NOSQL(Not Only SQL)

송무찬

mcsong@gmail.com

Agenda

- 1.정의
- 2.특징
- 3.배경
  - 3.1이론적 배경
  - 3.2환경적 배경
  - 3.3기술적 배경
- 4.Data Model별 분류
- 5.Q&A

1. 정의

- Not Only SQL의 약자
- 비 구조적인 데이터를 저장하기 위한 분산 저장 시스템

2. 특징

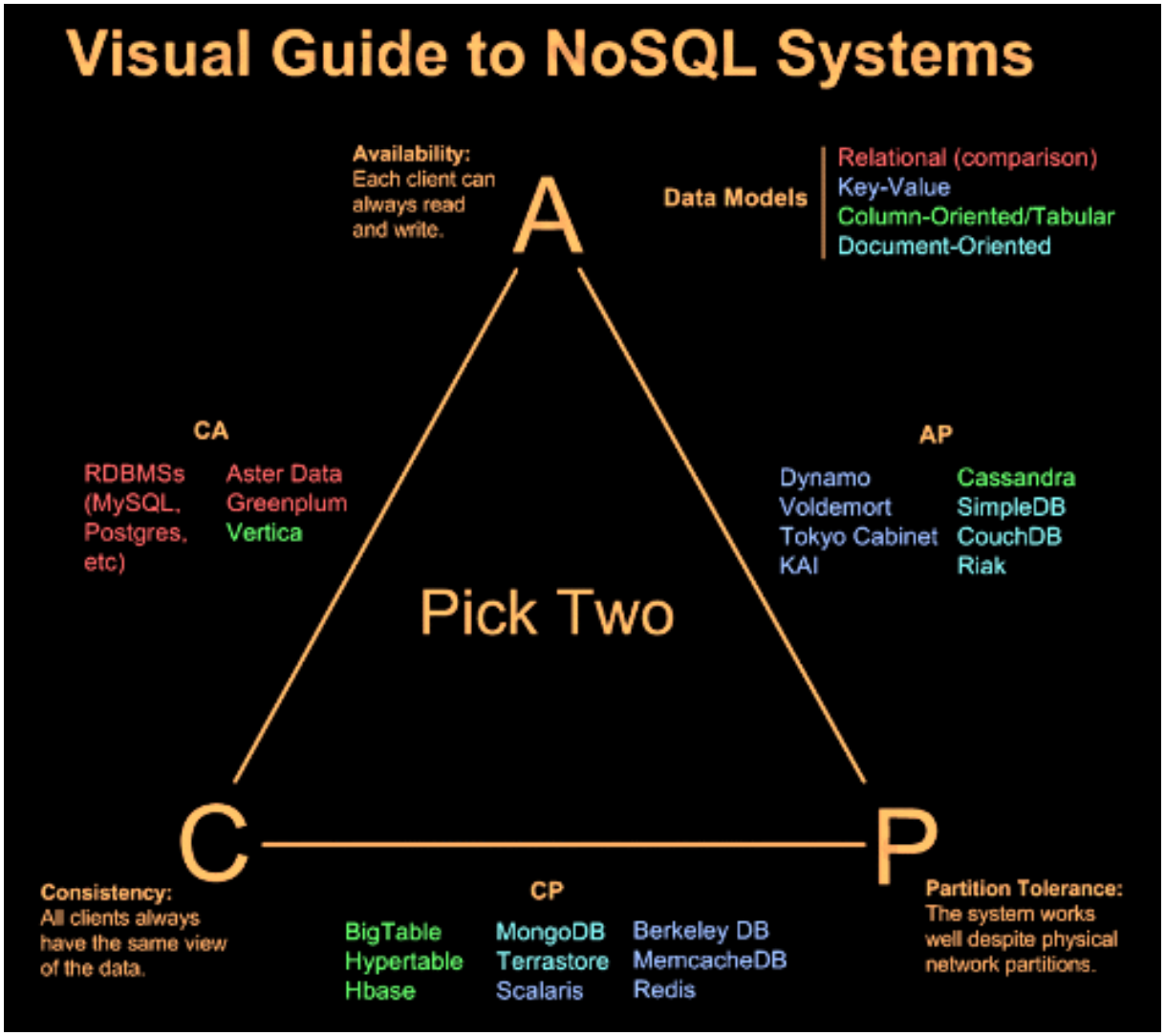
- 데이터 베이스라 부르기를 거부하고
- 기본적으로 key & value로 저장되고
- 분산 환경 지원
- call level interface 지원(DBMS에 접근하는 표준)
- 막대한 양의 데이터를 처리할 수 있는 대용량 데이터의 빠른 인덱싱
- 수평적 확장(horizontal scaling) 또는 scale-out의 형태로 Scalablity를 만족.
- 데이터의 스키마와 속성들을 동적으로 정의
- Join 미 지원으로 인한 데이터의 조합 미 지원

3.배경(1/19)

- 3.1 이론적 배경
  - 3.1.1 CAP Theorem
    - 분산 시스템이 갖추면 좋은 3가지의 특성
      - **C**onsistency : all nodes see the same data at the same time
      - **A**vailability : node failures do not prevent survivors from continuing to operate
      - **P**artition Tolerance : the system continues to operate despite arbitrary message loss
  - 3.1.2 Brewer's CAP Theorem
    - 분산 시스템은 위의 CAP 세 가지 특성을 동시에 만족하지는 못하기 때문에, 한 가지의 특성은 포기를 해야 한다.
    - 위의 CAP Theorem을 기준으로 RDB는 CA 특성을 만족시키는 시스템이라서, NOSQL 시스템들은 AP, CP의 특성을 가지는 분산 시스템들로 구성이 되어 있습니다.

3.배경(2/19)

- RDB = C+A
- Cassandra = A+P
- BigTable = C+P



3. 배경(3/19)

- 3.1.2.1 C-A 시스템
  - Traditional RDBMSs like Postgres, MySQL, etc (relational)

- 3.1.2.2 C-P 시스템
  - **BigTable (column-oriented/tabular, C++)**
  - Hypertable (column-oriented/tabular, C++)
  - **HBase (column-oriented/tabular, Java)**
  - MongoDB (document-oriented, C++)
  - Terrastore (document-oriented)
  - Redis (key-value, C)
  - Scalaris (key-value, Erlang)
  - MemcacheDB (key-value, C/C++)
  - Berkeley DB (key-value, C/C++/Java)

3. 배경(4/19)

- 3.1.2.3 A-P Systems
  - Dynamo (key-value)
  - Voldemort (key-value, Java)
  - Tokyo Cabinet (key-value, C)
  - KAI (key-value, Erlang)
  - **Cassandra (column-oriented/tabular, Java)**
  - CouchDB (document-oriented, Erlang)
  - SimpleDB (document-oriented)
  - Riak (document-oriented, Erlang)

3. 배경(5/19)

- 3.2 환경적 배경
  - 데이터의 엄청난 증가로 인한 RDBMS의 수용능력을 벗어남.
  - RDBMS의 고 비용을 감당하기 힘든 업체들의 데이터 관리 정책 전환.
  - 지리적인 분산(여러 지역의 데이터 센타 산재)으로 인한 분산 시스템의 요구.
  - 웹 어플리케이션 특징으로, Without transactions, Without strong consistency.
  - 웹 어플리케이션들의 데이터인 Non-structured data의 저장 시스템으로 RDBMS의 부 적절함.

3. 배경(6/19)

- 3.3 기술적 배경
  - Google의 GFS(2003), BigTable(2006), MapReduce Framework(2004), Sawzall(2005)을 통한 대용량 데이타 처리에 대한 기술 공개.

  - 3.3.1 GFS(2003, Apache Hadoop/HDFS, *http://hadoop.apache.org/hdfs/*)
    - 네트웍을 통해 파일을 읽고, 쓰는 시스템으로, chunk단위로 데이터를 관리하는 분산 시스템.
    - chunk의 기본단위는 64M, 수백M 이상의 대용량 파일 처리에 효과적으로 설계가 되어 있습니다.

3. 배경(7/19)

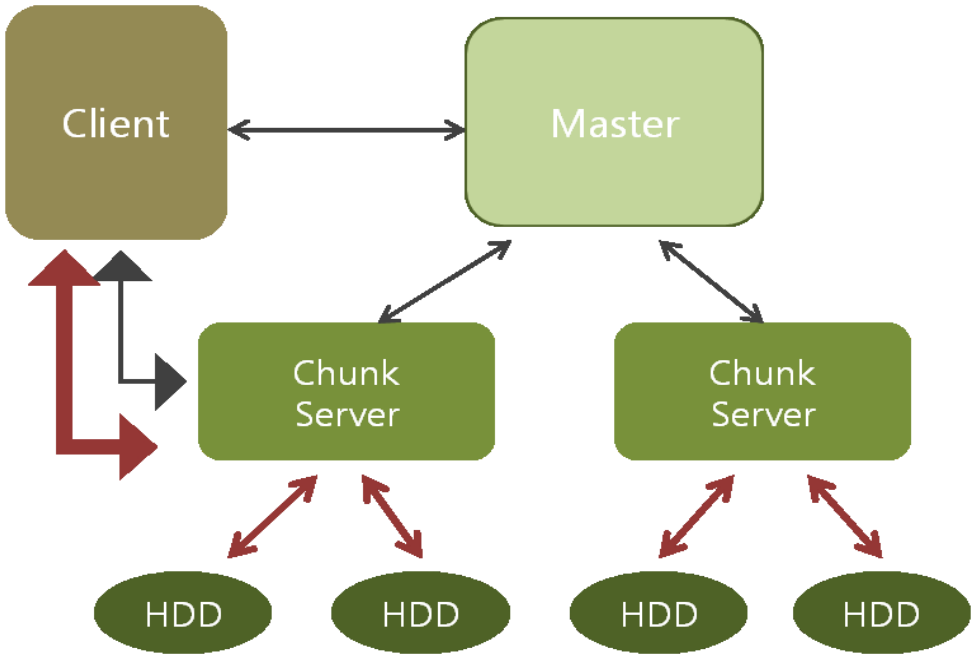
- 3.3.1.1 GFS vs FS

GFS	FS
다수의 하드 유지(효과적)	다수의 하드 유지(제한적)
읽기/쓰기/삭제만 가능	읽기/쓰기/업데이트/삭제 가능
항상 복사본 유지	

- 3.3.1.2 특징
  - Node(Chunk Server)의 다운이 서비스에 영향을 미치지 않음
  - NAS(Network Attached Storage)등 기존 Storage에 비해 월등히 저렴한 시스템
  - 파일 생성/쓰기에 대한 성능저하(파일복사를 동기적으로 처리하기 때문에, 3개의 파일이 완성 시 완료).
  - 3개의 복사본으로 인한 3배의 공간 소모

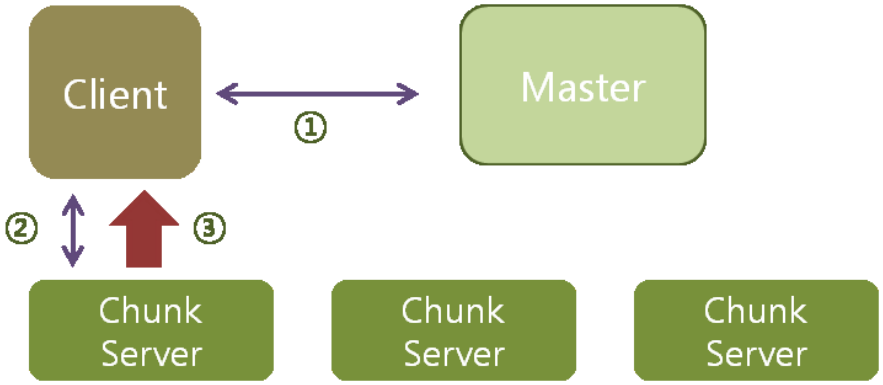
3. 배경(8/19)

- 3.3.1.3 Architecture
  - Master : GFS 전체 관리 및 통제 서버, Chunk Server의 정보관리, file과 chunk의 매핑 정보 유지
  - Chunk Server : 실제 HDD의 I/O 담당 서버, HDD는 chunk들의 집합으로 구성
  - Client : 클라이언트

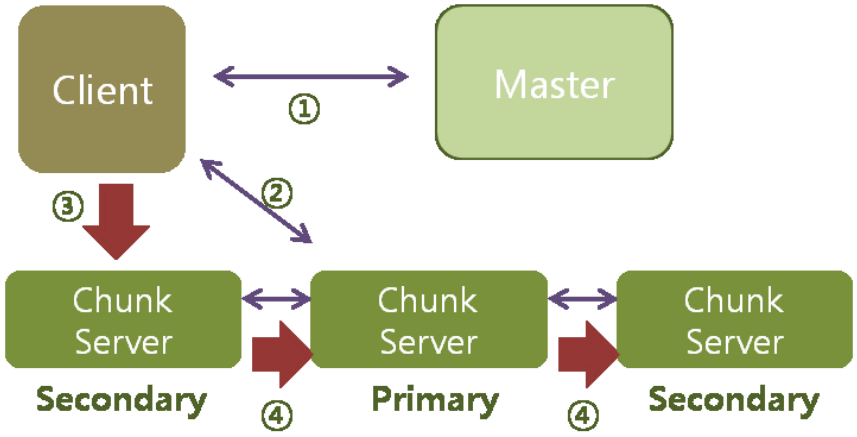


3. 배경(9/19)

- 읽기


Client
Master
①
②
③
Chunk Server
Chunk Server
Chunk Server

- 쓰기


Client
Master
①
②
③
Chunk Server
Chunk Server
Chunk Server
Secondary
Primary
Secondary
④
④

3.배경(10/19)

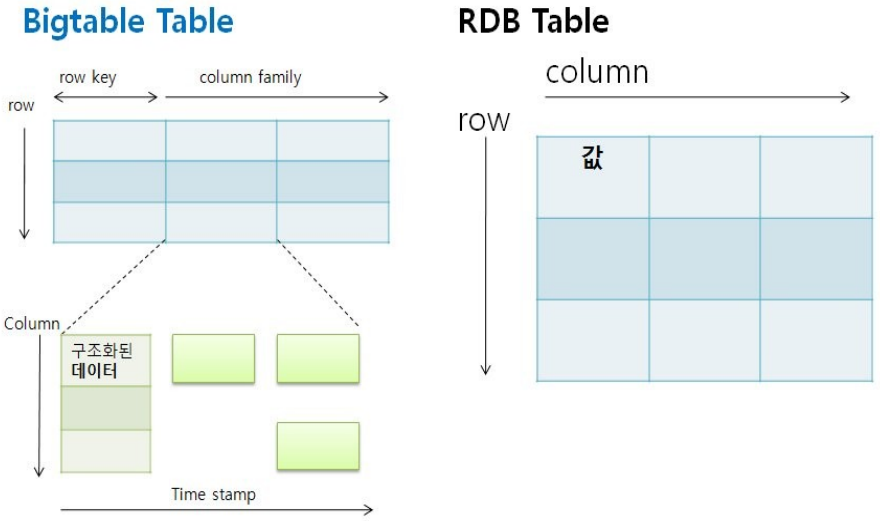
- 4.3.1.5 Problem & Solving
  - Disk 에러
    - Chunk에 대한 checksum 유지
  - Chunk Server 에러
    - Master가 관리대상에서 제외시키고, Chunk는 새로운 Server에 할당
  - Master 에러
    - 관리정보 갱신 시 Operation Log, Master 고장 시 Operation Log 읽고 재현

3. 배경(11/19)

- 3.3.2 BigTable(2006, Apache Hadoop/HBase, http://hadoop.apache.org/hbase/)
  - 수 천대 이상의 서버에서 페타(peta) 바이트에 달하는 매우 큰 용량의 구조화된 데이터(structured data)를 관리하기 위한 분산 저장 시스템(distributed storage system).
  - Google Analytics, Google Finance, Orkut, Personalized Search, Writely, Google Earth 등과 같은 60개 이상의 프로젝트에서 사용 중.

3. 배경(12/19)

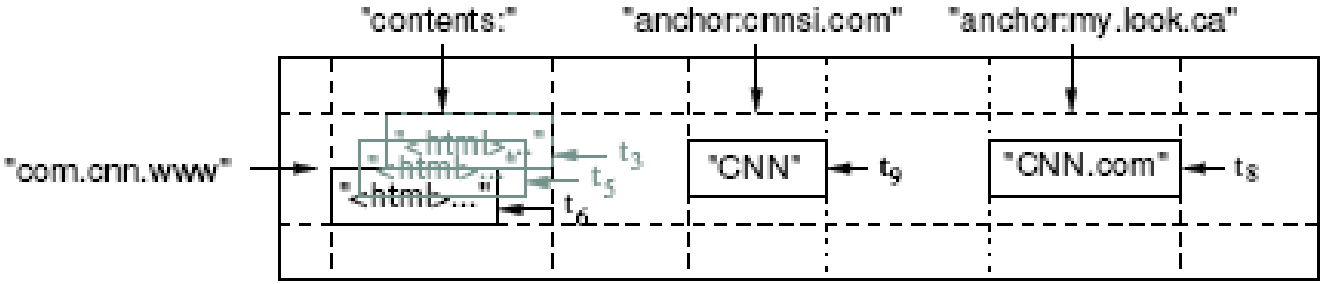
- 3.3.2.1 BigTable Table vs RDB Table



- 3.3.2.2 Data Model
  - sparse, distributed, multi dimensional sorted map.
  - indexed by a row key, column key, timestamp.
  - row key의 범위를 동적으로 분할하고, 분할된 범위는 tablet이라고 한다.

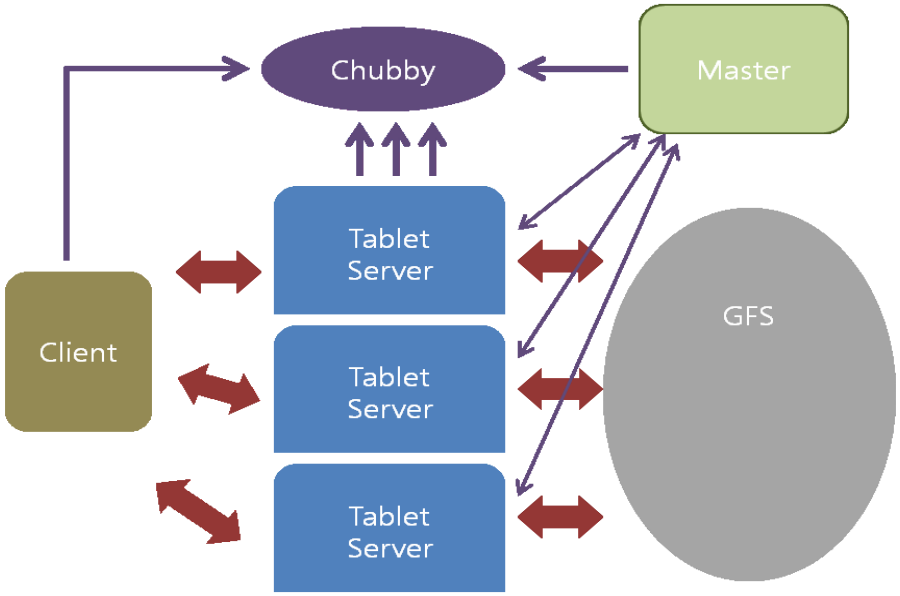
3. 배경(13/19)

- row key : 데이터의 읽기/쓰기의 기본적인 단위(최대 64KB의 임의의 문자열, 10~100 bytes 실제 데이터의 대부분). "com.cnn.www"
- column family : column의 논리적인 집합. "contents:", "anchor:"
- column : 실제 데이터의 저장 key. "anchor:cnnsi.com", "anchor:my.look.ca"
- timestamp : 64bit integer value. Master 서버의 Garbage Collection 기준으로 최근 3개의 데이터만 유지



3. 배경(14/19)

- 3.3.2.3 Architecture
  - Chubby : Tablet의 정보를 Client에게 제공, 분산 락(데이터의 깨짐 방지) 제공.
  - Master : Tablet을 Tablet 서버에 할당, 추가, 만료된 Tablet 제거 및 Garbage Collection 담당
  - Tablet Server : 할당된 Tablet 관리. 읽기, 쓰기에 대한 요청 처리.
  - Client : Chubby로부터 가져온, Tablet 서버에 대한 위치 캐싱 및 요청.



3. 배경(15/19)

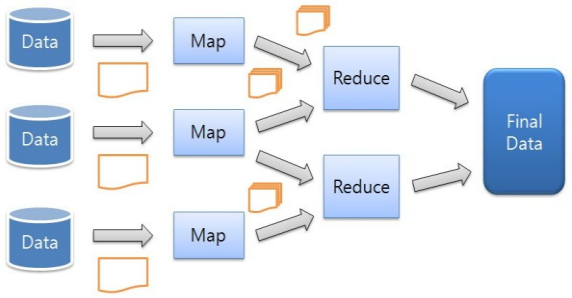
- 3.3.2.4 성능향상 기법
  - Locality Groups
    - 동시에 이용될 가능성이 큰 column family별로 묶음
  - Data compression
    - Locality group별로 묶인 데이터에 대한 자동 압축, 해제 기능
  - 읽기 캐싱
    - 가능한 Tablet Server의 메모리 상에 data 남겨 둠
  - Commit Log 일괄처리
  - 대용량 요청에 대한 일괄처리 기법

3. 배경(16/19)

- 3.3.3 MapReduce Framework(2004, Apache Hadoop/MapReduce, http://hadoop.apache.org/mapreduce/)
  - Map, Reduce는 Lisp과 같은 함수형 언어에서 유래한 용어로, GFS상에서 효율적인 데이터 처리를 위한 다수의 머신에서 병렬적으로 작업을 수행하게 하는 프로그래밍 모델.
  - Google의 경우, 웹 페이지 인덱스 생성하는 과정에서 방대한 양의 웹 페이지를 처리하기 위해 사용.
  - Map : 데이터 집합에서, 새로운 데이터를 생성하는 프로세스.
  - Reduce : Map에 의해 만들어진 데이터를 모아 최종적으로 원하는 결과로 만들어 내는 프로세스.

3. 배경(17/19)

- Ex) [1,2,3] 데이터를 샘플로..
  - map(2배, [1, 2, 3]) => [2, 4, 6]
  - reduce(덧셈, [2, 4, 6]) => **12**



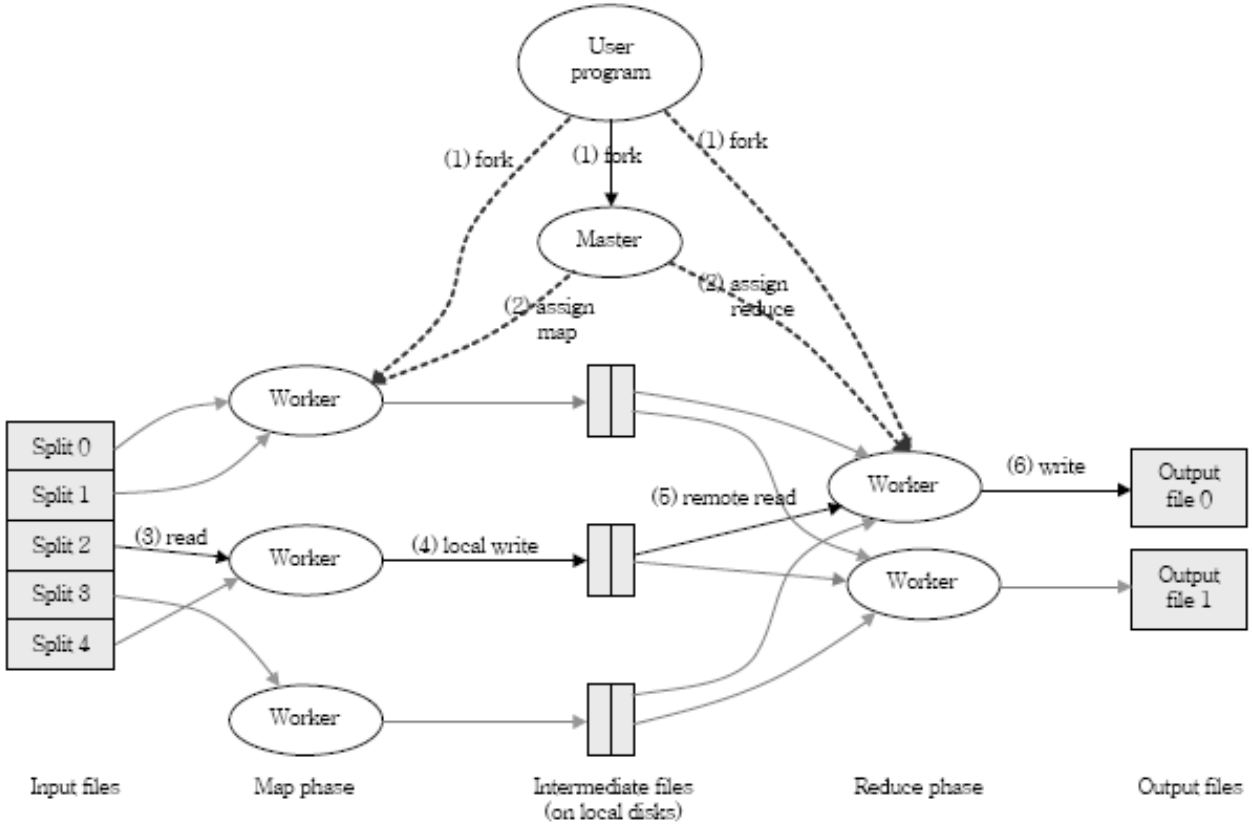
- Ex) Web log 데이터에 대한 map & reduce

```
212.122.174.13 - - [30/Oct/2009:18:14:32 +0100] "GET /foo HTTP/1.1" 200 18271
212.122.174.13 - - [30/Oct/2009:18:14:32 +0100] "GET /bar HTTP/1.1" 200 191726
212.122.174.13 - - [30/Oct/2009:18:14:32 +0100] "GET /baz HTTP/1.1" 200 198
74.119.8.111   - - [30/Oct/2009:18:14:32 +0100] "GET /egg HTTP/1.1" 200 43
74.119.8.111   - - [30/Oct/2009:18:14:32 +0100] "GET /moo HTTP/1.1" 200 91272
212.122.174.13 - - [30/Oct/2009:18:14:32 +0100] "GET /yay HTTP/1.1" 200 8371
```

```
212.122.174.13 210238
74.119.8.111   99643
```

3. 배경(18/19)

- 3.3.3.1 Architecture
  - Master : map, reduce 동작을 관리, Worker에게 작업을 분담.
  - Worker : map, reduce 작업을 수행.



3. 배경(19/19)

- 3.3.4 Sawzall(2005, Apache Hadoop/Pig, http://hadoop.apache.org/pig/)
  - 대규모 데이터에 대한 분석 처리를 위해서 개발한 interpreted 방식의 언어.
  - 외부 입/출력 데이터 형식 정의 - 입력 데이터에 대한 처리하는 기능 정의
  - 처리 결과를 중간 결과고 뽑아내는 기능(map)
  - 결과를 집계 처리하는 기능(reduce)

4.Data Model별 분류(1/3)

- 4.1 Relational
  - 관계로 데이터를 표시하는 DB
- 4.2 Key-Value
  - 키 기반의 get, put, delete 기능 제공
    - Amazon SimpleDB: **Misc**: not open source.
    - Azure Table Storage: Collections of free form entities (row key, partition key, timestamp). Blob and Queue Storage available, 3 times redundant. Accessible via REST or ATOM.
    - Chordless: API: **Java & simple RPC to vals**, Protocol: **internal**, Written in: **Java.**
    - Redis: API: **Tons of languages**, Written in: **C.**
    - Scalaris: Written in: **Erlang.**
    - Tokyo Cabinet / Tyrant:
    - GT.M: API: **C, Python, Perl**, Protocol: **native, inprocess C.**
    - Scalien: API / Protocol: **http** (text, html, JSON)**, C, C++, Python.**
    - Berkeley DB: API: **Many languages**, Written in: **C.**
    - Berkeley DB Java Edition: API: **Java**, Written in: **Java.**
    - MemcacheDB: API: Memcache protocol (get, set, add, replace, etc.), Written in: **C.**
    - HamsterDB:
    - NorthScale: based on Memcached.
    - Mnesia: (ErlangDB »).
    - LightCloud: (based on Tokyo Tyrant).
    - Pincaster: API: **HTTP/JSON**. Written in: **C**.
    - GenieDB:

4.Data Model별 분류(2/3)

- 4.3 column-oriented
  - 테이블 기반, 조인 미 지원, 컬럼기반(not like row-oriented databases)
    - Hadoop / HBase: API: **Java / any writer**, Protocol: **any write call**, Written in: **Java**
    - Cassandra: API: **Thrift** (Java, PHP, etc..), Protocol: **Thrift**, Written in: **Java**
    - Hypertable: API: **Thrift** (Java, PHP, Perl, Python, Ruby, etc.), Protocol: **Thrift**
    - Cloudera: Professional Software & Services based on Hadoop.
- 4.4 document-oriented
  - JSON, XML 형태의 구조적 문서 저장, 조인 미 지원
    - CouchDB: API: **JSON**, Protocol: **REST**, Written in: **Erlang**
    - MongoDB: API: **BSON(Binary JSON)**, Protocol: **lots of langs**, Written in : **C++**
    - Riak: API: **JSON**, Protocol: **REST**, Written in: **Erlang**
    - Terrastore: API: **Java & http**, Protocol: **http**, Written in : **Java**
    - OrientDB: Languages: **Java**, Written in: **Java**
    - RavenDB: .Net solution. Provides **HTTP/JSON** access. **LINQ** queries & **Sharding** supported. Misc:

4.Data Model별 분류(3/3)

- 4.5 graph database
  - 그래프로 데이터를 표현하는 DB
    - Neo4J: API: **lots of langs**, Protocol: **Java embedded / REST**, Written in: **Java.**
    - Sones: OO Graph DB, API: **.NET**, Protocol: **.NET embedded, REST, WebServices,** Written in: **C#**.
    - InfoGrid: API: **Java, http/REST**, Protocol: **as API + XPRISO, OpenID, RSS, Atom, JSON, Java embedded**, Written in: **Java.**
    - HyperGraphDB: API: **Java**, Written in:**Java**.
    - AllegroGraph: API: **Java, Python, Ruby, C#, Perl, Clojure, Lisp** Protocol: **REST**, Written in: **Common Lisp.**
    - Bigdata: API: **Java, Jini service discovery**, Written in: **Java**.
    - DEX: API: **Java**, Protocol: **Java Embedded**, Written in: Java / C++.
    - OpenLink Virtuoso: **Hybrid** DBMS covering the following models: **Relational, Document, Graph**.

5.Q&A

Q&A