. namenode에서 .bash\_profile 에 환경변수 추가
9. 하둡 설정파일 수정
10. JDK 폴더와 하둡 폴더를 다른 모든 노드들에 동기화 시킴
11. .bash\_profile 동기화
12. 모든 노드 재시작
13. namenode 포맷
14. 하둡 클러스터 시작
15. <http://master:50060/>으로 클러스터 내 데이터 노드 수 확인
16. /airline 폴더 생성
17. 데이터파일(2008.csv) 파일 업로드
18. `hdfs dfs -ls /airline` 으로 업로드 된 파일 확인

## 하둡 H/W 시스템 구성도

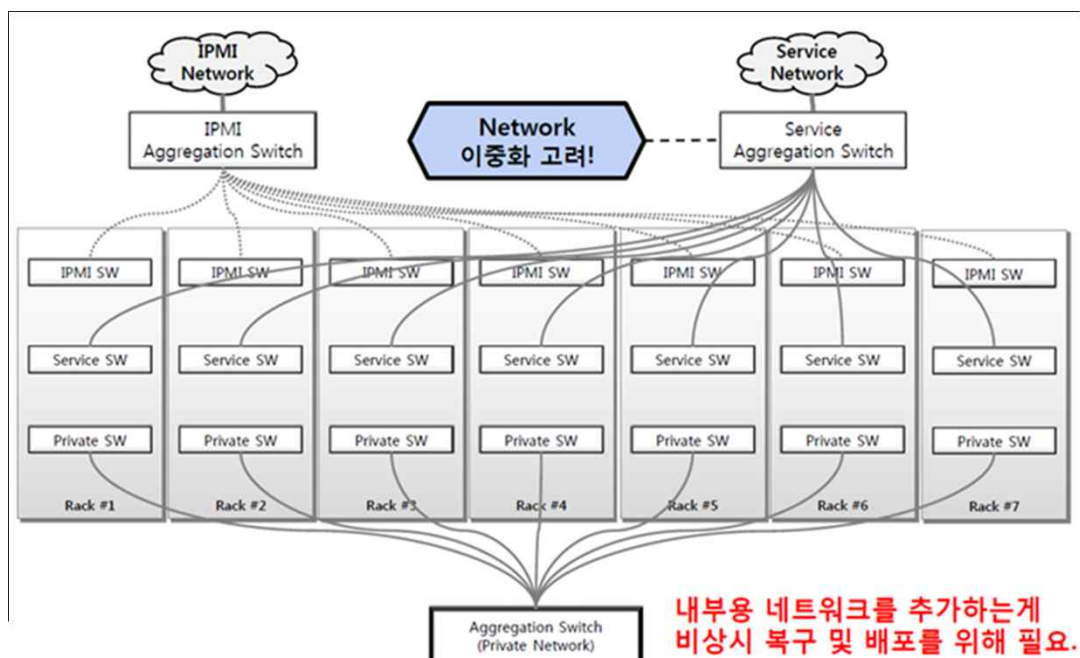


## H/W Planning

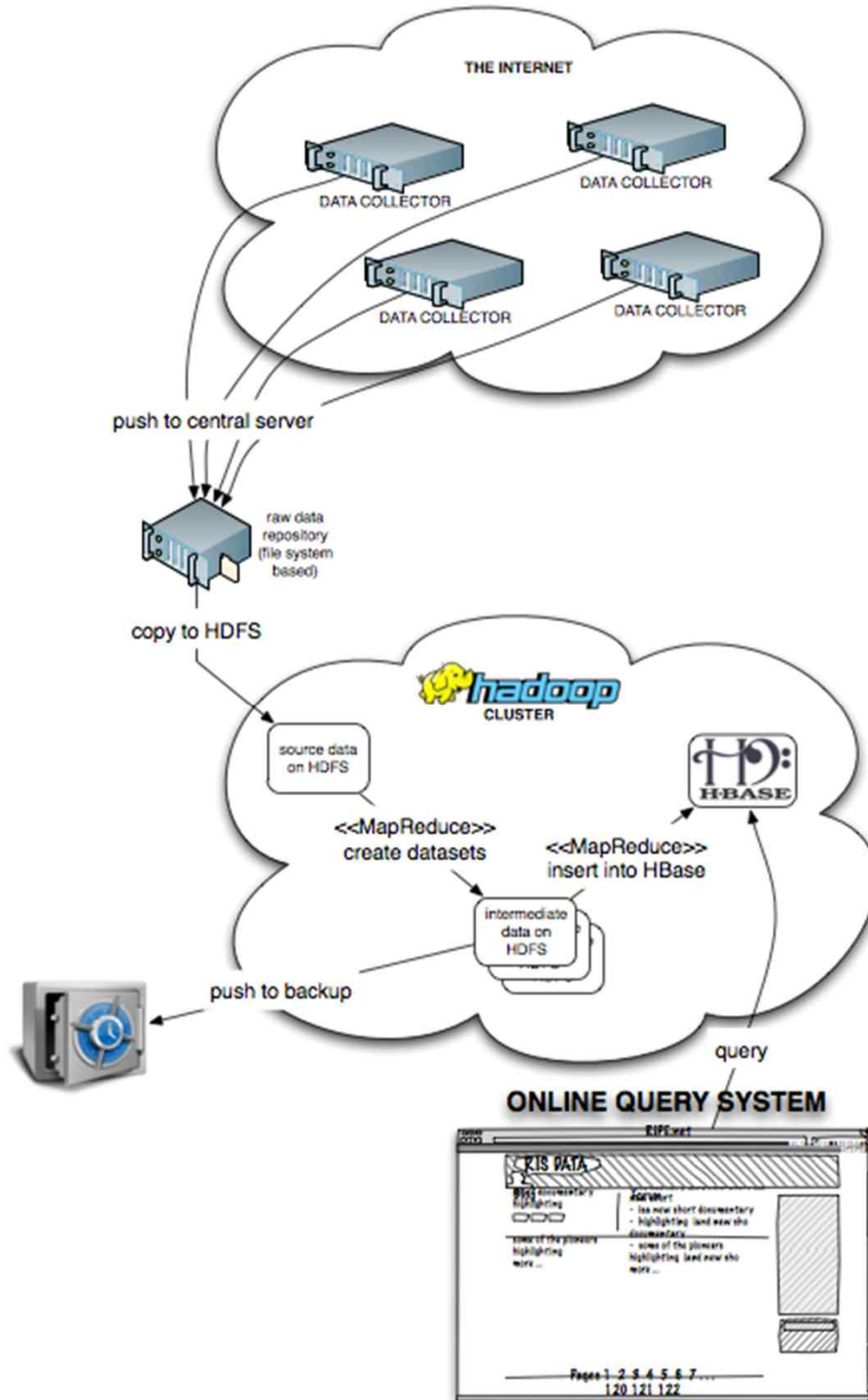
Unit	Selection				etc
Box	1U Hdd 2.5"=8~10, 3.5"=4		2U HDD 2.5"=20, 3.5"=10		1U:처리 성능 중심 2U: 적재량 중심
CPU	4core 30~60만원	8core 60~200만원	12core 160~240만원	16core 280~530만원	성능 검증 필요
RAM	32G	48G	64G	128G	많으면 많을 수록
HDD	2.5"		3.5"		2.5 7200*8=6T, 120만 2.5 SAS 1K*8=24T 272만 3.5 7200*4=12T, 64만 3.5 SAS 1K*4=4T 152만 3.5 SAS 1.5K*4=2.6T 272만
	SATA3 7.2K, 750G: 10만	SAS 10K, 300G: 34만	SATA3 7.2K, 3T: 16만	SAS 10K, 1T: 38만 15K, 650G: 68만	
Network	1G Switch: 48p 100~300만 NIC: 개당 1~20만		10G Switch: 48p 1000만 NIC: 개당 100만		엄청난 가격차이! 1G Bonding 고려
RAID	RAID-0 성능 중심	RAID-1 안정성 중심	RAID-5 안정성 강화	None	사용할 SW특성 고려

\* 예산에 맞는 선택요소의 면밀한 검토가 필요.(노드 당 400~500만원대 목표)

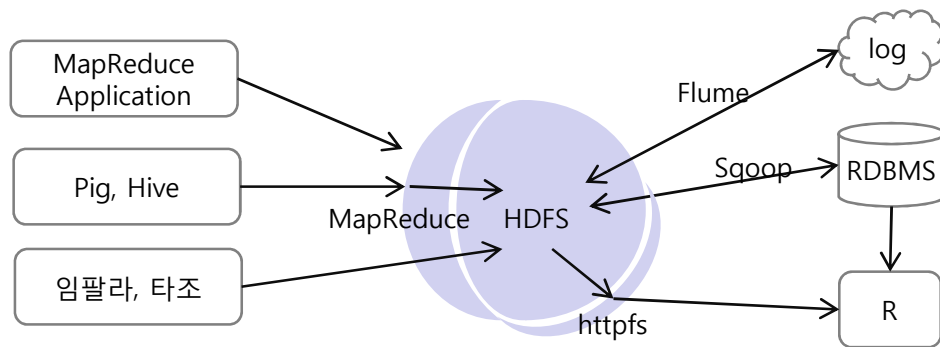
## N/W Planning



## 하둡 클러스터 사용 아키텍처



그림출처 : <http://blog.xebia.com/2010/11/29/going-nosql-at-ripe-ncc/>



이 페이지는 여백 페이지입니다.



# 3 HDFS 관리

---

## Objectives

- HDFS 명령어를 통해 파일 시스템을 사용하는 방법을 배웁니다.
- 안전모드에 대해 배웁니다.
- dfsadmin, fsck 도구에 대해서 배웁니다.
- 맵리듀스 어플리케이션 로깅에 대해서 배웁니다.

HDFS의 노드, 동작 방식

HDFS File System Metadata Backup

HDFS 관리

dfsadmin

안전모드

파일시스템 체크

노드 위임과 해제

하둡 클러스터 복구(Namenode disk failover)

로깅

이 페이지는 여백 페이지입니다.

## HDFS의 노드

- ◆ Namenode
  - 데이터노드 관리
  - 일종의 Master node로 파일에 대한 메타 데이터를 저장
  - SPOF(single point of failure)
  - Checkpoint 관리
- ◆ Secondary Namenode
  - Namenode의 fsimage와 editlog snapshot 저장
- ◆ Datanode
  - Datanodes는 실제 파일을 저장/읽기 수행. 하나의 파일을 블록이라는 단위로 나눠서 저장
  - Block단위는 임의 설정가능
  - Namenodes와 주기적으로 통신하여 저장하고 있는 블록에 대한 정보를 Namenodes에 저장
  - Datanodes에 저장된 파일들은 정책에 의해 자동 분산 저장되며 1대 node에 장애가 나도 서비스 영향 없음.(no single point of failure)

Namenode(네임 노드)는 다음과 같은 역할을 합니다.

- 네임 노드는 입력되는 파일에 대한 저장소선택, 복제 개수 지정 등 데이터 노드를 관리합니다.
- 일종의 Master node로 파일에 대한 메타 데이터를 저장하는 노드로, 디렉토리 구조, 파일에 대한 각종 메타 데이터, 그리고 물리적 파일이 저장되어 있는 위치 등을 저장합니다.
- 데이터 노드들에는 모든 블록에 대한 메타정보가 들어와 있기 때문에, 네임 노드가 장애가 나면 전체 HDFS 이 장애가 나게 됩니다. 이를 SPOF(single point of failure)라 부릅니다. 이것은 어떤 방법으로든 앞으로 해결해야 할 과제입니다.
- 파일 이름, 각 파일의 데이터 노드 위치 등 모든 메타데이터는 메모리 영역에 저장하므로 빠른 접근이 가능합니다.
- Checkpoint 관리를 수행합니다. 이것은 주기적으로 상태를 체크해 빠른 시간에 장애를 인지하고, 대처할 수 있게 도와줍니다
- fsimage : 파일 위치에 대한 매칭과 속성 등 namespace를 저장합니다.
- editlog : 파일 입출력에 대한 트랜잭션 로그를 저장합니다.

Secondary Namenode(보조 네임노드)는 다음과 같은 역할을 합니다.

- 네임노드의 fsimage와 editlog 스냅샷을 저장합니다.
- 보노 네임노드는 Master 서버(네임노드) 장애 시 복구하기 위한 기능이 아닙니다.
- Hadoop을 재 가동할 때 editlog의 트랜잭션 로그를 다시 작성(rebuild) 해주는 역할을 합니다.

## HDFS의 노드

Datanode(데이터 노드)

- 데이터 노드는 실제 파일을 저장/읽기 수행. 하나의 파일을 블록이라는 단위로 나눠서 저장하는 역할을 수행합니다.
- Block단위는 임의 설정이 가능합니다. 디폴트는 128MByte(134217728) 입니다.
- 네임노드와 주기적으로 통신하여 저장하고 있는 블록에 대한 정보를 네임노드에 저장하도록 합니다.
- 데이터 노드에 저장된 파일들은 정책에 의해 자동 분산 저장되며 1대 node에 장애가 나도 서비스 영향 없습니다.(no single point of failure)

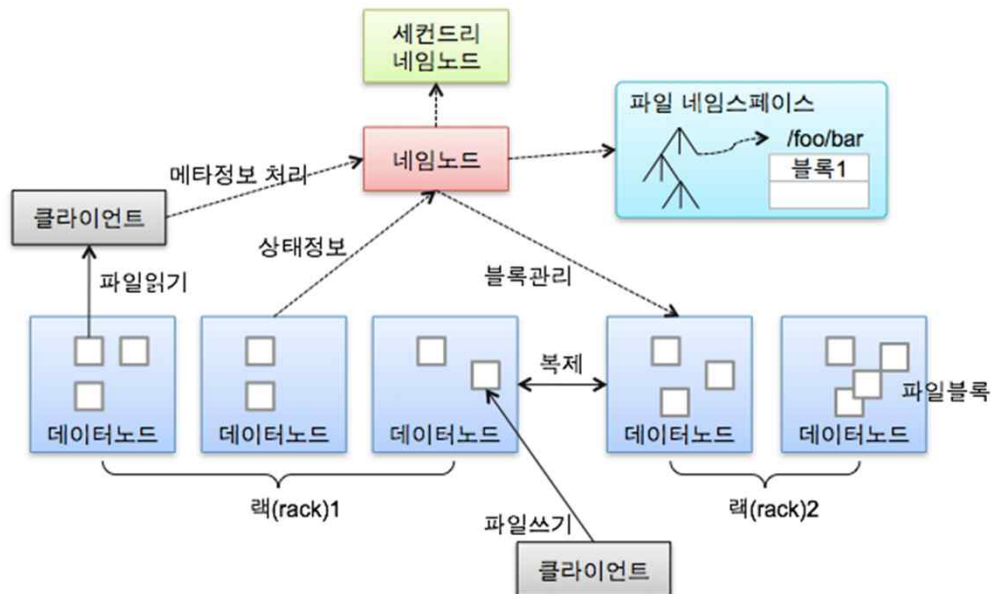
### 파일시스템 (fsimage)

fsimage는 파일시스템에 존재하는 모든 디렉토리와 파일 아이노드 정보를 바이트로 직렬화(디렉토리, 파일명, 상태 정보 등을 바이트 배열로 표현)한 파일입니다. 각 아이노드는 파일이나 디렉토리의 내부적인 표현이며 파일의 복제 단위, 변경 및 접근시간, 접근권한, 블록크기와 파일을 구성하는 블록조합들 같은 정보를 가집니다. 디렉토리는 변경 시간, 권한, 할당 크기 같은 메타데이터가 저장됩니다.

다음은 fsimage의 특징입니다.

- 파일시스템 메타데이터의 영속적인 체크포인트 파일이다.
- 개별 쓰기동작 때마다 갱신되지 않는다.
- 블록이 저장되는 데이터노드 정보를 기록하지는 않는다.
- 네임노드는 메모리상에 블록과 데이터 노드 매핑 정보를 유지합니다.

## HDFS 동작 방식 - Namenode, Datanode



1. (클라이언트) 네임 노드로 파일 패스에 대한 생성을 요청합니다.
2. (네임 노드) 메모리상에 해당 파일의 패스 정보를 생성하고 락(Lock)을 겁니다.
3. (네임 노드) 파일 데이터를 저장할 데이터 노드를 선택해 데이터 노드 목록을 클라이언트로 반환합니다.
4. (클라이언트) 첫 번째 데이터 노드로 전송하는 데이터의 첫 부분에 데이터 노드의 목록을 같이 포함하여 데이터를 전송합니다.
5. (데이터 노드) 데이터를 받은 첫 번째 데이터 노드는 데이터를 로컬 디스크에 저장하고 다음 데이터 노드로 데이터를 전송합니다. 이런 방식으로 두 번째 데이터 노드는 세 번째 데이터 노드로 복제본을 생성합니다.
6. (클라이언트) 정해진 블록 크기를 넘어서면 클라이언트는 네임 노드로 새로운 데이터 노드를 요청합니다.
7. (네임 노드) 메모리에 임시로 저장돼 있던 파일의 패스 정보를 메모리상에 있는 영구 파일 패스 정보로 이동시킵니다. 그리고 네임 노드 재시작 시에도 패스에 대한 정보가 존재하도록 네임 노드의 로컬 디스크에 파일 생성 관련 로그(edits)를 저장합니다.
8. (네임 노드) 클라이언트의 새로운 블록 생성 요청에 대해 블록을 저장할 데이터 노드 목록을 전달합니다.
9. (클라이언트) 4번~6번 과정을 반복합니다.
10. (클라이언트) 파일 전송이 완료되면 `close ()` 명령을 네임 노드로 전달합니다.

## HDFS 동작 방식 – Namenode, Datanode

11. (세컨드리 네임 노드) 주기적으로 7번에서 저장하고 있는 edits 파일을 네임 노드로 부터 다운로드 해서 네임스페이스 정보를 저장하고 있는 fsimage 파일 과 병합해 새로운 fsimage 파일을 생성해 네임 노드로 전송합니다.
12. (네임 노드) 기존의 fsimage 파일을 새로운 파일로 대체하고 edits 파일을 새로운 파일로 만듭니다.
13. (네임 노드) 시작 시에는 fsimage 파일을 읽어 메모리를 구성한 후 edits 파 일의 변경 대역을 하나씩 수행하는 방식으로 메모리를 재구성합니다.

### • 네임노드 디렉토리 구조

새롭게 포맷된 네임노드는 다음과 같은 디렉토리 구조를 생성합니다.

```
${dfs.name.dir}/current/VERSION
                        /edits
                        /fsimage
                        /fstime
```

### • VERSION

VERSION은 자바 속성 파일로 구동 중인 HDFS 버전에 대한 정보를 포함합니다.

```
#Tue Mar 18 19:21:36 GMT 2889
namespaceID=134368441
cTime=0
storageType=NAME_NODE
layoutVersion=-17
```

**namespaceID**는 파일시스템이 처음 포맷될 때 생성되는 파일시스템 식별자입니다.

**cTime** 속성은 네임노드의 저장소 생성시간을 표시합니다. 새롭게 포맷된 저장소는 항상 0이며, 이 값은 파일 시스템이 업그레이드될 때마다 타임스탬프로 수정됩니다.

**storageType**은 해당 저장소 디렉토리가 네임노드에 대한 데이터 구조를 포함하는지를 표시합니다.

**layoutVersion**은 영속적인 HDFS 데이터 구조의 버전을 음의 정수로 정의 합니다. -17 다음의 버전 숫자는 -18입니다. 이 숫자가 변경될 때는 HDFS가 반드시 업그레이드 되어야 합니다. 새로운 네임노드(또는 데이터노드)의 저장소 레이아웃이 이전 버전이면 동작하지 않기 때문입니다.

## HDFS 동작 방식 – Namenode, Datanode

- **edits**

파일 시스템 클라이언트가 쓰기(파일 생성이나 이동) 동작을 할 때, 먼저 에디트 로그에 기록합니다. 네임노드는 파일 시스템 메타데이터를 메모리로 올려 인메모리 자료구조로 관리하며, 에디트 로그가 수정된 후에 업데이트 합니다. 인메모리 메타데이터는 읽기 요청을 수행하는데 사용됩니다. HDFS에 쓰기 동작이 끝나고 나서 성공했다는 결과가 클라이언트로 반환되기 전에 에디트 로그를 플러시(파일에 반영)하여 동기화 시킵니다. 네임노드는 여러 개의 디렉토리에 쓰기 동작을 마치고 클라이언트에 결과를 반환하기 전에 변경된 값을 모든 에디트 로그 복제본에 플러시하고 동기화해야 합니다.

- **fsimage**

fsimage는 파일시스템 메타데이터의 영속적인 체크포인트 파일입니다. 이 파일은 개별 쓰기 동작 때마다 갱신되지는 않습니다. 만일 그렇게 되면 fsimage 파일이 너무 커져서 매우 느려지게 됩니다. fsimage 자체에 갱신 내용을 바로 반영하지 않더라도 장애복구능력이 저하되는 것은 아닙니다. 왜냐하면 네임노드가 실행 할 경우 fsimage를 메모리로 적재하고 에디트 로그에 기록된 모든 동작을 메모리에 반영하여 메타데이터의 최신 상태를 복원할 수 있기 때문입니다.

- **데이터노드 디렉토리 구조**

네임노드와는 달리 데이터노드는 명시적으로 포맷될 필요가 없습니다.

구동과정에서 자동으로 저장 디렉토리들을 생성합니다. 데이터 파일이 저장되는 디렉토리인 blockpoolID 는 namenode의 VERSION 파일에서 확인할 수 있습니다.

`${dfs.data.dir}/current/`

```
/blockpoolID/current/finalized/blk_<id 1>
/blockpoolID/current/finalized/blk_<id 1>.meta
/blockpoolID/current/finalized/blk_<id 2>
/blockpoolID/current/finalized/blk_<id 2>.meta
/blockpoolID/current/finalized/ ...
```

## HDFS 동작 방식 – Namenode, Datanode

- 보조 네임노드 디렉토리 구조

```
${dfs.namesecondary.dir}/current/VERSION
                                /edits
                                /fsimage
```

- 체크 포인팅

### 수동

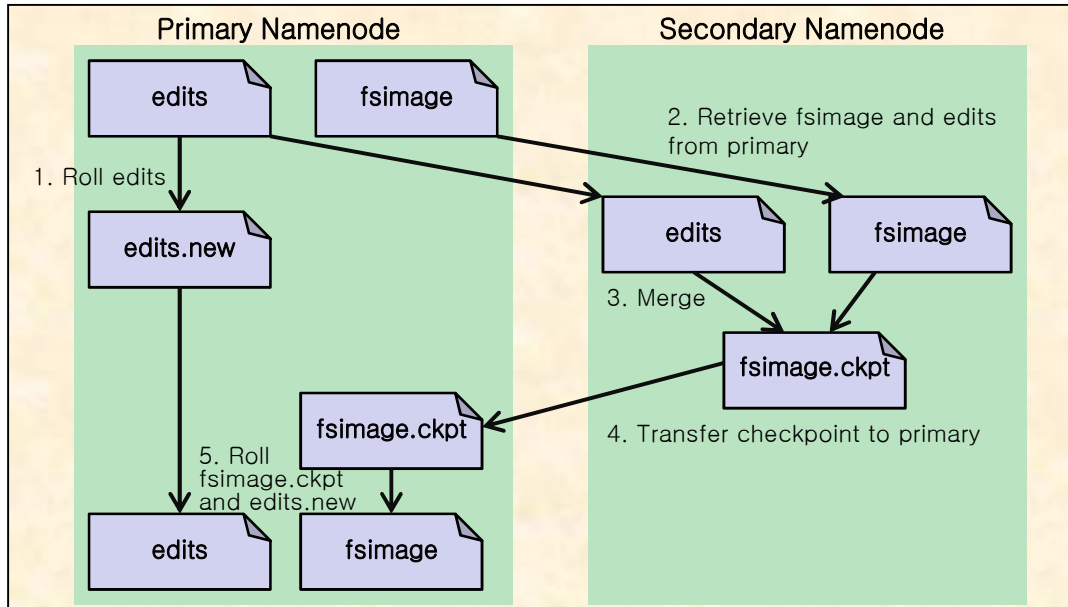
```
[hadoop@master sbin]$ hdfs dfsadmin -saveNamespace
saveNamespace: Safe mode should be turned ON in order to create namespace image.
[hadoop@master sbin]$ hdfs dfsadmin -safemode enter
Safe mode is ON
[hadoop@master sbin]$ hdfs dfsadmin -saveNamespace
[hadoop@master sbin]$ hdfs dfsadmin -safemode leave
Safe mode is OFF
```

### 자동

dfs.namenode.checkpoint.period 속성 : 매 시간 단위로 체크포인트 생성합니다. 디폴트는 3600(초)입니다.



## HDFS 동작 방식 – Secondary node



1. (네임노드) 세컨드리 네임노드는 네임 노드에 edits 로그 파일을 순환 사용하도록 요청합니다. 그 때문에 새로운 edits 로그는 새로운 edits 로그 파일에 저장됩니다.
2. (세컨드리 네임노드) HTTP GET을 이용해서 네임노드의 fsimage와 edits 로그를 가져옵니다.
3. (세컨드리 네임노드) fsimage를 메모리에 올리고 edits 로그의 각 동작을 반영합니다. 그리고 나서 새롭게 통합된 fsimage 파일을 생성합니다.
4. (세컨드리 네임노드) HTTP POST를 이용해서 새로운 fsimage 파일을 네임노드에 전송합니다.
5. (네임노드) 이전 fsimage 파일을 세컨드리 네임노드로부터 받은 새로운 이미지로 교체하며, 이전 edits 로그 파일을 1단계에서 시작한 새로운 edits 로그 파일로 교체합니다. fsimage 파일도 체크 포인트가 발생한 시간을 기록하기 위해 변경됩니다.

보조 네임노드는 네임노드의 인-메모리 메타데이터에 체크포인트를 생성합니다.

에디트 로그 (edits)는 네임노드의 로컬 디스크에 저장되는 파일 생성 관련 로그입니다.

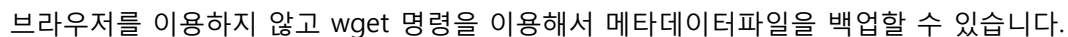
클라이언트의 쓰기(파일 생성이나 이동) 동작을 할 때마다 먼저 에디트 로그에 기록합니다. 네임노드는 파일시스템 메타데이터(fsimage)를 메모리에 올려서 인-메모리 자료 구조로 관리하며 에디트 로그가 수정된 후에 업데이트를 하며 인-메모리 메타데이터는 읽기 요청을 수행하는 데 사용합니다. 매번 쓰기 동작이 끝나고 나서 성공했다는 결과가 클라이언트로 반환되기 전에 에디트 로그를 플러시하여 동기화 시킵니다. 네임노드는 여러 개의 디렉토리에 쓰기 동작을 마치고 클라이언트에 결과를 반환하기 전에 변경 값을 모든 에디트 로그 복제본에 플러시 하고 동기화 해야 합니다. 이렇게 함으로써 컴퓨터 장애가 발생해도 동작 유실을 방지할 수 있습니다.

## Apache Hadoop 1.0/CDH3:

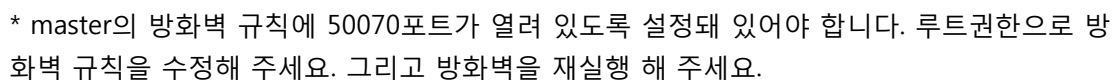
edits: <http://<namenode>:50070/getimage?getedit=1>

```
fsimage: http://<namenode>:50070/getimage?getimage=1&txid=latest
```

\* txid는 edits 또는 fsimage 파일명에 있는 숫자를 의미합니다.



다음 그림은 파일시스템을 백업했을 당시의 네임노드 메타데이터 파일입니다.



```
# service iptables restart
```

## HDFS File System Metadata Backup

wget 또는 curl 명령으로 메타데이터 파일을 받을 수 있습니다. fsimage와 edits 파일은 직렬화되어 있기 때문에 그 내용을 볼 수 없습니다.

내용을 보길 원한다면 hdfs oiv 와 hdfs oev 명령을 이용해 역직렬화 할 수 있습니다.

다음은 fsimage 파일을 역 직렬화 하여 그 내용을 보여주는 내용입니다.

```
[hadoop@slave1 Documents]$ ls -l
total 8
-rw-rw-r--. 1 hadoop hadoop 30 Mar 25 18:06 edits_0000000000000000137-0000000000000000138
-rw-rw-r--. 1 hadoop hadoop 2320 Mar 25 18:00 fsimage_0000000000000000138
[hadoop@slave1 Documents]$ file fsimage_0000000000000000138
fsimage_0000000000000000138: data
[hadoop@slave1 Documents]$ hdfs oiv -i fsimage_0000000000000000138 -o fsimage_deserialized.txt
[hadoop@slave1 Documents]$ head fsimage_deserialized.txt
drwxr-xr-x - hadoop supergroup 0 2014-03-22 22:19 /
drwxr-xr-x - hadoop supergroup 0 2014-03-25 17:28 /airline
drwxr-xr-x - hadoop supergroup 0 2014-03-22 22:19 /output
drwx----- - hadoop supergroup 0 2014-03-22 22:18 /tmp
-rw-r--r-- 1 hadoop supergroup 702878193 2014-03-25 17:28 /airline/2007.csv
-rw-r--r-- 1 hadoop supergroup 689413344 2014-03-22 22:18 /airline/2008.csv
-rw-r--r-- 1 hadoop supergroup 0 2014-03-22 22:19 /output/_SUCCESS
-rw-r--r-- 1 hadoop supergroup 281 2014-03-22 22:19 /output/delay.csv
drwx----- - hadoop supergroup 0 2014-03-22 22:18 /tmp/hadoop-yarn
drwx----- - hadoop supergroup 0 2014-03-22 22:19 /tmp/hadoop-yarn/staging
[hadoop@slave1 Documents]$
```

다음은 hdfs oev 명령을 이용해서 edits 파일을 역 직렬화 한 내용입니다.

```
[hadoop@slave1 Documents]$ file edits_0000000000000000137-0000000000000000138
edits_0000000000000000137-0000000000000000138: data
[hadoop@slave1 Documents]$ hdfs oev -i edits_0000000000000000137-0000000000000000138 -o
edits_137-138.txt
[hadoop@slave1 Documents]$ file edits_137-138.txt
edits_137-138.txt: XML document text
[hadoop@slave1 Documents]$ head edits_137-138.txt
<?xml version="1.0" encoding="UTF-8"?>
<EDITS>
  <EDITS_VERSION>-47</EDITS_VERSION>
  <RECORD>
    <OPCODE>OP_START_LOG_SEGMENT</OPCODE>
    <DATA>
      <TXID>137</TXID>
    </DATA>
  </RECORD>
</RECORD>
```

## HDFS 관리

- ◆ `hdfs dfs -cmd <args>`
- ◆ 파일 목록 보기 : `ls, lsr`
- ◆ 파일 용량 확인 : `du, dus`
- ◆ 파일 내용 보기 : `cat, text`
- ◆ 디렉토리 생성 : `mkdir`
- ◆ 파일 복사 : `put, get, getmerge, cp, copyFromLocal, copyToLocal`
- ◆ 파일 이동 : `mv, moveFromLocal`
- ◆ 파일삭제 : `rm, rmr`

- ◆ 카운트 값 조회 : `count`
- ◆ 파일의 마지막 내용 확인 : `tail`
- ◆ 권한 변경 : `chmod, chown, chgrp`
- ◆ 0바이트파일 생성 : `touchz`
- ◆ 통계 정보 조회 : `stat`
- ◆ 복제 데이터 개수 변경 : `setrep`
- ◆ 휴지통 비우기 : `expunge`
- ◆ 파일 형식 확인 : `test`

### 1. 파일 목록 보기

가. `ls` : 지정한 디렉토리에 있는 파일의 정보를 확인합니다.

```
hdfs dfs -ls [-R] PATH [PATH ...]
```

다. `lsr` : 하위 디렉토리 정보까지 출력합니다.

```
hdfs dfs -lsr PATH [PATH ...]
```

### 2. 파일 용량 확인

가. `du` : 디렉토리나 파일의 사용량을 확인합니다.(바이트단위)

```
hdfs dfs -du PATH [PATH ...]
```

나. `dus` : 전체 합계 용량

```
hdfs dfs -dus PATH [PATH ...]
```

### 3. 파일 내용보기

가. `cat`

```
hdfs dfs -cat FILE [FILE ...]
```

나. `text` : 파일의 텍스트 내용을 나타냅니다. 만약 파일이 텍스트 파일이라면 `cat` 명령과 동일합니다. 압축 형식(gzip과 하둡의 바이너리 시퀀스 파일 포맷)으로 알려진 파일들은 우선 압축을 해제합니다.

```
hdfs dfs -text FILE [FILE ...]
```

## HDFS 관리

### 4. 디렉토리 생성

가. mkdir

```
hdfs dfs -mkdir [-p] PATH [PATH ...]
```

### 5. 파일 복사

가. put : 파일이나 디렉토리를 목적지 경로로 복사합니다.

```
hdfs dfs -put LOCALSRC [LOCALSRC ...] DST
```

나. copyFromLocal : put과 동일합니다.

```
hdfs dfs -copyFromLocal LOCALSRC [ LOCALSRC ...] DST
```

다. get : HDFS에 저장된 데이터를 로컬파일 시스템으로 복사합니다. HDFS는 파일의 무결성을 확인하기 위해 체크섬 기능을 사용하는데 체크섬을 숨김 파일로 저장하고, 해당 파일을 조회할 때 체크섬을 사용해 무결성을 확인합니다.

```
hdfs dfs -get [-ignorecrc] [-crc] SRC [SRC ...] LOCALDST
```

라. getmerge : 지정한 경로에 있는 모든 파일의 내용을 합친 후, 로컬파일 시스템에 단 하나의 파일로 복사합니다.

```
hdfs dfs -getmerge SRC [SRC ...] LOCALDST [addnl]
```

마. cp : 지정한 소스 디렉토리 및 파일을 목적지 경로로 복사합니다.( 여러 개의 파일을 복사할 경우 반드시 디렉토리로 복사 되도록 설정하세요.)

```
hdfs dfs -cp SRC [SRC ...] DST
```

바. copyToLocal : get과 동일합니다.

### 6. 파일 이동

가. mv

```
hdfs dfs -mv SRC [SRC ...] DST
```

나. moveFromLocal : put과 동일하게 동작하지만 로컬파일 시스템으로 파일이 복사된 후 소스경로 파일은 삭제합니다.

```
hdfs dfs -moveFromLocal LOCALSRC [LOCALSRC ...] DST
```

다. moveToLocal : 아직 구현되지 않았습니다.

```
hdfs dfs -moveToLocal [-crc] SRC [SRC ...] LOCALDST
```

### 7. 파일 삭제

가. rm : 파일과 빈 디렉토리를 삭제합니다.

```
hdfs dfs -rm [-R] PATH [PATH ...]
```

나. rmr : 비어 있지 않은 디렉토리까지 삭제합니다.

```
hdfs dfs -rmr PATH [PATH ...]
```

## HDFS 관리

### 8. 카운트 값 조회

가. count : 지정 경로의 전체 디렉토리 개수, 파일개수, 파일 사이즈, 저장경로 이름이 출력합니다. HDFS는 디렉토리에서 생성할 수 있는 파일 개수나 파일 용량을 제한합니다..(-q 옵션을 사용하여 쿼터 정보를 조회)

```
hdfs dfs -count [-q] PATH [PATH ...]
```

### 9. 파일의 마지막 내용 확인

가. tail : 지정한 파일의 마지막 1KB의 내용을 화면에 출력합니다.

-f 옵션을 사용하면 해당 파일에 내용이 추가될 때 화면에 출력된 내용도 함께 갱신합니다.

```
hdfs dfs -tail [-f] FILE
```

### 10. 권한 변경

가. chmod : 지정한 경로의 권한을 변경합니다.

```
hdfs dfs -chmod [-R] MODE [, MODE ...] PATH [PATH ...]
```

나. chown : 지정한 경로의 소유주를 변경합니다.

```
hdfs dfs -chown [-R] [OWNER] [:[GROUP]] PATH [PATH ...]
```

다. chgrp : 지정한 경로의 소유그룹을 변경합니다.(-R 옵션 사용시 하위 디렉토리 정보도 모두 변경)

```
hdfs dfs -chgrp [-R] GROUP PATH [PATH ...]
```

### 11. 0바이트 파일을 생성합니다.

가. touchz

```
hdfs dfs -touchz FILE [FILE ...]
```

### 12. 통계정보조회

가. stat : 지정경로에 대한 통계 정보를 조회합니다.

%b - 블록단위의 파일 크기

%F - 디렉토리일 경우 directory, 일반 파일일 경우 regular File

%n - 디렉토리명 혹은 파일

%o - 블록크기

%r - 복제파일 개수

%y - 디렉토리 및 파일의 갱신일자를 yyyy-MM-dd-HH:mm:ss형식으로 출력

%Y - 디렉토리 및 파일의 갱신일자를 유닉스 타임스탬프 형식으로 출력

```
hdfs dfs -stat [FORMAT] PATH [PATH ...]
```

```
hdfs dfs -stat "%b %F %n %o %r %y" /airline/2008.csv
```

## HDFS 관리

### 13. 복제 데이터 개수 변경

가. setrep : 파일의 복제 데이터 개수를 변경합니다.(-R 옵션으로 하위디렉토리 파일까지 변경)

```
hdfs dfs -setrep [-R] [-w] REP PATH [PATH ...]
```

```
hdfs dfs -setrep 2 /ariline/2008.csv
```

위의 명령을 실행시키기 전과 후의 웹 UI(<http://master:50070>)에서 데이터 노드의 사용량을 확인해 보세요.

### 14. 휴지통비우기

가. expunge - HDFS에서는 삭제한 파일을 .Trash/라는 임시 디렉토리에 저장 후 일정시간 지난 후 완전히 삭제합니다.

```
hdfs dfs -expunge
```

core-site.xml 파일에 fs.trash.interval 속성으로 휴지통에 보관할 시간(분)을 지정할 수 있습니다. 기본값은 0입니다.

### 15. 파일의 형식 확인

가. test : 지정경로에 대해서 -ezd 옵션으로 경로가 이미 존재하는지, 파일 크기가 0인지 디렉토리인지 확인합니다. 체크결과가 맞을 경우 0을 출력합니다.

```
hdfs dfs -test [-ezd] PATH
```

-e PATH 존재 유무, PATH가 존재하면 0을 반환합니다.

-z 빈 파일 여부, 파일 길이가 0이면 0을 반환합니다.

-d PATH가 디렉토리이면 0을 반환합니다.

### 16. 도움말

가. help : 해당 명령어(CMD)에 대한 사용법을 나타냅니다. 만약 CMD 없이 해당 명령어를 사용하면 모든 명령어에 대한 사용법을 출력합니다.

```
hdfs dfs -help [CMD]
```

### 17. 사용자 디렉토리

가. 사용자 디렉토리를 생성하기 위해서는 /user/ 디렉토리에 사용자 아이디로 생성하면 됩니다. 그러면 이후 패스 지정을 상대경로로 지정할 수 있습니다.

```
hdfs dfs -mkdir -p /user/hadoop
```

## dfsadmin

- ◆ dfsadmin 도구는 HDFS 상태 정보를 조회하는 것 뿐만 아니라 HDFS 상에서 다양한 관리 동작을 수행하는 다목적 도구.
- ◆ `hdfs dfsadmin [GENERIC_OPTIONS]`
  - `[-report]`
  - `[-safemode enter | leave | get | wait]`
  - `[-refreshNodes]`
  - `[-finalizeUpgrade]`
  - `[-upgradeProgress status | details | force]`
  - `[-metasave filename]`
  - `[-setQuota <quota> <dirname>...<dirname>]`
  - `[-clrQuota <dirname>...<dirname>]`
  - `[-restoreFailedStorage true|false|check]`
  - `[-help [cmd]]`

### **[hadoop@master bin]\$ hdfs dfsadmin -help**

hdfs dfsadmin is the command to execute DFS administrative commands.

The full syntax is:

```
hdfs dfsadmin [-report] [-safemode <enter | leave | get | wait>]
               [-saveNamespace]
               [-refreshNodes]
               [-setQuota <quota> <dirname>...<dirname>]
               [-clrQuota <dirname>...<dirname>]
               [-setSpaceQuota <quota> <dirname>...<dirname>]
               [-clrSpaceQuota <dirname>...<dirname>]
               [-refreshServiceAcl]
               [-refreshUserToGroupsMappings]
               [-refreshSuperUserGroupsConfiguration]
               [-setBalancerBandwidth <bandwidth>]
               [-help [cmd]]
```

### **[hadoop@master bin]\$ hdfs dfsadmin -report**

HDFS의 기본적인 정보와 상태를 출력

### **[hadoop@master bin]\$ hdfs dfsadmin -safemode {enter | leave | get | wait}**

하둡을 재구동하면 로컬에 저장된 파일 시스템 이미지와 에디트 로그를 조회한 후 메모리에 있는 파일 시스템 이미지를 갱신합니다.

이때 데이터노드는 저장하고 있는 블록의 위치 정보를 네임노드에 전송하고, 네임노드는 메모리에 올라와 있는 파일 시스템 이미지와 데이터노드가 전송한 블록의 정보를 비교하는 작업을 수행합니다. 이러한 비교 작업을 블록 리포팅이라고 하며, 블록 리포팅이 완료되기 전까지의 상태를 안전모드(safemode)라고 합니다.



## dfsadmin

만약 블록 리포팅을 할 때 사용자가 파일을 저장한다면 블록 리포팅 결과에 오류가 생길 것입니다. 이처럼 왜곡이 생기는 것을 막기 위해 안전 모드 상태에서는 파일을 쓰거나 변경하는 작업이 금지됩니다. 대신 기존에 저장된 파일을 읽는 것은 허용됩니다.

하둡은 운영자가 직접 안전 모드를 제어할 수 있게 safemode 명령어를 제공합니다. 안전 모드를 시작할 경우에는 다음과 같이 enter 파라미터를 입력해서 명령어를 실행합니다.

### [hadoop@master bin]\$ hdfs dfsadmin -saveNamespace

하둡은 로컬 파일 시스템에 저장되어 있는 파일 시스템 이미지 파일과 에디트 로그를 현재 버전으로 갱신할 수 있는 saveNamespace 명령어를 제공합니다.

saveNamespace 명령어는 하둡을 구동할 수 있는 권한을 가진 사용자만 실행할 수가 있습니다. saveNamespace 명령어를 실행하려면 반드시 네임노드를 안전 모드 상태로 만들어야 합니다.

saveNamespace 명령어는 네임스페이스만 새롭게 저장해줄 뿐 안전 모드를 해제하지는 않습니다.

saveNamespace 정상 종료후에는 HDFS에 파일 쓰기가 허용되도록 네임노드의 안전모드를 해제합니다.

### [hadoop@master bin]\$ hdfs dfsadmin -setQuota

하둡은 HDFS의 디렉토리에 파일이 과도하게 생성되는 것을 제한할 수 있는 쿼터 설정 명령어를 제공합니다.

setQuota 명령어를 사용하면 디렉토리에 생성되는 파일과 하위 디렉토리 개수를 설정할 수 있습니다.

주의)

파라미터 값은 쿼터수가 지정된 디렉토리까지 포함한 숫자입니다. 만약 쿼터수를 1로 설정한다면 해당 디렉토리에 어떤 파일도 새로 저장할 수가 없습니다.

예) hdfs dfsadmin -setQuota 쿼터수 디렉토리명

[hadoop@master bin]\$ hdfs dfsadmin -setSpaceQuota

하둡은 특정 디렉토리가 지나치게 많은 용량을 차지하지 않도록 디렉토리에 저장할 파일 크기까지 설정할 수 있습니다.

예) hdfs dfsadmin -setSpaceQuota 용량 디렉토리명

용량은 단위를 붙이지 않을 경우 바이트로 인식합니다. 숫자 뒤에 m을 붙이면 MB, g를 붙이면 GB, t를 붙이면 TB입니다.

자세한 내용은 <http://cafe.naver.com/javaspecialistgroup/162> 를 참고하세요.

## 안전모드

- ◆ 하둡 실행 후 ^z 나 ^s 와 같이 비정상 종료를 할 경우 hadoop은 default로 safe모드로 진입하게 된다. 이 때, hdfs을 조작하면 다음과 같은 에러 메시지가 출력된다.
  - org.apache.hadoop.hdfs.server.namenode.SafeModeException: Cannot delete /output. Name node is in safe mode.
- ◆ 안전모드에서 파일 시스템 메타데이터로의 접근(디렉토리 목록 조회 같은)은 정상 동작한다. 만일 파일 읽기도 해당 블록이 클러스터에 있는 현재의 데이터 노드 상에 존재한다면 정상 동작한다. 그러나 파일 변경(쓰기, 삭제, 또는 이름 변경)은 항상 실패한다.
- ◆ 안전모드 진입과 해제
  - hdfs dfsadmin -safemode get
  - hdfs dfsadmin -safemode wait
  - hdfs dfsadmin -safemode enter
  - hdfs dfsadmin -safemode leave

속석명	타입	기본값	설명
dfs.namenode.replication.min	정수	1	쓰기 작업이 성공하기 위해 쓰여져야 하는 최소 복제본 수
dfs.namenode.safemode.threshhold-pct	실수	0.999	네임노드가 안전 모드에서 빠져나오기 위해 dfs.namenode.replication.min 에서 정의한 최소 복제 수준을 충족하는 시스템 내 블록의 비율. 이 값을 0이나 그 이하로 설정하면 네임노드는 안전모드 없이 시작된다. 이 값을 1보다 크게 하면 네임노드는 안전모드 상태에서 벗어나지 않는다(영원히 안전모드).
dfs.namenode.safemode.extension	정수	30,000	dfs.namenode.safemode.threshold 에서 정의한 최소 복제 수준이 만족되고 나서 안전모드 상태를 얼마나 더 유지할지를 밀리 세컨드(ms) 단위로 정함. 작은 클러스터의 경우(수십대 이하)에는 0초도 설정 가능하다.

## 파일시스템 체크

- ◆ 파일시스템 점검: fsck
  - 전체 파일 시스템에서 유실되거나 오염된 블록을 사전에 찾기 위함.
  - `hdfs fsck <path> [-move | -delete | -openforwrite] [-files [-blocks [-locations | -racks]]]`
- ◆ 파일 시스템 밸런서
  - 파일 시스템 데이터 노드의 저장된 데이터를 균등한 상태로 유지하기 위해 밸런서 도구를 주기적으로 실행하자.
  - `hdfs balancer [-threshold <threshold>]`
    - `-threshold <threshold>` : Percentage of disk capacity. threshold는 전체 데이터노드 사용량의 표준편차(stdev 값 보다 작아야 합니다.)
- ◆ 메타데이터 백업
  - 보조 네임노드의 이전 체크포인트 하위 디렉토리를 스크립트로 원격지에 주기적으로 저장
- ◆ 데이터 백업
  - 우선순위를 정하고 데이터는 항상 백업하자.

### 파일 시스템 점검 : fsck

- 모든 데이터 노드를 점검하여 사라지거나, 적게 또는 많이 복제된 블록을 찾아줍니다.
- `hdfs fsck /`
- 과잉 복제된 블록을 HDFS는 자동으로 삭제합니다.
- 부족하게 복제된 블록은 목표 복제 수를 충족할 때까지 생성합니다.
- `hdfs dfsadmin -metasave` 를 사용하면 복제되고 있는 블록에 대한 정보 조회할 수 있습니다.
- 잘못 복제된 블록(해당 블록의 복제본 모두가 같은 랙에 있을 경우)은 자동으로 잘못 복제된 블록을 재복제 합니다.
- `hdfs fsck /output/part-r-00000 -files -blocks -racks`

COMMAND_OPTION	Description
path	Start checking from this path.
-move	Move corrupted files to /lost+found
-delete	Delete corrupted files.
-openforwrite	Print out files opened for write.
-files	Print out files being checked.
-blocks	Print out block report.
-locations	Print out locations for every block.
-racks	Print out network topology for data-node locations.

\* 밸런서의 디폴트 Threshold는 10입니다.

## Lab – 데이터 파일 복제 개수 변경 및 밸런서 실행

\* HDFS 초기화 한 다음 2008.csv 파일을 HDFS에 저장한 다음 수행하세요.

1. 2008.csv 파일의 복제 데이터 개수를 확인해 보세요.
2. 웹 UI에서 각 노드별 데이터파일의 사용량(Used(%))을 기록하세요.
3. 2008.csv 파일의 복제 데이터 개수를 2로 변경해 보세요.
4. 웹 UI에서 각 노드별 데이터파일의 사용량(Used(%))을 기록하세요.
5. 2008.csv 파일의 복제 데이터 개수를 1로 변경해 보세요.
6. 웹 UI에서 각 노드별 데이터파일의 사용량(Used(%))을 기록하세요.
7. datanode2의 실행을 종료 시킨 다음 랩파일의 샘플 데이터(/home/hadoop/Lab/Data/) 디렉토리에 있는 파일을 모두 HDFS에 저장하세요.
8. Datanode2를 시작시키세요.
9. 웹 UI에서 전체 데이터 노드의 표준편차(stdev)를 확인하고 기록하세요.
10. 밸런서를 실행해서 전체 데이터 노드의 표준편차가 밸런서의 쓰레스홀드 이하가 되도록 하세요.
11. 웹 UI에서 표준편차를 확인하세요.

## Lab - 하둡 클러스터 복구(Namenode disk failover)

### Namenode 에러 시 하둡 클러스터 복구

#### 1. 하둡 클러스터 정상 종료시킴

```
$ sbin/stop-all.sh
```

#### 2. 하둡 클러스터 실행

```
$ sbin/start-all.sh
```

#### 3. 파일 목록 확인 후 파일 삭제(하둡 파일시스템 데이터 변경 후 Disk fail 가정하기 위함)

```
[hadoop@master Desktop]$ hdfs dfs -ls /airline
Found 2 items
-rw-r--r-- 1 hadoop supergroup 672068096 2014-04-23 05:18 /airline/2006.csv
-rw-r--r-- 1 hadoop supergroup 689413344 2014-03-22 22:18 /airline/2008.csv
[hadoop@master Desktop]$ hdfs dfs -rm /airline/2006.csv
14/04/23 05:41:22 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion
interval = 0 minutes, Empty interval = 0 minutes.
Deleted /airline/2006.csv
[hadoop@master Desktop]$
```

#### 4. Name 노드의 파일시스템 이미지 확인

```
[hadoop@master Desktop]$ cd /usr/local/hadoop-2.2.0/tmp/dfs/name/current/
[hadoop@master current]$ ls
edits_00000000000000000001-00000000000000000013
edits_000000000000000000112-000000000000000000113
edits_000000000000000000139-000000000000000000140
edits_000000000000000000141-000000000000000000149
...
edits_000000000000000003644-000000000000000003650
edits_000000000000000003651-000000000000000003652
edits_inprogress_000000000000000003653
fsimage_000000000000000003643
fsimage_000000000000000003643.md5
fsimage_000000000000000003652
fsimage_000000000000000003652.md5
seen_txid
VERSION
[hadoop@master current]$
```

## Lab – 하둡 클러스터 복구(Namenode disk failover)

### 5. Secondary name 노드 파일 시스템 이미지 확인

```
[hadoop@master Desktop]$ cd /usr/local/hadoop-2.2.0/tmp/dfs/namesecondary/current/
[hadoop@master current]$ ls
edits_00000000000000000001-00000000000000000013
edits_0000000000000000000112-000000000000000000113
edits_0000000000000000000139-000000000000000000140
...
edits_0000000000000000003644-0000000000000000003650
edits_0000000000000000003651-0000000000000000003652
fsimage_0000000000000000003643
fsimage_0000000000000000003643.md5
fsimage_0000000000000000003652
fsimage_0000000000000000003652.md5
VERSION
[hadoop@master current]$
```

### 6. Name 노드 메타정보삭제(파일시스템 손상 가정)

```
[hadoop@master current]$ pwd
/usr/local/hadoop-2.2.0/tmp/dfs/name/current
[hadoop@master current]$ rm *
[hadoop@master current]$ ls
[hadoop@master current]$
```

### 7. 하둡 클러스터 종료

```
$ sbin/stop-all.sh
```

### 8. Secondary name 노드의 메타데이터 name 노드의 current 디렉토리로 복사(리커버리)

```
[hadoop@master current]$ cp ../../namesecondary/current/* ./
[hadoop@master current]$ ls
edits_00000000000000000001-00000000000000000013
edits_0000000000000000000112-000000000000000000113
edits_0000000000000000000139-000000000000000000140
...
edits_0000000000000000003644-0000000000000000003650
edits_0000000000000000003651-0000000000000000003652
fsimage_0000000000000000003643
fsimage_0000000000000000003643.md5
fsimage_0000000000000000003652
fsimage_0000000000000000003652.md5
VERSION
```

## Lab – 하둡 클러스터 복구(Namenode disk failover)

### 9. 클러스터 시작

```
$ sbin/start-all.sh
```

### 10. 디렉토리 목록 확인(삭제된 파일 보임)

```
[hadoop@master current]$ hdfs dfs -ls /airline
```

```
Found 2 items
```

```
-rw-r--r-- 1 hadoop supergroup 672068096 2014-04-23 05:18 /airline/2006.csv
```

```
-rw-r--r-- 1 hadoop supergroup 689413344 2014-03-22 22:18 /airline/2008.csv
```

```
[hadoop@master current]$
```

### 11. 안전모드 상태 확인(복구 후 이므로 클러스터가 안전모드로 가동됨)

```
[hadoop@master current]$ hdfs dfsadmin -safemode get
```

```
Safe mode is ON
```

### 12. 파일 시스템 점검(Corrupt 된 파일 발견)

```
[hadoop@master current]$ hdfs fsck /
```

```
Connecting to namenode via http://master:50070
```

```
FCK started by hadoop (auth:SIMPLE) from /192.168.67.128 for path / at Wed Apr 23 05:52:22 PDT 2014
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742300
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742301
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742302
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742303
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742304
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742305
```

## Lab – 하둡 클러스터 복구(Namenode disk failover)

```
/airline/2006.csv: MISSING 6 blocks of total size 672068096 B.....Status: CORRUPT
Total size: 1361870184 B
Total dirs: 7
Total files: 9
Total symlinks: 0
Total blocks (validated): 17 (avg. block size 80110010 B)
*****
CORRUPT FILES: 1
MISSING BLOCKS: 6
MISSING SIZE: 672068096 B
CORRUPT BLOCKS: 6
*****
Minimally replicated blocks: 11 (64.70588 %)
Over-replicated blocks: 0 (0.0 %)
Under-replicated blocks: 0 (0.0 %)
Mis-replicated blocks: 0 (0.0 %)
Default replication factor: 1
Average block replication: 0.64705884
Corrupt blocks: 6
Missing replicas: 0 (0.0 %)
Number of data-nodes: 2
Number of racks: 1
FSCK ended at Wed Apr 23 05:52:22 PDT 2014 in 21 milliseconds
The filesystem under path '/' is CORRUPT
```

### 13. Corrupt 된 파일 삭제 시도하지만 안전모드 이므로 삭제 안됨

```
[hadoop@master current]$ hdfs dfs -rm /airline/2006.csv
14/04/23 05:53:45 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion
interval = 0 minutes, Emptier interval = 0 minutes.
rm: Cannot delete /airline/2006.csv. Name node is in safe mode.
```

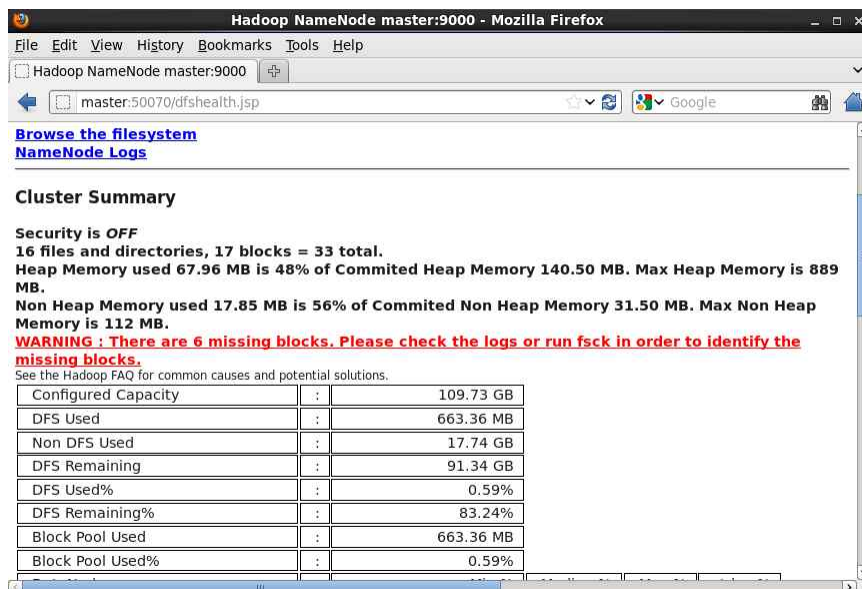
### 14. 안전모드 빠져나감

```
[hadoop@master current]$ hdfs dfsadmin -safemode leave
Safe mode is OFF
```



## Lab – 하둡 클러스터 복구(Namenode disk failover)

### 15. UI에서 Corrupt된 파일 확인(http://master:50070)



### 16. Corrupt 된 파일 모두 삭제

```
[hadoop@master current]$ hdfs fsck -delete /
```

```
Connecting to namenode via http://master:50070
```

```
FSCK started by hadoop (auth:SIMPLE) from /192.168.67.128 for path / at Wed Apr 23 06:00:14 PDT 2014
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742300
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742301
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742302
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742303
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742304
```

```
./airline/2006.csv: CORRUPT blockpool BP-685602400-192.168.224.135-1395551834987 block blk_1073742305
```

## Lab - 하둡 클러스터 복구(Namenode disk failover)

```
/airline/2006.csv: MISSING 6 blocks of total size 672068096 B.....Status: CORRUPT
Total size: 1361870184 B
Total dirs: 7
Total files: 9
Total symlinks: 0
Total blocks (validated): 17 (avg. block size 80110010 B)
*****
CORRUPT FILES: 1
MISSING BLOCKS: 6
MISSING SIZE: 672068096 B
CORRUPT BLOCKS: 6
*****
Minimally replicated blocks: 11 (64.70588 %)
Over-replicated blocks: 0 (0.0 %)
Under-replicated blocks: 0 (0.0 %)
Mis-replicated blocks: 0 (0.0 %)
Default replication factor: 1
Average block replication: 0.64705884
Corrupt blocks: 6
Missing replicas: 0 (0.0 %)
Number of data-nodes: 2
Number of racks: 1
FSCK ended at Wed Apr 23 06:00:14 PDT 2014 in 10 milliseconds
```

The filesystem under path '/' is CORRUPT

### 17. 파일 목록 확인(Corrupt 된 파일 제거됨)

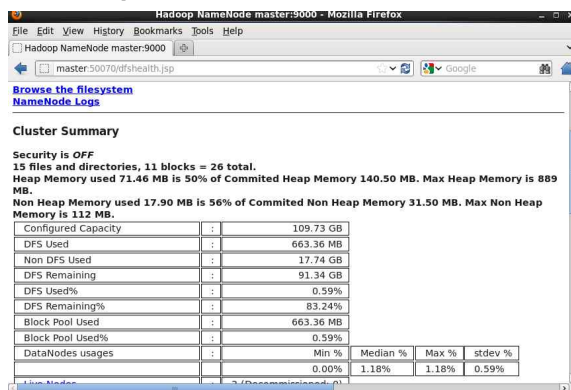
```
[hadoop@master current]$ hdfs dfs -ls /airline
```

Found 1 items

```
-rw-r--r-- 1 hadoop supergroup 689413344 2014-03-22 22:18 /airline/2008.csv
```

```
[hadoop@master current]$
```

### 18. 웹 UI(http://master:50070)



The screenshot shows the Hadoop NameNode web UI in a Mozilla Firefox browser window. The address bar shows 'master:50070/dfshealth.jsp'. The page title is 'Hadoop NameNode master:9000 - Mozilla Firefox'. The main content area displays the 'Cluster Summary' with the following information:

- Security is OFF
- 15 files and directories, 11 blocks = 26 total.
- Heap Memory used 71.46 MB is 50% of Committed Heap Memory 140.50 MB. Max Heap Memory is 889 MB.
- Non Heap Memory used 17.90 MB is 56% of Committed Non Heap Memory 31.50 MB. Max Non Heap Memory is 112 MB.

Below this summary is a table of metrics:

Metric	Value
Configured Capacity	109.73 GB
DFS Used	663.36 MB
Non DFS Used	17.74 GB
DFS Remaining	91.34 GB
DFS Used%	0.59%
DFS Remaining%	83.24%
Block Pool Used	663.36 MB
Block Pool Used%	0.59%
DataNodes usages	Min % Median % Max % stdev %
	0.00% 1.18% 1.18% 0.59%

\* 파일이 추가된 경우에는 secondary namenode의 fsimage가 갱신됩니다.

## 노드 위임

◆ 클러스터에서 노드를 추가하기 위해서...

1. include 파일에 새 노드의 네트워크 주소를 추가한다.

dfs.hosts와 mapreduce.jobtracker.hosts.filename속성을 통해 하나의 공유 파일을 참조한다.

2. 네임노드에 허가된 데이터 노드 집합을 반영한다.

```
$ hdfs dfsadmin -refreshNodes
```

3. 새로 허가된 노드매니저 집합을 리소스매니저에 반영한다.

```
$ yarn rmadmin -refreshNodes
```

4. 새 노드가 하둡 제어 스크립트에 의해 장치 클러스터에서 사용될 수 있게 slaves 파일을 갱신한다.

5. 새로운 데이터 노드와 태스크 트래커가 웹 UI에 나타나는지 확인한다.

## 노드 해제

◆ 클러스터에서 노드를 제거하기 위해서...

1. 해제할 노드의 네트워크 주소를 exclude 파일에 추가한다. 이때 include 파일은 수정하지 않는다.(dfs.hosts.exclude와 mapreduce.jobtracker.hosts.exclude.filename에서 지정한 파일에)

2. 허가된 새로운 데이터 노드 집합을 가지고 네임노드를 갱신한다.

```
$ hdfs dfsadmin -refreshNodes
```

3. 허가된 새로운 노드매니저 집합을 가지고 리소스매니저를 갱신한다.

```
$ yarn rmadmin -refreshNodes
```

4. 웹 UI에 접속해서 해제할 데이터 노드의 관리 상태가 "Decommissioning in Progress"로 변했는지 확인한다.

5. 모든 데이터 노드가 블록 복사를 완료하면 관리 상태가 "Decommissioned"로 변한다. 그러면 해제된 노드를 중단시킨다.

6. slaves 파일에서 해당 노드를 삭제한다.

## Lab – 노드 해제

### 1. `sbin/stop-all.sh`로 클러스터 종료

### 2. `hdfs-site.xml`에 `exclude`와 `include` 속성 추가(The full pathname of the file must be specified)

```
[hadoop@master hadoop]$ vi hdfs-site.xml
... 생략
<property>
  <name>dfs.hosts.exclude</name>
  <value>/usr/local/hadoop-2.2.0/etc/hadoop/exclude</value>
</property>
<property>
  <name>dfs.hosts</name>
  <value>/usr/local/hadoop-2.2.0/etc/hadoop/include</value>
</property>
</configuration>
```

### 3. `mapred-site.xml`에 `exclude`와 `include` 파일을 지정하는 속성 추가(The full pathname of the file must be specified)

```
[hadoop@master hadoop]$ vi mapred-site.xml
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.jobtracker.hosts.exclude.filename</name>
    <value>/usr/local/hadoop-2.2.0/etc/hadoop/exclude</value>
  </property>
  <property>
    <name>mapreduce.jobtracker.hosts.filename</name>
    <value>/usr/local/hadoop-2.2.0/etc/hadoop/include</value>
  </property>
</configuration>
```

### 4. 하둡 설정파일 디렉토리에 `include` 파일과 `exclude` 파일을 만들어줍니다.

```
[hadoop@master hadoop]$ vi include
[hadoop@master hadoop]$ vi exclude
[hadoop@master hadoop]$ ls *clude
exclude  include
```

## Lab - 노드 해제

5. 클러스터를 시작합니다.

```
[hadoop@master hadoop-2.2.0]$ sbin/start-all.sh
```

6. **exclude** 파일에 제거할 노드의 아이피를 입력합니다.

```
[hadoop@master hadoop]$ vi exclude
slave2
```

7. 새로운 데이터 노드 집합으로 네임노드를 갱신합니다.

```
hdfs dfsadmin -refreshNodes
```

8. 새로운 데이터 노드 집합으로 잡 트래커를 갱신합니다.(필요하다면...)

```
hdfs mradmin -refreshNodes
```

9. 웹 UI에서 데이터 노드의 관리 상태가 "Decommissioning in Progress"로 변했는지 확인합니다.(잠시 기다린 다음 새로고침 하세요)

The screenshot shows the Hadoop NameNode web UI in a Mozilla Firefox browser. The page title is 'Hadoop NameNode master:9000 - Mozilla Firefox'. The URL bar shows 'master:50070/dfshealth.jsp'. The main content area displays various DFS metrics and node status.

Metric	Value
DFS Remaining%	86.78%
Block Pool Used	1.70 GB
Block Pool Used%	3.04%
DataNodes usages	Min %: 1.18%, Median %: 1.92%, Max %: 1.92%, stdev: 0.37
Live Nodes	2 (Decommissioned: 1)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Number of Under-Replicated Blocks	0

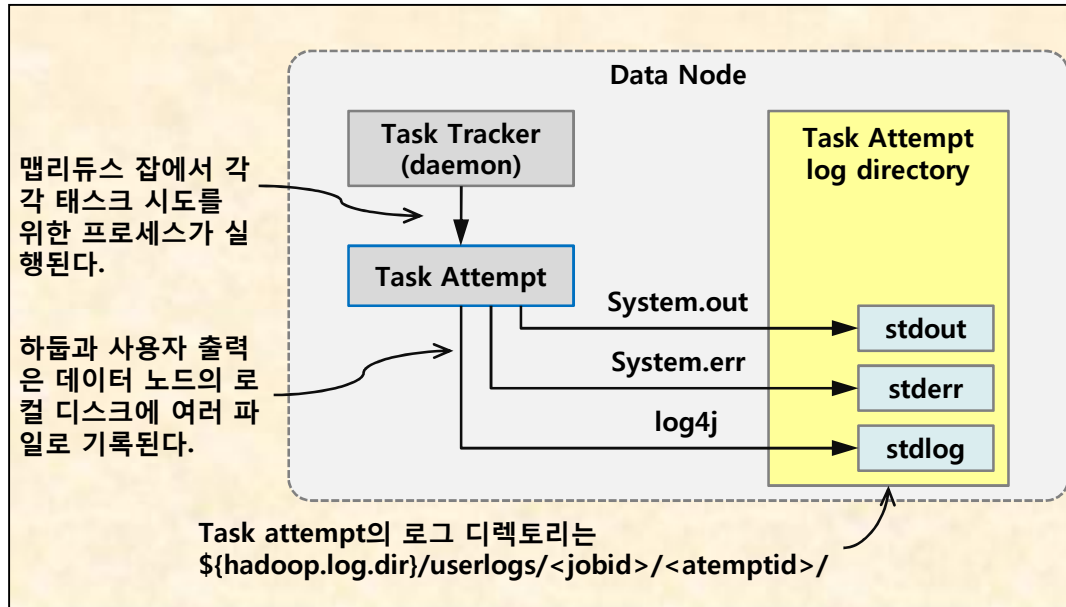
Below the metrics, the 'NameNode Journal Status' is shown with 'Current transaction ID: 161'. A 'Journal Manager' table is also visible with columns for 'Journal Manager' and 'State'.

10. slave1의 데이터 디렉토리에 블록이 복사된 것을 확인 할 수 있습니다.

```
hadoop@slave1:usr/local/hadoop-2.2.0/tmp/dfs/data/current/BP-632179431-192.168.60.128-140653296206$ ls
current in use.lock
[hadoop@slave1 data]$ cd current/
[hadoop@slave1 current]$ pwd
/usr/local/hadoop-2.2.0/tmp/dfs/data/current
[hadoop@slave1 current]$ ls
BP-632179431-192.168.60.128-140653296206 VERSION
[hadoop@slave1 current]$ cd BP-632179431-192.168.60.128-140653296206/current/finalized/
[hadoop@slave1 finalized]$ ls
blk_1073741852 blk_1073741854 blk_1073741856
blk_1073741852_1028.meta blk_1073741854_1030.meta blk_1073741856_1032.meta
blk_1073741853 blk_1073741855 blk_1073741857
blk_1073741853_1029.meta blk_1073741855_1031.meta blk_1073741857_1033.meta
[hadoop@slave1 finalized]$
```

11. slaves 파일에서 제거할 데이터노드 정보를 삭제합니다.

## 로깅



매퍼의 catch 블록에 다음과 같이 로그를 남기는 문장을 남겼을 경우

```
}catch(ArrayIndexOutOfBoundsException e) {
    System.out.println("*****" + e.getMessage());
    log4j.error("Exception- " + e.getClass().toString());
    log4j.info("Info : reason-" + e.getMessage());
}
```

\* log4j는 멤버변수에 `private Logger log4j = Logger.getLogger(AirPollution.class);` 로 선언되어 있으며, Logger 클래스는 `org.apache.log4j.Logger` 클래스입니다.

\* `hadoop/lib` 에 있는 `log4j-1.2.17.jar` 파일을 빌드패스에 추가해야 합니다.

\* `setup` 메서드에 로그 레벨을 지정하세요. `log4j.setLevel(Level.INFO);`

위 코드가 포함된 예제를 실행시키면 다음과 같이 `userlogs` 디렉토리에 `jobid` 디렉토리가 만들어지고 그 아래 `taskid` 디렉토리가 만들어집니다.

```
[hadoop@master logs]$ cd /usr/local/hadoop-2.2.0/logs/userlogs/
```

```
[hadoop@master userlogs]$ ls
```

```
application_1395966791119_0003
```

```
[hadoop@master userlogs]$ cd application_1395966791119_0003/
```

```
[hadoop@master application_1395966791119_0003]$ ls
```

```
container_1395966791119_0003_01_000001 container_1395966791119_0003_01_000003
```

```
container_1395966791119_0003_01_000002
```

```
[hadoop@master application_1395966791119_0003]$ cd container_1395966791119_0003_01_000002
```

```
[hadoop@master container_1395966791119_0003_01_000002]$ ls -l
```

```
total 1608
```

```
-rw-rw-r--. 1 hadoop hadoop      0 Mar 27 17:47 stderr
```

```
-rw-rw-r--. 1 hadoop hadoop 172040 Mar 27 17:47 stdout
```

```
-rw-rw-r--. 1 hadoop hadoop 1467753 Mar 27 17:47 syslog
```

```
[hadoop@master container_1395966791119_0003_01_000002]$
```

\$ ls -lR ./ 명령으로 어떤 attemptid  
폴더에 로그가 저장되었는지 찾을  
수 있다.

## 로깅

```
[hadoop@master container_1395966791119_0003_01_000002]$ cat stdout | head -5
```

```
*****7
*****7
*****7
*****7
*****7
```

```
[hadoop@master container_1395966791119_0003_01_000002]$ cat syslog | head -35
```

```
2014-03-27 17:47:15,910 WARN [main] org.apache.hadoop.conf.Configuration: job.xml:an attempt to override
    final parameter: mapreduce.job.end-notification.max.retry.interval; Ignoring.
2014-03-27 17:47:15,911 WARN [main] org.apache.hadoop.conf.Configuration: job.xml:an attempt to override
    final parameter: mapreduce.job.end-notification.max.attempts; Ignoring.
2014-03-27 17:47:16,121 INFO [main] org.apache.hadoop.metrics2.impl.MetricsConfig: loaded properties from
    hadoop-metrics2.properties
2014-03-27 17:47:16,183 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: Scheduled snapshot
    period at 10 second(s).
2014-03-27 17:47:16,183 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: MapTask metrics
    system started
...
2014-03-27 17:47:17,997 INFO [main] org.apache.hadoop.mapred.MapTask: mapreduce.task.io.sort.mb: 100
2014-03-27 17:47:17,997 INFO [main] org.apache.hadoop.mapred.MapTask: soft limit at 83886080
2014-03-27 17:47:17,997 INFO [main] org.apache.hadoop.mapred.MapTask: bufstart = 0; bufvoid = 104857600
2014-03-27 17:47:17,997 INFO [main] org.apache.hadoop.mapred.MapTask: kvstart = 26214396; length =
    6553600
2014-03-27 17:47:18,054 ERROR [main] hadoop.seoul.airpollution.AirPollutionMapper: Exception- class
    java.lang.ArrayIndexOutOfBoundsException
2014-03-27 17:47:18,054 INFO [main] hadoop.seoul.airpollution.AirPollutionMapper: Info : reason-7
2014-03-27 17:47:18,054 ERROR [main] hadoop.seoul.airpollution.AirPollutionMapper: Exception- class
    java.lang.ArrayIndexOutOfBoundsException
2014-03-27 17:47:18,054 INFO [main] hadoop.seoul.airpollution.AirPollutionMapper: Info : reason-7
2014-03-27 17:47:18,054 ERROR [main] hadoop.seoul.airpollution.AirPollutionMapper: Exception- class
    java.lang.ArrayIndexOutOfBoundsException
2014-03-27 17:47:18,054 INFO [main] hadoop.seoul.airpollution.AirPollutionMapper: Info : reason-7
...
```

<http://hadoop.apache.org/docs/r2.2.0/api/index.html>

## 로깅

- ◆ Task attempt 로그에 의해 디스크가 full되는 현상을 방지하기 위해 다음 3가지 방법을 사용할 수 있다.
- ◆ `mapreduce.job.userlog.retain.hours`
  - `mapred-default.xml` 문서에 디폴트 24(시간)로 설정되어 있다.
  - 1.x는 `mapred.userlog.retain.hours`
- ◆ `mapreduce.task.userlog.limit.kb`
  - `mapred-default.xml` 문서에 0으로 설정되어 있다.
  - 단위는KB, 0으로 하면, 이 속성을 사용하지 않음을 의미한다.
  - 1.x는 `mapred.userlog.limit.kb`
- ◆ 설정파일에서 로그 레벨 변경
  - 하둡은 log4j 로그 설정파일을 통해 로그 수준을 지정
  - `log4j.properties`
  - FATAL, ERROR, WARN, INFO, DEBUG, TRACE

log4j는 자바기반 로깅 유틸리티입니다. 디버그용 도구로 주로 사용되고 있습니다. log4j의 최근 버전에 의하면 높은 등급에서 낮은 등급으로의 6개 로그 레벨을 가지고 있습니다. 설정 파일에 대상별(자바에서는 패키지)로 레벨을 지정이 가능하고 그 등급 이상의 로그만 저장하는 방식입니다.

```
[hadoop@master hadoop]$ vi log4j.properties
# Custom Logging levels
#log4j.logger.org.apache.hadoop.mapred.JobTracker=DEBUG
log4j.logger.org.apache.hadoop.mapred.TaskTracker=ERROR
#log4j.logger.org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit=DEBUG
```

위 내용처럼 디폴트는 잡트래커나 태스크트래커의 로그 레벨 설정이 주석처리 되어 있습니다. 주석을 풀고 하둡을 재 시작 시키면 로그 레벨이 변경됩니다.

특정 태스크의 로그 레벨을 지정하려면 다음과 같이 할 수 있습니다. `com.example.mr`은 로그를 남길 대상 패키지입니다.

```
log4j.logger.com.example.mr=ERROR
```

CLI환경에서 하둡 `daemonlog` 명령을 이용하면 로그 레벨을 지정할 수 있습니다.

```
hadoop daemonlog -setlevel <host:port> com.example.mr ERROR
```

로그 레벨은 FATAL, ERROR, WARN, INFO, DEBUG, TRACE등이 있습니다.



## 마스터의 장애 대책

- ◆ Hadoop 2.x 부터 한 쌍의 네임노드를 Active-Standby 로 구성하여 HDFS 고가용성을 지원할 수 있다.
- ◆ 네임노드는 에디트 로그를 공유하기 위해 공유된 스토리지를 사용한다.
  - 마스터는 특별한 장애가 없음 / 고장시에 MapReduce 다시시작
- ◆ 데이터 노드의 장애 대책
  - 데이터 노드의 수가 많은 만큼 고장 확률이 높음
  - 마스터의 수시 검사 / 고장시 관리 대상 제외
  - 장애 발생 데이터 노드의 수행 Map은 다른 워커에 의해 처음 부터 시작
- ◆ Map과 Reduce 장애 대책
  - 문제가 명확하면 수정하지만 그렇지 않으면 무시하고 넘어감

고 가용성을 위해 네임노드의 hdfs-site.xml 파일에 여러 속성들을 설정해야 합니다.

- dfs.nameservices - 새로운 네임 서비스의 논리적 이름
- dfs.ha.namenodes.[nameservice ID] - 네임서비스 내에서 각 네임노드의 유일 식별자(로 구분해서 입력)
- dfs.namenode.rpc-address.[nameservice ID].[name node ID] - 각 네임노드의 RPC 주소
- dfs.namenode.http-address.[nameservice ID].[name node ID] - 각 네임노드의 HTTP 주소
- dfs.namenode.shared.edits.dir - 공유 디렉토리 경로(file://가 경로 앞에 붙는다)
- dfs.client.failover.proxy.provider.[nameservice ID] - HDFS 클라이언트가 액티브 네임노드에 접근하기 위해 사용하는 자바 클래스 이름  
(org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider)
- dfs.ha.fencing.methods - Failover 기간 동안 활성화 네임노드를 막기 위한 스크립트나 자바 클래스 목록
- fs.defaultFS - 하둡 FS 클라이언트에서 사용하는 기본 경로 접두어

고 가용성을 위한 Active-Standby 구성에 대한 자세한 내용을 다음 문서를 참고하세요.

- <http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithNFS.html#Architecture>

## 메모리 설정 - 데이터 노드

- ◆ 데몬당 1,000MB 메모리 할당
- ◆ `hadoop-env.sh`의 `HADOOP_HEAPSIZE`에 의해서 제어
- ◆ 태스크 트래커는 맵과 리듀스 태스크를 실행시킬 별도의 자식 JVM을 생성하므로 이러한 JVM에 필요한 메모리도 워커 컴퓨터의 물리적인 총 메모리 범위 내에서 계수화 시켜야 함
- ◆ 하나의 태스크트래커에서 한꺼번에 실행할 수 있는 맵 태스크의 최대 개수는 `mapreduce.tasktracker.map.tasks.maximum`에 의해 제어, 디폴트 2
- ◆ 리듀스 태스크도 `mapreduce.tasktracker.reduce.tasks.maximum`으로 설정, 디폴트 2
- ◆ 태스크를 수행하는 각 자식 JVM에 할당되는 메모리는 `mapred.child.java.opts`를 통해 변경, 기본 값은 `-Xmx200m`

하나의 태스크 트래커 상에서 동시에 실행될 수 있는 태스크의 개수는 컴퓨터에서 가용한 프로세스와 연관이 있습니다. 맵리듀스 잡은 일반적으로 입출력 성능에 의해 제한되므로 가용률을 더 좋게 하기 위해서 프로세서 보다 더 많은 태스크를 실행하는 것이 바람직 합니다. 초과 요청할 수 있는 태스크의 양은 실행 잡의 CPU 사용률에 의해 영향을 받지만, 경험적으로 좋은 방법은 프로세서 수 보다 하나 또는 둘 정도(맵과 리듀스 태스크의 개수를 세서) 더 많이 갖는 것입니다. 예를 들어 8개의 프로세서가 있고, 각 프로세서당 두 개의 프로세스를 실행시키고 싶다면 `mapreduce.tasktracker.map.tasks.maximum`과 `mapreduce.tasktracker.reduce.tasks.maximum`의 값을 모두 7(8이 아닙니다. 데이터 노드와 태스크 트래커가 각각 한 개의 슬롯을 점유 합니다.)로 설정하면 됩니다. 또한 할당 가능한 메모리를 400MB로 증가 시켰다면 총 메모리 사용량은 7,600MB가 될 것입니다.

JVM	기본 메모리 사용량(MB)	8개 프로세서에서 사용되는 메모리( 자식 JVM당 400MB를 사용할 경우)(MB)
DataNode	1,000	1,000
TaskTracker	1,000	1,000
자식 맵 태스크	2 * 200	7 * 400
자식 리듀스 태스크	2 * 200	7 * 400
총 합	2,800	7,600

위의 결과에 의해서 자바 메모리를 8GB로 할당한 것이 적합한지 여부는 해당 컴퓨터에서 실행되는 다른 프로세스에 의해 좌우 됩니다. 정확한 메모리 설정은 클러스터에 의존적일 수밖에 없습니다. 그래서 시간을 두고 클러스터 상에서 메모리 사용량을 지속적으로 관찰해야 하며, 이러한 경험적인 수치에 의해 최적화 될 수 있습니다. Ganglia 같은 도구를 사용하면 이러한 정보를 수집할 수 있습니다.

## 메모리 설정 - 네임노드와 보조네임노드

### ◆ 네임노드

- hadoop-env.sh파일의 HADOOP\_NAMENODE\_OPTS에 JVM 명령으로 설정
- 각 파일의 개별 블록에 대한 참조 값이 메모리로 관리
- 파일당 블록 개수, 파일명 길이, 파일 시스템의 디렉토리 수에 의존적
- 경험적으로 백만 블록당 1,000MB 정도
- 8TB 디스크, 200노드 클러스터, 블록사이즈 128MB, 복제수준 3일 경우
  - $8 \times 1024 \times 1024 \times 200 / (128 \times 3) = 4,369,066.67$
  - -Xmx4500m

### ◆ 보조 네임노드

- HADOOP\_SECONDARYNAMENODE\_OPTS
- 네임노드와 동일하게 설정

## 메모리 설정 - 셔플 메모리

- ◆ 각각의 맵 태스크는 환형 구조의 메모리 버퍼를 가지고 있다.
- ◆ mapreduce.task.io.sort.mb속성에 버퍼의 크기를 지정, 디폴트 100(MB)
- ◆ mapreduce.map.sort.spill.percent속성에 지정한 값(디폴트 0.80(80%))에 도달하면 백그라운드 스래드는 그 내용을 디스크로 Spill 함
  - 스페일은 mapreduce.cluster.local.dir 속성으로 명시된 디렉토리에 라운드로빈 방식으로 보내진다.
  - 스페일 하는 동안 맵 출력은 계속해서 버퍼에 쓰기를 진행함
  - 만약 이 시간동안 버퍼가 가득 차면 맵은 디스크로 보내는 작업이 완료될 때까지 블록됨

이 페이지는 여백 페이지입니다.



