

# 0816 하둡

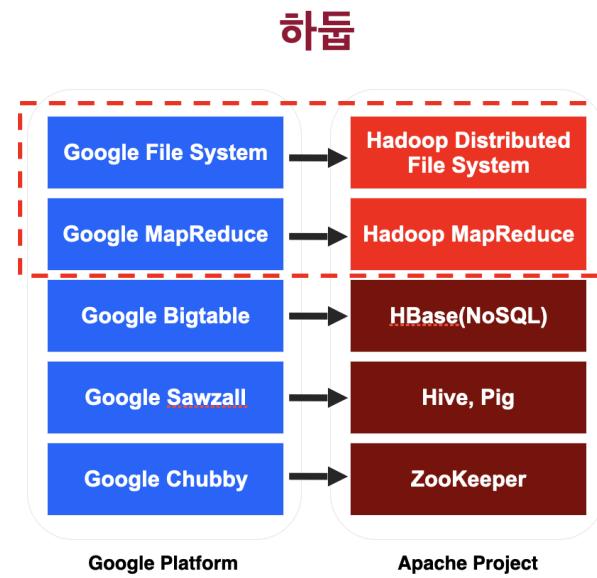
하둡은 2007년 탄생 이후 3점대 버전까지 나온 굉장히 성숙한 기술이다.

## 하둡의 탄생

인덱싱 라이브러리 lucene 오픈소스로 공개

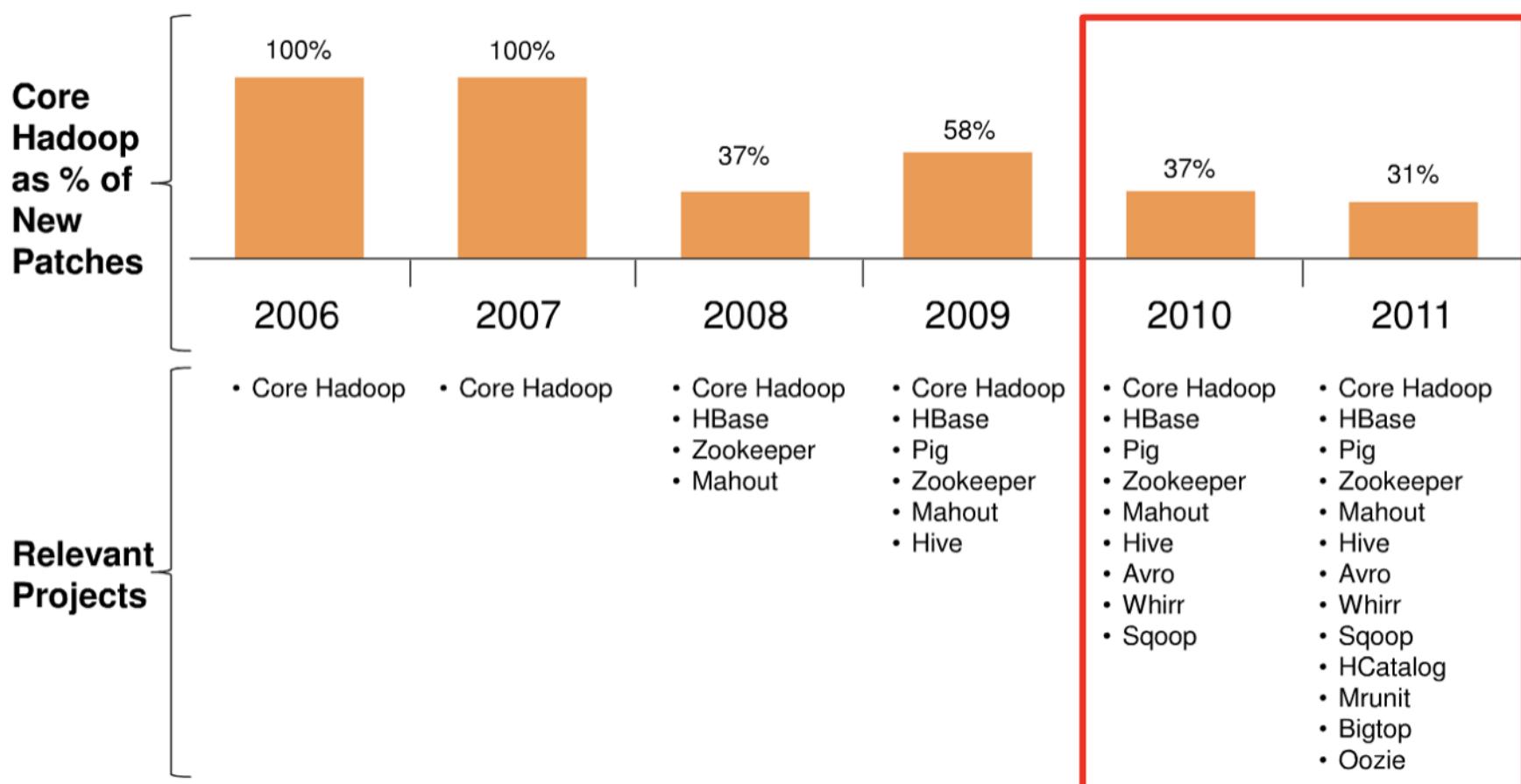
이것의 아들 프로젝트가 nutch가 나오고 이것의 아들 하둡이 나옴

- GFS : 2003년 구글 파일 시스템 논문 공개 더그 커팅이 gfs(GoogleFileSystem)과 GoogleMapReduce를 보고 nutch의 파일 시스템 개선한게 하둡임
- MapReduce : 구글에서 발명한 병렬분산처리를 하기위한 알고리즘
- 하둡 코어 : HDFS, HMR 이 둘을 합친 것을 말함



그 외에도 하둡은 구글에서 많이 가져온 것들이 있음

- bigtable (비정형 데이터 다루게 하는 것인데 이를 고대로 HBase로 가져와씀)
- hive는 pig의 단점을 개선한 것 (비정형 데이터는 어느정도는 정형 데이터로 저장해놓음 따라서 정형데이터를 다뤄야 한다. pig는 하둡 스크립트 언어인데 이를 sql로 쉽게 다룰 수 있게 한 것이 hive이다.)



- Sqoop : 관계형 데이터베이스를 하둡으로 보내거나 하둡 데이터를 관계형 데이터베이스로 보내거나 도와주는 것
- Oozie : etl(데이터를 가져오는 과정)에서 한번에 쿼리로 가져오지 않음(여러 커리로 전처리도 진행하고 가져옴) 이를 조율하는 스케줄러 요즘은 ariFlow를 더 많이 사용
- 주키퍼를 3대 또는 5대 구성하는 것을 Quorum이라 함

## 왜 빅데이터가 화두가 되었을까

Hadoop 덕분에 큰 데이터를 저장하기 위해 큰 비용이 들지 않음

큰 데이터 분석을 위해 많은 비용이 들지 않게 되어 이를 사람을 통해 분석하려고 했지만

이를 머신러닝을 통해 좀더 많은 분석을 하려고 한다.

데이터 분석 환경 구축이 중요해짐

## 왜 온프레미스를 사용할까?

클라우드 환경이 더 좋지만 비용적인 부분에서 클라우드가 비쌈

## 하둡 HDFS 이해

master slave 구조

master 역할 daemon과 slave 역할 daemon 있음

- 프로세스 : 작업의 단위 흐름의 단위
  - 프로세스가 계속 떠있는 것 : daemon
- 이제 노드들을 다 daemon이라고 부르게 됨

HFS master 와 HFS chunkserver 가 있음

HFS master는 meta 정보만 담름

- 마스터 slave 구조

하둡의 master daemon 이름은 name node라 불림

실제 데이터가 뜨는 HFS chunkserver(slave daemon)는 하둡에서 DataNode라 불림

- slave 노드의 쉬운 확장 가능 : 일반적인 환경에서 데이터 저장 공간을 늘리려면 아예 프로그램을 다시 구성해야 하지만 하둡은 서버만 늘리면 쉽게 확장 가능
- 신뢰성 보장 - 장애가 나더라도 복제가되어서 자동 관리가 됨

## 하둡 블록이란

- 하나의 파일을 여러개의 블록으로 저장
- 블록은 64mb 혹은 128mb(디폴트)로 저장됨
- 이때 하나의 블록은 기본적으로 3개 다른 서버에 똑같은(카피) 블록이 저장됨

ex) 100mb 저장한다면 300mb 저장 서버가 필요함(물론 디폴트 값임)

서버 하나가 고장나더라도 다른 서버에서 가져오면 됨 그리고 서버가 죽으면 다른 서버에 데이터 카피한다

따라서 최소 3개 서버에 데이터를 저장한다.

기본적으로 분산 시스템들은 기본적으로 서버에 데이터들을 여러개 카피하여서 저장하여 신뢰성을 보장한다.

하나의 서버 다운되면 underreplicated 되기 때문에 다른 서버에 블록 내용을 카피하게 됨

이때 다운된 서버를 다시복구하면 4개 서버에 4개의 파일이 저장되어 overreplicated 되기 때문에 main node에서 4개 서버 중 하나의 서버의 파일을 지우도록 명령함 —> 이것이 하둡내에서 자동을 동작됨

- **replication** : 데이터를 3개의 block에 디폴트로 할당하는 것

## 블록이 큰 이유는?

1. main(master) node의 과부화 방지
2. 탐색 비용 최소화할 수 있다.

## 로컬리티

- 서버1의 데이터는 서버1에서 처리한다는 것
- 서버2에서 서버1의 데이터를 처리를 원할 경우 속도저하 및 자원이 많이 소비되게 된다.

## 네임노드(NN)

- Name Node

## 보조 네임노드(SNN)

- Second Name Node

파일이 만들어지면 그 로그를 edits에 넣고 메모리에 올라감

보조 네임노드는 edits, fsimage를 합쳐주는 역할을 한다.

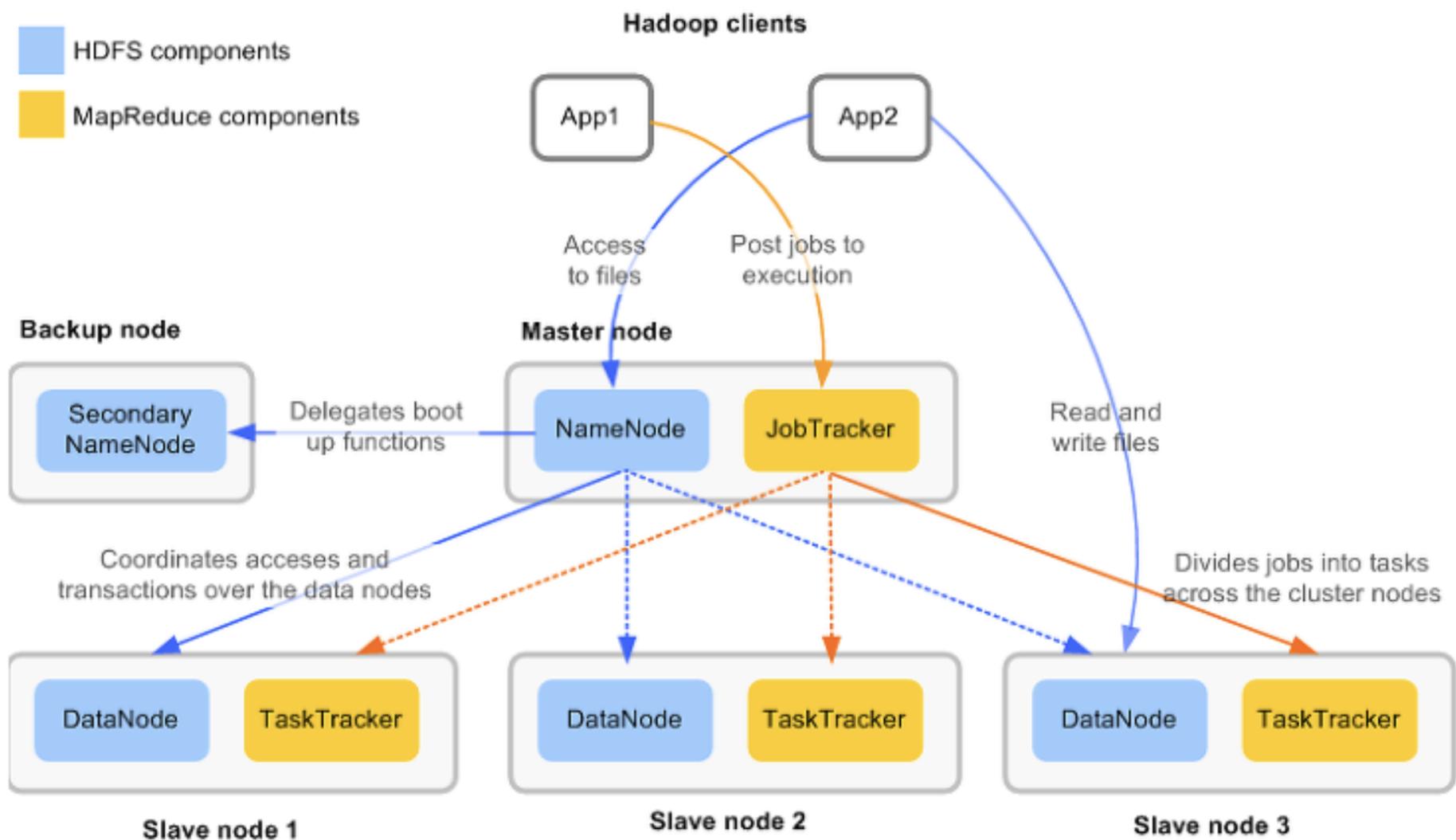
fsimage는 그동안 edits 파일 로그들을 램에 두는게 아니라 파일로 만들어주는 것

이를 보조 네임노드가 fsimage.ckpt로 만들어 줌

네임노드(NN)의 edits 로그는 보조 네임노드가 작동하면서도 계속 쌓이게 됨

역할 : 네임노드(NN)을 끄거나 오작동이 나면 log가 다 사라지게 되는데 이를 보조 네임노드(SNN)가 fsimange.ckpt 파일로 만들어서 계속 저장하면서 해결하게 됨(NN의 램 역할)

## 데이터 노드



- 보통 namenode 서버 따로 JobTracker 노드 따로 서버를 만듬

## 하둡 아키텍처

NN 가 다운되면 상황되서 전체 서버 다운을 방지하기 위해서

NN active 과 NN standby 를 둔다.

하지만, NN이 다운되면 NN standby가 실행되도록 로그도 전해주고 해야하는데 이를

Journal node(JN) : namenode의 edits 정보를 저장하고 공유하는 기능을 한다.

NN standby는 JN을 통해서 NN active의 정보를 공유 받게된다(ex: active NN의 log)

<https://cyberx.tistory.com/148>

## HDFS Federation

- 파일, 블록 개수가 많아지면 NN의 메모리가 부족해지는 상황이 발생
- 여러 대 하둡을 두어 위와같은 오류발생 방지

대부분 기업에서 사용 안함

## 맵리듀스

- HDFS를 다룰 수 있는 알고리즘
- Map function과 Reduce Function으로 구성됨
- key-value 구조가 핵심인 알고리즘

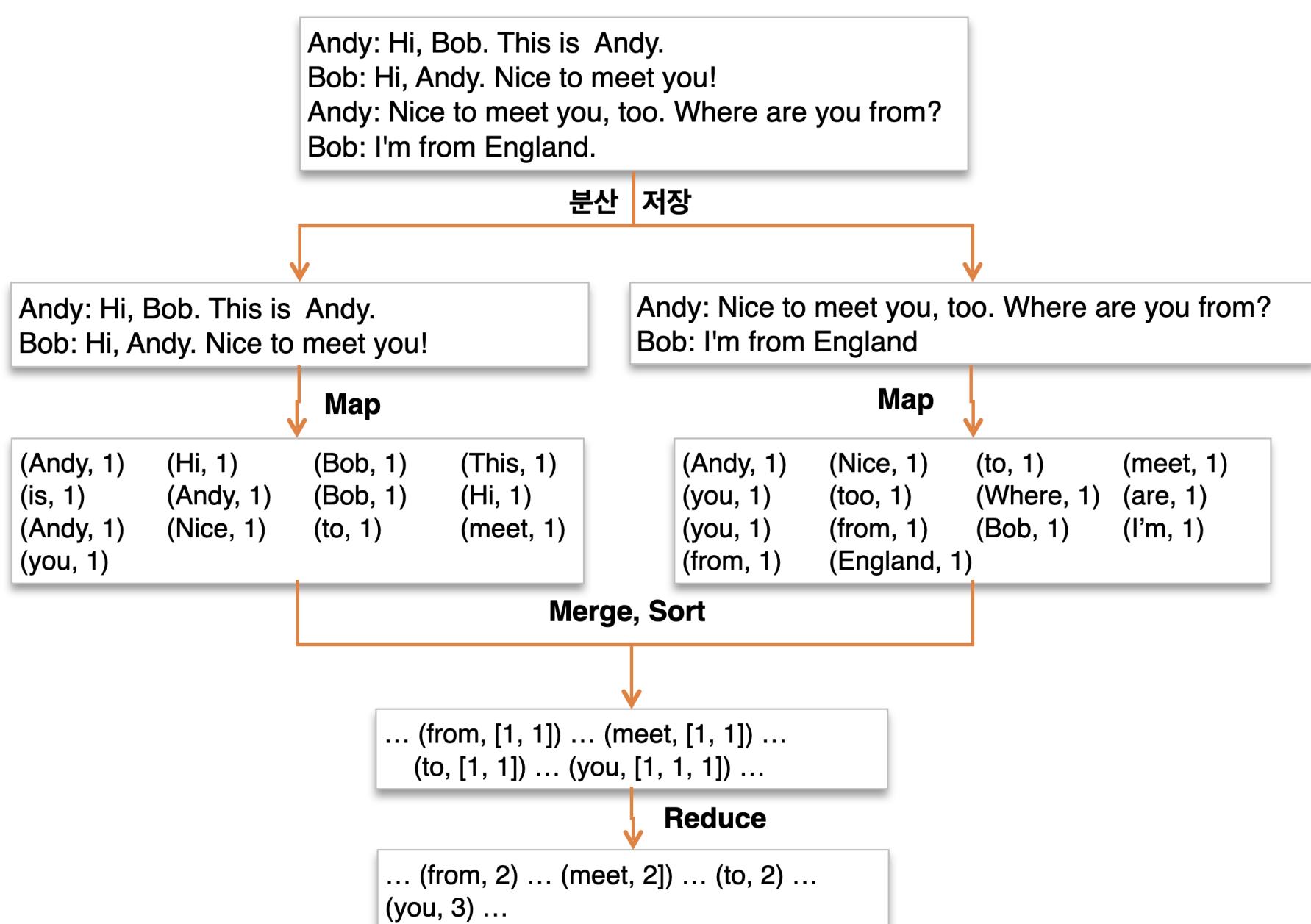
## 맵리듀스 구동 방식

classic : 1점대 버전에서 사용됨 지금은 버전 3임 사용 안함



하나의 파일을 128MB로 나누어 블록 단위로 저장하며 replication(같은 block 3개 카피)가 적용된다

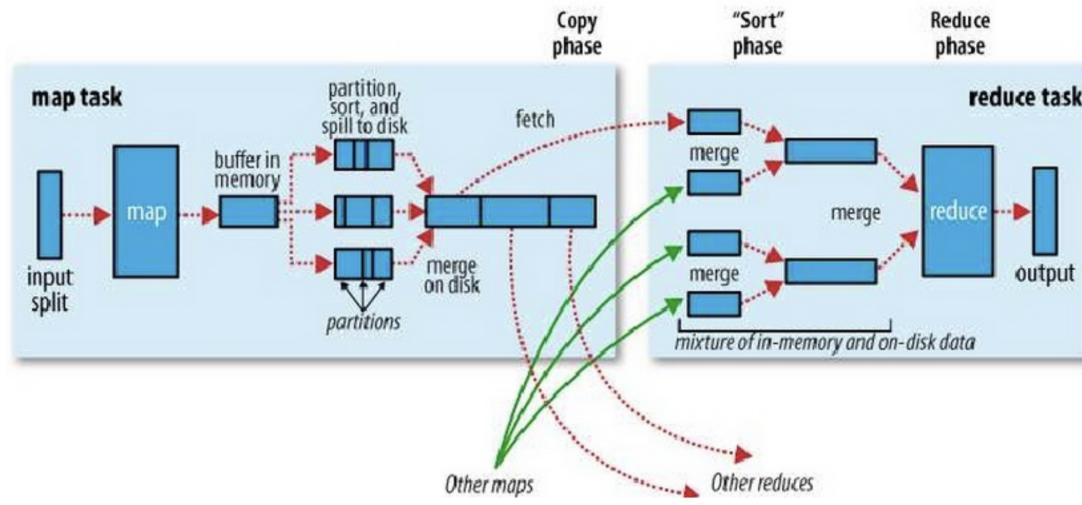
# 하둡 맵리듀스(MapReduce) 이해



- word를 카운트하여 계산하고자 함
1. Map : key, value로 저장하는 연산 진행, merge, sort
  2. Reduce : 같은 key의 대한 연산을 진행함

이런식으로 분산 처리를 진행하게 됨

## 맵리듀스(MapReduce) 데이터 처리흐름



input split이라는 class에 데이터를 저장

## 맵리듀스(MapReduce) 구현을 위한 인터페이스(Interface)

<b>Input</b>	TextInputFormat	Share
<b>Mapper</b>	$(k_1, v_1) \rightarrow (k_2, v_2)$	
<b>Combiner</b>	$(k_2, \text{list}(v_2)) \rightarrow (k_2, v_2')$	
<b>Partitioner</b>	$(k_2, v_2', \#reducer) \rightarrow \#partition$	
<b>Shuffle/sort</b>		
<b>Reducer</b>	$(k_2, \text{list}(v_2')) \rightarrow (k_3, v_3)$	
<b>Output</b>	TextOutputFormat	

- combiner는 reduce 작업을 하기 전에 같은 파티션에서 reduce 연산을 미리 진행함

이후 연산량이 줄어들어 다시 reduce 진행하면 연산량이 낮아짐

## YARN

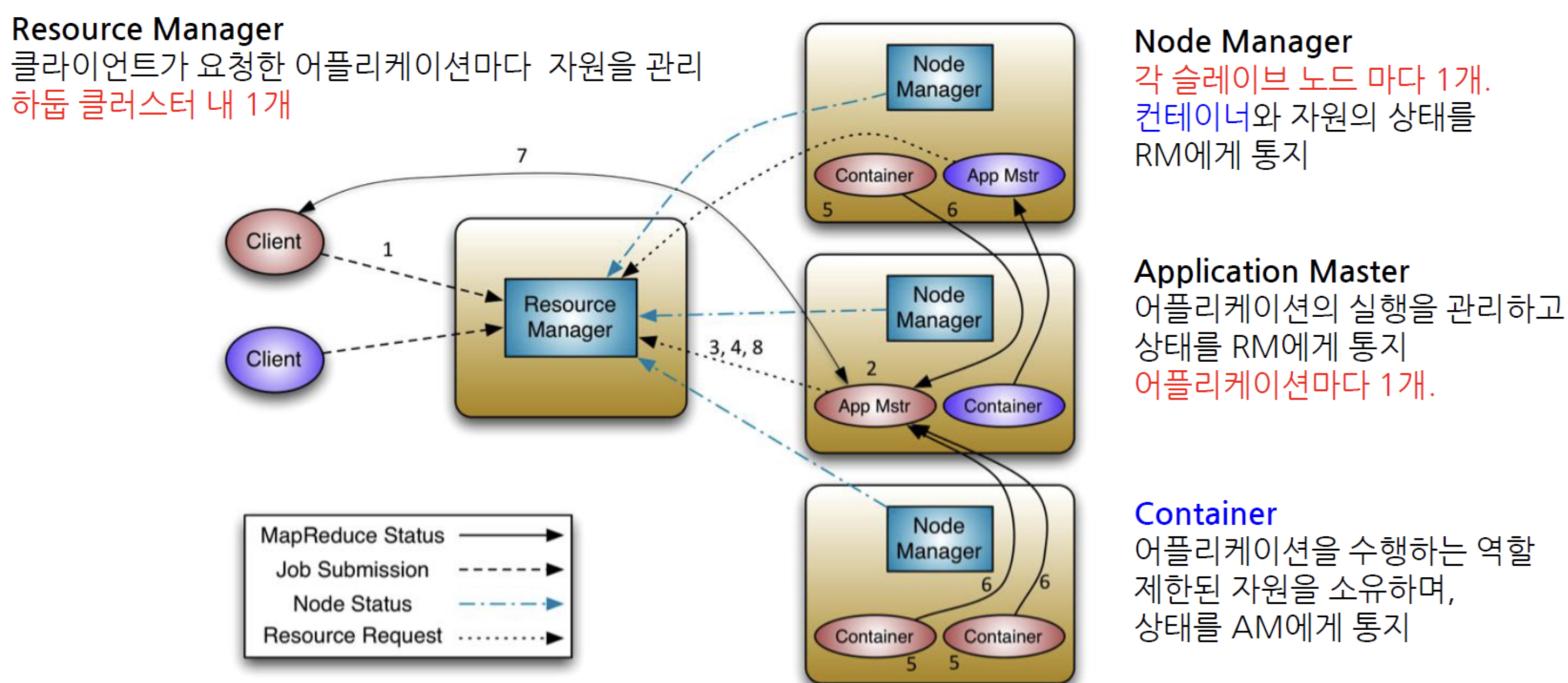
- 버전 2로 가서 바뀜

YARN이 나와서 Jobtracker와 tasktracker 도 사라짐 그 자리에 resource manager와 node manager가 들어가게 됨

## MapReduce I 과 YARN 의 차이점 (1/4)

- 이전 버전의 **MapReduce** 시스템은 4,000 노드 이상의 매우 큰 클러스터 상에서 동작 시 병목현상 이슈가 발생 함 (**JobTracker** 에 발생)
- 확장성 문제를 해결하기 위해 **JobTracker** 의 책임을 여러 **컨포넌트로** 분리
  - **ResourceManager** : 클러스터의 컴퓨팅 리소스 이용 상태를 관리하고 할당하는 것을 조정함
  - **ApplicationMaster** : 클러스터에서 실행중인 Job 의 Life Cycle 을 관리
  - **NodeManager** : 컨테이너를 모니터링하고, Job 이 할당 받은 그 이상의 리소스가 사용되지 않도록 보장
- **JobTracker** 와 다르게 응용 프로그램의 각 인스턴스는 **ApplicationMaster** 를 고정적으로 할당시켜 응용 프로그램의 지속성을 유지

## MapReduce I 과 YARN 의 차이점 (4/4)



[https://www.slideshare.net/TaeYoungLee1/20141029-25-hive?qid=63278cce-9697-4f54-b40c-8549b07f52a6&v=&b=&from\\_search=10](https://www.slideshare.net/TaeYoungLee1/20141029-25-hive?qid=63278cce-9697-4f54-b40c-8549b07f52a6&v=&b=&from_search=10)

- 클라이언트가 요청하면 resource manager가 다른 노드 매니저에게 명령을 전달하고 application 수행하고 지정 된 application 연산은 container들에서 수행된다.
- 모두 application 작업을 수행하면 컨테이너는 자동으로 종료된다.

## 버전3 하둡

- namenode가 standby와 active가 있지만 active active만 존재한다.

빠르게 active 노드가 다운되도 바로 실행되도록

## 하둡은 언제 효과를 볼까?

- 주관적인 생각

수백 TB 되야 효과를 볼 수 있다.

200 노드는 사용해야 하둡의 효과를 볼 수 있다.