HADOOP

18기 분석 김가영

하둡 개요

High-Availability Distributed Object-Oriented Platform

대용량 데이터를 분산 처리할 수 있는 자바 기반의 오픈소스 프레임워크

범용 컴퓨터 여러대를 `클러스터화`하고, 큰 크기의 데이터를 클러스터에서 병렬로 `동시 처리`하여 처리속도를 높임

HDFS라는 데이터 저장소와 맵리듀스라는 분석 시스템을 통해 분산 프로그램 수행

하드 디스크의 용량은 증가, 데이터를 읽는 속도는 느림 ➡ 단일 디스크: 너무 많은 시간이 걸림, 쓰는 것은 더 느림 병렬로 작업하자! ➡

디스크에 데이터를 병렬로 쓰거나 읽을 때 고려할 점
하드웨어 장애: 하둡에서의 HDFS
- 장애로 인해 데이터 손실이 일어날 가능성이 높음
- 데이터를 여러 곳에 복제하는 방식 = RAID
데이터 결합: 하둡에서의 MapReduce
- 데이터 일치를 지키는 것이 어려움
- key-value 쌍의 계산으로 변환

하둡 특징&장단점

Distributed File System	분산 및 병렬 처리에 적합한 구조로 파일 저장
Distributed/Parallel Computing Framework	Map, Reduce 기반의 컴퓨팅 플랫폼, 쉽게 개발 가능
Scalablility	장비를 증가시킬수록 선형적으로 성능 향상
Data Locality	데이터가 있는 곳으로 소스를 보내 로직을 수행
Fault Tolerant	데이터 블록의 복사본 중복 저장
Open Source Project	-

[장점]
- 오픈 소스로 라이선스 비용 부담 적음
- 시스템 중단 없이 장비 추가가 용이
- 안정적이고 확장성이 높음
- 대용량 데이터의 일괄 처리 가능

[단점]
- HDFS에 저장된 데이터의 변경이 불가
- 실시간 데이터 분석에는 부적합
- 설치와 설정에 대한 어려움 존재

하둡 구성요소

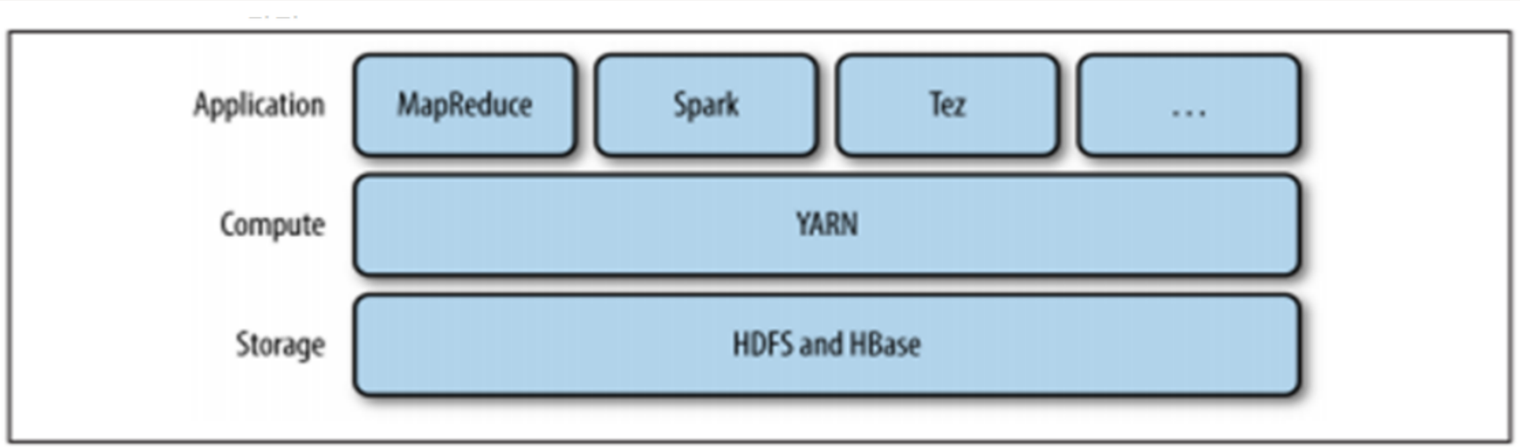


Figure 4-1. YARN applications

- Common: 하둡의 다른 모듈을 지원하기 위한 공통 컴포넌트 모듈

- HDFS: 분산 저장을 처리하기 위한 모듈, 여러 개의 서버를 하나의 서버처럼 묶어서 데이터 저장

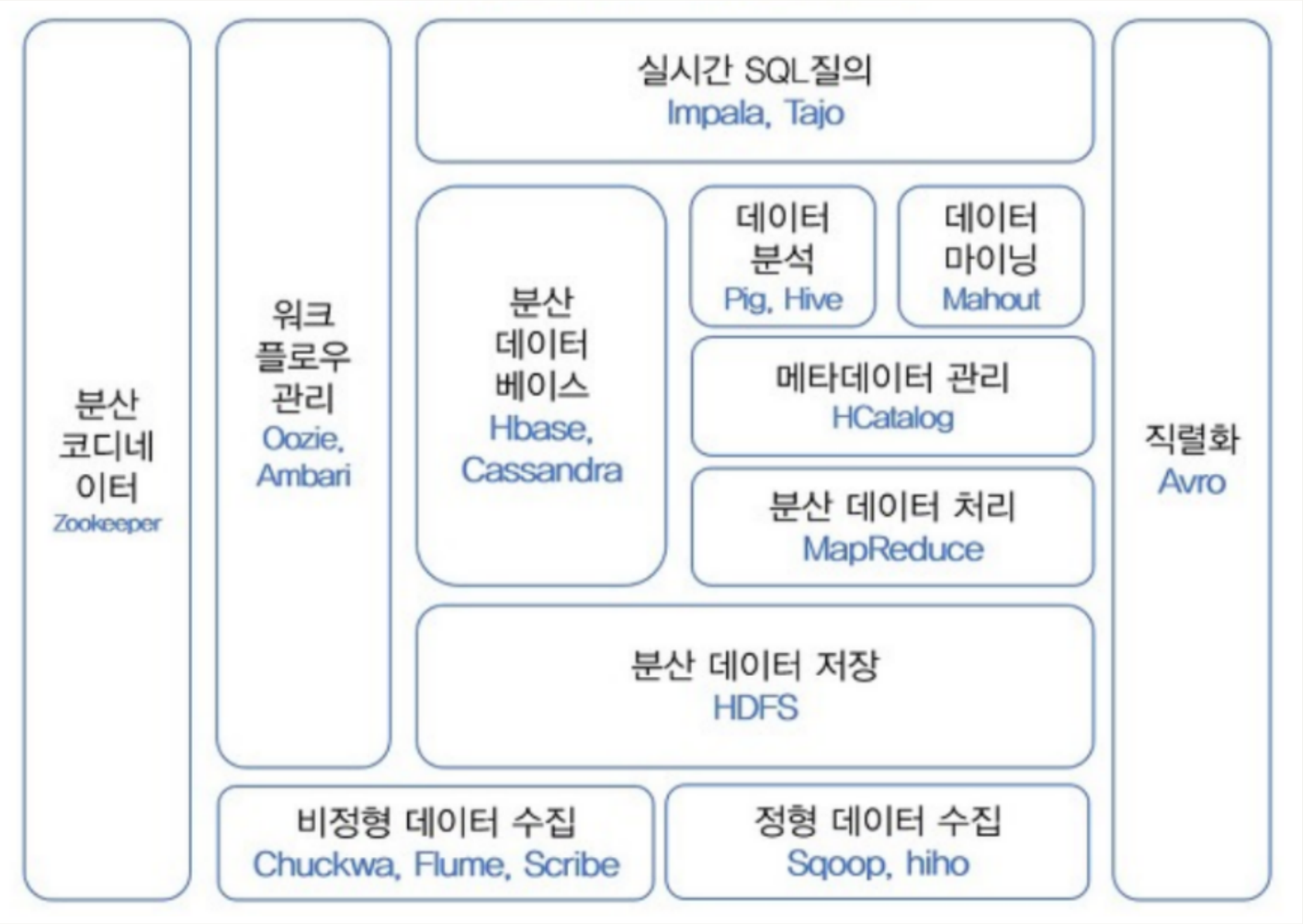
- MapReduce: 분산 저장된 데이터를 병렬 처리할 수 있게 해주는 분산 처리 모듈

- YARN: 병렬 처리를 위한 클러스터 자원 관리 및 스케줄링 담당

하둡 버전별 특징

V1	분산저장(네임노드, 데이터노드), 병렬처리 프레임워크를 정의 작업처리를 슬롯 단위(맵, 리듀스 슬롯)로 처리
V2	YARN 도입으로 병렬 처리 구조 변경 작업처리를 컨테이너 단위로 처리
V3	이레이져 코딩 도입(블록 복제 대체→HDFS 사용량 감소) JAVA8 지원 Ozone추가 고가용성을 위해 2개 이상의 네임노드 지원

하둡 에코시스템



분산 컴퓨팅과 데이터 처리를 위한 기반 시설

비즈니스에 효율적으로 적용할 수 있도록 제공되는 다양한 서브 프로젝트들의 모임

하둡 vs RDBMS

표 1-1 RDBMS와 맵리듀스의 비교

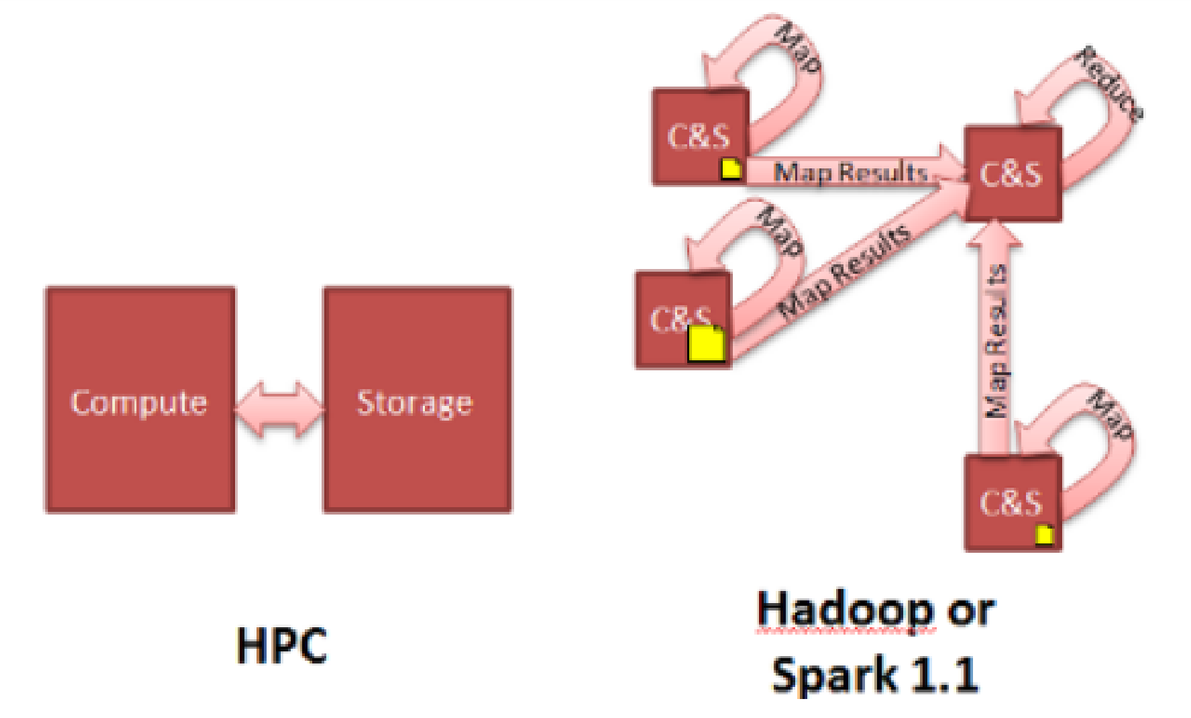
	전통적인 RDBMS	맵리듀스
데이터 크기	기가바이트	페타바이트
접근 방식	대화형과 일괄 처리 방식	일괄 처리 방식
변경	여러 번 읽고 쓰기	한 번 쓰고 여러 번 읽기
트랜잭션	ACID	없음
구조	쓰기 기준 스키마	읽기 기준 스키마
무결성	높음	낮음
확장성	비선형	선형

RDBMS
: 관계형 데이터베이스 관리 시스템

활용 측면에서
- RDBMS: 특정 쿼리와 데이터 변경에 적합
- MapReduce: 전체 데이터셋을 분석할 필요가 있는 문제에 적합

데이터 구조 측면에서
- RDBMS: 정형 데이터 처리에 초점
- MapReduce: 비정형/반정형 데이터에서도 잘 처리 가능

하둡 vs 그리드 컴퓨팅



[그리드 컴퓨팅]
: 모든 컴퓨팅 기기를 하나의 초고속 네트워크로 연결
- 메시지 전달 인터페이스(MPI)와 같은 API를 이용하여 대규모 데이터 처리
- SAN으로 연결된 공유 파일시스템에 접근하는 클러스터 머신 여러 대에 작업을 분산
- 계산 중심의 작업에서는 좋은 결과
- 계산 노드들이 대용량 데이터에 접근해야 할 때 병목 현상 발생, 놀고 있는 계산 노드 발생

하둡
- 데이터 지역성(Data Locality): 계산 노드에 데이터를 함께 배치
- 데이터가 로컬에 있어 접근이 빠름

HDFS 특징

Hadoop Distributed File System

배치처리를 위해 설계된 범용 하드웨어에서 동작하고,
장애 복구성을 가지는 분산 파일 시스템

1. 블록 단위 저장
- 큰 데이터를 나누어 저장
→ 단일 디스크보다 큰 파일도 저장 가능
- 블록사이즈보다 작은 파일은 기존 파일의 사이즈로 저장

2. 블록을 복제하여 중복 저장
- 블록에 문제가 생겨 데이터가 손실되는 경우 방지
- 블록의 기본 복제 단위는 3
- 같은 랙(Rack)의 서버와 다른 랙의 서버로 복제되어 저장

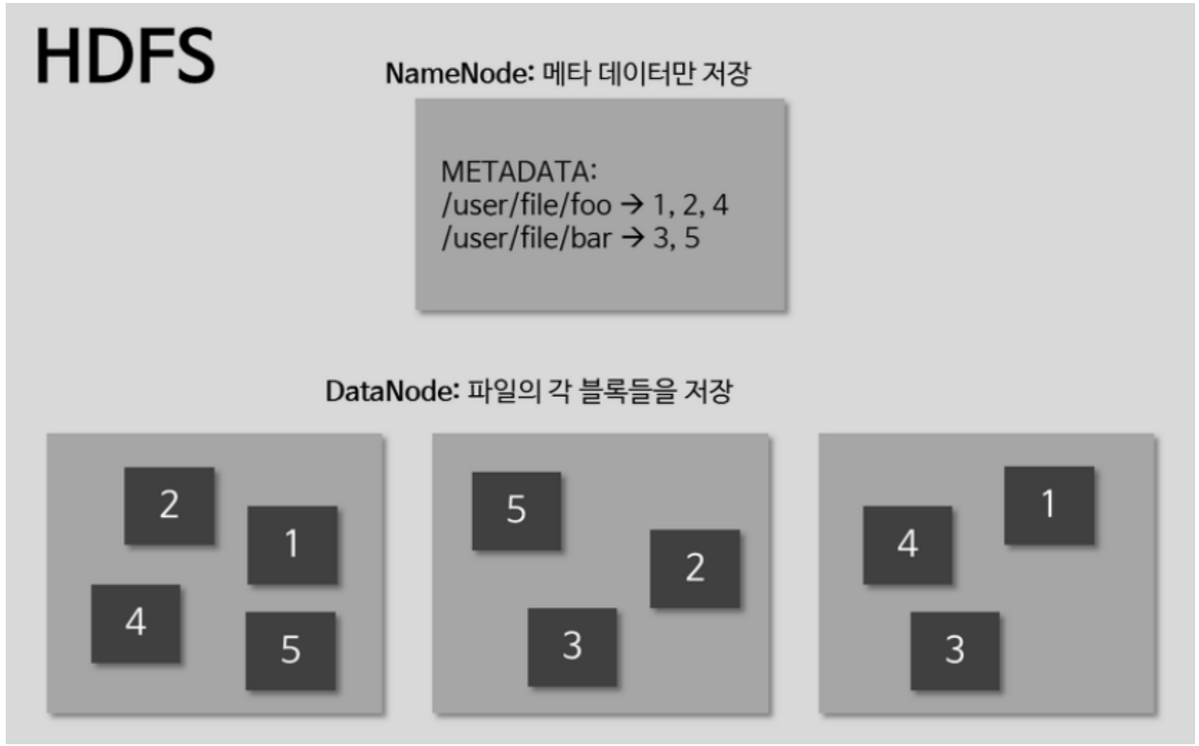
3. 읽기 중심
- 한 번 쓰면 여러번 읽는 것을 목적으로 한다.
- 파일의 수정 지원 X

4. 데이터 지역성
데이터의 위치에서 알고리즘을 처리
→ 데이터를 이동시키는 비용 감소

5. 마스터-슬레이브 구조

BOAZ

HDFS 구성



[NN (Name Node)]
- 파일시스템의 네임스페이스를 관리
- 파일시스템 트리와 그 트리에 포함된 모든 파일과 디렉터리에 대한 메타데이터를 유지
- namespace image + edit log 로 로컬 디스크에 영속적으로 저장
- 파일을 구성하는 블록들의 목록과 위치 정보 저장
- 시스템이 시작할 때 모든 데이터노드로부터 받아서 재구성 → 디스크에 영속적으로 저장하지는 않음
- 파일을 읽거나 쓰는 작업의 시작점 역할

[DN (Data Node)]
- 파일시스템의 실질적인 일꾼
- 클라이언트나 네임노드의 요청이 있을 때 블록을 저장하고 탐색
- 저장하고 있는 블록의 목록을 주기적으로 네임노드에 보고

BOAZ

HDFS 네임노드와 데이터 노드 간 통신 방식

Handshake

데이터 노드를 네임 노드에 등록할 때 수행하는 통신

적절한 SW 버전 확인 → 적합한 데이터 노드다 → 데이터 노드를 식별할 수 있는 고유한 id 부여

block report

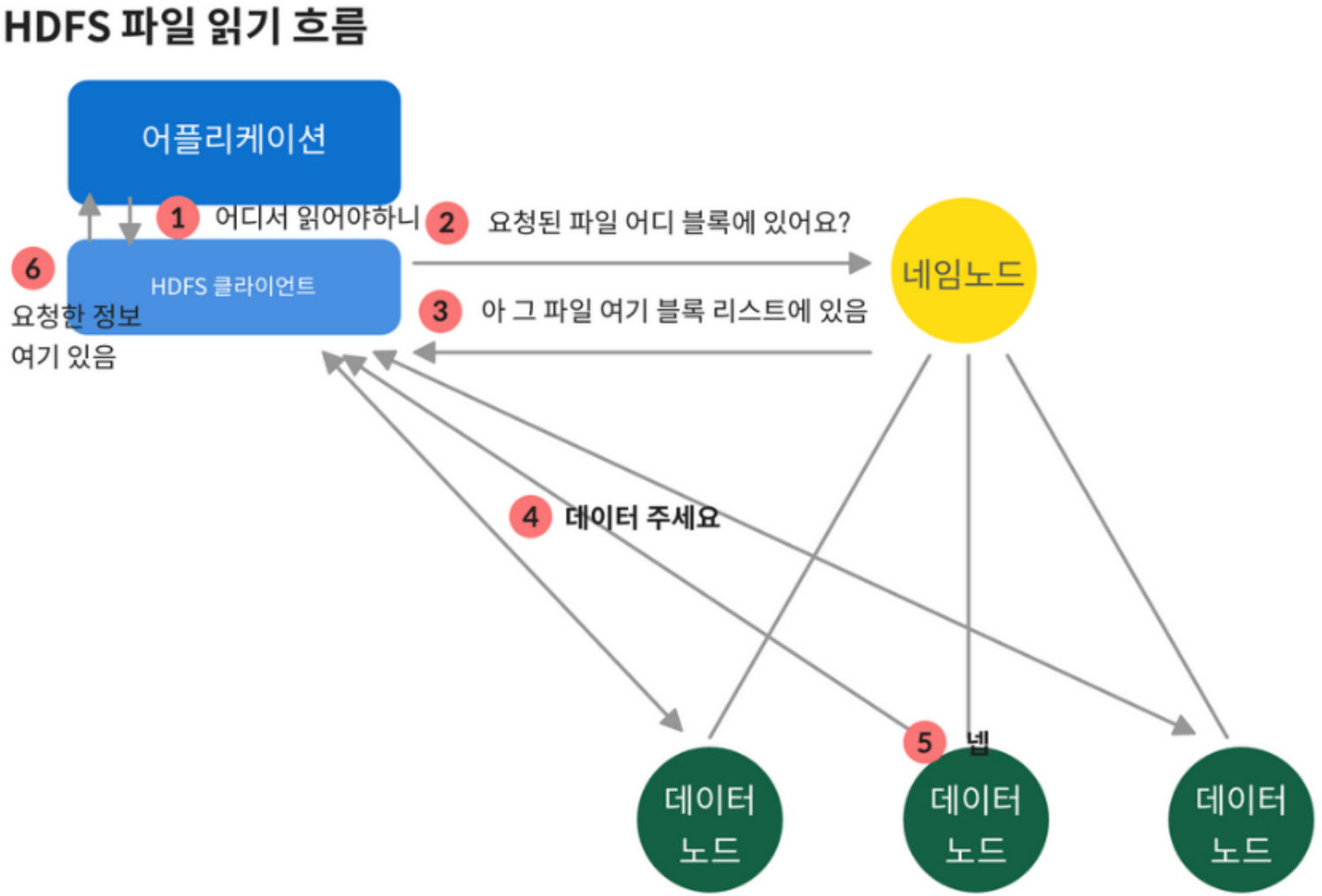
블록이 저장된 위치를 실제로 알 수 있게 해줌
각 데이터 노드들은 매 시간마다 네임노드에 저장되어 있는 위치와 현황을 네임 노드에게 전달

Heartbeat

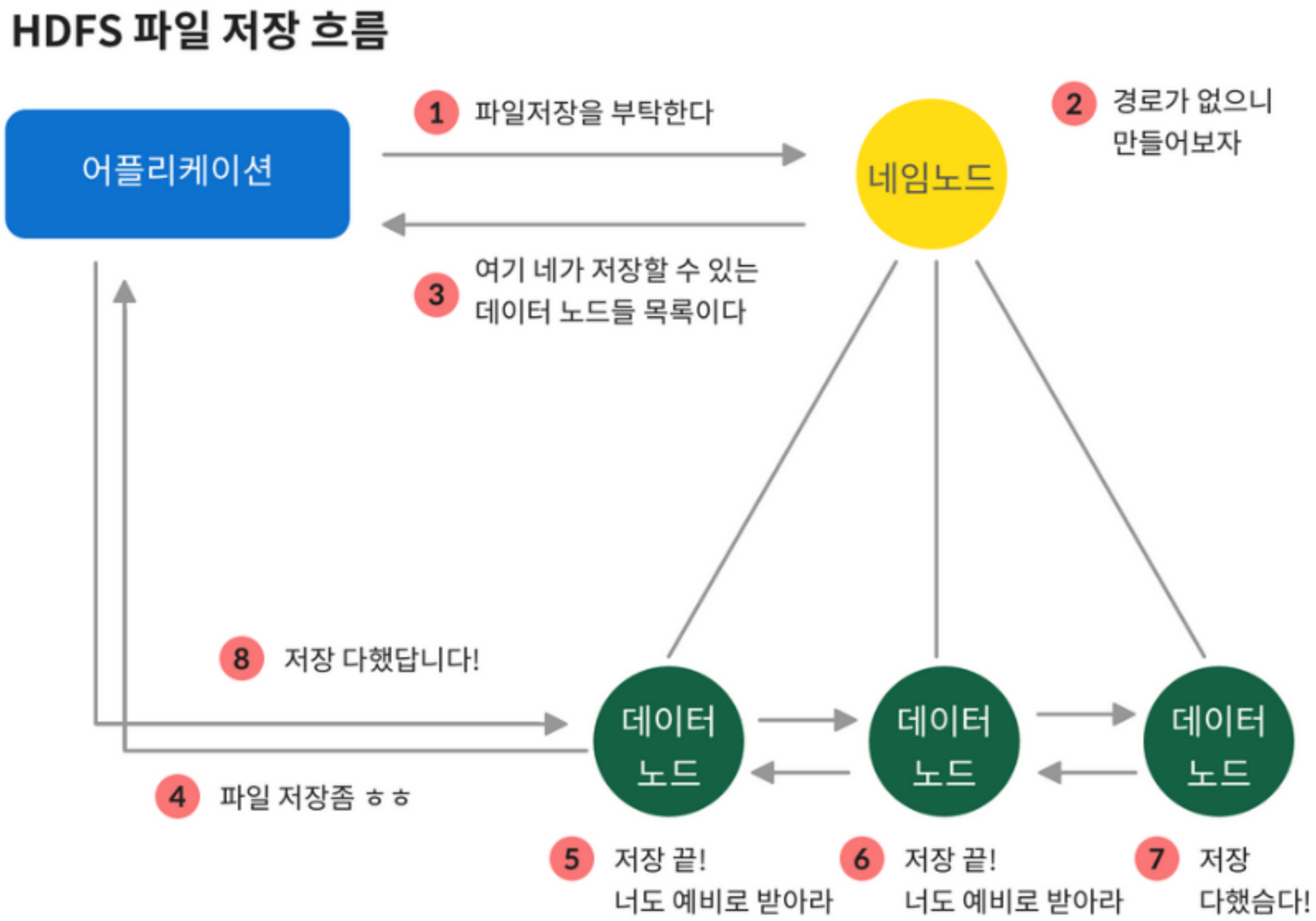
각 데이터 노드는 3초에 한 번씩 heartbeat를 네임 노드에 보내 동작하고 있음을 확인시켜 줌

BOAZ

HDFS 파일 읽기 흐름


HDFS 파일 읽기 흐름
어플리케이션
1 어디서 읽어야하니
2 요청된 파일 어디 블록에 있어요?
네임노드
HDFS 클라이언트
6
요청한 정보
여기 있음
3 아 그 파일 여기 블록 리스트에 있음
4 데이터 주세요
5 넵
데이터 노드
데이터 노드
데이터 노드

HDFS 파일 저장 흐름


HDFS 파일 저장 흐름
어플리케이션
1 파일저장을 부탁한다
네임노드
2 경로가 없으니
만들어보자
3 여기 네가 저장할 수 있는
데이터 노드들 목록이다
8 저장 다했답니다!
데이터
노드
데이터
노드
데이터
노드
4 파일 저장좀 ㅎㅎ
5 저장 끝!
너도 예비로 받아라
6 저장 끝!
너도 예비로 받아라
7 저장
다했슴다!

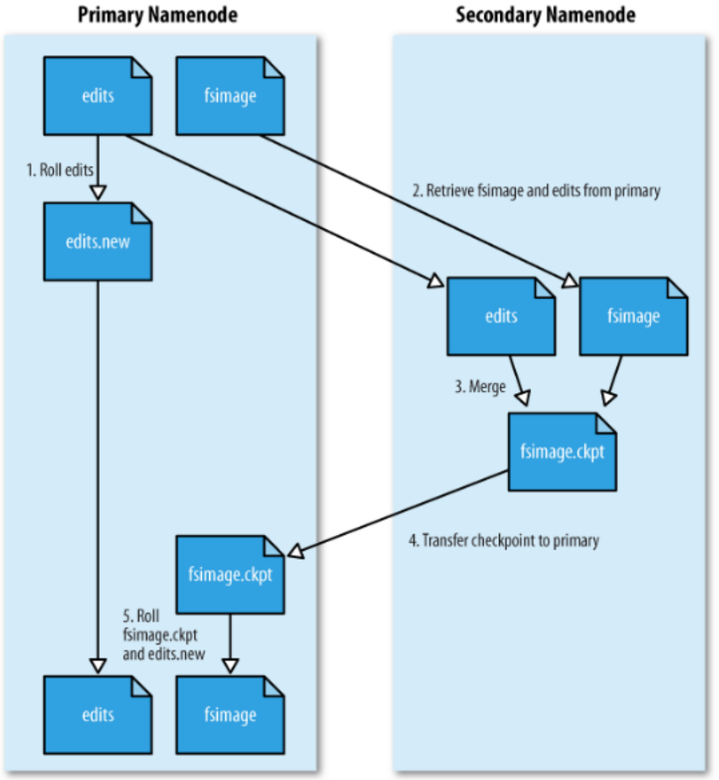
HDFS 네임노드의 장애와 복구 기능

네임노드를 실행하는 머신이 손상 → 파일 시스템의 어떤 파일도 찾을 수 없음

장애 복구 기능

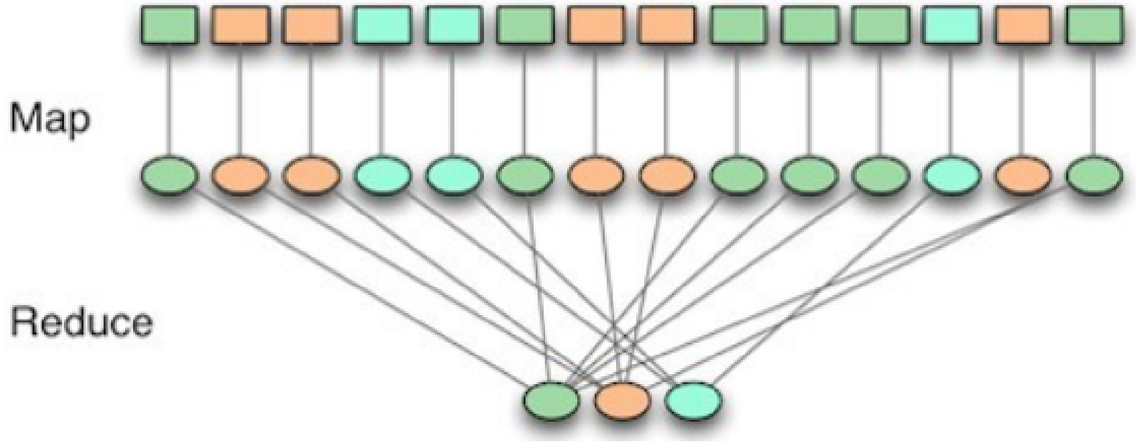
1. 파일을 백업
- 네임노드가 다수의 파일시스템에 영구적인 상태를 저장하도록 하둡을 구성
- 백업 작업은 동기화되고 원자적으로 실행됨
- 권장 방법: 로컬 디스크 + 원격의 NFS 마운트 동시 백업

2. secondary namenode 운영
- edit log: HDFS의 메타데이터에 대한 모든 변화를 기록하는 로그파일
- edit log가 너무 크다면: 메모리에 올라와 있는 파일 시스템 이미지를 갱신하는 단계를 진행할 때 많은 시간 소요
- 주기적으로 namespace image를 edit log와 병합
  → 새로운 namespace image 제작
- 네임스페이스 이미지의 복제본을 보관
  :약간의 시간차를 두고 복제
  :주 네임노드에 장애 발생 시 어느 정도의 데이터 손실은 불가피



MR (MapReduce) & 특징

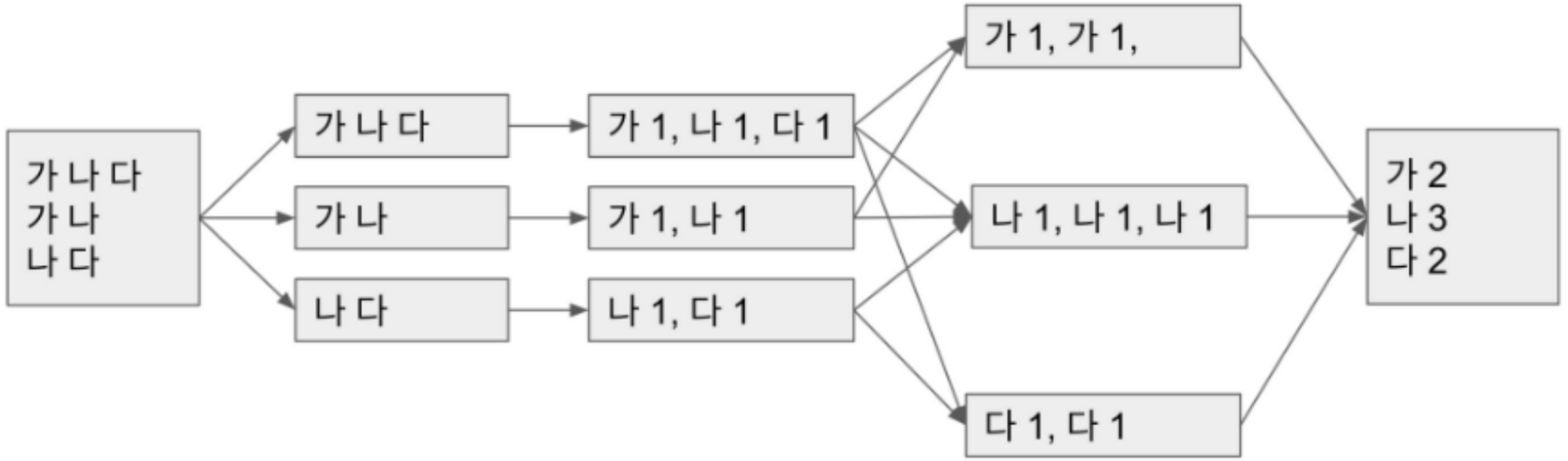
- Hadoop 클러스터의 데이터를 처리하기 위한 병렬 처리 프로그래밍
- 구글에서 대용량 데이터를 분산 처리하기 위해 제작
- 여러 컴퓨터에서 동시에 작업을 처리하여 속도를 높일 수 있다.



- 일괄 처리 시스템
- 각 단계는 입력과 출력으로 key-value 쌍을 가짐
- 데이터의 입출력과 병렬 처리 등 기반 작업을 프레임워크가 알아서 처리
- 타입은 프로그래머가 선택, map 함수와 reduce 함수 또한 프로그래머가 작성

BOAZ

MR (MapReduce)



[Map]
- Map은 분산되어있는 컴퓨터에서 처리하는 것
- 흩어져 있는 데이터를 key(몇 번째 데이터인지), value(값을 추출한 정보)로 데이터를 묶어줌

[Reduce]
- 최종적인 통합관리를 위해 Reduce를 해주는 것
- 만약 A컴퓨터에서 데이터가 5번나오고 B컴퓨터에서 데이터가 10번 나왔다면, Reduce는 데이터가 총 15번 나왔다고 통합을 해주는 것
- Reduce단계는 Map단계의 key를 중심으로 필터링 및 정렬한다. 하둡에서는 이 Map과 Reduce를 함수를 통해서 구현하고 맵리듀스 잡을 통해 제어한다.

MR (MapReduce) 처리단계

1. input
- 데이터 입력
- HDFS로부터 파일을 가져옴

2. split
- 전체 input data를 분할
- 하둡은 큰 데이터가 들어오면
 64MB 단위 블록으로 분할
- 분할된 모든 블록은 같은 Map 작업 수행, 이후 Reduce 작업을 통해 합쳐짐

3. Map
- 데이터를 (key, value) 쌍의 형태로 연관성 있는 데이터 분류를 묶는 작업
- 필요한 분석 대상만 추출
- 잘못된 레코드를 제거

MR (MapReduce) 처리단계

4. Combiner
- Mini-Reducer
- Map 프로그램에서 나온 출력에 대해 Reduce로 보내기 위한 shuffle/sort가 발생하기 전에 reducer를 적용하는 것
- reducer로 가는 데이터의 크기를 줄이는 것
- 필수 단계는 아님
- 성능 개선 가능

5. Partition
- 키를 기준으로 디스크에 분할 저장하는 것
- Hash partitioning이 기본
- 각 파티션은 키를 기준으로 정렬됨
- mapper의 local file system에 저장됨
- 분할된 각 파일은 각각 다른 reduce task에 저장됨

MR (MapReduce) 처리단계

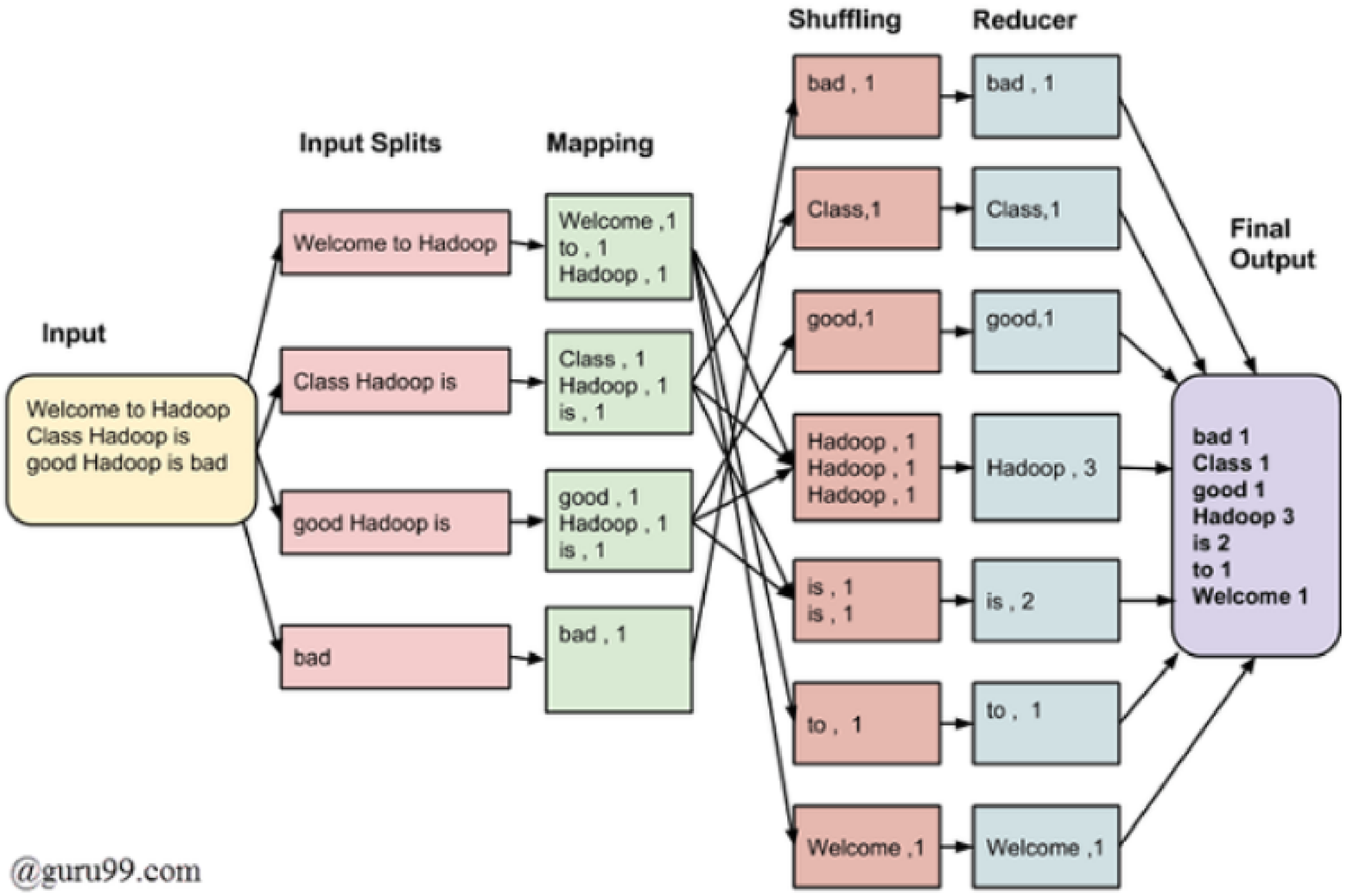
6. Shuffle
- Reducer로 데이터를 이동시키는 것
- Key 값을 기준으로 리스트 형태로 조합
- Key와 연관되어 있는 모든 값은 reducer로 보내짐

7. Sort
- Reducer로 전달된 데이터를 key값을 기준으로 정렬

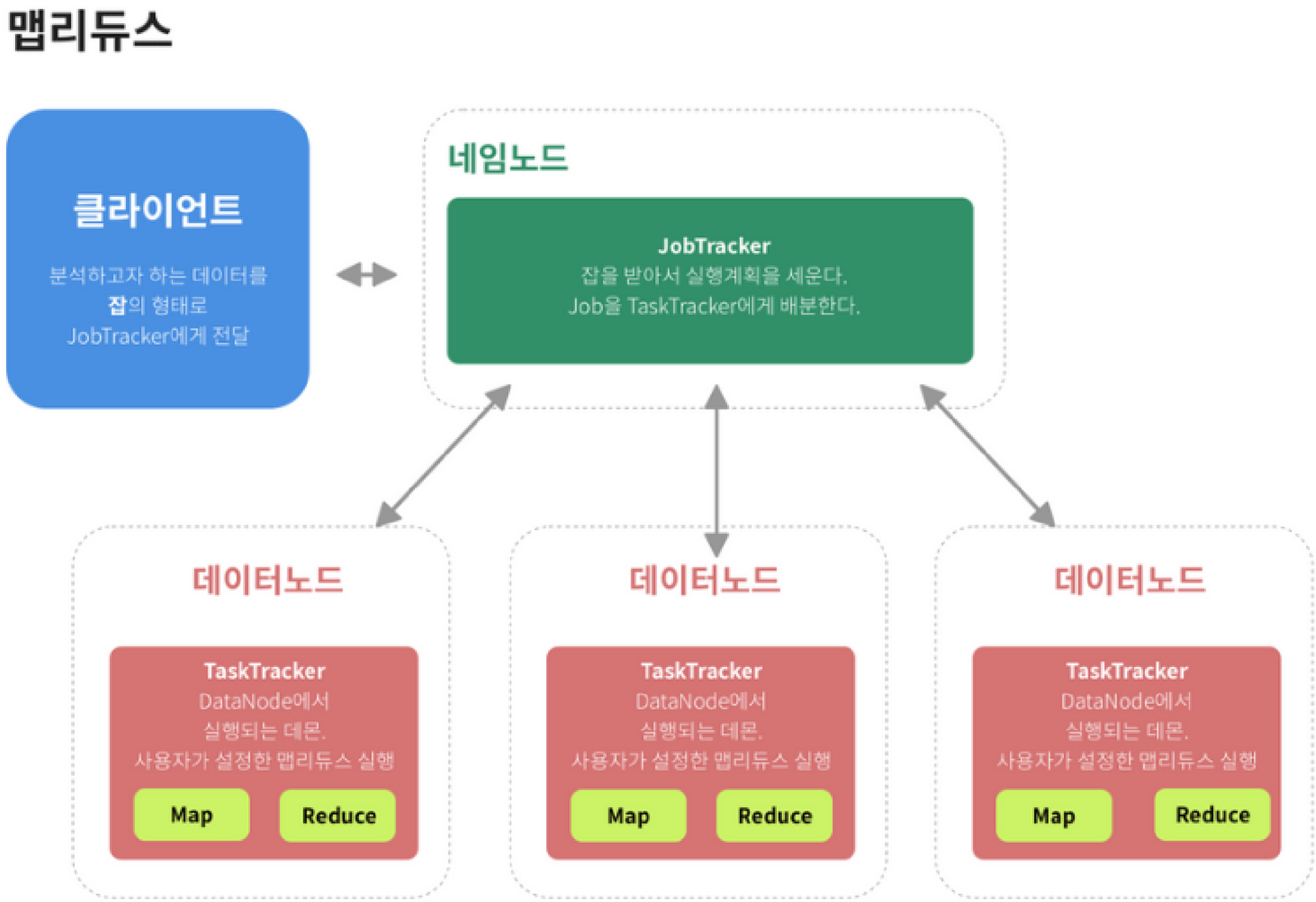
8. Reduce
- Map 단계에서 넘어온 데이터에서 중복 데이터를 제거하고 원하는 데이터를 추출하는 작업
- Map 작업을 수행한 각각의 블록의 결과를 합치는 작업

9. Output
- Reducer의 결과를 정의된 형태로 저장

MR (MapReduce) 처리단계 예시



MR (MapReduce) 구성



잡(job): 클라이언트가 하둡으로 실행을 요청하는
MapReduce 프로그램의 작업 단위

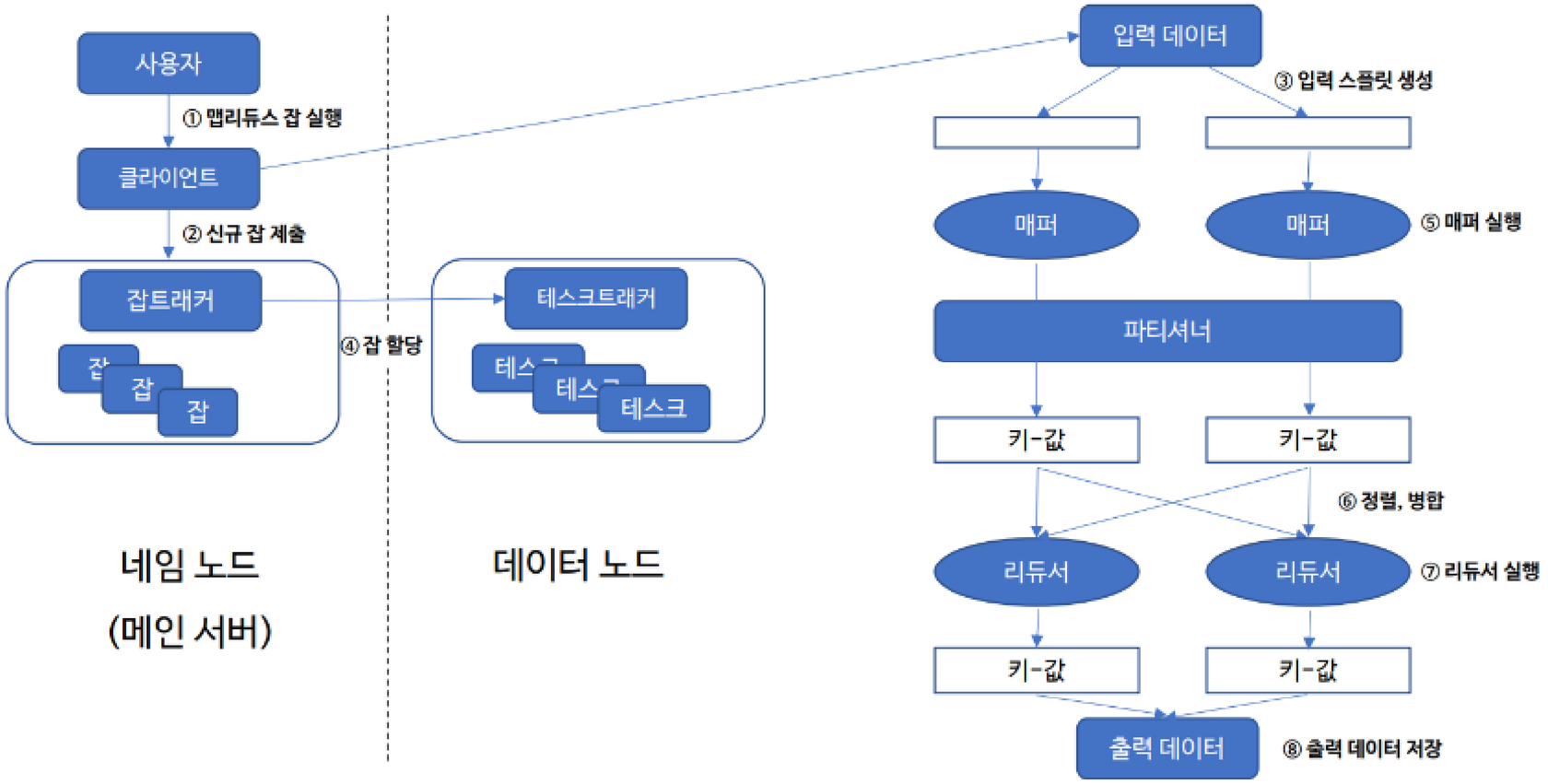
MR (MapReduce) 구성

클라이언트
- 사용자가 실행한 맵리듀스 프로그램
- 혹은 하둡에서 제공하는 맵리듀스 API
- 구현된 MapReduce 잡을 제출하는 실행주체
- 분석하고자 하는 데이터를 잡의 형태로 잡 트래커에게 전송

잡 트래커(Job Tracker)
- 하둡 클러스터에 등록된 전체 잡의 스케줄링 관리 및 모니터링
- 전체 하둡 클러스터에서 하나의 잡트래커 실행
- 클러스터의 다른 노드들에게 map 및 reduce task를 할당

태스크 트래커(Task Tracker)
- 사용자가 설정한 MapReduce 프로그램 실행
- 하둡의 데이터노드에서 실행되는 데몬
- 잡 트래커의 작업 요청을 받고, 잡 트래커가 요청한 map과 reduce의 개수만큼 맵 태스크와 리듀스 태스크 생성
- 진행 상황을 잡 트래커에게 보고

MR (MapReduce) 실행 과정



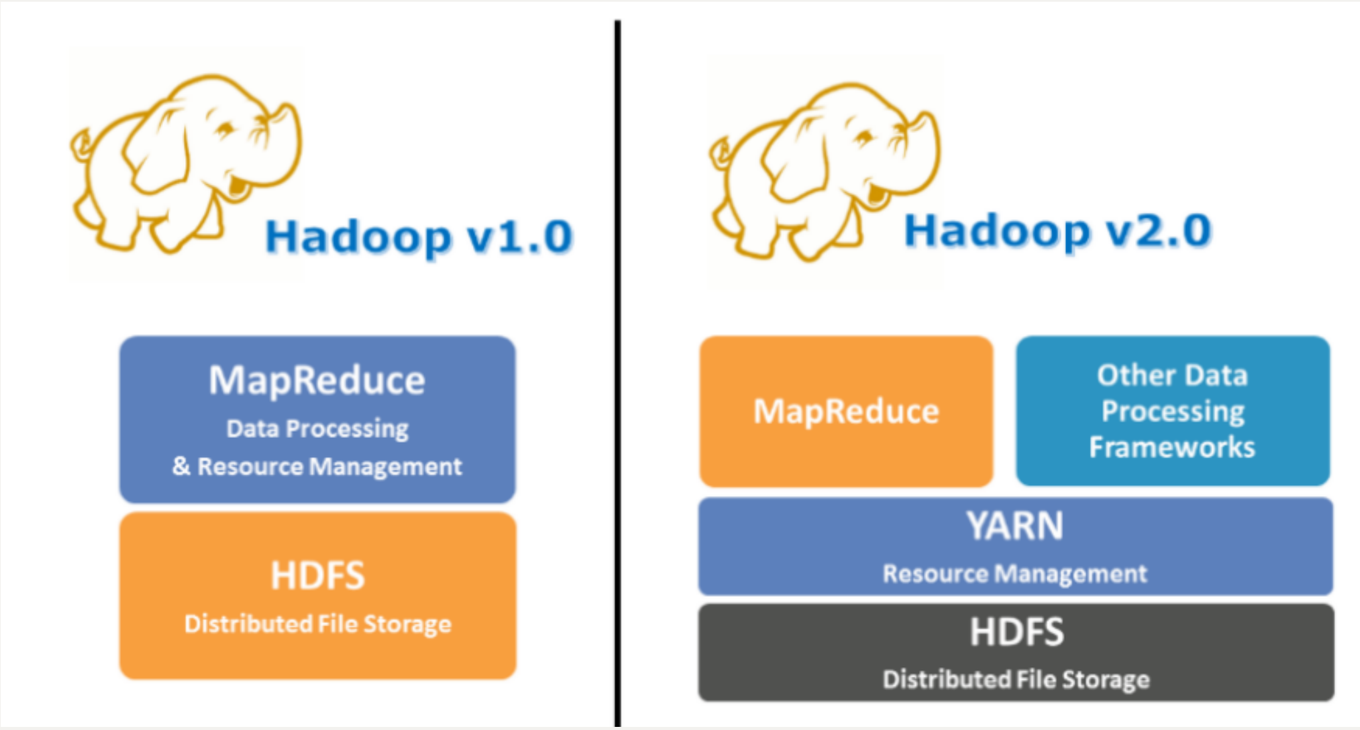
MR (MapReduce) 장단점

장점
- 단순하고 사용이 편리
- 유연성: 특정 데이터 모델에 의존하지 않아 비정형 데이터 모델 지원 가능
- HDFS 이외에도 다양한 저장구조 지원
- 데이터 복제 기반의 데이터 내구성 지원
- 높은 확장성

단점
- 동적 스키마 → RDBMS보다 불편
- 낮은 성능
- 단순한 데이터 처리
- 개발환경의 불편, 운영 노하우 부족

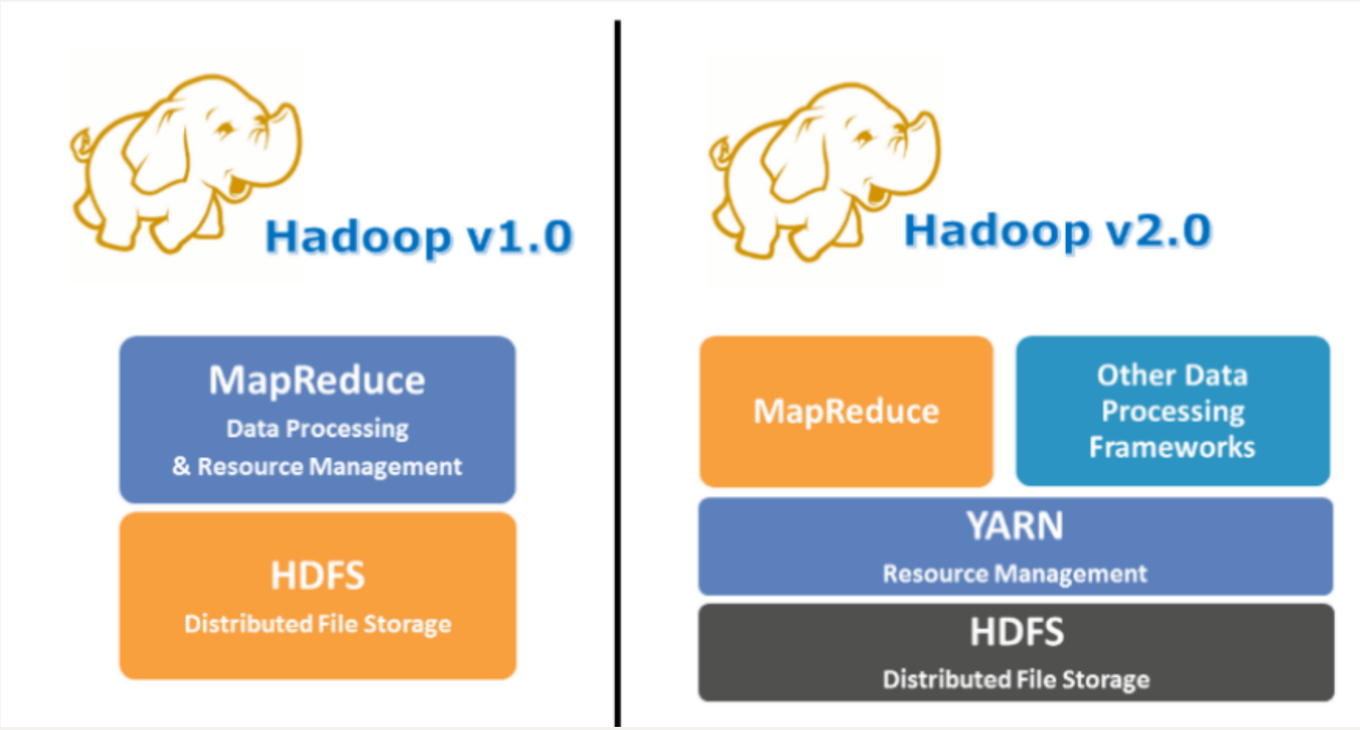
YARN

- Yet Another Resource Negotiator
- 클러스터 자원 관리 시스템
- Hadoop 2에서 처음 도입
- 맵리듀스 등 분산 컴퓨팅 도구 지원
- 클러스터의 자원을 요청하고 사용위한 API 제공

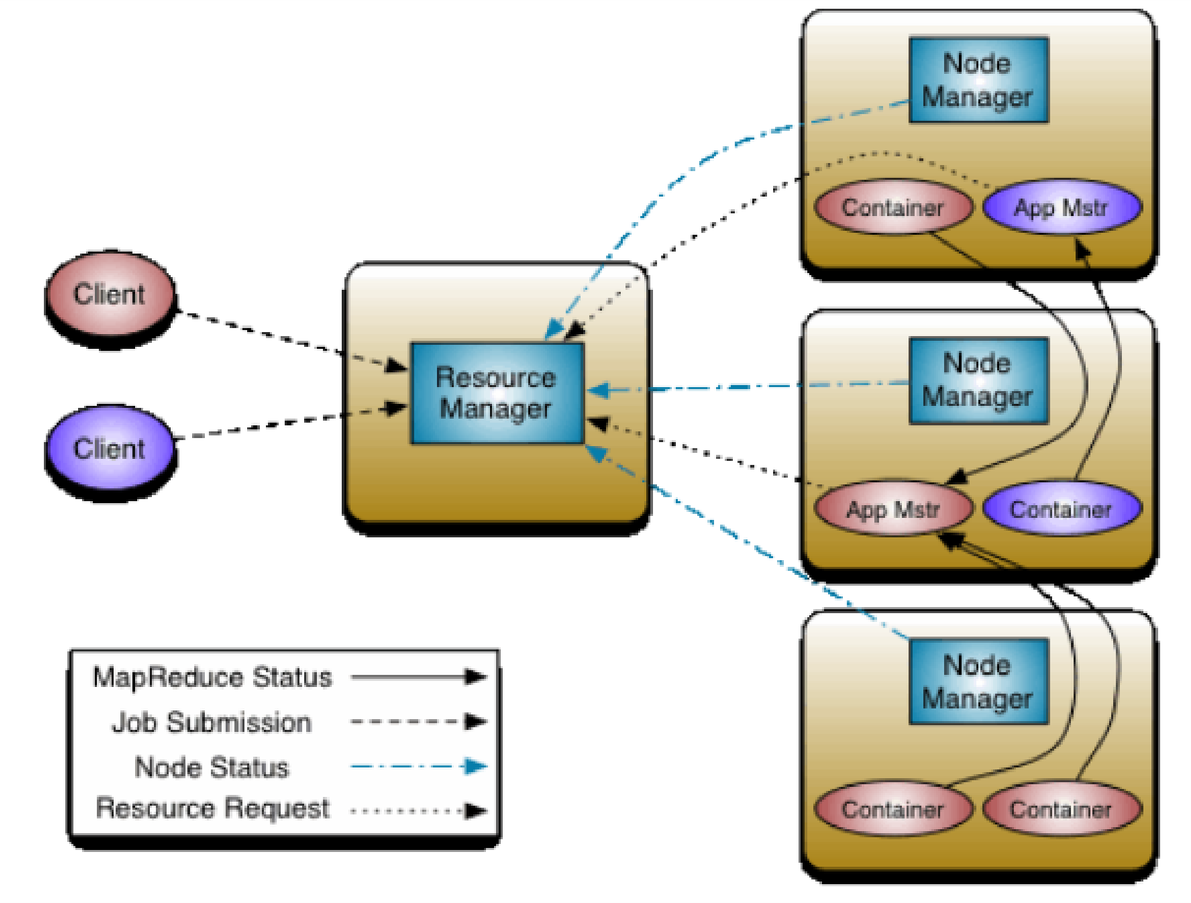


YARN

- Yet Another Resource Negotiator
- 클러스터 자원 관리 시스템
- Hadoop 2에서 처음 도입
- 맵리듀스 등 분산 컴퓨팅 도구 지원
- 클러스터의 자원을 요청하고 사용위한 API 제공



YARN 구성

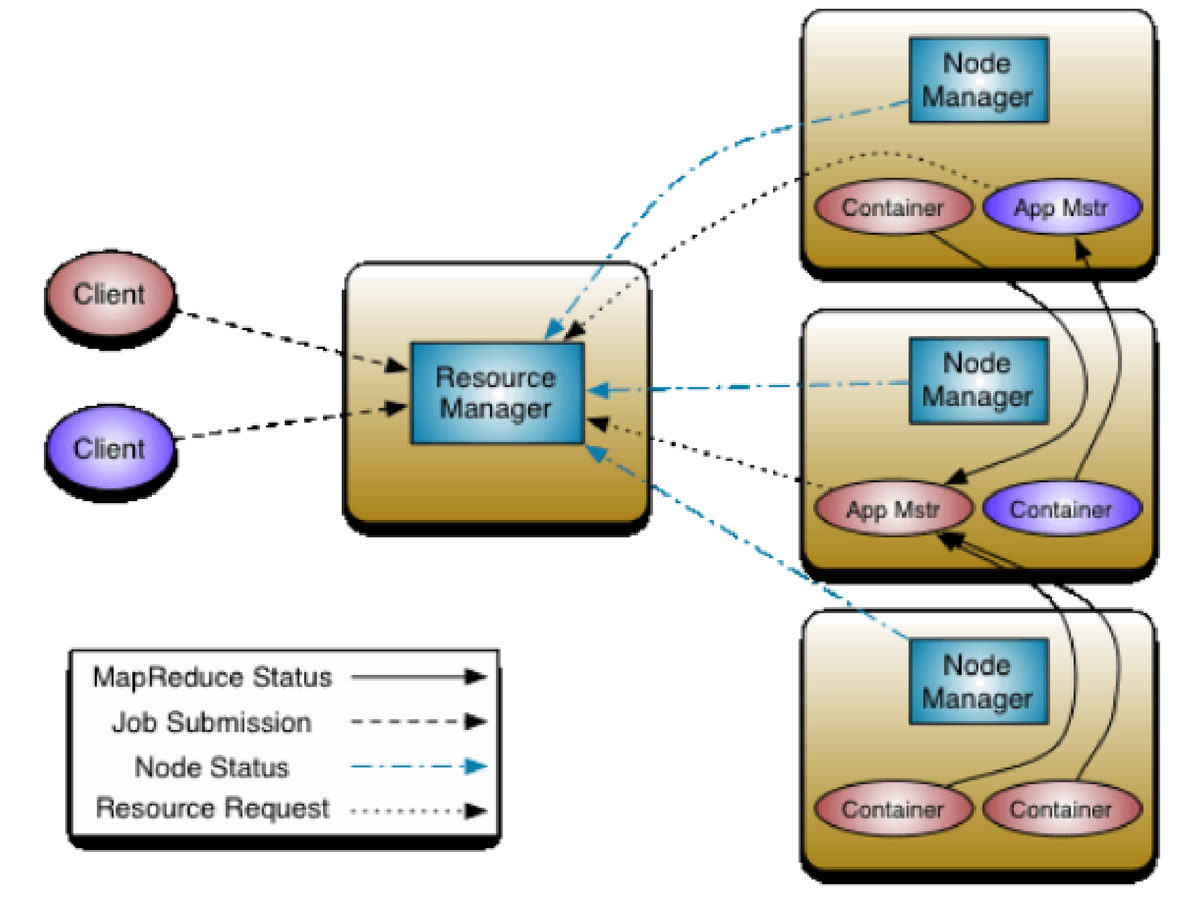


[RM (Resource Manager)]
- 클러스터에서 유일
- 클러스터 전체 자원의 사용량 관리
- 애플리케이션 마스터와 함께 동작
- YARN 클러스터의 리소스를 사용하고자 하는 다른 플랫폼으로부터 요청을 받아 리소스 할당 (스케줄링)
  - 크게 Scheduler 와 Applications Manager로 구성

  [Scheduler]
  - 실행되고 있는 애플리케이션들에 리소스 할당
  - 애플리케이션 실행에 관여하지 않음

  [Applications Manager]
  - 잡의 제출을 받으면 애플리케이션 마스터 실행
  - 애플리케이션 실행에 관여

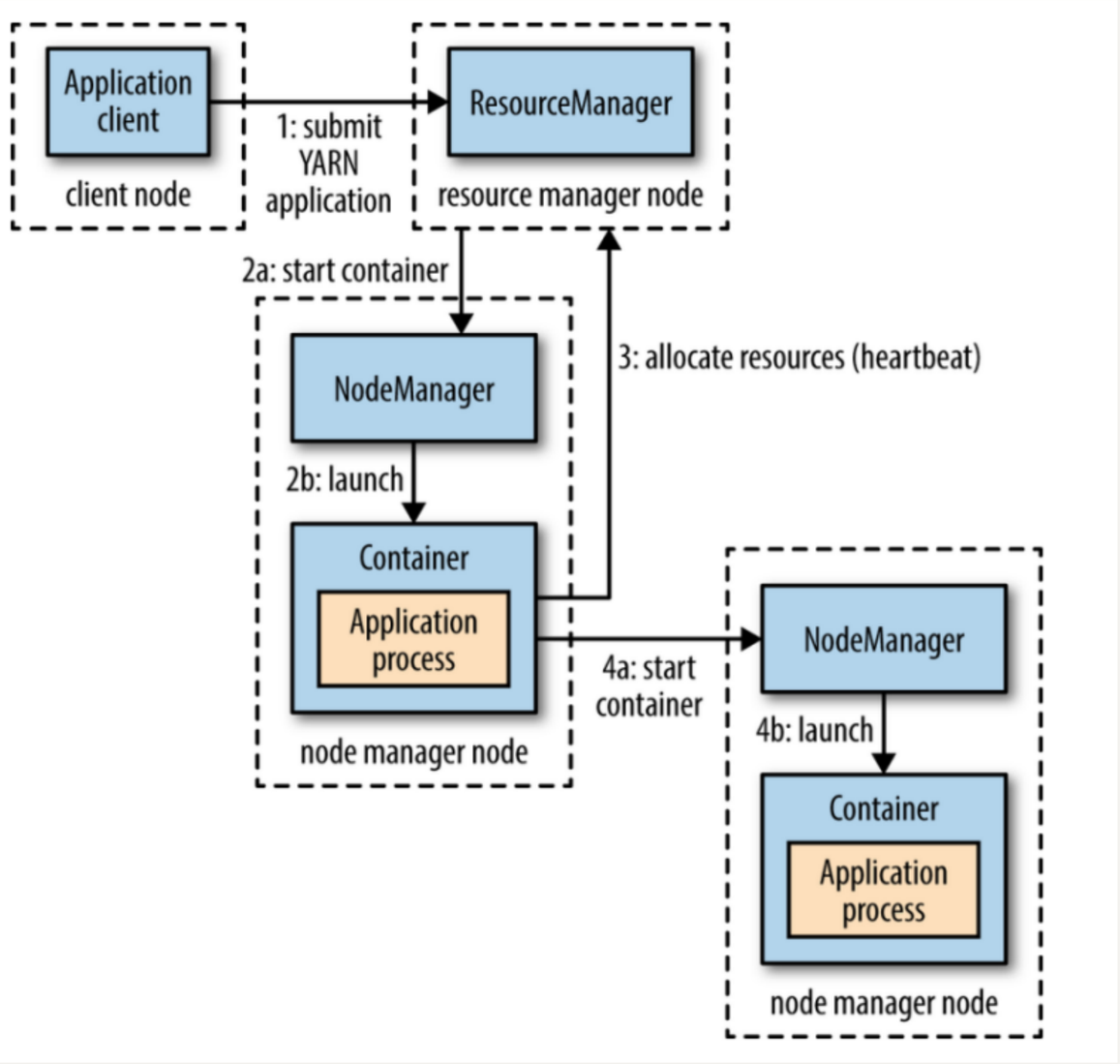
YARN 구성



[NM (Node Manager)]
- 클러스터 각 노드마다 실행
- YARN 클러스터의 Worker 서버
- 머신마다 존재하는 프로세스, 머신 컨테이너들의 리소스 사용량을 스케줄러에 reporting
- 컨테이너를 구동하고 모니터링하는 역할
  :컨테이너 장애 상황 또는 컨테이너가 요청한 리소스보다 많이 사용하고 있는지 감시 → 요청한 리소스보다 많이 사용하면 해당 컨테이너를 kill

[AM(Application Manager)]
- 애플리케이션의 실행 및 관리를 담당
- 애플리케이션마다 존재
- RM과의 통신을 통해 리소스를 할당 받음
- NM과의 통신을 통해 애플리케이션 내의 태스크 감독

YARN 수행과정



1. 클라이언트가 리소스 매니저에 접속하여 애플리케이션 마스터 프로세스 구동 요청

2. 리소스 매니저가 컨테이너에서 애플리케이션 마스터를 시작할 수 있는 노드 매니저를 하나 찾음

3. 애플리케이션 마스터는 리소스 매니저에 자원 요청

4. 각 노드에 컨테이너를 실행하고 작업 진행

5. 작업이 종료되면 역순으로 알리고 자원 해제

YARN 장점

확장성(Scalability)

수용가능한 단일 클러스터 규모 확대, 애플리케이션 마스터와 리소스 매니저 분리로 한계 극복

가용성(Availability)

리소스 매니저와 애플리케이션 마스터에 고가용성 제공

효율성(Utilization)

노드 매니저가 리소스 풀을 관리, 유동적인 자원할당 가능

다중 사용성 (Multitenancy)

다양한 애플리케이션 수용, 다른 버전의 맵리듀스 동시 실행 가능

BOAZ

Reference

https://han-py.tistory.com/361
(하둡 개요)
https://wikidocs.net/22654
(하둡)
http://www.incodom.kr/%ED%95%98%EB%91%A1
(하둡)
https://yes90.tistory.com/62
([BigData]하둡의 분산파일처리시스템 HDFS)
https://charsyam.wordpress.com/2011/04/06/hadoop-%EC%97%90%EC%84%9C-secondary-namenode-%EC%9D%98-%EC%97%AD%ED%99%9C/
(Hadoop 에서 Secondary Namenode 의 역활)
https://juneyr.dev/2018-10-31/hadoop
(하둡 이란)
https://mangkyu.tistory.com/127
([Hadoop] YARN의 구조와 동작 방식)
https://magnificent-tarragon-491.notion.site/Hadoop-eeaba8c2b3544a48bc7762616c976a54
(박지윤님의 하둡 이론 정리)