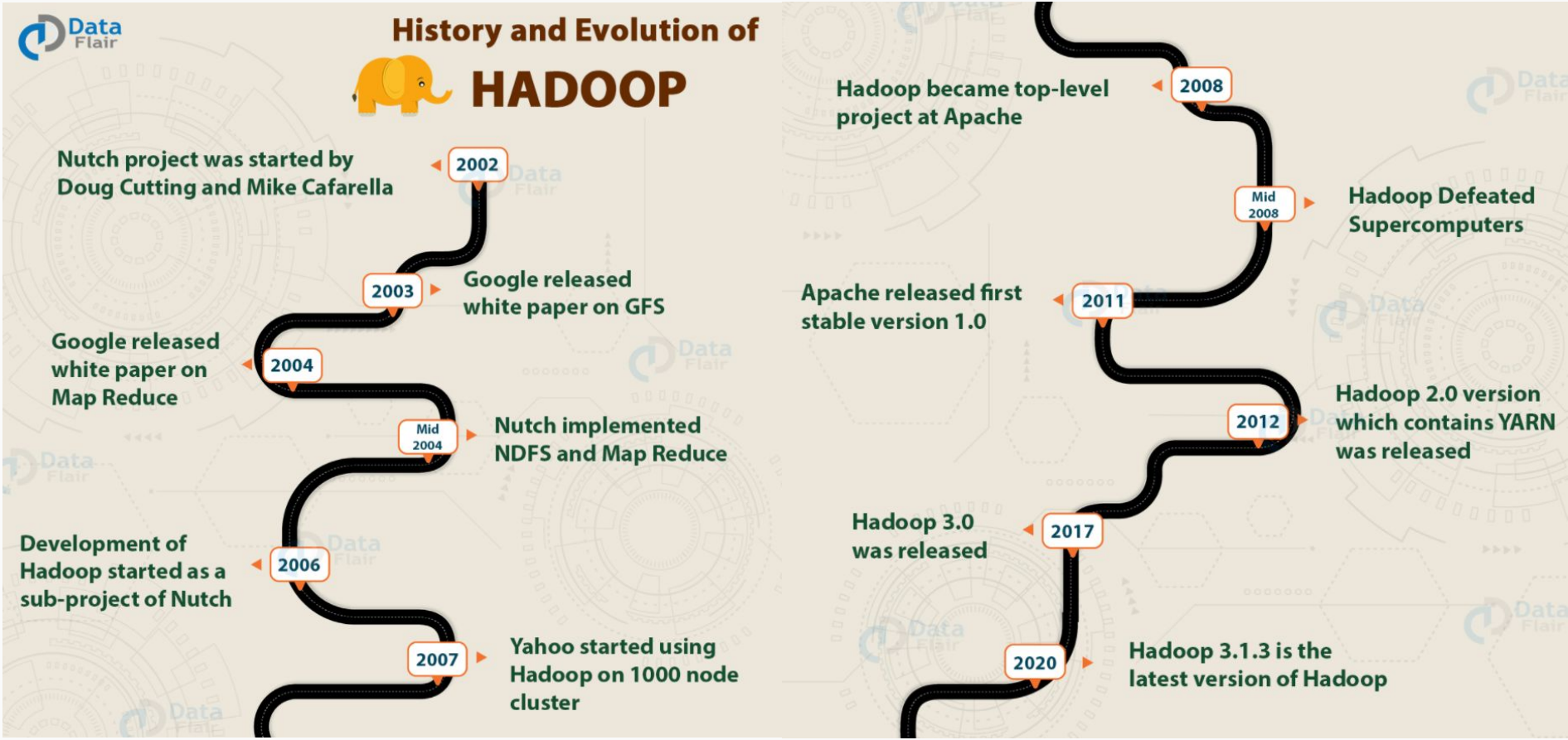
zum internet

하둡의 이해

빅데이터팀 김정익

Index

하둡의 진화과정

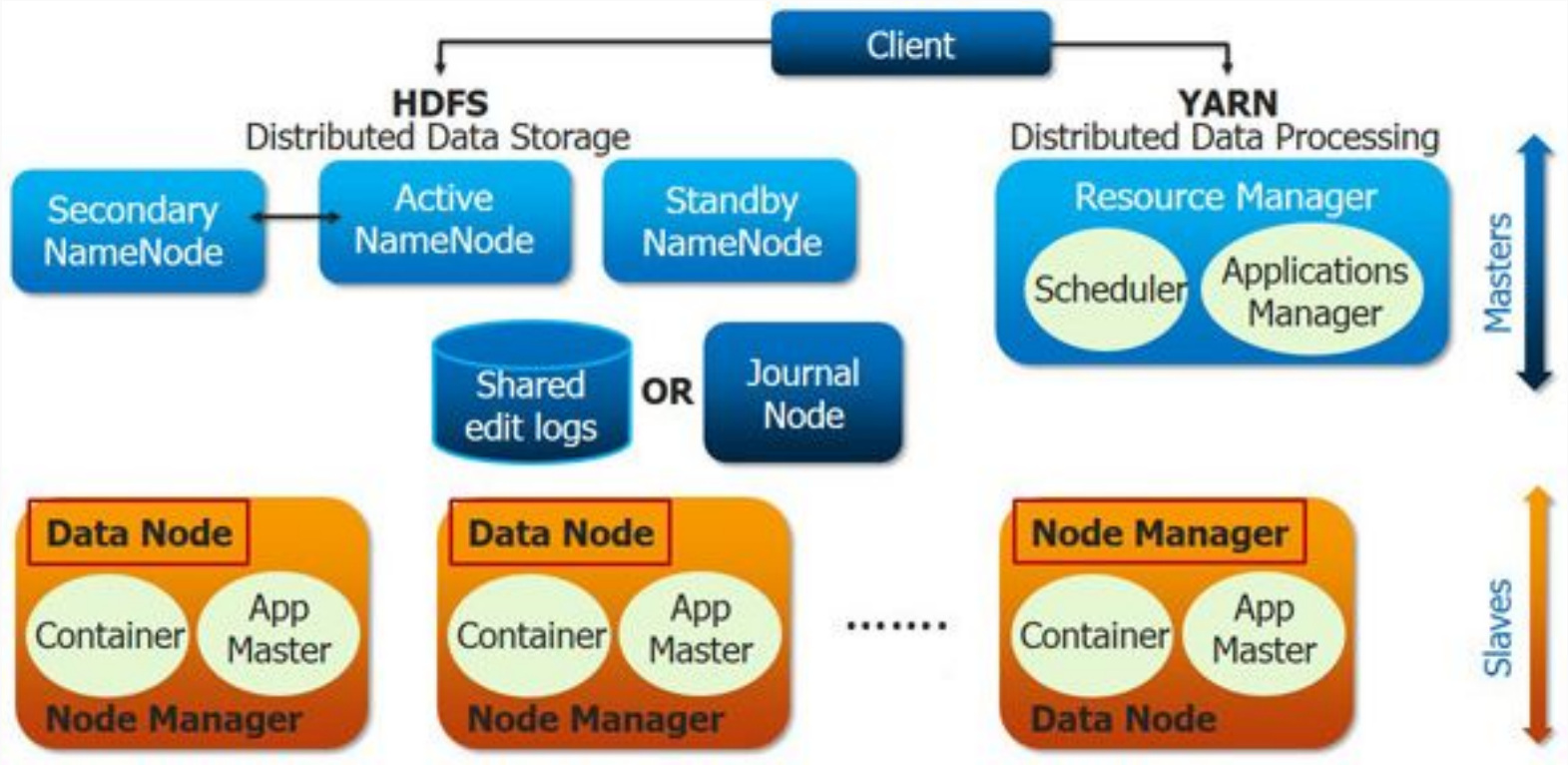

Data Flair
History and Evolution of
HADOOP
Nutch project was started by Doug Cutting and Mike Cafarella
2002
2003
Google released white paper on GFS
Google released white paper on Map Reduce
2004
Mid 2004
Nutch implemented NDFS and Map Reduce
Development of Hadoop started as a sub-project of Nutch
2006
2007
Yahoo started using Hadoop on 1000 node cluster
Hadoop became top-level project at Apache
2008
Mid 2008
Hadoop Defeated Supercomputers
Apache released first stable version 1.0
2011
2012
Hadoop 2.0 version which contains YARN was released
Hadoop 3.0 was released
2017
2020
Hadoop 3.1.3 is the latest version of Hadoop

하둡이 왜 필요할까?

“ 데이터 홍수의 시대 ”

- 빅데이터를 다루기 위한 안정성 있고 확장성 높은 플랫폼
- 큰 데이터를 저장하기 위해 큰 비용이 들지 않음
- Data로부터 새로운 Insight와 사업기회를 찾기 위한 노력과 시장 확대

Hadoop 2.0 구성도



HDFS 설계

1. 장애 복구

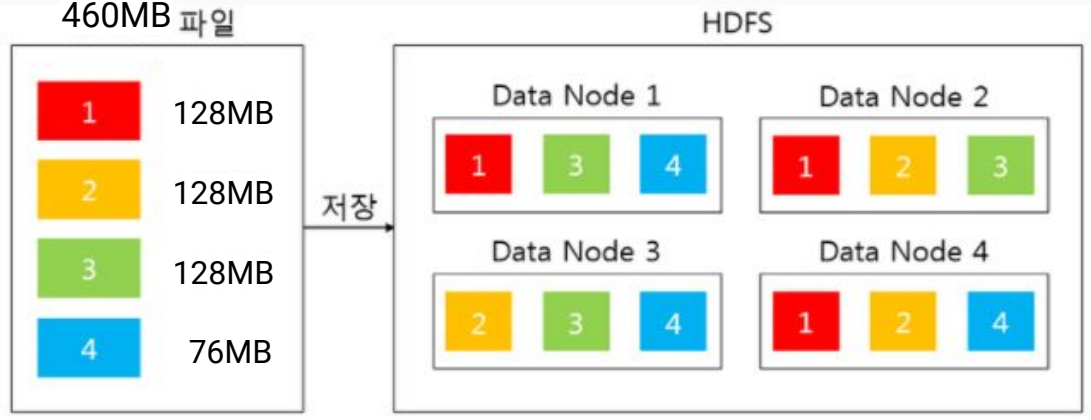
2. 스트리밍 방식의 데이터 접근

3. 대용량 데이터 저장

4. 데이터 무결성, 정합성

블록

하나의 파일을 여러 개의 Block으로 저장



블록이 큰 이유는?

탐색 비용을 최소화 하여 데이터 전송에 더 많은 시간을 할당하기 위해

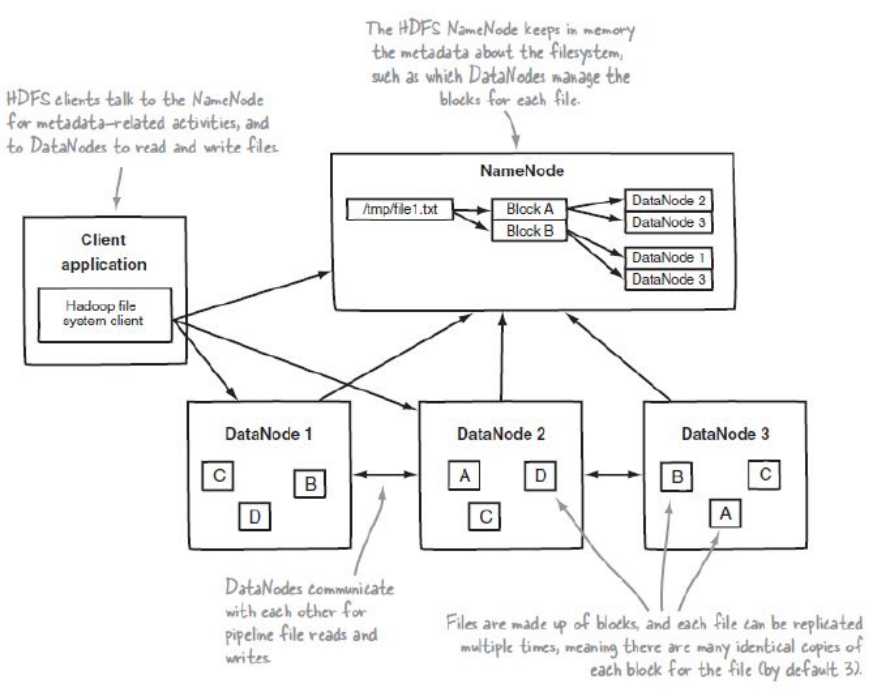
블록 추상화 개념의 장점

- 파일 하나의 크기가 단일 디스크 용량보다 커질 수 있다.
- 스토리지의 서브 시스템이 단순하다.
- replication 구현에 적합하다.

블록의 지역성(Locality)

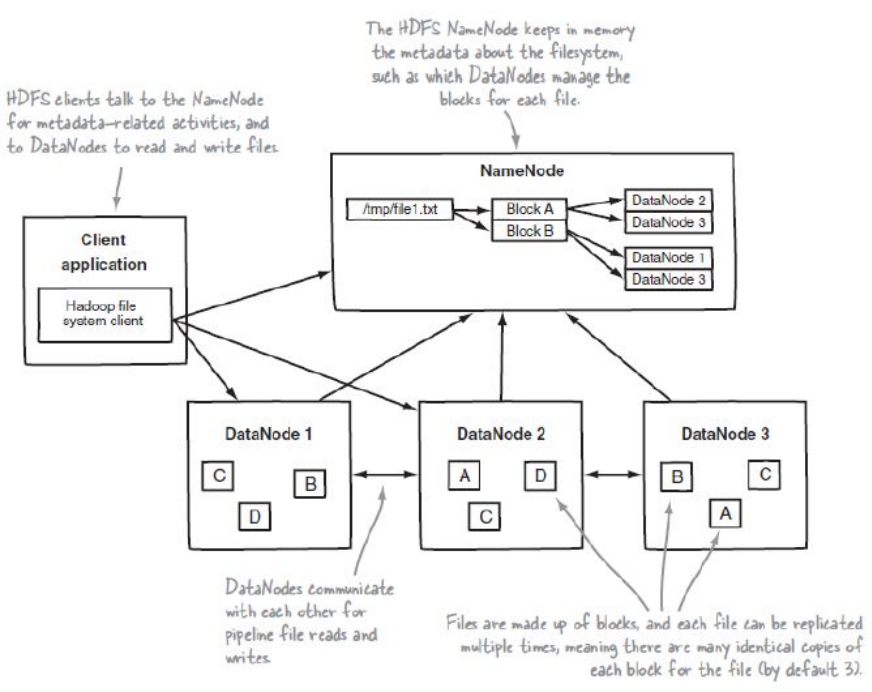
- 전통적인 분산 시스템은 스토리지에서 데이터를 프로세서로 이동
- 하둡은 데이터 지역성을 통해 네트워크를 이용한 데이터 전송 시간 감소
- 데이터가 있는 곳으로 소스를 보내 처리하는 방식
- 대용량 데이터 확인을 위한 디스크 탐색 시간 감소

Name Node



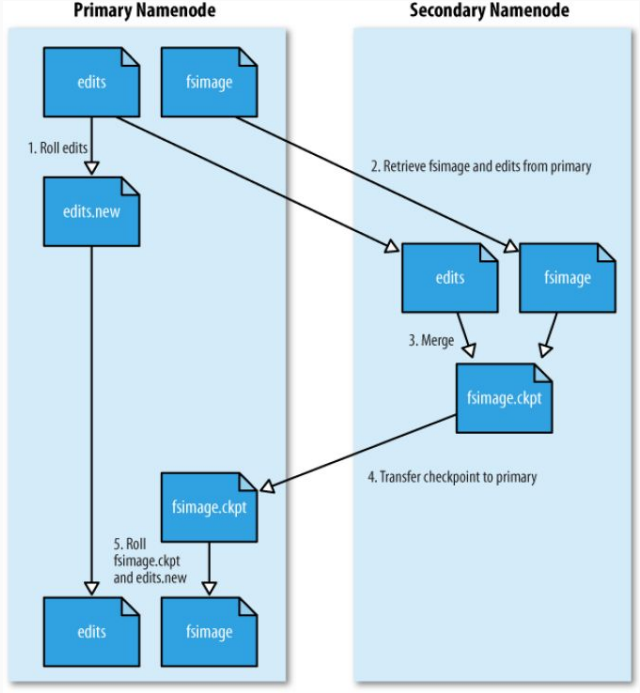
- 전체 HDFS에 대한 Name space를 관리
- DataNode로 부터 Block 리포트를 받음
- Data 에 대한 Replication 유지를 위한 커맨더 역할 수행
- 파일 시스템의 fsimage , edit logs 관리

Data Node



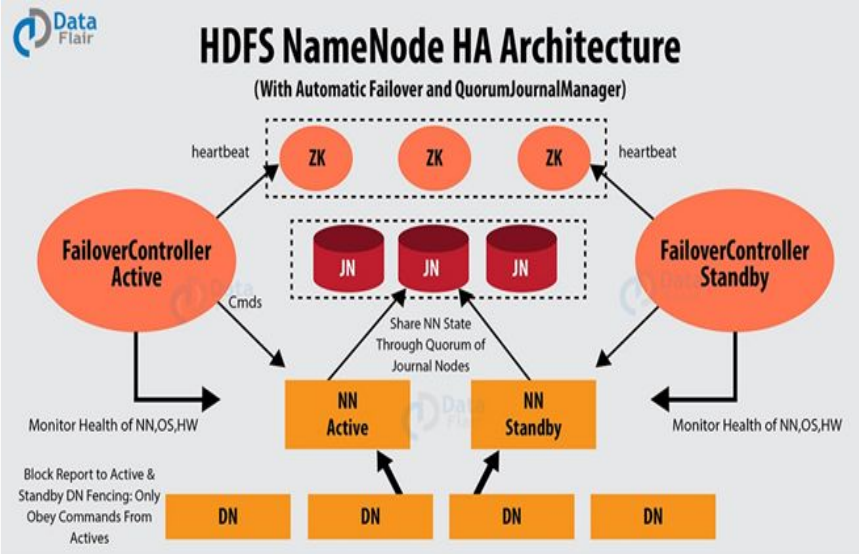
- 물리적으로 로컬파일 시스템에 HDFS 데이터를 저장
- 블록리포트를 주기적으로 네임노드에 보고

Secondary namenode



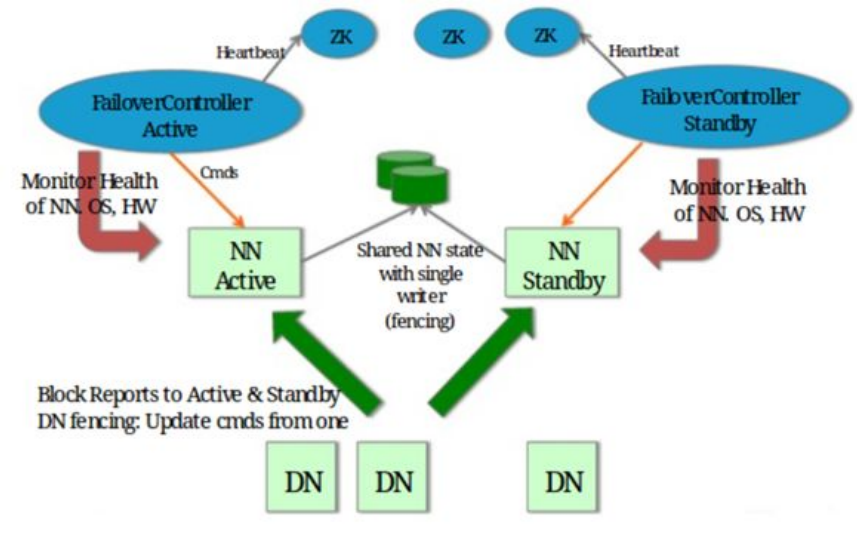
- edit log가 너무 커지지 않도록 주기적으로 fsimage와 edit log를 merge하여 새로운 fsimage를 생성 (Check Point)
- 네임노드에 장애가 발생할 것을 대비해서 네임스페이스 이미지의 복제본을 보관하는 역할
- fs.checkpoint.period 속성값으로 설정가능

HDFS 고가용성(High Availability)



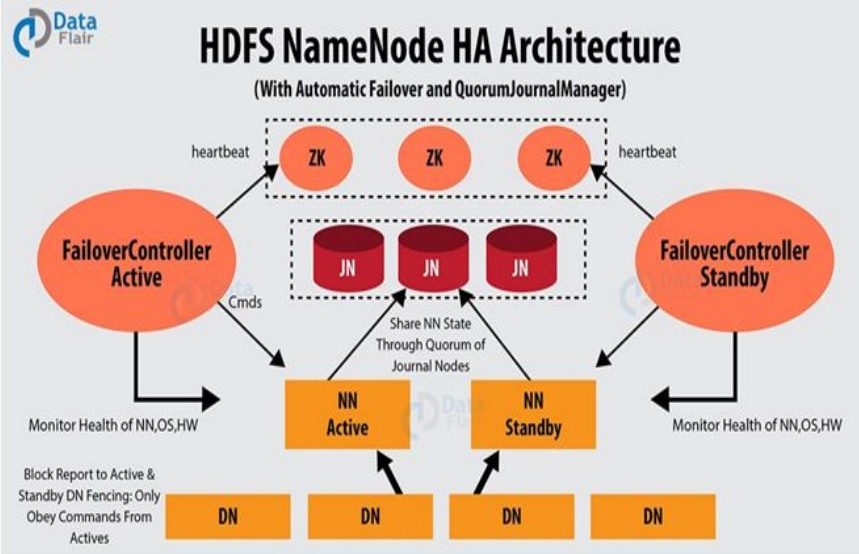
- Active NN와 Stanby NN를 이용하여 지원
- 동일한 메타데이터를 유지하고, 공유 스토리지를 이용하여 에디트파일을 공유
- 공유 방식은 NFS 방식과 QJM 방식이 있음
- 액티브 네임노드에 문제가 발생하면 스탠바이 네임노드가 액티브 네임노드로 동작
- 주키퍼를 이용하여 장애 발생시 자동으로 변경될 수 있도록 구성

NFS 공유 방식



- edit 파일을 공유 스토리지를 이용하여 공유하는 방법
- 네트워크 장애의 경우, 기존 Active NameNode가 Shared Storage와만 통신이 되는 상황이라면 기존 Active NameNode 는 여전히 Live한 상태로 SplitBrain 발생 가능성 존재함

QJM 공유방식



- 고가용성 edit log를 지원하기 위한 목적으로 설계됨
- QJM은 저널 노드 그룹에서 동작하며 (defalut 3) 각 edit log는 전체 저널 노드에 동시에 쓰여짐
- HDFS 고가용성은 active NN를 선출하기 위해 주키퍼를 이용

Journal Node 사용 시 Failover 절차

1. Active NameNode는 edit log 처리용 epoch number를 할당 받는다. 이 번호는 unique하게 증가하는 번호로 새로 할당 받은 번호는 이전 번호보다 항상 크다.

2. Active NameNode는 파일 시스템 변경 시 JournalNode로 변경 사항을 전송한다. 전송 시 epoch number를 같이 전송한다.

3. JournalNode는 자신이 가지고 있는 epoch number 보다 큰 번호가 오면 자신의 번호를 새로운 번호로 갱신하고 해당 요청을 처리한다.

4. JournalNode는 자신이 가지고 있는 번호보다 작은 epoch number를 받으면 해당 요청은 처리하지 않는다.
- 이런 요청은 주로 SplitBrain 상황에서 발생하게 된다.
- 기존 NameNode가 정상적으로 Standby로 변하지 않았고, 이 NameNode가 정상적으로 fencing 되지 않은 상태이다.

5. Standby NameNode는 주기적(1분)으로 JournalNode로 부터 이전에 받은 edit log의 txid 이후의 정보를 받아 메모리의 파일 시스템 구조에 반영

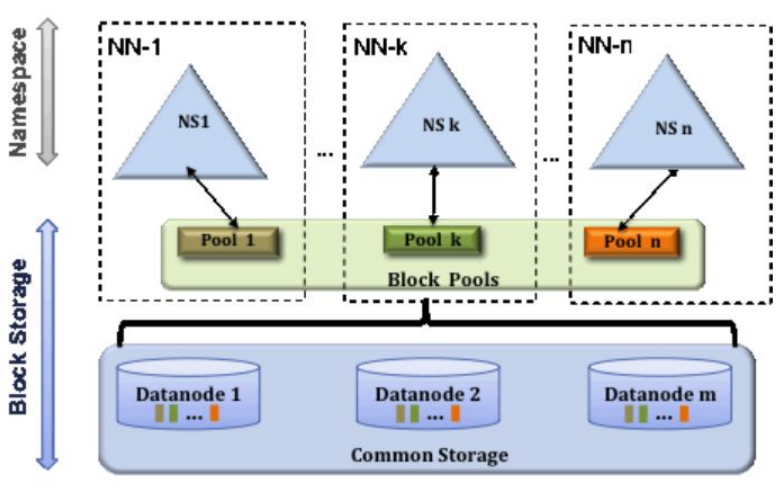
6. Active NameNode 장애 발생 시 Standby NameNode는 마지막 받은 transaction id 이후의 모든 정보를 받아 메모리 구성에 반영 후 Active NameNode로 상태 변환

7. 새로 Active NameNode가 되면 1번 항목을 처리한다.

블록 캐싱

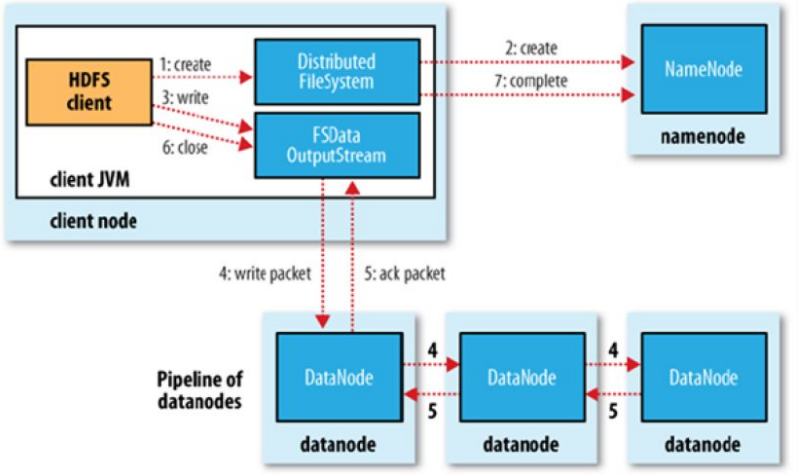
- 데이터 노드에 저장된 데이터 중 자주 읽는 블록은 블록 캐시(block cache)라는 데이터 노드의 메모리에 명시적으로 캐싱
- 파일 단위로 캐싱할 수도 있어서 조인에 사용되는 데이터들을 등록하여 읽기 성능을 높일 수 있음

HDFS 페더레이션



- 하나의 네임노드에서 관리하는 파일, 블록 개수가 많아지면 메모리에 병목이 생김
- 디렉토리 정보를 가지는 네임스페이스와 블록 정보를 가지는 블록 풀을 각 네임노드가 독립적으로 관리

HDFS 쓰기 연산 처리 메커니즘

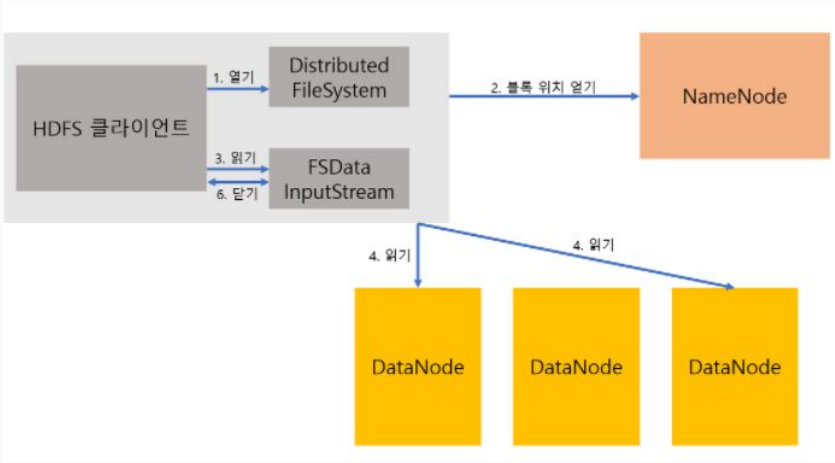


hdfs dfs –put {input_data} {input_location}

1. Client에서 DistributedFileSystem의 create()를 호출해 파일을 생성
2. DistributedFileSystem은 네임스페이스에 새로운 파일을 생성하기 위해 NN에 **RPC** 요청을 보냄
3. 요청이 NN의 검사를 통과하면 NN은 새로운 파일의 레코드를 만듦
4. DistributedFileSystem은 데이터를 쓰기 위해 FSDataOutputStream을 반환하고, DN와 NN의 통신 처리를 하는 **DFSOutputStream**으로 래핑됨.
5. DFSOutputStream은 데이터를 **패킷으로 분리**하여 **데이터 큐**로 보냄
6. **DataStreamer**는 패킷을 처리하기 위해 **NN에 DN 목록을 요청**하고 DN 목록은 파이프라인을 형성하여 **연쇄적으로 데이터를 전달**하여 저장
7. DFSOutputStream은 **ack큐**를 유지하며, 파이프라인의 모든 DN로부터 ack 응답을 받아 제거됨
8. 데이터 쓰기 완료 시 Client는 close() 메서드를 호출
9. 모든 패킷이 완전히 전송되면 NN에 신호를 보내 완료

HDFS 읽기 연산 처리 메커니즘

hdfs dfs –get {input_data} {input_location}

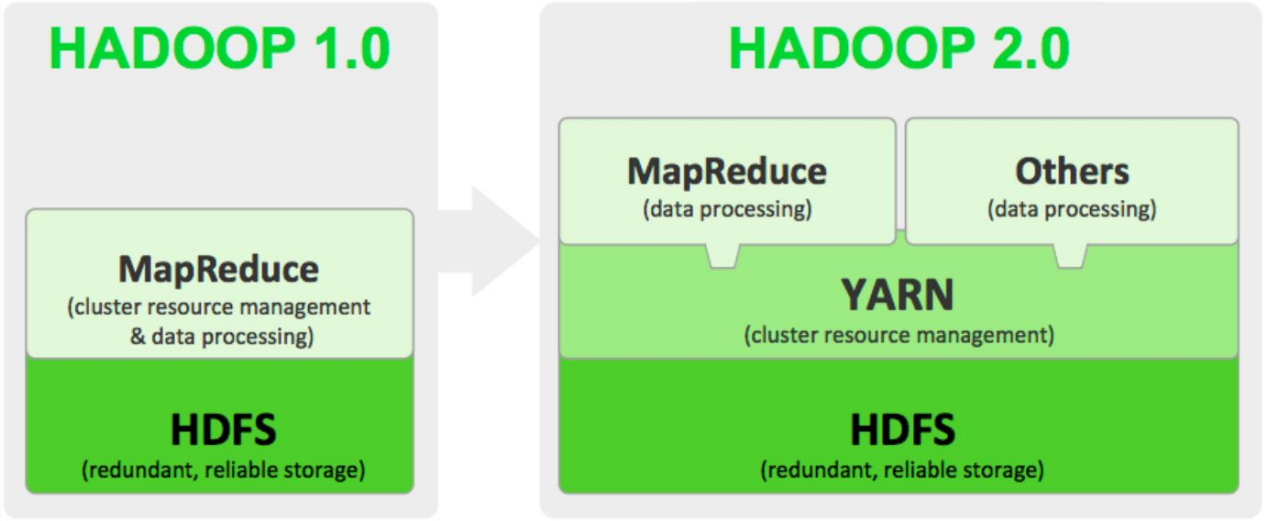


1. HDFS Client는 **DistributedFileSystem**의 인스턴스인 Filesystem 객체의 open() 메소드를 호출하여 원하는 파일을 연다.
2. Client는 **RPC**를 이용하여 NN를 호출하고 DN의 주소를 반환한다.
3. DistributedFileSystem은 클라이언트가 데이터를 읽을 수 있도록 **FSDataInputStream**을 반환하고, FSDataInputStream은 DN와 NN의 I/O를 관리하는 **DFSInputStream**을 래핑함.
4. 해당 스트림에 대해 read() 메서드를 반복적으로 호출하여 DN로부터 데이터 전송
5. Client는 스트림을 통해 block을 순서대로 하나씩 읽음
6. 모든 block 읽기가 끝나면 Client는 FSDataInputStream의 close() 메소드를 호출

YARN

- **HADOOP 1**
  - 클러스터의 규모와 상관없이 **1개의 Job Tracker**가 모든 노드의 job을 관리

- **HADOOP 2 YARN**
  - **Job Tracker**의 기능을 **Resource Manager**와 **Application Master**로 분리하여 확장성 제공
  - 맵리듀스 알고리즘 이외에 또 다른 분산처리 알고리즘을 처리할 수 있는 호환성 제공



YARN Daemons

Resource Manager

- master node에서 동작
- application들의 자원 요구의 할당 및 관리

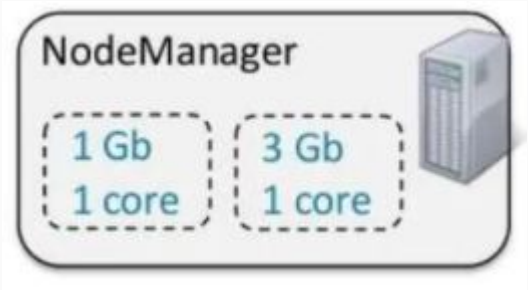
NodeManager

- slave node에서 동작
- node의 자원을 관리
- container에 node의 자원을 할당

YARN Daemons

Containers

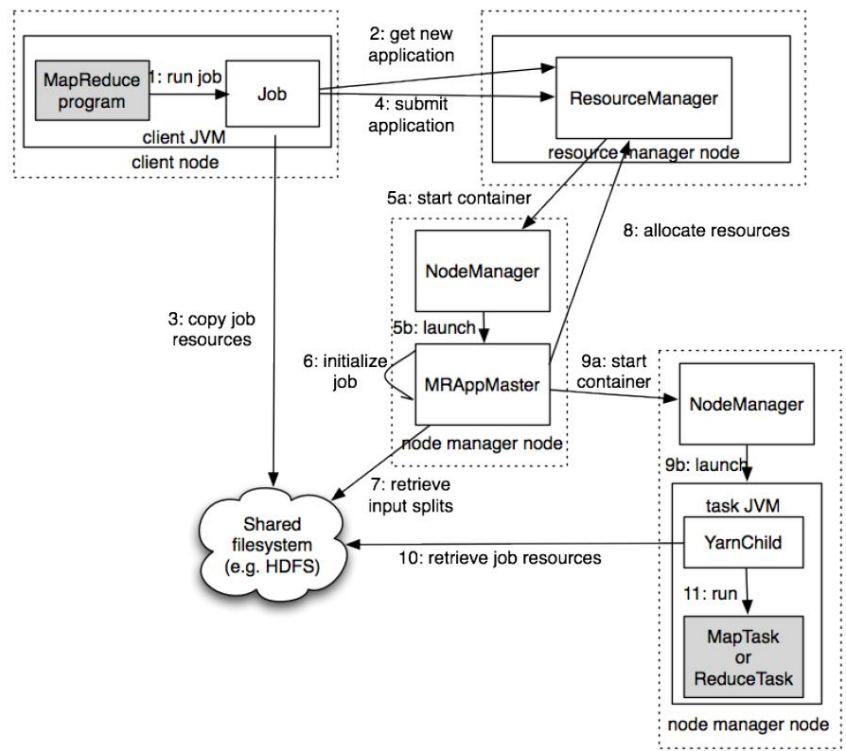
- RM의 요청에 의해 NM에서 할당
- slaver node의 CPU core, memory의 자원을 할당
- applications은 다수의 container로 동작



Application Master

- application당 한 개씩 존재
- application의 spec을 정의
- container에서 동작
- application task를 위해 container할당을 RM에게 요청

YARN의 동작 방식

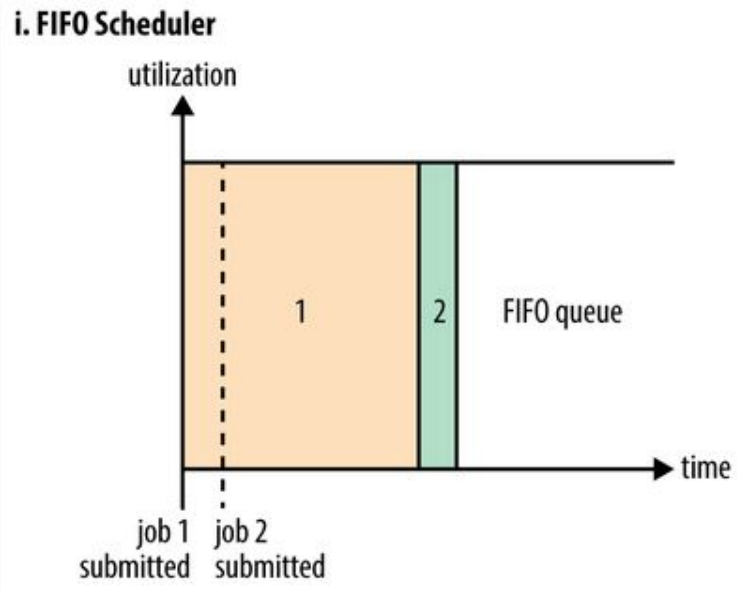


1. 클라이언트가 어플리케이션 실행을 Resource Manager에게 요청
2. Resource Manager는 Node Manager에게 어플리케이션 실행을 요청
3. Node Manager는 새로운 컨테이너(Application Master)를 실행
4. Application Master는 Resource Manager에게 리소스를 요청
5. Resource Manager는 클러스터를 구성하는 노드들의 가용 자원 목록을 전달
6. Application Master는 할당받은 Node Manager에게 컨테이너 실행을 요청
7. Node Manager들은 Container를 생성하여 Task를 실행
8. 어플리케이션이 종료되면 Application Master가 종료되고 Resource Manager는 Application Master가 할당 받았던 자원을 회수함

YARN 스케쥴러

FIFO Scheduler

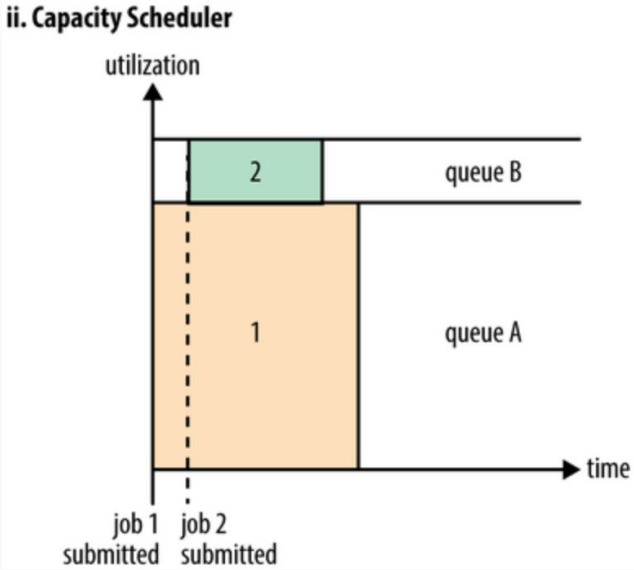
- 큐에 들어오는 순서대로 잡을 순차적으로 실행
- 하나의 잡을 처리할 때 속도가 빠름
- 병렬처리에 부적절함



YARN 스케쥴러

Capacity Scheduler

- Hadoop 2.x YARN의 기본 스케쥴러
- 트리 형태로 계층화된 큐를 선언하고 전체 클러스터의 지정된 가용량을 미리 할당.
- 각각의 큐는 클러스터의 가용량을 공유
- 만약 클러스터의 자원에 여유가 있다면 설정을 이용하여 각 큐에 설정된 용량 이상의 자원을 이용하게 할 수 있고, 운영 중에도 큐를 추가할 수 있는 유연성
- 대형 job은 FIFO보다 늦게 끝난다.



YARN 스케쥴러

Capacity Scheduler

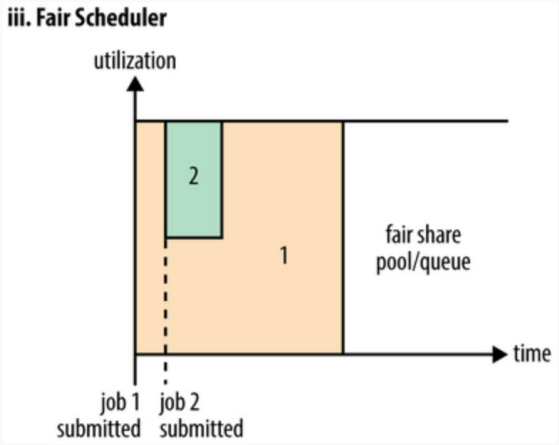
설정값	비고
yarn.scheduler.capacity.maximum-applications	PRE, RUNNIG 상태로 설정 될 수 있는 최대 애플리케이션의 개수
yarn.scheduler.capacity.maximum-am-resource-percent	애플리케이션 마스터(AM)에 할당 가능한 최대 비율. AM은 실제 작업이 돌지 않고 작업을 관리하는 역할을 하기 때문에 작업에 많은 컨테이너를 할당하기 위해 이 값을 적당히 조절해야 함
yarn.scheduler.capacity.root.queues	root 큐에 등록하는 큐의 이름. root큐는 하위에 동록할 큐를 위해 논리적으로만 존재
yarn.scheduler.capacity.root.[큐이름].maximum-am-resource-percent	큐에서 AM이 사용할 수 있는 자원의 비율
yarn.scheduler.capacity.root.[큐이름].capacity	큐의 용량 비율
yarn.scheduler.capacity.root.[큐이름].user-limit-factor	큐에 설정된 용량 * limit-factor 만큼 다른 큐의 용량을 사용할 수 있음. 클러스터의 자원을 효율적으로 사용할 수 있음. maxmimum-capacity 이상으로는 이용할 수 없음.
yarn.scheduler.capacity.root.[큐이름].maximum-capacity	큐가 최대로 사용할 수 있는 용량

YARN 스케쥴러

Fair Scheduler

- 페어 스케줄러는 제출된 작업이 동등하게 리소스를 점유할 수 있도록 지원
- 작업 큐에 작업이 제출되면 클러스터는 자원을 조절하여 모든 작업에 균등하게 자원을 할당
- 메모리와 CPU를 기반으로 자원을 설정 가능

- 페어 스케줄러는 트리 형태로 계층화된 큐를 선언하고, 큐별로 사용가능한 용량을 할당하여 자원을 관리함



설정값	기본값	비고
yarn.scheduler.fair.allocation.file	fair-scheduler.xml	설정파일의 이름
yarn.scheduler.fair.user-as-default-queue	true	큐이름을 지정하지 않았을 때 기본큐의 사용 여부
yarn.scheduler.fair.preemption	false	우선순위 선점의 사용 여부

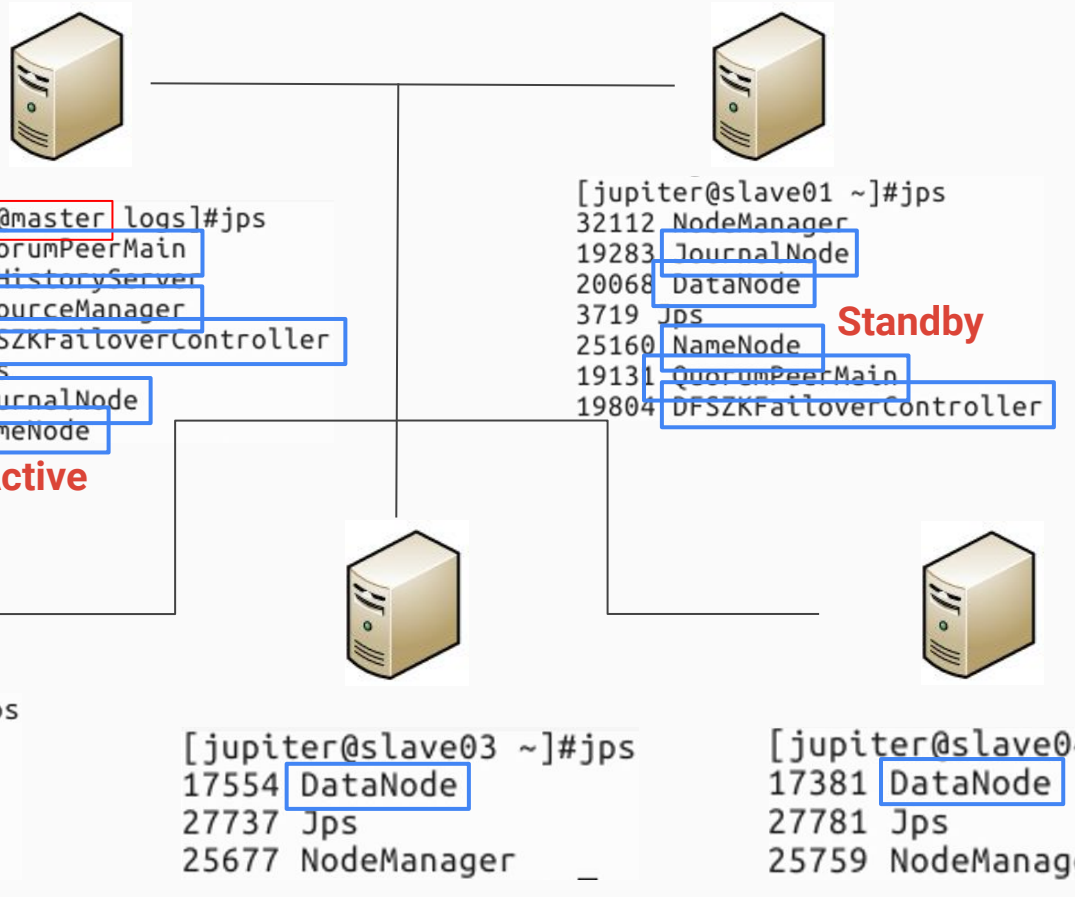
클러스터 구성

$HADOOP_HOME/sbin/start-dfs.sh

$HADOOP_HOME/sbin/start-yarn.sh

$HADOOP_HOME/bin/hdfs namenode -bootstrapStandby

$HADOOP_HOME/sbin/hadoop-daemon.sh start namenode



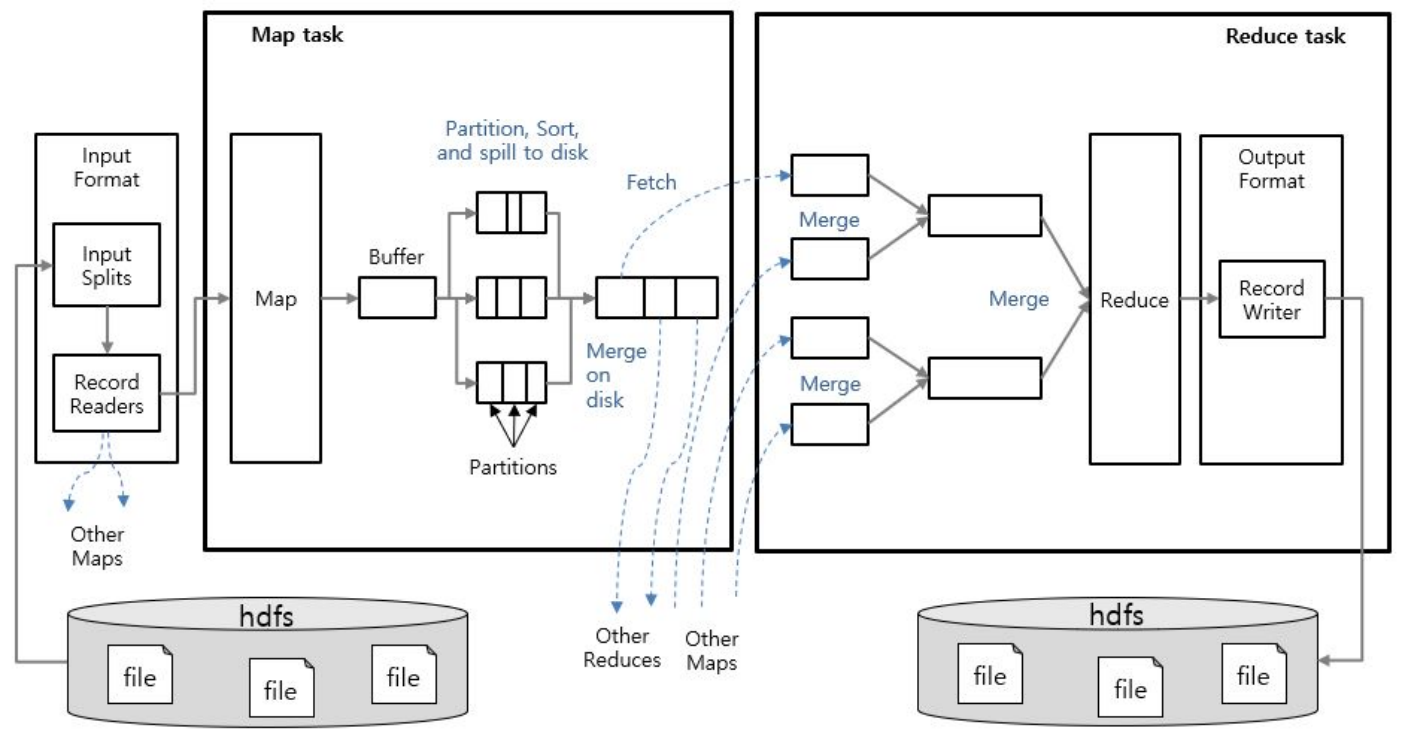
Map Reduce

맵리듀스란?

- 2004년 구글에서 처음 소개된 Large Data Processing 알고리즘
- Hadoop MapReduce 프레임워크로 구현됨
- Map Function과 Reduce Function 으로 구성
- 특정 스키마, 질의에 의존적이지 않아 유연함
- Job에 대한 처리는 내부적인 로직에 의해 이루어짐

맵리듀스 작동방법

Job 클라이언트가 수행하는 작업의 기본 단위로, Map task와 Reduce task로 나눠서 실행

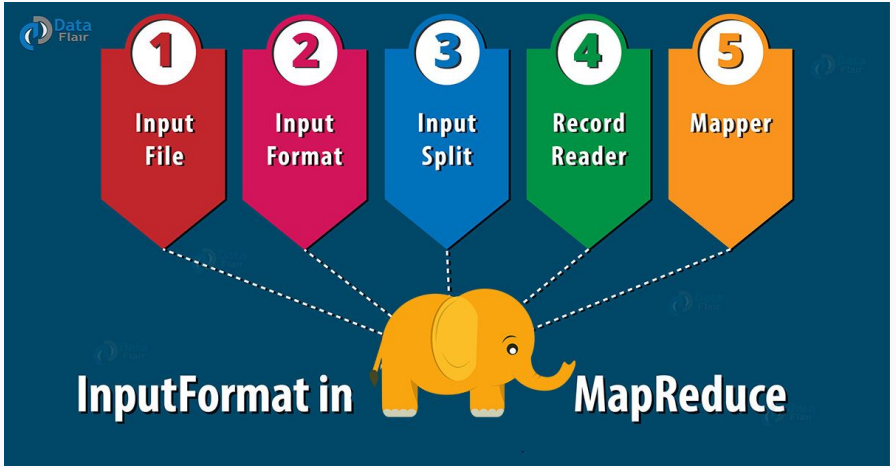


Map task 단계

데이터를 읽어 Key, Value 구조의 입력 레코드를 구성한 뒤 중간 단계의 레코드로 변환하는 작업

InputFormat

- 입력 파일을 분할하여 InputSplit을 생성한 뒤 각 개별 Mapper에 할당
- Mapper가 처리할 입력 레코드를 InputSplit에서 어떻게 추출해야 할 지 명시하는 RecordReader 구현

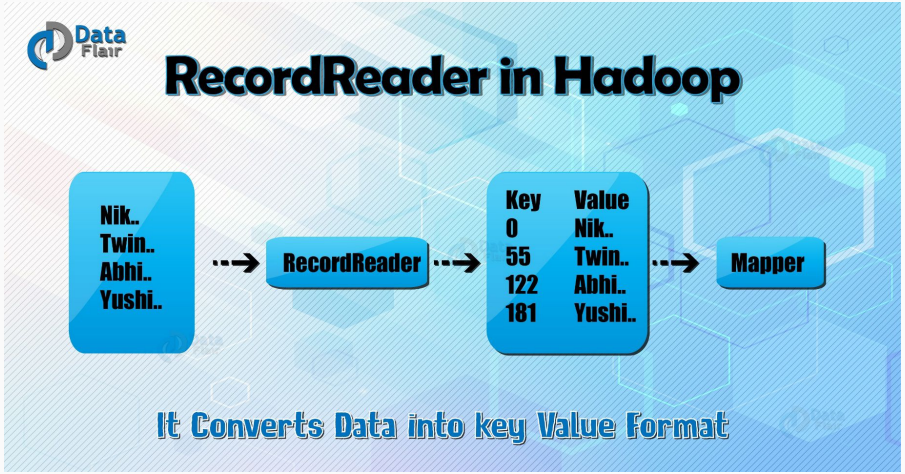


Map task 단계

데이터를 읽어 Key, Value 구조의 입력 레코드를 구성한 뒤 중간 단계의 레코드로 변환하는 작업

RecordReader

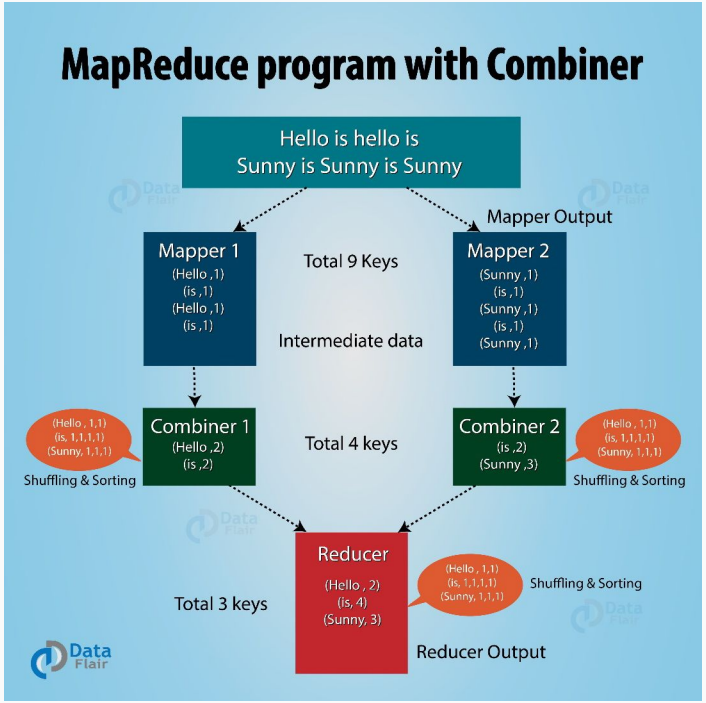
- 크기 기준으로 분리된 InputSplit을 Mapper에서 Key, Value 형태로 처리할 수 있도록 레코드 기반의 형태로 변환하는 작업 수행
- LineRecordReader는 TextInputFormat이 제공하는 기본 레코드 입력기



Mapper 의 역할

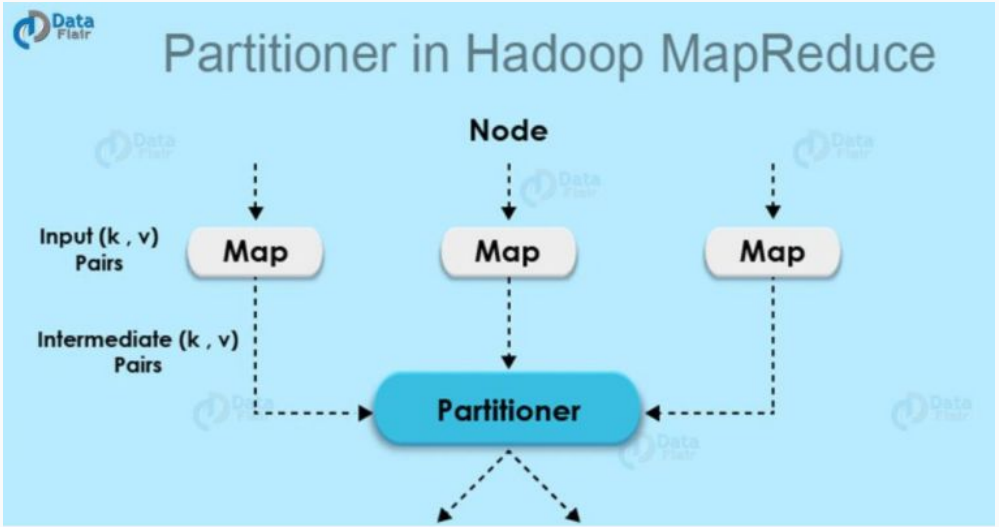
- 맵에서는 입력 데이터를 읽어 **map** 함수에서 사용자가 정의한 연산을 수행함
- 맵 함수의 처리 결과는 우선 메모리 버퍼에 저장되며 버퍼의 내용이 한계치에 도달하면 일괄로 디스크에 저장됨.
- 이 때, 맵의 출력 데이터는 키 값에 의해 정렬되고 리듀서 수에 맞게 파티션별로 나눠 저장
- **컴바이너** 함수의 결과 데이터에 대해 정렬과 **파티셔닝** 작업이 발생
- 셔플링 과정에서 발생하는 트래픽이 발생함

Combiner 의 역할



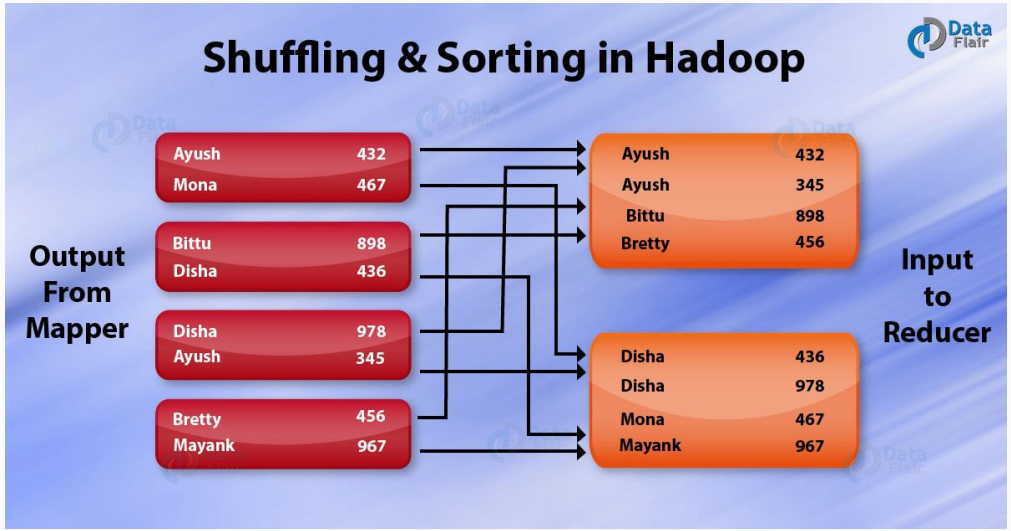
- Mapper의 중간 출력 결과를 받아서 Reducer에게 전달해주는 역할
- Mapper와 Reducer 사이의 셔플할 데이터의 양을 줄여 물리적 트래픽을 감소시키는 역할

Partitioner 의 역할



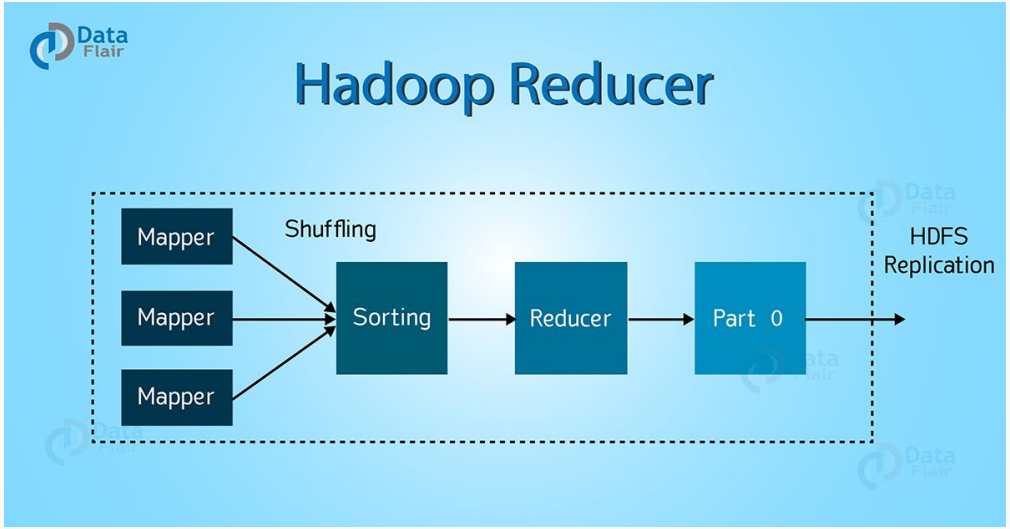
- 서로다른 Mapper에서 생성된 중간결과 Key-Value 쌍을 Key 중심으로 같은 Key를 갖는 데이터는 물리적으로 동일한 Reducer 서버로 보내기 위한 용도로 사용
- 기본 파티셔너는 데이터의 Key 값을 hash처리 하고 Reducer의 개수만큼 모듈러 연산

Shuffling & Sort 과정



- Mapper의 결과 데이터 쌍이 Reducer로 전달 되는 Shuffling과정에서 트래픽이 발생
- 서로 다른 Mapper로 부터 받은 데이터를 Key 중심으로 Sorting

Reducer의 역할



- Mapper의 출력 결과를 입력으로 받아서 데이터를 처리
- 처리된 데이터를 OutputFormat의 형태에 맞게 결과로 출력
- RecordWriter로 key, value 쌍을 출력

Map Reduce

$HADOOP_HOME/bin/yarn jar
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.10.1.jar wordcount /input /output

```
[jupiter@master /]#$HADOOP_HOME/bin/yarn jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.10.1.jar wordcount /input /output
21/12/07 20:18:57 INFO input.FileInputFormat: Total input files to process : 1
21/12/07 20:18:57 INFO mapreduce.JobSubmitter: number of splits:1
21/12/07 20:18:58 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1638859252625_0001
21/12/07 20:18:58 INFO conf.Configuration: resource-types.xml not found
21/12/07 20:18:58 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
21/12/07 20:18:58 INFO resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
21/12/07 20:18:58 INFO resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
21/12/07 20:18:59 INFO impl.YarnClientImpl: Submitted application application_1638859252625_0001
21/12/07 20:18:59 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1638859252625_0001/
21/12/07 20:18:59 INFO mapreduce.Job: Running job: job_1638859252625_0001
21/12/07 20:19:10 INFO mapreduce.Job: Job job_1638859252625_0001 running in uber mode : false
21/12/07 20:19:10 INFO mapreduce.Job:  map 0% reduce 0%
21/12/07 20:19:16 INFO mapreduce.Job:  map 100% reduce 0%
21/12/07 20:19:26 INFO mapreduce.Job:  map 100% reduce 100%
21/12/07 20:19:26 INFO mapreduce.Job: Job job_1638859252625_0001 completed successfully
21/12/07 20:19:26 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=1836
                FILE: Number of bytes written=427105
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1453
                HDFS: Number of bytes written=1306
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=4006
                Total time spent by all reduces in occupied slots (ms)=6115
                Total time spent by all map tasks (ms)=4006
                Total time spent by all reduce tasks (ms)=6115
                Total vcore-milliseconds taken by all map tasks=4006
                Total vcore-milliseconds taken by all reduce tasks=6115
                Total megabyte-milliseconds taken by all map tasks=4102144
                Total megabyte-milliseconds taken by all reduce tasks=6261760
        Map-Reduce Framework
                Map input records=31
                Map output records=179
                Map output bytes=2055
                Map output materialized bytes=1836
                Input split bytes=87
                Combine input records=179
                Combine output records=131
                Reduce input groups=131
                Reduce shuffle bytes=1836
                Reduce input records=131
                Reduce output records=131
                Spilled Records=262
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=228
                CPU time spent (ms)=1330
```

```
[jupiter@master /]#$HADOOP_HOME/bin/hdfs dfs -cat /output/*
(BIS),  1
(ECCN)  1
(TSU)   1
(see    1
5D002.C.1,      1
740.13) 1
<http://www.wassenaar.org/>     1
Administration  1
Apache  1
BEFORE  1
BIS     1
Bureau  1
Commerce,       1
Commodity       1
Control 1
Core    1
Department      1
ENC     1
Exception       1
Export  2
For     1
Foundation      1
Government      1
Hadoop  1
Hadoop, 1
Industry        1
Jetty   1
License 1
Number  1
Regulations,    1
SSL     1
Section 1
Security        1
See     1
Software        2
Technology      1
```

zum internet

감사합니다.