- 출처 : 빅데이터 기사

데이터의 정의

정량적 데이터(정형 데이터)

- Structed Data
- 수치로 표현할 수 있는 숫자, 도형, 기호 등의 데이터를 의미
- 저장, 검색, 분석 활용에 용이
- 온도, 습도, 풍향, 기압 등은 정량적 데이터

정성적 데이터(비정형 데이터)

- Unstructed Data
- 언어, 문자 등의 정형화되지 않은 데이터를 의미
- '오늘 무릎이 무척 시리니 비가 올 거 같다'라는 예측을 한다면, 여기서 무릎이 시린 정도는 수치로 정형화할 수 없는 정성적 데이터
- 영화 감상평, 실시간 SNS 검색어 등의 정성적 데이터 수집과 분석에는 상대적으로 많은 비용과 기술적 투자가 필요
- 이미지, 텍스트 데이터, 동영상이 대표적인 비정형 데이터의 종류

반정형 데이터

- Semi-structred Data
- 일반적인 규칙화된 형식을 갖지 않는 웹문서, 신문 등과 같은 데이터
- 데이터 내부에 정형 데이터의 스키마에 해당되는 메타 데이터를 가지고 있으며, 보통 파일 형태로 저장
- 데이터 내부의 규칙성을 파악해 데이터를 파싱할 수 있는 파싱 규칙을 적용
- 데이터의 구조는 일관성이 없으므로 테이블의 형식에도 샘플들의 속성 순서가 모두 다를 수 있음
    - HTML, XML, JSON

전통적인 데이터베이스 시스템

- 지난 수년 동안 기업들 스스로가 데이터를 조직화하기 위한 표준 방식을 고수하기 때문에 온라인 트랜잭션 데이터베이스, 데이터 마트, 데이터 웨어하우스에 의존해 구조화된 데이터라고 불리는 비즈니스 데이터를 저장하고 분석해 옴

- 이런 전통적인 데이터 저장소들은 여전히 엄청나게 유용하고 빅데이터가 만연한 오늘날에도 자신들만의 기능을 하고 있음
- 그러나 지난 15여년간에 걸친 일부 변화들은 새로운 데이터 저장 방법과 프로세싱 프레임웍크의 필요성을 느끼게 되었음

데이터의 변화

- 비정형 데이터 또는 일부만 정형화된 데이터 같은 새로운 데이터들(트위터, 서버 로그, 비디오, 사진, 음성)은 데이터를 저장하고 데이터에 포함돼 있는 정보들을 효과적으로 이용하기 위한 새로운 방식이 필요
- 웹 사이트나 다른 자료들로부터 유입되는 막대한 양의 정보가 점차 기존의 DW 방식으로 저장하고 처리할 수 있는 한계를 벗어남
- 새로운 컴퓨팅 패러다임은 NoSQL DB를 사용해 큰 데이터 세트를 분석하게 됨
- 기업 시스템에서 유입되는 대량의 데이터를 저장하고 분석하는 데 따른 비용이 증가함
- 상당한 크기의 저장 공간, 처리 능력, 분석에 대한 필요성이 증가
- 전통적인 DB가 갖고 있는 큰 문제는 스케일 업 아키텍처를 사용한다는 점
- 늘어나는 데이터를 저장하기 위해서는 CPU, RAM등 하드웨어를 업그레이드 해야 함
- 업그레이드로 끝나는게 아니라 CPU 및 I/O 병목 현상 문제도 해결해야 함
- 하둡의 HDFS 파일 시스템은 데이터 사이즈가 커질수록 서버를 계속 추가시켜 전체적인 저장 능력을 증가시키는 방식인 스케일 아웃(Scale Out) 방식을 사용

하둡

- 하둡은 방대한 양의 데이터를 처리하는 플랫폼
- 2005년에 오픈소스 프로젝트로 소개
- 하둡이 소개된 이래, 매우 효율적이면서 신뢰할 수 있는 컴퓨터 아키텍처의 기반이 되는 병렬 처리 알고리즘과 간단한 데이터 프로세싱 모델을 이용해 많은 양의 데이터를 처리하는 빅데이터 처리 시스템의 표준

역사

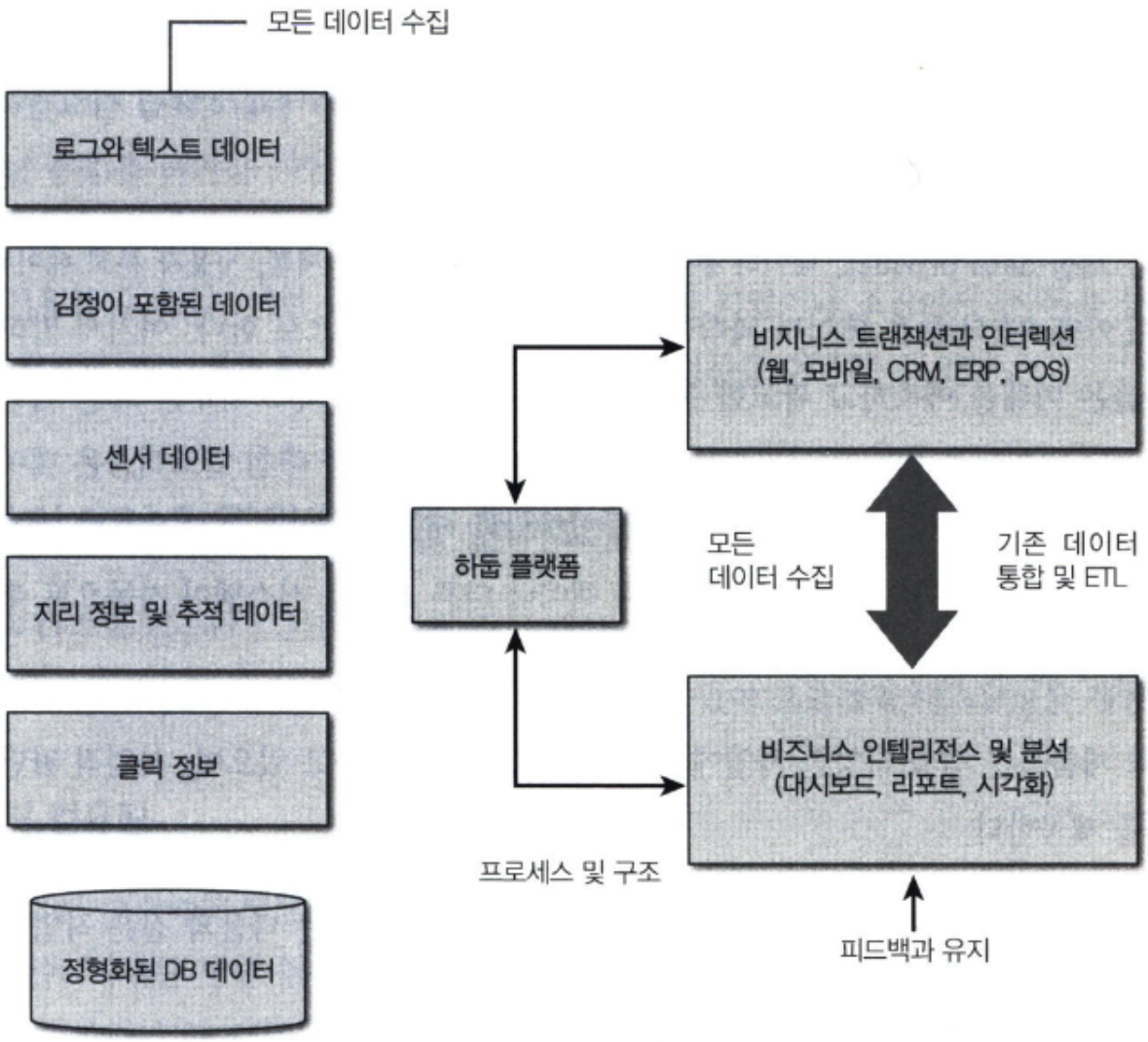
- 텍스트 검색 라이브러리로 널리 활용되는 아파치 루씬의 창시자인 더그 커팅이 개발
- 하둡은 루씬 프로젝트의 일환으로 개발된 오픈 소스 웹 검색 엔진인 아파치 너치의 하부 프로젝트로 시작
- 2003년 구글은 구글 파일 시스템(GFS)을 발표하였고 이것이 하둡의 분산 저장 개념의 토대가 됨
- GFS는 하둡의 분산 저장인 MapReduce라는 처리 기술을 개발하는데 많은 영감을 줌
- 2006년 더그 커팅과 톰 화이트를 중심으로 팀을 구성해 하둡을 개발하기 시작
- 하둡의 이름은 더그 커팅의 아들이 갖고 놀던 장난감 봉제 인형 코끼리의 이름에서 가져옴

특징

- 대용량 데이터 처리 능력
- 장애 허용(Fault tolerance)
- 높은 장애 대응력(fault reliant)
- 데이터의 스트리밍 액세스
- 간단한 데이터 일관성 모델

데이터 레이크

- 데이터의 출처나 데이터를 활용하는 분석 프레임웍크에 관계없이 조직의 모든 데이터를 중심부에서 저장하도록 하는 것을 의미
- 데이터 레이크는 조직의 데이터 처리와 분석 그리드로 사용될 수 있음
- 데이터 레이크는 기업에 들어오는 모든 데이터의 종착지로 활용될 수 있음
- 기업들은 데이터 레이크를 통해서 전통적인 데이터 처리 방식으로 얻을 수 없는 통찰력을 얻을 수 있음



[그림 1-1] 하둡 기반 데이터 호수 아키텍처. 다양한 소스에서 들어오는 데이터가 하둡으로 흘러들어가 처리되고, 처리된 데이터가 ETL과 BI 컴포넌트로 들어가는 구조

- 출처 : 빅데이터 하둡 가이드

하둡의 역할

- 전통적인 정형화 데이터를 저장하고 처리할 뿐만 아니라 일부만 정형화되거나 완전 비정형 데이터도 저장하고 처리
- 사실상 무제한으로 데이터를 저장할 수 있음
- SQL과 NoSQL 프로세싱 프레임워크도 사용할 수 있음
- 병렬 처리 프로세싱 전략을 사용해 방대한 데이터세트를 효과적으로 처리
- 하둡 환경의 거의 모든 요소가 오픈 소스 소프트웨어이기 때문에 비용에 있어 효율적
- 방대한 양의 데이터 분석을 하기 위해 탄생

아키텍처

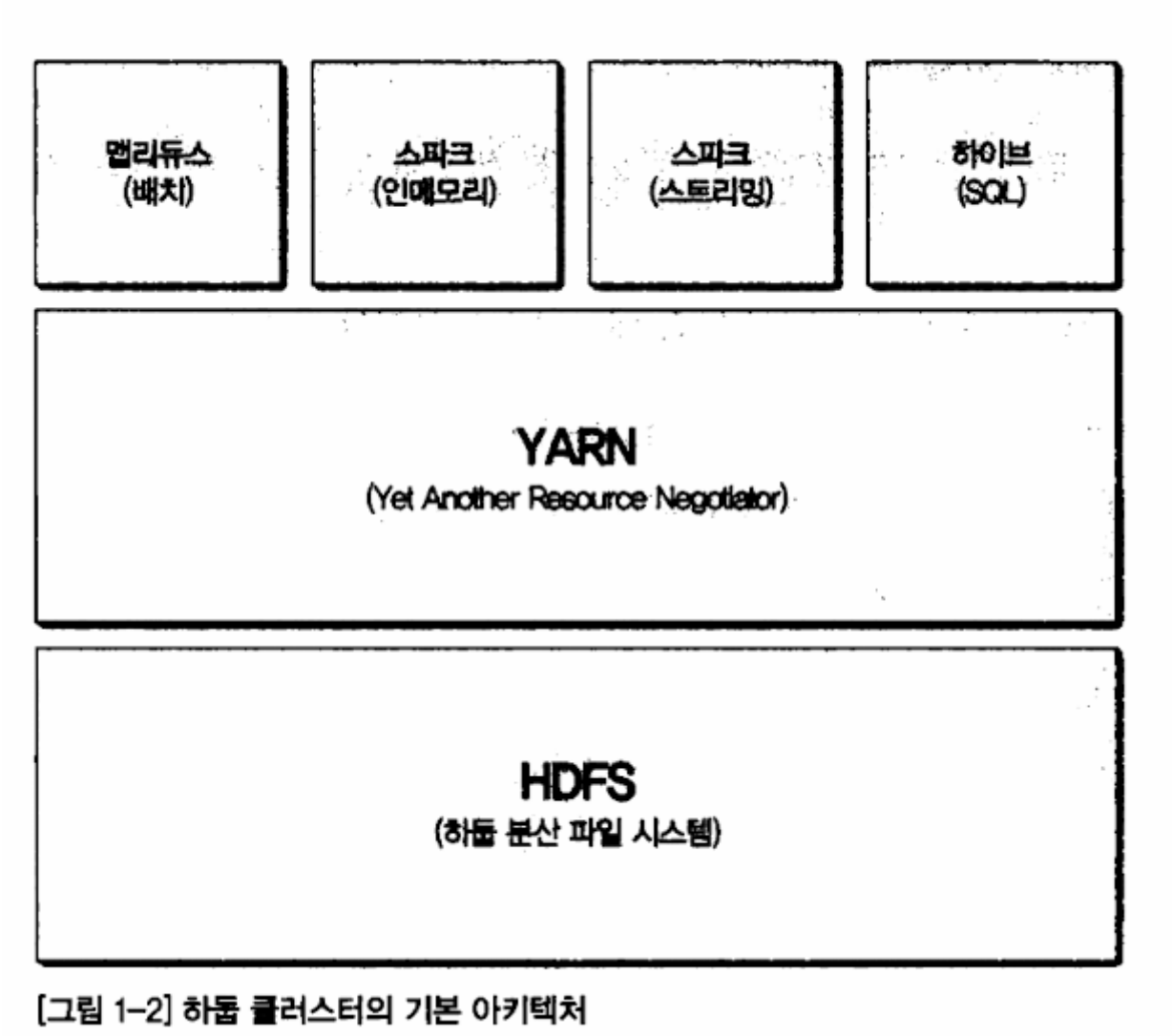
클러스터 컴퓨팅

- 하둡은 다수의 컴퓨터 노드를 이용해 큰 스케일의 데이터를 관리하는 최고의 시스템
- 하둡이 사용하는 수 많은 노드는 큰 데이터를 처리하는데 있어 효율성과 내고장성을 제공
- 하나의 컴퓨터를 사용하는 것이 아니라 평균적으로 6대에서 64대의 노드를 가지고 사용
- 클러스터 컴퓨터 모델에서 서버가 고장나는 일은 흔한 경우로 간주하여 장애 복구 능력이 뛰어남
  - 데이터 블록들을 여러 개 만들어 각각 다른 공간에 있는 서버에 저장
  - 하둡은 기본적으로 복제 개수를 3개로 설정
- 큰 작업들을 작은 단위의 작업들로 나누는 것
- 작은 단위의 작업들이 실패했을 때 다른 노드에서 쉽게 다시 실행할 수 있음
- 기존의 OLAP 시스템과 다르게 클러스터 모델에서는 한 번 쓰고 여러 번 읽는 형태의 작업을 다룸
- 대부분 데이터들은 파일 시스템에 한 번 기록되면 거의 변경되지 않음

하둡 클러스터

- 하둡 클러스터는 하둡이 실행되는 머신들과 그 머신이 데이터를 저장하고 프로세싱 하도록 하는 운영 시스템인 데몬, 소프트웨어 프로세스들로 구성
- 하둡 클러스터는 쉽게 확장하거나 반대로 줄일 수 있음

- 하둡 클러스터의 핵심은 HDFS임



[그림 1-2] 하둡 클러스터의 기본 아키텍처

- HDFS의 분산 파일 시스템은 데이터 청크들을 신속하게 처리하기 위해 일반적으로 매우 큰 데이터 블록을 사용
- 장애에 대해서 데이터를 중복해서 저장
- 맵리듀스라는 프로그램 모델을 이용해 데이터를 처리
- 하둡 클러스터는 기본적으로 리눅스 서버에서 데몬과 독립적인 JVM(Java Virtual Machine) 내에서 동작

하둡 생태계

- 하둡은 기본적으로 몇 가지 컴포넌트로 구성
- 이 기본 컴포넌트 위에 다른 컴포넌트를 추가할 수 있음

기본 컴포넌트

- HDFS : 데이터에 접속해 높은 처리량을 보여주는 파일 시스템
- Yarn : 작업을 스케줄링하고 리소스를 관리하는 프레임웍크

- 맵리듀스 : 크기가 큰 데이터 세트를 병렬 처리하기 위한 프레임웍크

확장 컴포넌트

- Flume : 스트리밍 데이터를 하둡으로 이동하는 데이터 이동 툴
- HBase : HDFS를 스토리지에 사용하는 분산 컬럼 지향의 데이터베이스(NoSQL DB)
- Hive : HDFS 데이터를 위한 분산 데이터 웨어하우스로 SQL 기반 쿼리를 제공
- Hue : 사용자와 관리 인터페이스로 하여금 HDFS 파일을 검색하게 하고, Pig나 Hive 쿼리를 작동시키고 Oozie를 통한 워크플로를 스케줄링 함
- Kafka : 많은 양의 실시간 데이터를 처리하는 메시지 큐 프레임웍크
- Pig : 대규모 데이터세트를 분석하기 위한 프레임웍크, 데이터 파이프라인을 만들 수 있게 함
- Sqoop : HDFS와 관계형 데이터베이스 사이의 데이터를 이동하기 위한 툴
- Zookeeper : 하둡, HBase, Hive, Kafka와 같은 분산형 애플리케이션에 의해 사용되는 조정 서비스

HDFS

-하둡 분산 파일 시스템

- 하둡 버전의 GFS
- 빅데이터를 클러스터의 컴퓨터에 분산 저장하는 시스템
- 클러스터들의 저장소를 하나의 거대한 파일 시스템으로 사용
- 데이터가 저장될 때 여분의 복사본까지 생성
- 장애가 일어났을 때 백업 본을 사용하여 회복

블록

- 물리적인 디스크는 블록 크기란 개념이 존재
- 블록 크기는 한 번에 읽고 쓸 수 있는 데이터의 최대의 크기이다.
- 단일 디스크를 위한 파일시스템은 디스크 블록 크기의 정수배인 파일시스템 블록 단위로 데이터를 다룬다.
- 파일시스템 블록의 크기는 보통 수 킬로바이트고, 디스크 블록의 크기는 기본적으로 512바이트
- 사용자는 파일의 크기와 상관없이 파일을 읽고 쓸 수 있으며, 특정 파일시스템에 구애받지도 않는다.

HDFS의 블록

- 기본적으로 128MB와 같이 굉장히 큰 단위

- HDFS의 파일은 단일 디스크를 위한 파일시스템처럼 특정 블록 크기의 청크(chunk)로 쪼개지고 각 청크는 독립적으로 저장
- 단일 디스크를 위한 파일시스템은 디스크 블록 크기보다 작은 데이터라도 한 블록 전체를 점유하지만, HDFS 파일은 블록 크기보다 작은 데이터일 경우 전체 블록 크기에 해당하는 하위 디스크를 모두 점유하지 않음
  - 예를 들어 HDFS의 블록 크기가 128MB고 1MB 크기의 파일을 저장한다면 128MB의 디스크를 사용하는 것이 아니라 1MB의 디스크만 사용
- HDFS의 블록의 크기가 큰 이유는 탐색 비용을 최소화하기 위해서임
- 블록이 매우 크면 블록의 시작점을 탐색하는 데 걸리는 시간을 줄일 수 있고 데이터를 전송하는데 더 많은 시간을 할애
- 따라서 여러 개의 블록으로 구성된 대용량 파일을 전송하는 시간은 디스크 전송 속도에 크게 영향을 받는다.
- 블록은 내고장성(fault tolerance)과 가용성(availability)을 제공하는 데 필요한 복제(replication)를 구현할 때 적합
- 블록의 손상과 디스크 및 머신의 장애에 대처하기 위해 각 블록은 물리적으로 분리된 다수의 머신(기본값 3대)에 복제
- 블록이 손상되거나 머신의 장애로 특정 블록을 더 이상 이용할 수 없으면 또 다른 복사본을 살아 있는 머신에 복제하여 복제 계수(replication factor)를 정상 수준으로 돌아오게 할 수 있음

YARN

- Yet Another Resource Negotiator
- 리소스 교섭자라는 의미를 가짐
- YARN은 데이터 처리 부분을 담당. 즉 클러스터의 리소스를 관리하는 시스템
- 누가 작업을 언제 실행했고 어떤 노드가 추가 작업을 할 수 있고 누구는 할 수 없고 등을 결정

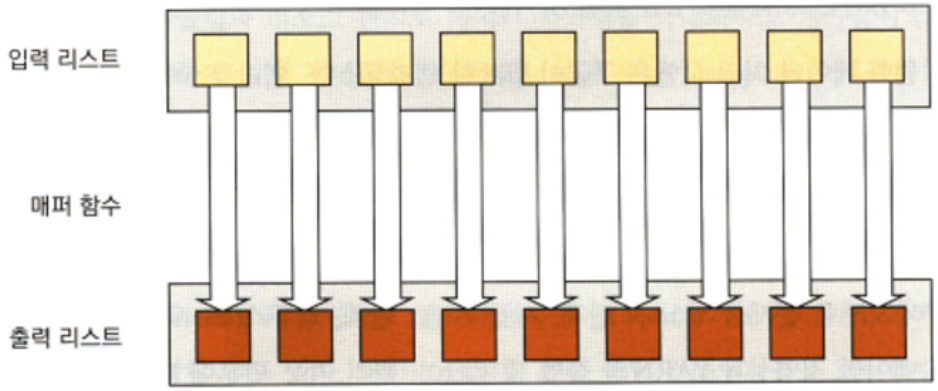
MapReduce

- 데이터를 클러스터 전체에 걸쳐 처리하도록 하는 프로그래밍 모델
- Map와 Reduce로 구성
- Mapper는 클러스터에 분산돼 있는 데이터를 효율적으로 동시에 변형시킬 수 있음
- Reducer는 그 데이터를 집계
- 원래는 MapReduce와 YARN이 거의 같은 역할을 하였지만, 현재는 분리
- 병렬 처리 모델은 문제를 map, shuffle, reduce 단계로 나눠 수행

map

- 입력 데이터가 클러스터에서 병렬로 처리되며 이 맵 단계를 수행하는 mapper 함수는 원시 데이터를 key와 value의 쌍으로 변환

❯ 그림 3-3 맵 단계: 입력 리스트는 독립적인 블록으로 나뉘어 HDFS에 저장돼 있다. 매퍼 함수는 각 블록에 맞게 병렬로 실행된다. 출력 리스트는 키-값 쌍의 집합이다.



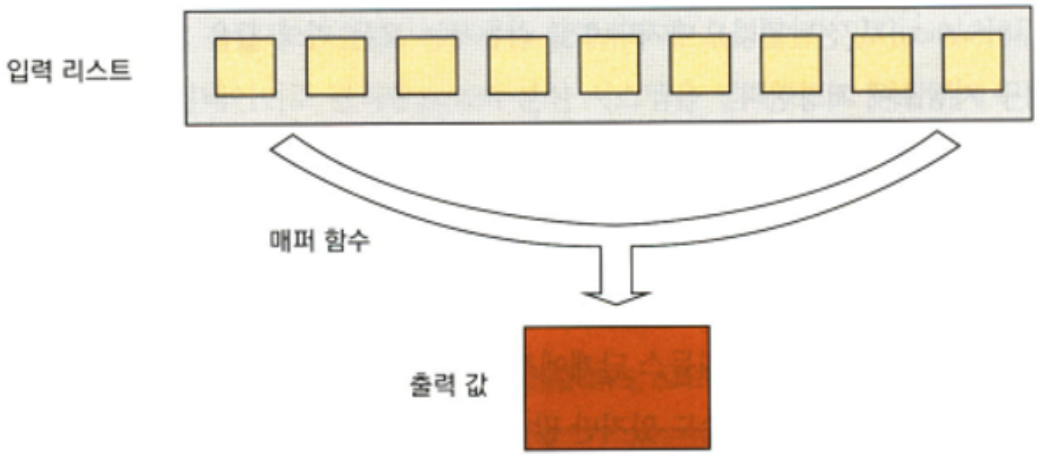
shuffle

- 변환된 데이터는 키를 기준으로 정렬돼 bucket으로 셔플링된다.

reduce

- 모든 키의 값을 처리하며 결과를 HDFS나 다른 영구 저장소에 저장

❯ 그림 3-4 리듀스 단계: 맵 단계의 출력 리스트가 리듀스 단계의 입력 리스트로 연결된다. 여러 리듀서가 사용될 경우, 입력 리스트는 키 값으로 묶여 특정 리듀서로 전달된다. 리듀서는 입력 리스트를 출력 값으로 병합(리듀스)한다.



마스터 노드와 워커 노드

마스터 노드

- 클러스터의 작업을 중재
- 컴퓨팅을 하기 위해 마스터 노드
- 클러스터들은 구성 크기에 따라 3~6개 정도의 소수의 마스터 노드들을 구성

워커 노드

- 마스터 노드의 지시에 따라 명령을 수행
- 대부분의 클러스터 노드들을 워커 노드에 해당
- 워커 노드들을 실제로 데이터가 저장되고 프로세싱하는 노드

하둡 서비스

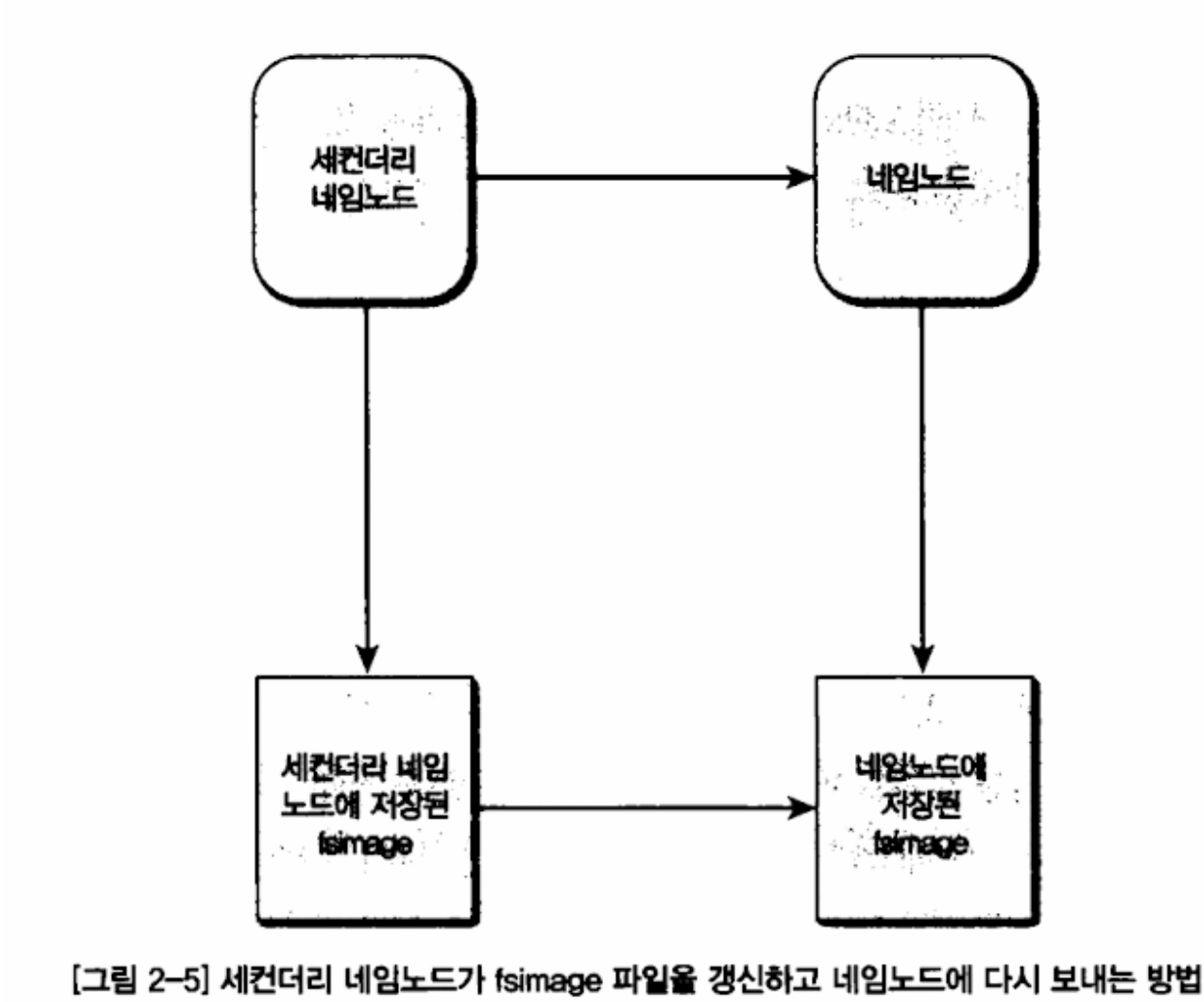
네임노드

- 마스터 노드에서 동작하면서 HDFS 파일 시스템 디렉토리 트리와 파일의 위치와 같은 HDFS 스토리지에 관련된 메타데이터를 유지
- 클라이언트가 HDFS를 읽거나 쓰려고 할 때, HDFS는 네임노드 서비스에 접속하고 네임노드는 HDFS 내에 있는 파일 위치와 같은 정보를 제공
- HDFS에서 마스터 역할을 하며, 워커 역할을 하는 데이터노드에게 I/O 작업을 할당한다.
- 손상되면 데이터노드에 저장된 블록으로부터 파일을 재구성하는 방법을 알 수 없기 때문에 파일시스템의 모든 파일을 찾을 수 없게 됨.
- 데이터노드와 주기적으로 모니터링 하면서 데이터노드의 용량이 다 차면 다른 데이터노드로 블록을 이동 가능하도록 만듬
- 파일시스템 트리와 그 트리에 포함된 모든 파일과 디렉터리에 대한 메타데이터를 유지
- 이 정보는 네임스페이스 이미지와 에디트 로그라는 두 종류의 파일로 로컬 디스크에 저장
- fsimage라는 파일 시스템 이미지를 저장하며 아래와 같은 정보를 가지고 있음
  - HDFS에 있는 모든 파일 이름
  - HDFS 디렉토리 구조
  - HDFS에 있는 모든 파일의 퍼미션 정보

세컨더리 네임노드

- 네임노드가 없다면 어떤 데이터노드에 어느 블록들이 파일로 저장돼 있는지 알 방법이 없음
- 네임노드가 손상될 경우 대비하여 이중화를 구성
- 세컨더리 네임노드는 주기적으로 fsimage 파일의 체크포인트를 실행
- 체크포인트는 세컨더리 네임노드가 fsimage와 edit 파일을 통합하는 작업
  - edit 파일은 fsimage 파일이라는 네임노드 메타 데이터 파일에 대한 트랜잭션 로그 파일
    - 파일 생성
    - 파일 삭제
    - 파일 이름 변경
    - 파일 권한 변경
- 네임노드가 구동될 때 로컬에 저장된 fsimage와 editslog를 조회하고, 메모리에 fsimage를 로딩하여 파일 시스템 이미지를 생성합니다

- 이후, 메모리에 로딩된 파일 시스템 이미지에 editslog에 기록된 변경 이력을 적용하고, 이를 이용해 fsimage 파일을 갱신합니다
- edits 파일의 크기가 커지면, fsimage를 만드는 데 시간이 많이 소요
- 이러한 문제를 해결하기 위해 보조 네임노드가 fsimage를 갱신하는 역할을 담당
- 이 과정을 체크포인트라고도 부름
- 이 체크포인트 과정에서 edits 파일의 변경 이력이 fsimage 파일에 반영되고, edits 파일은 초기화됨

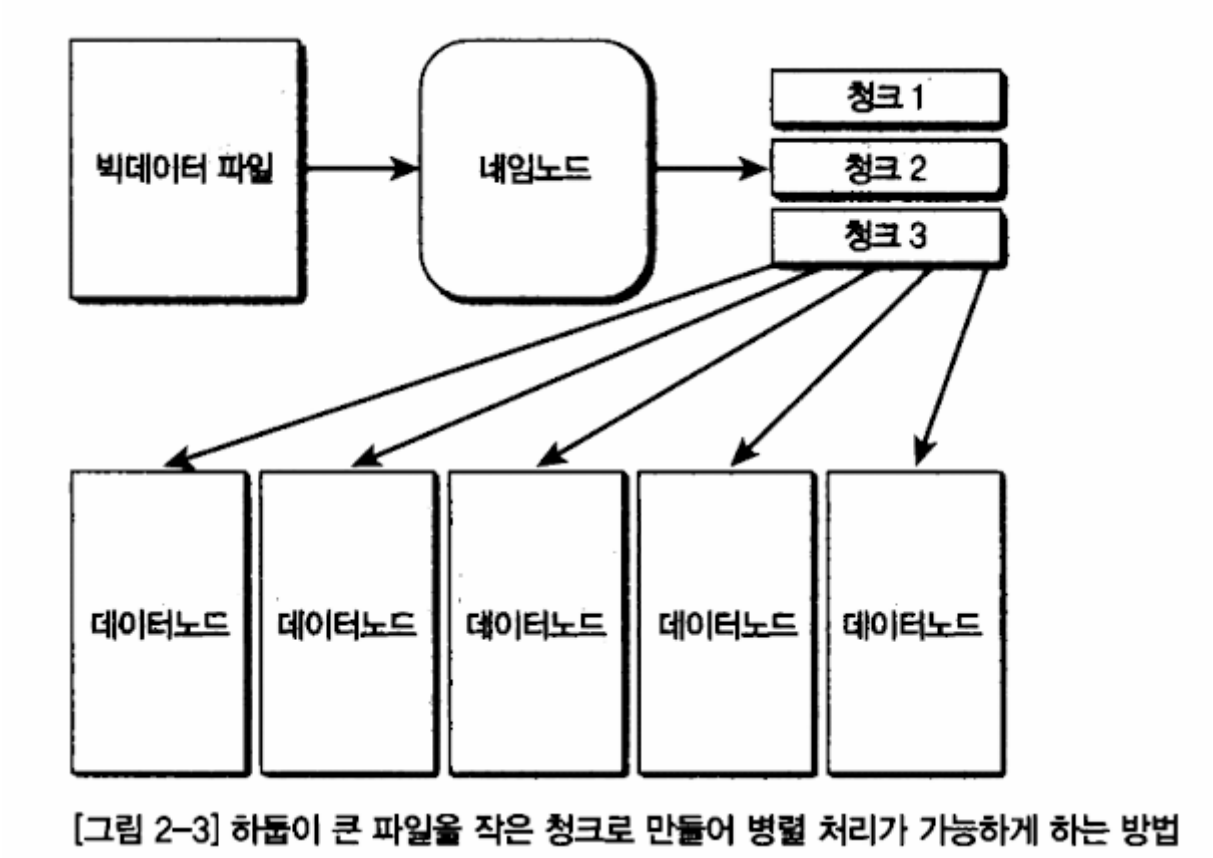


[그림 2-5] 세컨더리 네임노드가 fsimage 파일을 갱신하고 네임노드에 다시 보내는 방법

데이터노드

- 워커 노드에서 실행되는 서비스
- 클라이언트나 네임노드의 요청이 있을 때 블록을 저장하고 탐색하며, 저장하고 있는 블록의 목록을 주기적으로 네임로드에 저장
- 데이터노드들은 네임노드와 접속을 계속 유지하고 파일 시스템에서 일어나는 모든 변화를 네임노드에 업데이트함
- 데이터 블록들을 생성 및 삭제

- 클러스터에 걸쳐 데이터 복제



[그림 2-3] 하둡이 큰 파일을 작은 청크로 만들어 병렬 처리가 가능하게 하는 방법

- 큰 파일을 여러 개의 청크의 크기로 저장. 이때 청크의 크기는 블록 사이즈에 의해 결정

얀 서비스

리소스 매니저

- Resource Manager
- 클러스터에 하나만 실행되는 서비스로, 마스터 노드 중 하나에서 실행
- 클러스터의 리소스를 나눠주는 역할을 함
- 워커 노드에 태스크를 스케줄링하는 일도 담당
- 리소스 매니저는 스케줄러와 애플리케이션매니저라는 2개의 핵심 컴포넌트를 가짐

역할

- 모든 얀 애플리케이션의 시작을 위한 처리
- 잡 스케줄링과 실행 관리
- 모든 데이터노드의 리소스 할당

주요 기능

- 애플리케이션을 위한 첫 번째 컨테이너를 만듬. 바로 이 컨테이너에서 애플리케이션마스터가 실행됨

- 데이터노드를 관리하기 윟 노드매니저의 하트 비트를 관리
- 클러스터 사이의 리소스 할당을 결정하기 위해 스케줄러를 실행
- 클러스터 레벨 보안을 관리
- 애플리케이션에서 온 리소스를 요청 처리
- 애플리케이션마스터의 상태를 관찰하고, 실패할 경우 컨테이너를 재시작
- 애플리케이션이 완료되거나 소멸된 후, 컨테이너 할당을 해제

애플리케이션 마스터

- Application Master
- 마스터 서비스로, 클러스터에서 실행되는 애플리케이션마다 하나씩 실행
- 애플리케이션 마스터는 클러스터에 있는 애플리케이션의 실행을 조율하고 애플리케이션을 위한 리소스들에 대해 리소스매니저와 협상하면서 리소스를 조절

노드 매니저

- Node Manager
- 워커 노드에서 동작하는 서비스
- 노드매니저 서비스는 태스크를 워커 노드에서 실행하고 관리
- 노드매니저는 리소스매니저와 긴밀한 관계를 유지하고 노드 상태 및 자신들이 동작시키고 있는 태스크 상태를 업데이트

기능

- 하트비트와 컨테이너 상태 알림을 통해 리소스매니저와 정보를 주고 받음
  • 애플리케이션 프로세스를 등록하고 시작
  • 애플리케이션마스터를 시작하고, 애플리케이션 매니저의 응답에 따라 애플리케이션 리소스 컨테이너 의 나머지들(컨테이너에서 동작하는 맵과 리소스 태스크)도 시작
  • 애플리케이션 컨테이너의 라이프 사이클을 관리
  • 컨테이너의 리소스 사용량(CPU/메모리)과 관련된 정보를 관찰, 관리, 제공
  • 데이터노드의 상태 추적
  • 컨테이너 리소스의 사용량 모니터링 및 불필요한 프로세스 제거
  • 잡의 로그들을 수집하고 통합해 HDFS에 저장하는 등의 로그 관리
  • 얀 애플리케이션을 위한 보조 서비스를 제공
- 맵리듀스 프레임워크에서 사용되는 정렬이나 셔플 같은 기능으로 애플리케이션에서 사용하는 서비스를 제공
  • 노드 레벨의 보안 유지

얀 아키텍처

- 얀은 리소스매니저에 의존

- 리소스 매니저는 하둡 클러스터에서 동작하는 모든 애플리케이션 사이에서 중재자 역할을 함
- 그리고 클러스터에 있는 모든 워크 노드를 관리하는 노드매니저와도 협력

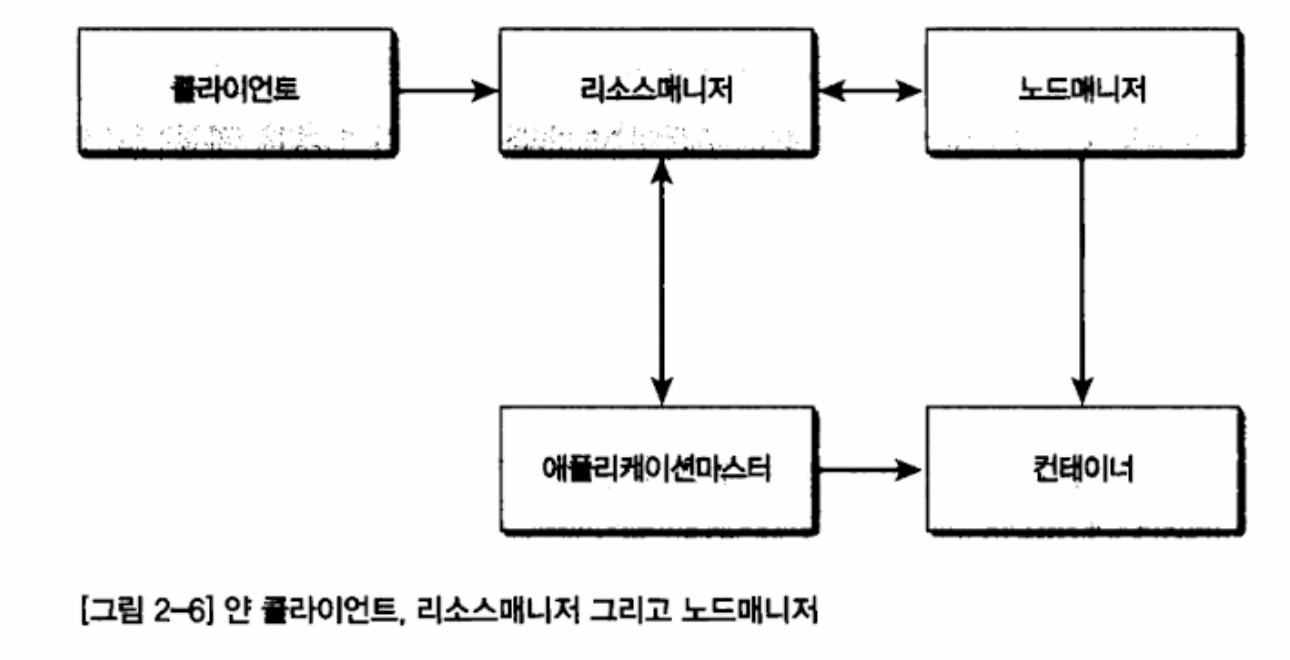
용어 정리

클라이언트

- 클러스터에 얀으로 실행할 잡을 서밋하는 프로그램
- 간혹 클라이언트 프로그램을 동작시키는 게이트웨이 머신을 의미하기도 함

잡

- 하나 이상의 태스크를 포함하는 애플리케이션을 의미
- 예를 들어, 맵리듀스 잡은 맵퍼, 리듀스 그리고 프로세싱할 입력 리스트를 포함
- 맵리듀스 잡을 동작시킬 때, 태스크는 맵퍼와 리듀스가 수행한 작업들
- 애플리케이션은 맵퍼와 리듀서 둘 모두 사용하거나 리듀스는 사용하지 않고 맵퍼만 사용할 수 있음
- 각 맵퍼와 리듀스 태스크는 컨테이너 내에서 동작
- 관리자는 컨테이너 크기를 설정하고 얀은 맵퍼와 리듀스의 수를 결정
- 컨테이너는 램, CPU와 같은 리소스를 추상화한 단위



[그림 2-6] 얀 클라이언트, 리소스매니저 그리고 노드매니저

얀 컨테이너

- 얀은 컨테이너를 사용해 애플리케이션을 실행

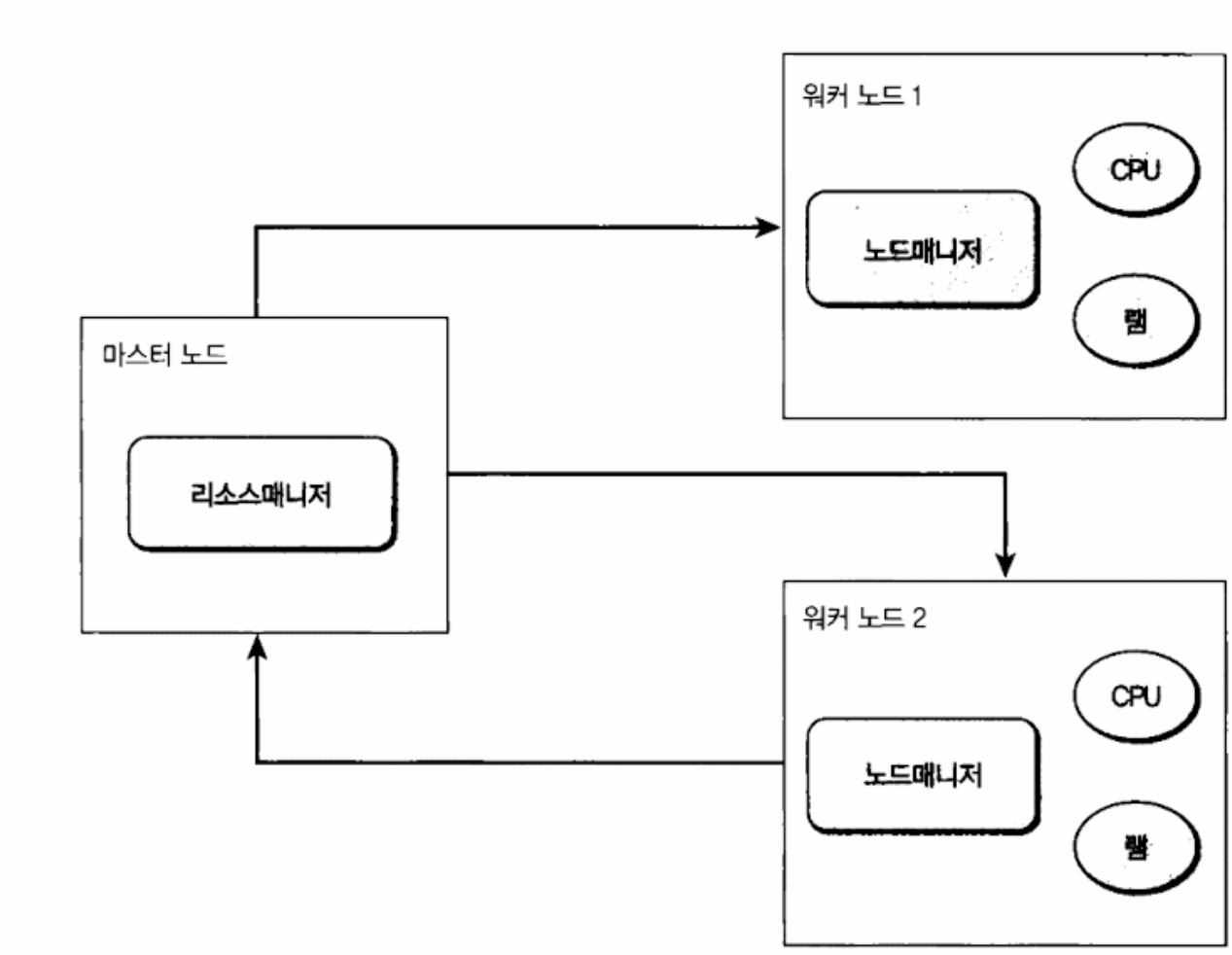
- 컨테이너는 메모리와 프로세싱 코어와 같은 다른 리소스의 양을 구체적으로 표현하는 논리적 구조
- 예를 들어, 어떤 컨테이너는 2GB의 메모리와 2개의 프로세싱 코어로 나타낼 수 있음
- 얀에 있는 모든 애플리케이션은 컨테이너 내에서 동작
- 각가의 하둡 잡은 태스크를 갖고 있고, 각 태스크들은 자신이 속한 컨테이너 내에서 동작
- 컨테이너는 태스크가 시작될 때 생성되고 완료되면 소멸하며 갖고 있던 리소스들은 다른 태스크에 할당

애플리케이션마스터

- 각각의 얀 애플리케이션에는 하나의 전용 애플리케이션마스터가 있음

주요기능

- 태스크 스케줄링과 실행 관리
- 애플리케이션 태스크를 위해 로컬리소스 할당
- 애플리케이션 마스터는 실행 중인 애플리케이션과만 관련되어 있음



[그림 2-7] 노드매니저가 노드의 리소스를 관리하고 애플리케이션 컨테이너를 감독하기 위해 리소스 매니저와 연동하는 방법

리소스 매니저와 협력

- 애플리케이션마스터가 맵과 리듀스 태스크를 실행하기 위해 리소스매니저에게 리소스를 요청할 때, 다음과 같은 것들을 매우 구체적으로 요청
  - 잡을 처리하는데 필요한 파일 블록
  - 애플리케이션을 위해 만들어야 하는 컨테이너 수
  - 컨테이너 크기(CPU core Ram등)
  - 네임노드를 통해 리소스가 할당된 위치를 할당받고, 데이터 블록이 저장되는 위치도 요청
  - 리소스 요청 우선순위

잡히스토리서버

- 전체 클러스터에 잡히스토리서버는 하나만 존재
- 잡히스토리서버는 모든 얀 잡의 평가 지표들과 메타데이터를 보관하고 있고, 잡히스토리서버 웹 UI를 통해 이 정보들을 확인
- 클러스터에 잡히스토리서버가 없어도 되지만, 잡히스토리 서버 없이는 잡 로그나 잡 히스토리를 보기 어려움

설치

hadoop 설정 파일

환경 설정 파일

- hadoop-env.sh
- mapred-env.sh
- yarn-env.sh

기본 설정 파일

- core-default.xml
- hdfs-default.xml
- mapred-default.xml
- yarn-default.xml

수정할 수 있는 설정 파일

- hdfs-site.xml
- core-site.xml

- mapred-site.xml
- yarn-site.xml

데몬

- HDFS 데몬 : 네임노드, 세컨더리 네임노드, 데이터 노드
- YARN 데몬 : 리소스매니저, 노드매니저, 잡히스토리서버
- 데이터노드와 노드매니저 이 두 데몬은 모든 워커 노드에서 실행

환경변수

- JAVA_HOME
- HADOOP_CLASSPATH
- HADOOP_HEAPSIZE
- HADOOP_LOG_DIR
- HADOOP_PID_DIR

읽기 전용 디폴트 설정 파일

- core-default.xml
  - 코어 하둡 기본 세팅으로 core-site.xml 파일의 설정값으로 오버라이드됨
- hdfs-default.xml
  - HDFS 관련 서비스들을 위한 디폴트 세팅. hdfs-site.xml의 세팅으로 오버라이드 됨
- mapred-default.xml
  - 맵리듀스 v2 디폴터 세팅. mapred-site.xml에 있는 설정값으로 오버라이드 됨
- yarn-default.xml
  - 얀을 위한 디폴트 세팅. Yarn-site.xml에 있느 설정값으로 오버라이드 됨

core-site.xml

- fs.defaultFS는 네임노드 서비스를 위해 디폴트 파일 시스템의 호스트와 포트 정보를 명시
- 클러스터에서 사용할 디폴트 파일 시스템 이름을 명시

```
<configuration>
 <property>
  <name>fs.defaultFS</name>
   <value>hdfs://namenode:8020/</value>
 </property>
```

mapred-site.xml

- 아래 매개변수는 맵리듀스 잡을 실행하기 위한 런타임 프레임웍
- local, classic, yarn 중에서 하나를 선택함

```
<property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
</property>
```

- 잡히스토리서버는 이 매개변수의 지정된 주소/포트 값을 이용해 내부 통신을 한다
- 기본값은 0.0.0.0:10020

```
<property>
                <name>mapreduce.jobhistory.address</name>
                <value>namenode:10020</value>
</property>
```

- 잡히스토리서버는 웹 UI에 접근하는 액세스 포인터로, 이 매개변수의 주소/포트를 사용

```
<property>
                <name>mapreduce.jobhistory.webapp.address</name>
                <value>namenode:19888</value>
</property>
```

yarn-site.xml

- 노드매니저에게 맵리듀스 컨테이너로 하여금 맵 태스크에서 리듀스 태스크로 셔플해야 한다는 것을 알려주는 것

```
<property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
</property>
```

hdfs-site.xml

- 하둡은 기본적으로 파일을 작성할 때 각각의 데이터 블록을 세 벌씩 복사
- 디폴트 값은 3이지만, 단일 노드를 가지고 있다면 1로 설정해야 함

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

```
        <value>3</value>
</property>
```

- 데이터 노드가 로컬 파일 시스템의 어느 디렉토리에 블록들을 저장할 지 지정
- HDFS 데이터를 저장하려면 디렉토리에 있어야 함
- 네임노드를 포맷하게 되면, 네임노드는 이 디렉토리를 로컬 리눅스 파일 시스템이 아닌 HDFS로 관리되는 뭔가로 변환

```
<property>
        <name>dfs.datanode.data.dir</name>
        <value>file:///home/hadoop/data</value>
</property>
```

- 하둡에게 fsimage, edits 파일과 같은 네임노드의 핵심 메타데이터 파일을 저장할 위치를 지정
- 네임노드 서비스만 이 메타데이터에 읽기/쓰기를 할 수있음
- 해당 폴더의 내용이 손실되면 HDFS의 내용을 복구할 수 없음

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

<value>file:///home/hadoop/data/name1,file:///home/hadoop/data/name2</value>
</property>
```

- 세컨더리 네임노드는 fsimage와 edit 로그를 저장하기 위해서 아래 매개변수의 폴더를 사용

```
<property>
        <name>dfs.namenode.checkpoint.dir</name>
        <value>file:///home/hadoop/data/namesecondary</value>
</property>
```