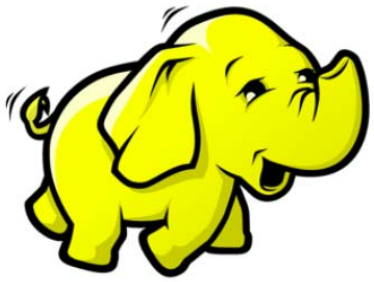


아파치 하둡 소개

순천향대학교 컴퓨터공학과
이 상 정

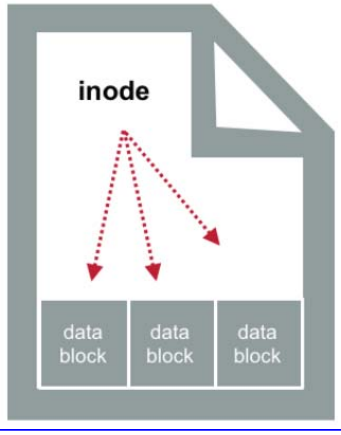
학습 내용

1. 로컬 파일 시스템의 문제점
2. 아파치 하둡 소개
3. 하둡 에코 시스템
4. 응용 사례

1. 로컬 파일 시스템의 문제점

로컬 파일 시스템 (Local File System)

- 로컬 파일 시스템의 각 파일은 inode(메타 데이터)와 데이터 블록으로 구성
- inode
  - 파일에 관한 속성 (파일 타입, 권한, 소유자, 파일 이름, 갱신 시간)을 나타내는 메타 데이터
  - 파일 저장된 데이터 블록의 위치를 가리키는 포인터
- 데이터 블록
  - 파일의 실제 내용이 저장



데이터 손실

- 하드 드라이브의 고장 시 모든 파일의 데이터가 손실
- 동일한 로컬 드라이브를 설치하여 미러링 (mirroring)
- 컴퓨터 자체가 고장 나거나 잃어 버리면 여전히 데이터 손실

- 클라우드에 중요한 파일들을 미러링하여 데이터 손실 방지

인간 실수

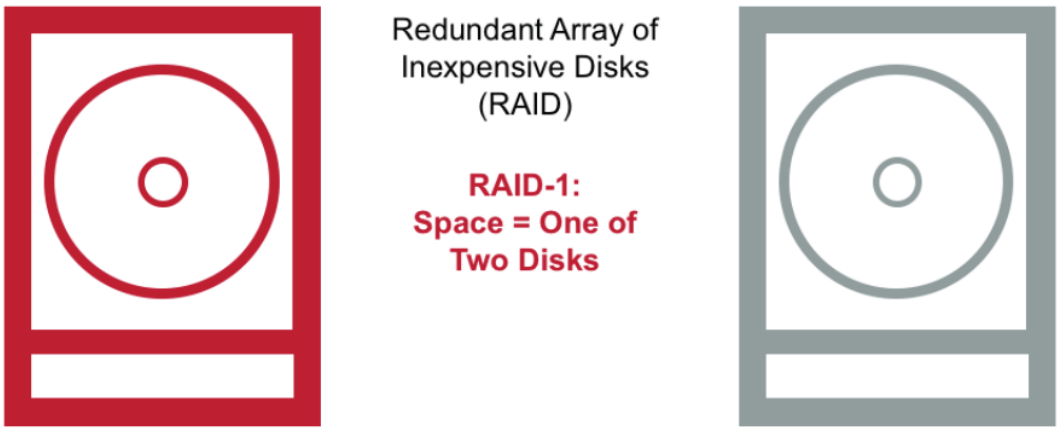
- 작업자가 실수로 주요한 파일이나 디렉토리를 삭제한 경우 미러링은 도움이 안됨
  - 미러링은 로컬 또는 원격에서 동기화되므로 한 쪽에서 삭제하면 다른 쪽도 삭제됨
- 위의 방지를 위해 백업 드라이브에 정기적으로 백업
  - 운영체제는 증분 백업(incremental backup)을 지원
    - 변경 또는 갱신된 부분만 백업하여 공간 및 처리 시간을 줄임
    - 백업된 시간 이전으로 복구도 가능

저장 공간 (1)

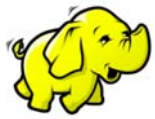
- 로컬 드라이브의 비디오, 오디오등의 데이터 등으로 저장 공간 부족은 흔히 발생
- 새로운 하드 드라이브를 파일 시스템에 추가하여 확장
  - 이를 RAID-0라고 함

로컬 파일 시스템의 문제점 – 저장 공간 (2)

- 로컬 미러(local mirror)는 RAID-1 이라고 함
  - 전체 공간은 두 개의 디스크 중 하나의 크기

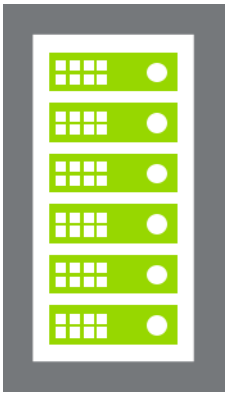


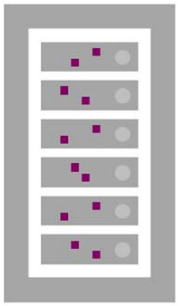
2. 아파치 하둡 소개



아파치 하둡 (Apache Hadoop) 소개

- 아파치 하둡은 대용량 데이터의 분산 저장 및 처리를 위한 오픈 소스 프레임워크
  - 특수한 전용 하드웨어가 아닌 일반 범용 머신들로 클러스터의 노드 구성
  - 여분의 리던던시(redundancy)로 구현되어 높은 고장 감내(fault-tolerant) 시스템
    - 한 디스크가 노드가 고장 나도 데이터 손실 없이 나머지 노드가 작업을 수행

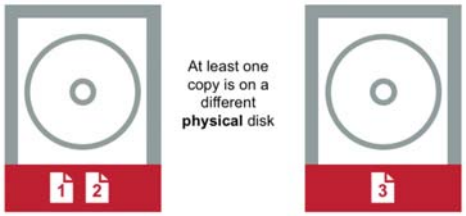




하둡 분산 파일 시스템 (HDFS) – 복제

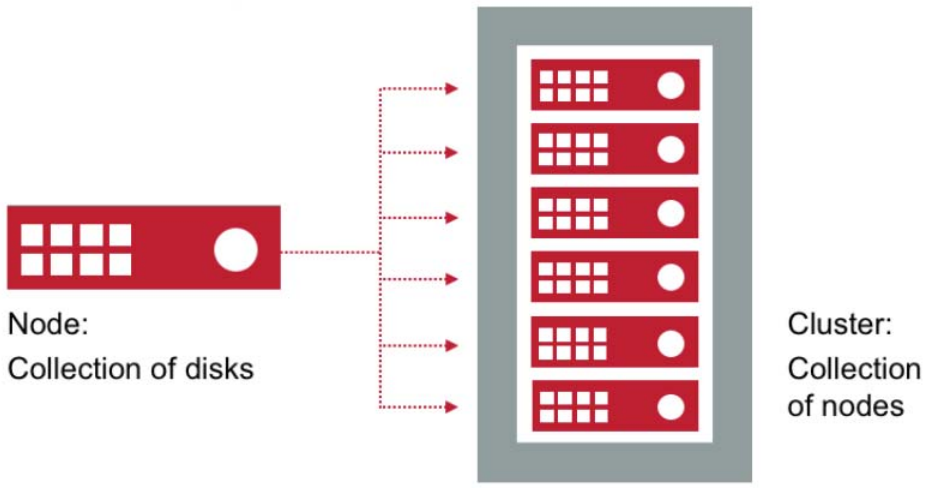
- 하둡 분산 파일 시스템 (Hadoop Distributed File System)
  - 디폴트로 데이터가 3번 복제(replication)

  - 최소한 하나는 다른 노드에 위치



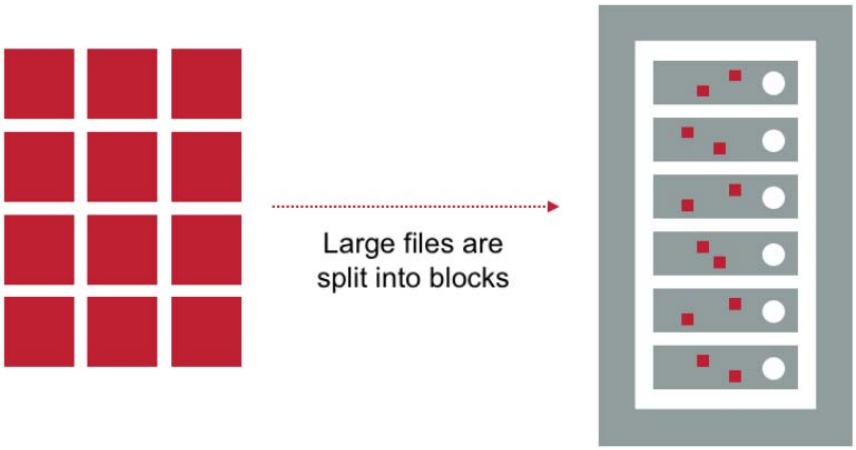
하둡 분산 파일 시스템 – 노드와 클러스터

- 물리적 디스크는 노드와 클러스터로 구성
  - 노드 (node)는 디스크들로 구성
  - 클러스터 (cluster)는 노드들로 구성



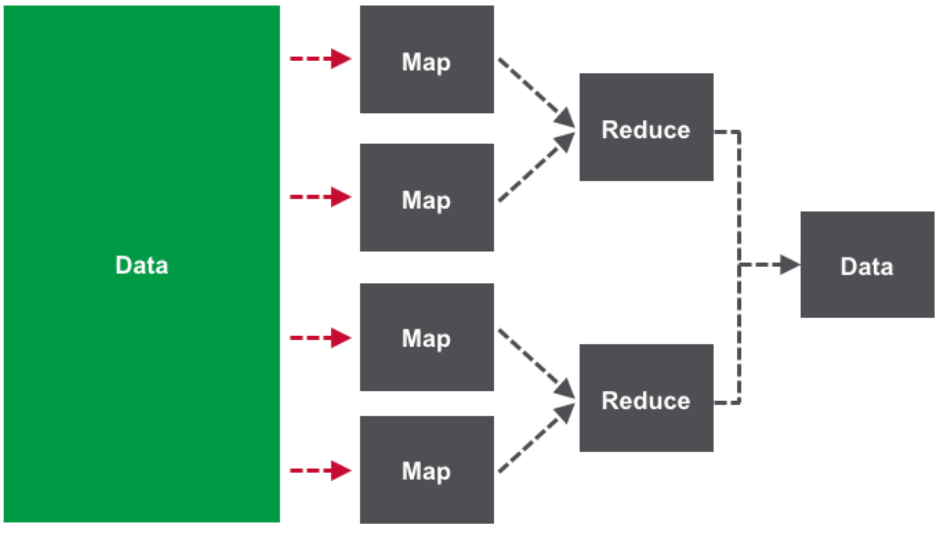
하둡 분산 파일 시스템 – 블록

- 대규모 파일들이 블록으로 분할
- 블록 단위로 데이터가 복제, 읽기, 쓰기 수행
  - 예를들어 1 TB의 파일의 경우 블록으로 분할되고 복제되어 여러 노드들에 분산되어 저장



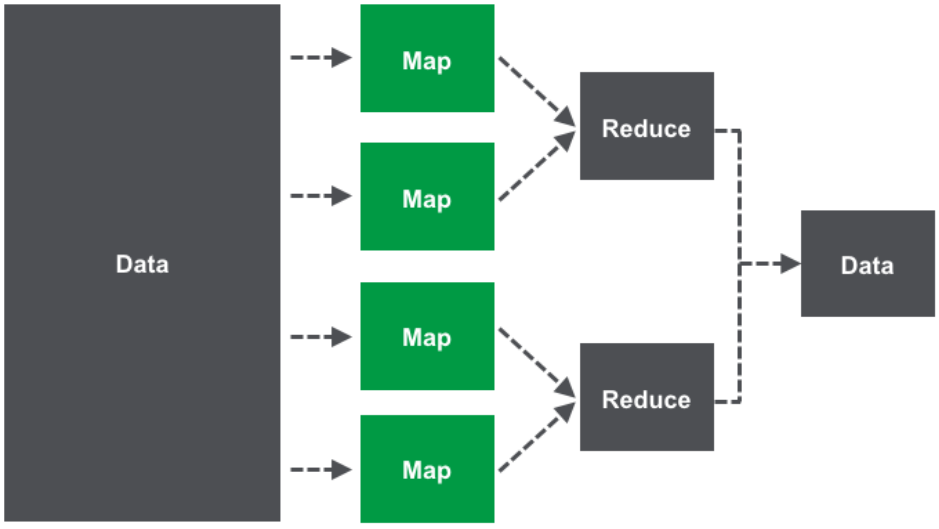
분산 처리 – 맵리듀스 (MapReduce) 알고리즘

- 하둡의 분산 처리를 위한 맵리듀스 알고리즘은 맵(Map), 셔플(Shuffle), 리듀스(Reduce)의 3 단계로 구성



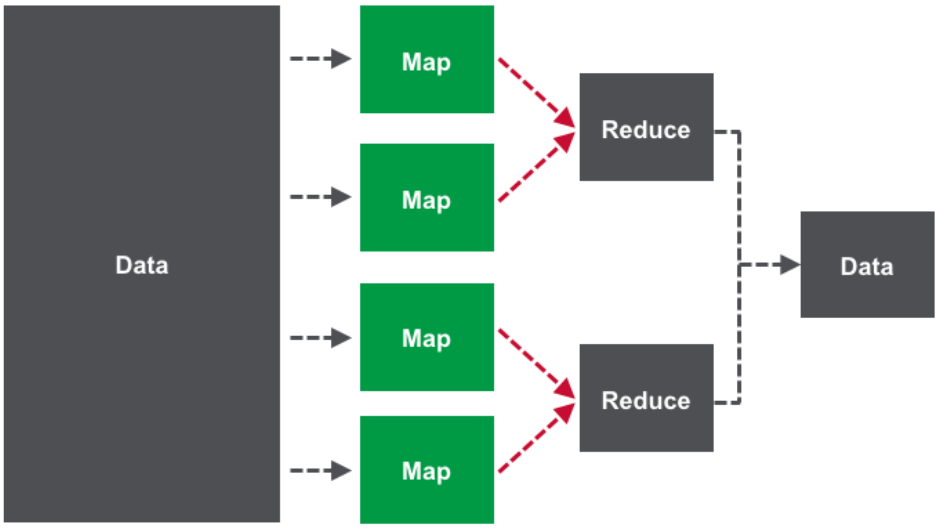
맵리듀스 알고리즘 – 맵 (Map)

- 맵 단계에서는 데이터를 분할하고, 맵 함수를 적용하여 키-값 쌍 (key-value pair)을 생성



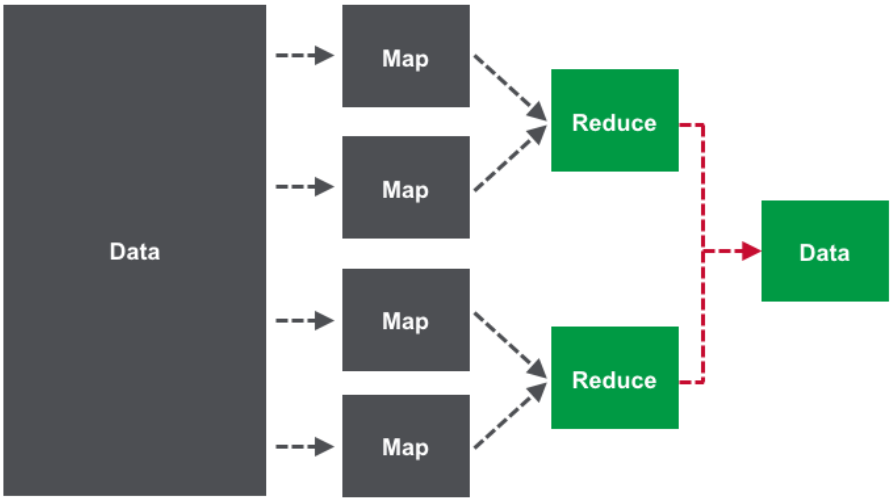
맵리듀스 알고리즘 – 셔플 (Shuffle)

- 셔플 단계에서는 맵 단계 출력의 키/값들의 쌍들을 정렬하고 분할하여 리듀스 단계로 전달



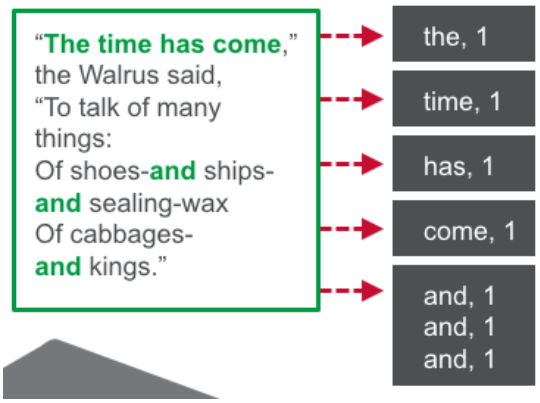
맵리듀스 알고리즘 – 리듀스 (Reduce)

- 리듀스 단계에서는 각 분할에 대해 리듀스 함수를 적용
- 맵리듀스 알록리즘은 분할정복 (divide-and-conquer) 기반 알고리즘으로 하나의 큰 작업을 여러 개의 작은 작업으로 나누어 병렬 처리



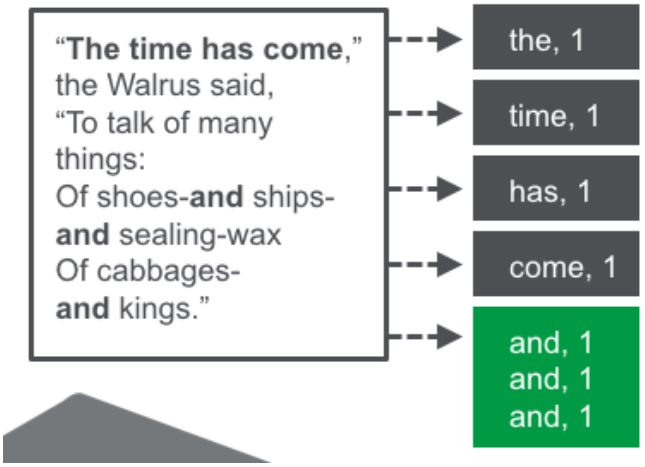
맵리듀스 단어 카운트 예 – 입력

- 맵리듀스 알고리즘의 “Hello World” 버전인 단어 카운트 (word count) 예로 동작 설명
  - Lewis Carroll 의 모든 책에 대한 단어를 카운트
- 하둡은 먼저 데이터 파일들을 분할하여 데이터 노드들에 분배
  - 분배 받은 한 노드 예



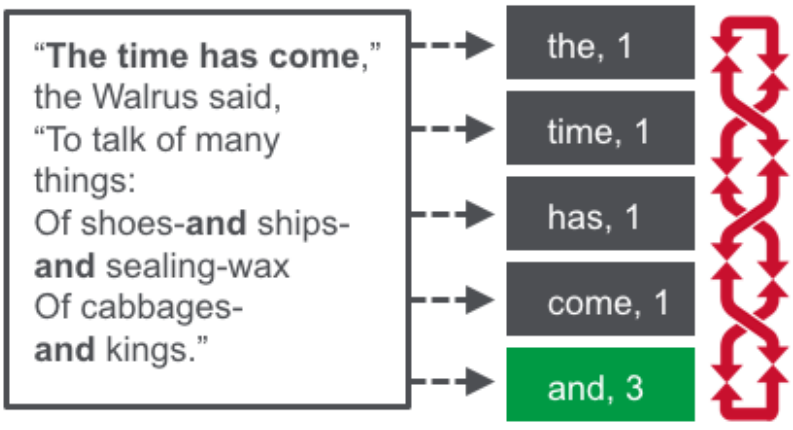
맵리듀스 단어 카운트 예 – 맵

- 맵 단계에서 입력 문자열을 토큰화하여 모든 단어에 대해 키-값 쌍을 출력
  - 이 예에서는 3개의 “and”가 3개의 다른 키-값 쌍으로 생성



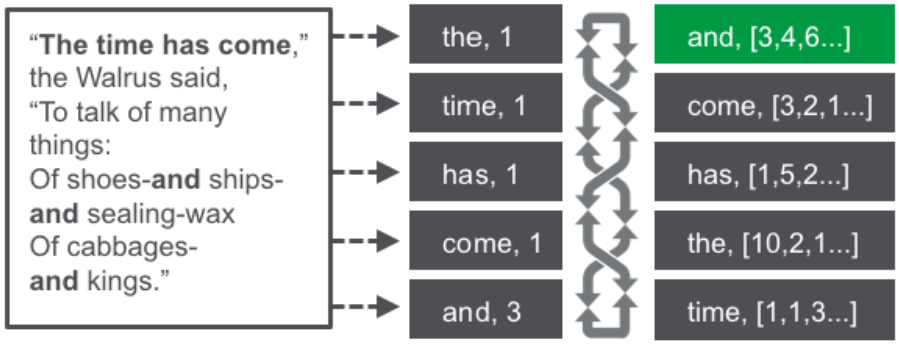
맵리듀스 단어 카운트 예 – 병합 및 셔플

- 병합 및 셔플 단계에서는 중복된 같은 키-값 쌍을 병합한 후에 셔플



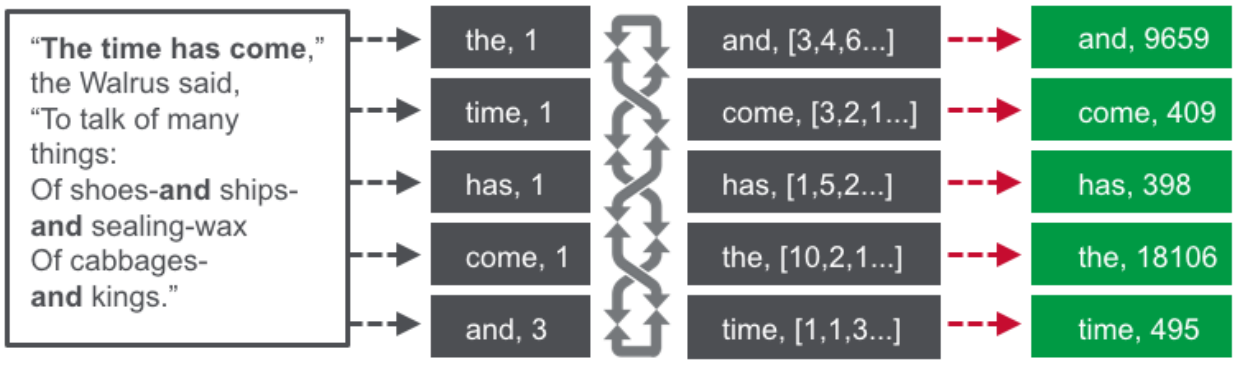
맵리듀스 단어 카운트 예 – 리듀스

- 리듀스 단계에서 다른 노드의 맵 결과들을 취합하여 통합



맵리듀스 단어 카운트 예 – 출력

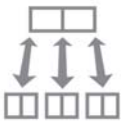
- 마지막으로 하둡 프레임워크는 출력을 취합하여 파일 시스템에 저장



3. 하둡 에코 시스템

하둡 에코 시스템

- 하둡의 핵심 2 요소
  - HDFS: 데이터 저장
  - 맵리듀스: 데이터의 처리



- 하둡은 다양한 소스의 구조화/비구조화 데이터를 하둡 클러스터에 가져와 처리하는 응용들을 제공
  - 이들 데이터 처리 응용들의 일부는 하둡 에코 시스템 (Hadoop ecosystem)으로 알려짐

빅 데이터 파이프라인

- 하둡 에코 시스템은 시스템 관리자, 프로그램 개발자, 데이터 분석가 들이 빅 데이터를 데이터의 소스로 부터 수집(ingestion) 되어 처리(processing) 및 분석(analysis) 의 파이프라인 단계의 과정을 수행할 수 있는 툴을 제공

하둡 에코 시스템 – 클러스터 관리

- 하둡 관리자는 주키퍼(Zookeeper)를 사용하여 클러스터를 관리하고, 클러스터에 실행되고 있는 맵리듀스나 YARN의 버전을 파악하고 있어야 함

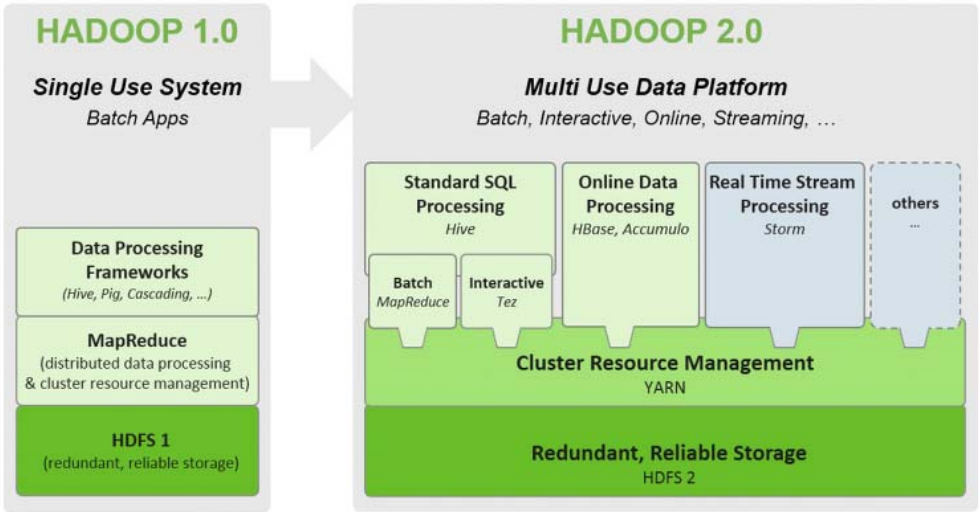
아파치 주키퍼 (Apache Zookeeper)

- 아파치 주키퍼는 클러스터의 각 노드들 사이의 서비스들을 관리하고 조정
  - 하둡에서 처음 실행되는 서비스
  - 서비스들 간의 동기화
  - 장애 상황 판단 및 복구
  - 네임서비스를 통한 부하분산
  - 환경 설정 관리

- **Coordinates other services**
  - Maintains configuration information
  - Monitors cluster heartbeats
- **Runs on an odd number of nodes**
  - Requires a majority for tie-breakers
- **Starts before other services**

YARN

- YARN (Yet Another Resource Negotiator)
  - 하둡 클러스터의 각 응용에 자원을 할당하고 모니터링
  - 다양한 응용들이 하둡 클러스터 자원의 공유 지원
    - 초기에는 하둡에 맵리듀스만 지원



하둡 에코 시스템 – 작업 기술 및 데이터 수집

- 개발자는 자바, 파이썬, 스칼라 등을 사용하여 수행할 작업을 기술하고, 플럼(Flume), 우지(Oozie), 스쿱(Sqoop) 등의 툴을 사용하여 데이터를 수집

아파치 플럼 (Apache Flume)

- 아파치 플럼은 하둡 클러스터에 스트리밍 데이터를 수집하는 오픈소스 서비스
  - 시스템 로그(syslog), 웹 서버 로그, 소셜 미디어 피드 등의 외부 소스로 부터의 스트림 이벤트 데이터들을 하둡 클러스터에 가져옴

- Ingests data into Hadoop cluster
- Commonly used with log files

아파치 스쿱 (Apache Sqoop)

- 아파치 스쿱은 하둡 클러스터와 외부 데이터 저장소 (RDBMS, NoSQL 데이터) 사이에 데이터를 가져오고 내보내는 툴

- Transfers data between HDFS and external data stores
- Can be used with RDBMS and NoSQL

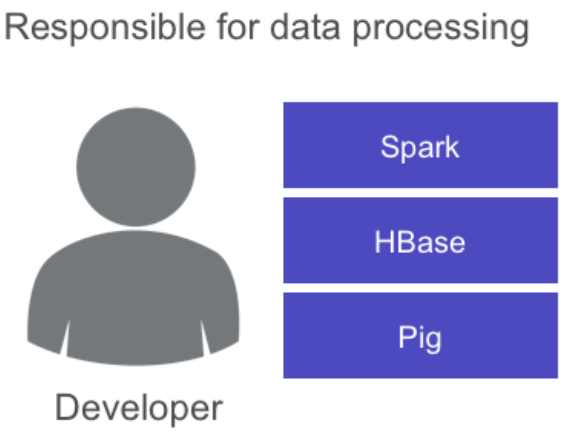
아파치 우지 (Apache Oozie)

- 아파치 우지는 하둡의 워크 플로우(workflow, 작업 흐름)를 생성하고 스케줄링하는 툴
  - 하둡의 워크 플로우는 여러 응용들이 직렬 또는 병렬로 수행하여 복잡함
  - 워크플로우에는 자바나 스크립트 뿐만 아니라 맵리듀스, 피그(Pig), 하이브(Hive), 스쿱(Sqoop) 등과 같은 다수의 수행 작업들이 포함

- Schedules workflows
- Manages multiple jobs

하둡 에코 시스템 – 데이터 처리

- 개발자는 데이터 수집 뿐만 아니라 스파크(Spark), HBase, 피그(Pig) 등의 툴을 사용하여 데이터를 처리



아파치 스파크 (Apache Spark)

- 아파치 스파크는 하둡 등과 같은 분산 컴퓨팅 클러스터 상에서 범용 데이터 분석을 수행하는 프레임워크
  - 메모리 상에 캐시된 데이터를 사용하여 반복적인 작업을 수행하여 기계학습 등과 같은 복잡한 분석을 맵리듀스 보다 빨리 처리
  - 스칼라(Scala), 파이썬, 자바로 응용을 작성

아파치 HBase

- 아파치 HBase는 구글의 빅테이블 모델을 따라 개발된 오픈 소스 NoSQL 데이터베이스
  - 시스템 측정 값, 사용자 클릭 등과 같은 대용량 데이터를 행과 열들을 저장
  - 채팅, 이메일과 같은 일관성이 없는 값들을 갖는 데이터를 열에 저장
  - 웹 응용이나 검색 색인과 같이 랜덤 읽기/쓰기 접근이 연속으로 발생하는 데이터 저장
  - 자바로 Hbase 응용 작성

- **NoSQL database**
  - Capture large amounts of data
  - Store sparse data with inconsistent values
  - Continuous data with read/write access
- **Written in Java**

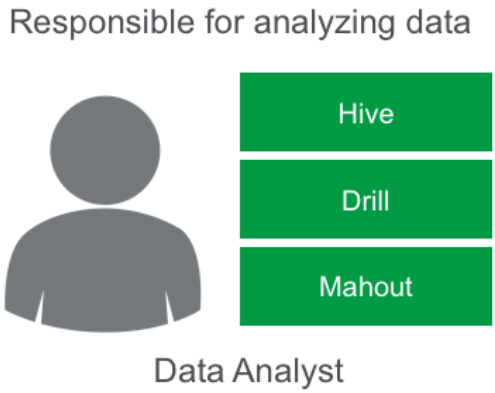
아파치 피그 (Apache Pig)

- 아파치 피그는 데이터를 분석하는 플랫폼
  - 데이터 플로우 스크립트 언어 “Pig Latin” 제공
  - 스크립트를 맵리듀스 프로그램 시퀀스로 변환
  - 파이썬, 자바, 자바스크립트, 루비로 함수 작성

- Includes scripting language called Pig Latin
- Transform and analyze large datasets
- Write functions in Python, Java, JavaScript, Ruby

하둡 에코 시스템 – 데이터 분석

- 데이터 분석가는 하이브(Hive), 드릴(Drill), 머하웃(Mahout) 등의 툴을 사용하여 데이터를 분석
  - 데이터 마이닝, 추출, 정규화, 필터링, 집계, 질의, 해석, 도식화, 머신러닝을 이용한 예측



아파치 하이브 (Apache Hive)

- 아파치 하이브는 하둡 상에서 구축되는 데이터웨어하우스 (data warehouse) 인프라스트럭처
  - HCatalog를 사용하여 데이터를 테이블에 저장하고, Hive Metastore를 사용하여 데이터를 추적
  - HiveQL이라는 SQL과 같은 질의 언어를 사용하여 맵리듀스 프로그램을 생성하고 실행

- Stores data in HCatalog and Hive Metastore
- Hive Query Language (HQL)
  - SQL-like query language
- Uses MapReduce

아파치 드릴 (Apache Drill)

- 아파치 드릴은 빅데이터 탐색을 위한 질의 엔진
  - SQL을 사용하여 HDFS나 하이브에 저장된 구조화, 반구조화, 비구조화 데이터들에 대한 동적 질의를 수행

- Structured, semi-structured, and unstructured data
- Multiple data sources and data types
- ANSI-SQL compliant

아파치 머하웃 (Apache Mahout)

- 아파치 머하웃은 기계학습 라이브러리
  - 군집화(clustering), 분류(classification), 협업 필터링(collaborative filtering) 알고리즘 지원
  - 대규모 빅데이터를 알고리즘에 적용하여 추천 등과 같은 예측을 시도

- Clustering
- Classification
- Collaborative filtering

4. 응용 사례

응용 사례

- 3가지 하둡 응용 사례 소개
  - 데이터 웨어하우스 최적화
    (Data Warehouse Optimization)
  - 추천 엔진
    (Recommendation Engine)
  - 대규모 로그 분석
    (Large-Scale Log Analysis)

데이터 웨어하우스란?

- 데이터 웨어하우스 (Data Warehouse)
  - 다양한 소스의 데이터를 저장하는 중앙 저장소 (centralized repository)
    - 마케팅, 재정, 인사관리, 웹 사이트, 외부 데이터 저장

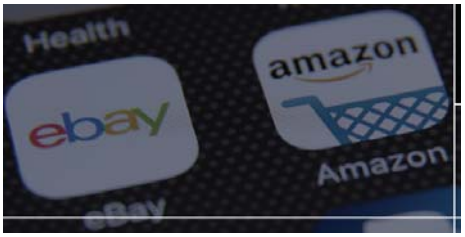
  - 분산 운영되는 각각의 다양한 데이터 소스를 통합 관리
    - 모든 데이터가 단일의 소스인 것처럼 질의, 처리
  - 통합 관리를 위해 공통의 구조화된 형식으로 데이터를 저장

데이터 웨어하우스 ETL

- 데이터 웨어하우스는 각각의 소스에 대해 ETL (Extract, Transform, Load) 을 수행
  - 데이터 수집 시 저장 및 처리 전에 데이터 웨어하우스의 스키마와 일치되도록 변환이 필요
- 데이터의 속도, 다양성, 규모 (3V)가 증가됨에 따라 ETL 처리 비용 증가

온라인 쇼핑의 데이터 웨어하우스 예 – 다양성

- 온라인 쇼핑 업체는 자신의 상품의 판매와 관련하여 발생되는 모든 정보의 이해를 원함
  - 각 품목의 메타데이터를 추적하고, 고객 평가와 감성분석(sentiment analysis)의 연관성을 알고자 함
  - 크기, 중량, 색상, 제조업자 등의 상품 관련 정보는 JSON과 같은 구조화 또는 반-구조화 형식으로 기술
  - 감성 분석으로 소셜 미디어 데이터를 활용
    - 소셜 미디어 데이터는 반-구조화 또는 비구조화된 데이터로 저장 및 처리가 어려움



온라인 쇼핑의 데이터 웨어하우스 예 – 속도

- 데이터는 수시로 변함
  - 소셜 미디어 데이터는 고객이 새로운 평가를 작성할 때마다 변경
  - 상품 정보도 변경되고, 새로운 기능이 추가될 때마다 상품의 용어도 변경
- 기업의 데이터 웨어하우스에서 빈번한 데이터의 변경은 처리 비용 및 시간 증대
  - 변경될 때마다 데이터를 변환하여 저장

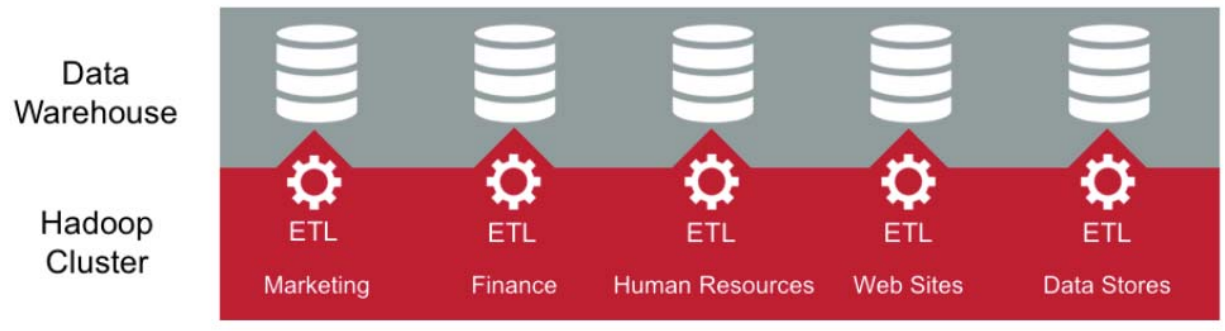
데이터 웨어하우스 최적화

- 기존의 질의 방식 및 기법들을 그대로 활용하면서 효율적인 비용으로 확장 가능한 솔루션을 어떻게 구축하는가?

How do we build a **cost-effective, scalable solution** to consume big data while taking advantage of **legacy query work and skills** our data analysts have developed?

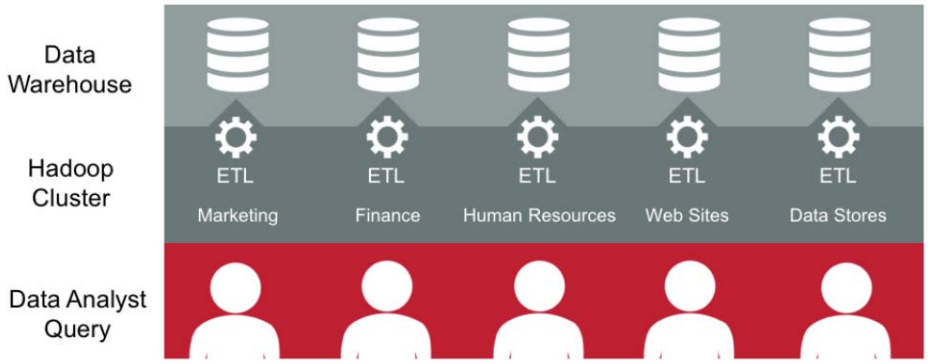
하둡과 데이터 웨어하우스 (1)

- 하둡은 데이터 웨어하우스에서 ETL 과정을 분리하여 최적화된 솔루션 제공
  - 데이터 웨어하우스는 저장에만 초점
  - 데이터 추출 및 변환 단계에서 아파치 플럼(Flume)이나 피그(Pig)를 사용
    - 데이터 웨어하우스는 변환된 데이터 만을 저장하기만 하여 비용 절감



하둡과 데이터 웨어하우스

- 데이터 분석가들이 기존의 툴과 코드를 사용하여 데이터 웨어하우스에 질의 가능
- 데이터 웨어하우스에 변환된 데이터를 저장하면서 수집된 데이터는 원래 형식대로 하둡 클러스터에 저장
  - 데이터를 다른 방식으로 처리한 후 데이터 웨어하우스에 저장도 가능

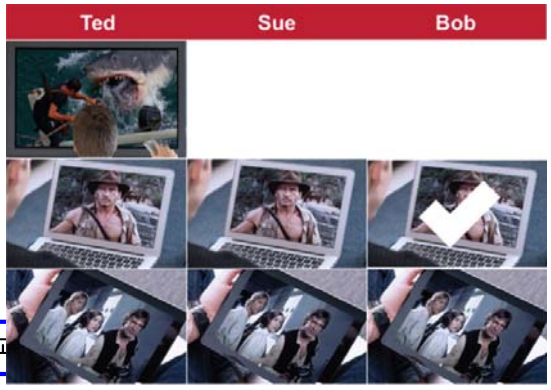


추천 (Recommendation)

- 아마존과 같은 사이트에서 쇼핑 시 상품을 추천
  - 고객의 취향에 부합하는 상품을 추천
  - 추천은 고객의 참여 및 만족도, 구매 이력 등을 반영하여 웹 사이트 방문 시 마다 갱신
- 성공적인 추천은 개별 고객의 취향에 맞추어서 적절한 시점에 해야함
  - 개별 고객의 취향을 정확하게 예측하기 위해 머신 러닝(machine learning) 알고리즘을 적용

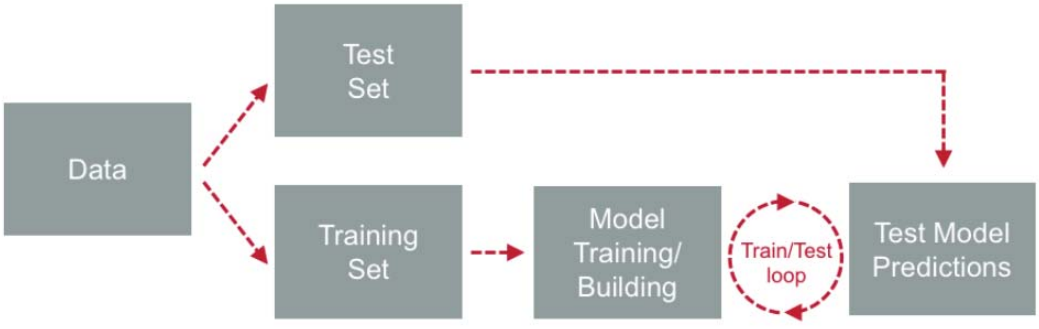
협업 필터링 (Collaborative Filtering)

- 머신 러닝 알고리즘 중 협업 필터링을 적용하여 추천 시스템 개발
  - 협업 필터링 알고리즘은 서로 다른 사용자들 사이의 선호도를 비교하여 추천이나 예측에 상요되는 모델을 생성
  - 추천 예
    - 3명의 사용자 선호도
      - Ted는 죠스, 스타 워즈, 인디아나 존스를 선호
      - Sue는 스타워즈, 인디아나 존스 선호
      - Bob은 스타 워즈 선호
    - 스타 워즈를 선호하는 사용자는 인디아나 존스를 선호할 것이라고 예측
    - Bob에게 추천되는 영화는 인디아나 존스



하둡을 이용한 추천 시스템 구축

- 과거 사용자의 선호(구매) 히스토리를 나타내는 대규모 데이터 세트를 오프라인으로 하둡 클러스터에 적재
  - 데이터는 테스트와 훈련 데이터로 분리
  - 이들 데이터 세트가 협업 필터링 모델을 학습(훈련)에 사용
  - 아파치 스파크(Spark)나 머하웃(Mahout)과 같은 머신 러닝 툴을 사용
  - 사용자 데이터가 추가될 수록 추천은 정확해짐



로그 분석 (Log Analysis) 사용 사례

- 사용자의 웹사이트 사용 패턴을 파악하기 위해 클릭 스트림 데이터가 로그파일에 저장
- 인터넷 TV의 셋탑 박스가 사용 중일 때 일련의 사용 기록 로그를 기록
- 서버의 로그는 컴퓨터 시스템의 사용자 접속, 사용한 응용 프로그램, 에러 등과 같은 시스템과 응용의 이벤트를 기록

Clickstream Data
Log Files

Set-Top Box
Streaming Logs

Server Logs
User Activity

서버 로그 분석 예

- 서버의 로그
  - 특정 기간 동안 수집되어 실시간으로 분석한 후 폐기
    - 이들 데이터의 폐기로 향 후에 유용하게 활용될 수 있는 데이터 손실
  - 이벤트의 발생 날짜와 시간 단위로 조직화되어 중첩된 형식(nested format)으로 저장
    - SQL과 같은 분석 툴을 사용하여 중첩된 데이터를 처리하기 어려움
  - 효율적인 비용으로 모든 히스토리의 대규모 로그들을 저장하고 분석하는 새로운 로그 분석 기법이 필요
    - 하둡 에코 시스템 아파치 드릴(Drill)은 효율적인 비용으로 다양한 소스로부터의 반구조화된 대규모 로그 데이터를 배치와 실시간으로 분석

Discarding data loses
valuable future insights

Difficult to process
nested data

서버 해킹 탐지 예

- 데이터가 하둡 클러스터에 저장되면 머신 러닝 및 통계 분석으로 데이터를 분석하여 지능적이고 유용한 정보를 제공할 수 있음
  - 서버 해킹 탐지 예
    - 서버의 비정상적인 동작을 감지하고 해킹 공격과의 연관성을 탐지
    - 의심되는 사용자의 행동 및 트랜잭션 감지
    - 네트워크 상의 디바이에서 비인가된 접속 발견
    - 히스토리 데이터 상에서 포렌식 분석 또는 조사(과학 수사, 범죄 규명)를 수행

과제

- 아파치 하둡 에코 시스템 중 하나를 선택하여 상세히 조사
  - 앞에서 소개한 또는 소개하지 않은 에코 시스템
    - 맵리듀스, YARN 포함
  - 응용 사례

참고 자료

- MapR Academy, http://learn.mapr.com/
  - Apache Hadoop Essentials
    - https://learn.mapr.com/series/essentials-series/ess-105-apache-hadoop-essentials
- Apache Hadoop
  - http://hadoop.apache.org/

ZK ZooKeeper: https://zookeeper.apache.org/
YA YARN: http://hadoop.apache.org/
FL Flume: http://flume.apache.org/
OO Oozie: http://oozie.apache.org/
SQ Sqoop: http://sqoop.apache.org/
SP Spark: http://spark.apache.org/
HB HBase: https://hbase.apache.org/
PI Pig: https://pig.apache.org/
HI Hive: https://hive.apache.org/
DR Drill: https://drill.apache.org/
MA Mahout: http://mahout.apache.org/