**The Ultimate hands on Hadoop**

**Lecture 1-1**

**Hadoop 개요**

하둡은 방대한양의 데이터셋을 변환하고 분석하는데 강력한 툴이다.

하둡은 25개의 유틸리티와 연관되어있다.

Set up Hadoop

Virtual Box 설치, Virtual Box는 가상환경을 나의 PC에서 사용하게 해준다.

이 강의에선 Hortonworks vendor를 사용하지만 Cloudera 역시 유명한 vendor이다.

이 강의에선 Hortonworks Data Platform Sandbox 2.5를 사용한다.

Ambari

* Hadoop에선 Ambari를 이용해 Hadoop Cluster를 Visualize한다.
* Ambari를 활용하면 나의 Hadoop Ecosystem의 다양한 것들을 볼 수 있다.

Hive(Similar with Database)

* Hive를 이용해 데이터를 Hadoop에 집어넣는다.
* 이때 데이터는 Relational Database에 직접 저장되는 것이 아니라. Entire cluster에 distribute된다.
* Hive view에서 Data를 Handling 할 수 있다.
* e. g) Data upload : Upload Table, Query 작성 : Query

**Lecture 1-2**

**Hadoop Ecosystem**

What is Hadoop?

* Open source
* Software platform that runs on a cluster of computers so as opposed to just to running.
* Multiple PC를 이용하여 BigData를 다룰 수 있게 해준다.
* Distributed Storage(Main things that Hadoop provide).
* Map reduce = 분산되어져 저장되어 있는 정보를 위한 Hadoop의 Solution.

Why Hadoop?

* 데이터의 크기가 너무 커졌다 - terabytes per day

그러므로 Single PC가 다룰 수 없게되었다.

* Disks seek time = 하드웨어 성능의 한계
* Hardware failures = 각 종 사고 및 백업의 문제
* Processing times = CPU의 한계, Hadoop에 참여한 CPU들의 Parallel processing으로 극복
* Horizontal scaling is linear = 더 많은 data나 process, 더 나은 성능을 원할 시 PC를 섞거나 추가해주면 된다는 것을 의미.

**Lecture 1-3**

**Core Hadoop Ecosystem**

HDFS(Hadoop Distribute File System)

* HDFS는 GFS와 비슷하며 Hadoop version의 Files System을 의미한다.
* Cluster of Computers에 Big Data를 위한 Distribute the storage를 가능하게 해준다.

그럼으로써 모든 Hard Drive는 Cluster에서 거대한 File System처럼 보이게 된다.

또한 중복된 Data를 가짐으로서 다양한 Issue에 대응한다. e. g)자연재해, Shut down…

YARN(Yet Another Resource Negotiator)

* YARN은 Data Processing에 시작점이다.
* 기본적으로 Computing cluster 자원들을 관리한다.
* -e. g) 어떤 데이터를 받을지, 어떤 task를 실행할지, 어떤 node가 사용가능한지 아닌지 등..

Map Reduce

* Entire Cluster에 Programming model이 data를 process하게 해준다.
* Mapper와 Reducer로 구성되어져 있다.

Pig

* JAVA and Python Language를 사용하지 않고 좀더 친숙한 SQL style syntax를 가진다.
* Map Reduce 위에서 작동한다.

HIVE

* Pig와 비슷하지만 SQL database와 더욱 비슷하다.
* SQL Query를 사용하며, HDFS를 SQL Database처럼 보게 해준다.
* 하지만 Relational Database는 아니다.

Apache Ambari

* Ambari는 실제적으로 Hadoop Ecosystem의 가장 상위에서 동작한다.
* 모든 것[e. g) State of Nodes, HBase …] 들에 대한View를 제공한다.

Cluster의 상태를 View안에서 확인할 수 있다.

* Hortonworks사용 시 사용가능.

Mesos

* YARN의 대안책.
* Managing Resources of our cluster.
* YARN과 함께 사용가능하다.

Spark

* Map reduce와 같은 층에서 동작.
* Data를 process할 시 빠르고 효율적이며 믿을 만하다.
* SQL query를 다룰 수 있으며 Machine learning을 Cluster에서 가능케 해준다.

또한 Streaming data를 실시간으로 다루기 좋다.

HBase

* Transactional Platform이다. 즉, 데이터의 추출, 입력, 수정 등을 담당한다.
* NoSQL database이다.

매우 많은 트랜잭션을 다루는데 매우 빠른 데이터베이스이며, Spark나 Map Reduce로 Data를 transform시 다른 Cluster에 빠르게 expose해준다.

Apache Storm

* Streaming data를 processing하기 위한 기초적인 방법이다.
* Storm을 활용하여 Streaming data를 real-time으로 받아올 수 있다.

Oozie

* Cluster의 Job Scheduling하는 것 중 하나이다.
* 많은 다른 시스템들이 각 Task를 Hadoop cluster에서 수행할 경우 필요하다.

Ref) Window의 Job Scheduling과 유사

Task process를 해야 할 경우 Scheduling을 통해 Sorting하여 수행할 필요가 있음.

Zookeeper

* Cluster를 조직화 한다.
* Cluster 안에 있는 Nodes를 tracking하고 각 node들의 상태들을 공유한다.

Sqoop

* HDFS를 ODBC, JDBC 같은 형태로 Transform 해준다.
* 즉 Legacy Databases와 Hadoop 사이의 Connector 역할을 한다.

Flume

* 방대한 Web logs를 Transport 해준다.
* Web logs를 Real-time으로 받아들이고, Real-time으로 Cluster에 배포한다.

**External Data Storage**

MySQL

Cassandra

Mongo DB

**Query Engines**

Drill

* Hbase, Cassandra, MongoDB에서 사용가능한 Query

Hue

Phoenix

Presto

Zepplin

**Lecture 2-1**

**HDFS(Hadoop Distributed File System)**

* 하둡의 가장 아랫층을 담당한다.
* 빅데이터가 전체 클러스터에 분할되어 저장하게 해준다.
* 또한 Application이 분석하기 위해 분할된 데이터에 빠르게 Access 하게 해준다.
* Handling Big Data
* By breaking them into blocks
  + Blocks은 128MB가 기본값이다.
  + Blocks을 나누어 저장하므로 Big Data를 효율적으로 활용 가능하다.
* Stored across several commodity computers
  + Blocks을 나누어 복제하여 나누어 저장함으로써 다양한 Problems에 대응한다.
* HDFS Architecture
  + Name node : 모든 Data nodes를 track하여 기록하고 유지한다.
  + Data node : 실제로 모든 Data와 File을 저장한다.
* Reading a File

1. Client Node에서 Name Node에 필요한 Data를 요청.
2. Name Node에서 Blocks의 위치와 Data nodes정보를 응답.
3. Data node에 Blocks을 요청한다.

* Writing a File

1. Client Node에서 Name Node에 새로운 Data Writing을 요청.
2. Name node에서 새로운 Entry생성하여 Client Node에 응답.
3. Data Node에 새로운 Data 작성.
4. 활성화된 Nata Node끼리 Writing된 Data 공유
5. Data Nodes에 Data가 저장된 후 Client Node로 저장정보 전송.
6. Client Node는 Name Node에 Data Node에 저장된 정보 전송.

* Name Node Resilience
  + Back Up metadata
  + Secondary Namenode
  + HDFS Federation
    - Each namenode manages a specific namespace volume
* Zookeeper : Zookeeper를 사용하여 Name node가 Shut down되는 상황에 대응할 수 있다.
  + Zookeeper는 Active namenode를 track한다.

**Lecture 2-3**

Ambari를 사용하여 Web UI처럼 HDFS를 사용할 수 있다.

Putty를 사용하여 Command Prompt처럼 HDFS를 사용할 수 있다.

e. g) maria\_dev@127.0.0.1 (Host Name, IP address), Port Number is used ‘2222’ on this particular sandbox.

**Command Prompt 명령어**

Hadoop fs -ls : HDFS안에 List를 확인할 때 사용하는 명령어

Hadoop fs – mkdir : HDFS안에 디렉토리를 만들 때 사용한다.

Wget http:// ~~ : http Server에서 파일을 다운로드할 때 사용한다.

Hadoop fs -copyFromLocal : Local에서부터 HDFS로 파일 복사 시 사용한다.

Hadoop fs -rm : Hadoop 파일 삭제 명령어

Hadoop fs -rm dir : 디렉토리 삭제 명령어

Hadoop fs : Hadoop 명령어 List 조회

Python ~~.py -r Hadoop –hadoop-straming-jar /usr/h에/current/Hadoop-mapreduce-client/Hadoop-streaming.jar 저장할파일위치: Python파일 실행 후 Hadoop에서 삭제하기.

**Lecture 2-4**

**Conceptual MapReduce**

* 클러스터 안에서 데이터 분할 processing 제공
* Data를 Mapper and Reducer 기능으로 Mapped(transformed) and Reduced(aggregated) 한다.
* Mapper

1. Mapper는 Raw source data를 Key/value 쌍으로 변환한다,

e. g) K1:V K2:V K3:V, K1:V, K1:V

1. Key로 데이터를 추출하고 정의한다.
2. 전체 클러스터에 Spreading 되어있는 Data를 Key값을 활용해 Value를 추출한다.

* Reducer

1. 우리가 원하는 Output 값을 만들어낸다.
2. 즉 Mapper가 추출하고 Shuffle and Sort 과정을 거친 후 Reducer가 우리가 원하는 값을 만들어 내는 것이다.

**Lecture 2-5**

**MapReduce**

* 각각 다른 파티션에 있는 Data를 Mapper가 각각 추출한 후 Shuffle and Sort Operation에서 합친 후 Reducer가 Output값을 Return한다.
* MapReduce processing

1. Client Node에서 YARN Resource Manager(작동하고 있는 Machine Manage 즉, Node들의 이용상태를 track한다.)로 Message전달
2. NodeManager(MapReduce Application Master: Node가 하고있는 일 or 실제 수행중인 stuff 즉, 현재 Node의 동작상태에 대한 관리를 한다.)가 어떤 Data에 대하여 mapping and reducing 작업 수행 시 HDFS cluster로 실제적으로 Data를 process and output the data results을 읽기 및 출력한다.

* For about mappers and reducers

1. MapReduce는 자바언어로 작성되었다.
2. 다른 언어로의 interfacing이 가능하다. 즉 Java로 작성되었지만 Python or C등의 Language도 사용이 가능하다는 이야기.

* Handling Failure

1. Application master가 errors or hanging을 위해 worker tasks를 모니터한다.

Error를 발견하면 필요하다면 Restart를 하고 가능하다면 다른 Node를 찾는다.

1. Application master가 down된 경우

YARN이 Restart를 시도할 것이다.

1. 전체 Node가 down된 경우

Resource manager가 Restart를 시도할 것이다.

1. Resource Manager가 down된 경우(가장 끔찍한 경우)

이런 경우를 위해 Zookeeper에서 제공하는 High Availability(HA)를 설정한다.

HA를 설정한 경우 down 되었을 시에 Second Resource Manager로 Zookeeper가 Redirect할 것이다.

**Lecture 5-1**

For about Hive

On the Hive view we can use the SQL Queries.

**Lecture 5-2**

**How Hive works**

* Hive maintains a “metastore” that imparts a structure you define on the unstructured data that is stored on HDFS etc.

e. g) Prompt에서 Hive 코드 작성법

CREATE TABLE ratings (

UserID INT,

movieID INT,

ratings INT,

time INT)

ROW FORMAT DELIMTED ###행의 형태 정의

FIELDS TERMINATED BY ‘\T’ ###

STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH … OVERWRITE INTO TABLE ratings; ##Overwrite란 덮어쓰는 것을 의미한다.

Ref) LOAD DATA : 분할된 파일시스템으로부터 데이터를 Hive로 이동한다.

LOAD DATA LOCAL : 로컬 파일시스템으로부터 데이터를 Hive로 복사한다.

* Partitioning

1. We can store our data in partitioned subdirectories
   * + Huge optimization if your queries are only on certain partitions.

e. g)

CREATE TABLE customers (

Name STRING,

Address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>

)

PARTITIONED BY (country STRING);

…/customers/country=CA/

…/customers/country=GB/

* Ways to use Hive
  + Interactive via hive> prompt / Command line interface (CLI)
  + Saved query files like “hive -f /somepath/queries.hql
  + Through Ambari / Hue
  + Through JDBC/ODBC server
  + Through Thrift service but we need to remember, Hive is not suitable for OLTP
  + Via Oozie

**Lecture 5-6**

**What’s MySQL?**

* 인기있는 free relational database
* MySQL은 OLTP, online transaction processing에 적절하다.

**Sqoop**

* Data를 추출 입력을 다루기 위해 MapReduce가 실제로 시작한다.
* Export data from Hive to My SQL
  + Sqoop import –connect jdbc:mysql://localhost/movielens –drivercom.mysql.jdbc.Driver –table movies

: Data를 MySQL에서 HDFS로 옮기고 싶을 때 사용하는 구문

: --hive-import

* Incremental imports
  + Sqoop을 이용하여 Relational database와 Hadoop의 Sync를 유지할 수 있다.
  + --check-column
  + -–last-value
* Export data from Hive to My SQL
  + Sqoop export –connect jdbc:mysql://localhost/movielens -m 1 –drivercom.mysql.jdbc.Driver –table exported\_movies –export-dir/apps/hive/warehouse/movies –input-fields-terminated-by ‘\0001’
  + Target table은 MySQL안에 반드시 이미 존재하여야 하며, columns들은 정렬되어 있는 것이 좋다.

**Lecture 5-7**

**Activity import movie data**

* SET NAMES ‘utf8’ : NAMES를 utf8 type으로 설정
* SET CHARACTER ‘utf8’ : CHARACTER를 utf8 type으로 설정
* use tablename : SET한 설정을 table에 적용한다
* source movielens.sql; : SET한 설정을 table에 적용하는 source 실행

**Lecture 5-8**

**Activity Import Data from MySQL to Hadoop**

* GRANT ALL PRIVILEGES ON movielens.\* to ‘’@’localhost’;

: localhost에 있는 movielens database에 대한 모든 권한(For connecting to localhost)을 준다.

* Sqoop import –connect jdbc:mysql://localhost/movielens –driver com.mysql.jdbc.Driver –table movies -m 1

Interpret code

Sqoop import –connect jdbc:mysql://localhost/movielens

: import한다. jdbc:mysql을 이용하여 localhost의 movielens에 연결

–driver com.mysql.jdbc.Driver –table movies

: Database와 Message Drive를 위해 선언하는 구문

-m 1

: 오직 하나의 mapper만 실행하기를 원한다.

Reason : localhost에 연결했기 때문에 오직 하나의 mapper를 가짐. 만약 Cluster의 연결한다면 “-m”만 선언해주면 된다.

**Lecture 6-1**

**Why NoSQL?**

* 많은 양의 데이터를 가지고 있을 때, Relational Database로 처리 불가능할 시에 사용한다.
* 많은 양의 데이터를 처리 할 시에 사용하는 방법.
  + 극도로 많은 양의 수치들을 Relational Database Management System으로 지속적으로 처리 -> 이 방법은 Hadoop과 NoSQL전에 사용하였다.
  + Relational Database는 Normalization 된 Data를 요구하지만 현시대의 대다수의 Data는 Denormalization되어 있다.
  + Cashing layer : Cash Layer에서 Format에 적합한 Data를 걸러서 Database에 저장하는방법.
  + Master/slave setups : Slave에 Database를 두고 Master로 Handling한다.
  + Sharding : Adding more Hard Drive.
  + Meterialized : Serices가 원하는 format으로 정렬한 후 Database에 저장하는 형태.

**Use the right tool for the job**

* For analytic queries, Hive, Pig, Spark, etc. 가 일하기에 좋다. 또한 저런 것들은 HBase or 다른 NoSQL윗층에서 작동가능하다.
* MySQL에서 데이터를 다수의 Application에서 추출하는 것은 plenty하고 fast하다.

하지만 만약 거대한 Scale에서, Web applications에서 작동할 때에는 Non-realational database를 사용하는게 빠르고 scalable serving하기 좋다.

**Lecture 6-2**

**HBase**

* Hadoop cluster의 구성요소 중 하나이다.
* Non-relational, scalable database built on HDFS
* HBase is Google Big Table concept. It is just different name for opensource.
* 별도의 Query language를 가지고 있지 않다.
* HBase의 모든 것은 API가 가지고 있다. 또한 API는 CRUD(Create ,Read, Update, Delete) operations라고 불리는 것을 할 수 있다.
* HBase architecture
  + Split Region Server는 HBase의 핵심이다. 그리고 Regions을 이야기할 때 위치는 이야기하지않는다. 왜냐하면 Automatically 하게 분할되어 적용되기 때문이다. 그러므로 데이터가 늘어나게 되면, Repartition을 수행하고 만약 Server가 추가된다면 그 또한 자동으로 Runtime을 다룬다.
  + 즉, 자동으로 parallel 한 Region Server에 Data가 자동으로 분할되어 속하게 해준다는 것을 알면 된다.
  + Key는 분할되어 각각 다른 Region server에 속하게 된다.
  + 실제 데이터는 HDFS안에 저장된다.
  + HMaster = Master node

: 자료가 저장되어 있는 위치 등을 관리한다.

* + Zookeeper

: 현재 동작중인 Resources를 관리한다.

* HBase data model
  + 행의 개념을 지니고 있다.
  + 각 행은 식별할 수 있는 고유의 Key(Similar with primary Key at the RDB)와 Value를 가지고 있다.
  + Column을 Relational Database처럼 굳이 정의할 필요가 없다. 대신에 column families를 정의하고, 각 column family는 많은 수의 column을 contain 하고 있다.
  + Column이 Sparse Data를 가지고 있어도 상관없다.
  + 각 CELL은 Timestamp와 함께 많은 Version을 가질 수 있다.
  + Version은 각 CELL의 History라고 생각하면 된다.
* Some ways to access HBase
  + HBase shell
  + JavaAPI – Wrappers for Python, scala, etc.
  + Spark, Hive, Pig
  + REST service
  + Thrift service
  + Avro service

**Lecture 6-3**

HBase는 Row 지향적이다.

HBase Running

e. g) /usr/h에/current/hbase-master/bin/hbase-daemon.sh start rest -p 8000 –infoport 801

REST server를 HBase 위에 올려 구동하는 구문

* Starbase
  + Python에서 사용
  + HBase를 위한 REST client

**Lecture 6-4**

**Integrating Pig with HBase**

* HBase 저장시에는 반드시 Unique Key를 설정하고 Column Family를 만든 후 Key와 value를 설정하여 수행한다.
* “hbase shell’ : Interactive hbase shell
  + “create ‘user’, ‘userinfo’” : ‘user’ 테이블을 생성하고 ‘userinfo’라는 column family를 생성한다.
  + Drop ‘tablename’ : table 삭제 구문, HBase shell 안에서 사용한다
* For about Pig file
  + Ratings = LOAD ‘\ …’ ##LOAD를 통해 local에 있는 FILE 불러온다.

USING PigStorage(‘|’) ##PigStorage를 통해 File의 determinant 형태 구분

As (userID:int, age:int, gender:chararray, occupation:chararray, zip:int);

## 파일의 형식 선언

STORE ratings INTO ‘hbase://users’ ##HBase 안에 저장

USING org.apache.pig.backend.hadoop.hbase.HBaseSotrage (

‘userinfo:age, userinfo:gender, userinfo:occupation, userinfo:zip’);

##Column Family 설정 구문

**Lecture 6-5**

**Cassandra Overview**

* HBase와 달리, Single point of failure, Master node가 존재하지 않는다.

즉, Every node는 같은 Software에서 동작하고 같은 functions을 수행한다.

* Data model은 Big Table / HBase와 비슷하다.
* Non-relational 이다. 그러나 Interface로써 CQL query language한계를 가진다.
* CAP(Consistency, Availability, Partition, Tolerance) theorem
* **C**onsistency : Singlecopy ACID consistency를 의미한다.  
  **A**vailability : 가용성  
  **P**artition Tolerance : Partition이 나누어져 있음에도 System이 살아있음을 의미
* CA without P: 분산 트랜잭션 데이터베이스는 네트워크 파티션이 없는 경우 수행할 수 있다.
* CP without A: 파티션 상황에서 파티션이 복구되기 전까지는 ACID 데이터베이스의 트랜잭션은 중단된다. 병합충돌을 피하기 위해서이다. 일관성을 피하기 위함이다.
* AP without C: HTTP Web caching 은 문서 복제를 통해 클라이언트-서버 파티션 복원을 제공한다. 그러나 클라이언트-서버 파티션은 만료된 복제에 대해 최신데이터인지 검증을 못하게 한다. 일반적으로 분산 데이터베이스 문제는 만료 기반의 캐시를 사용해서 AP 를 실현하거나 복제본과 다수결을 통해 PC를 얻는다. 소수는 안된다. (PC를 얻는게 아니고 CP를 얻는다 일까?)
  + Cassandra는 Availability와 Partition Tolerance를 중요시하고 Consistency를 어느정도 감수할 수 있을 때 사용한다.
  + HBase, mongoDB는 Partition Tolerance and Consistency를 중요시한다.
  + MySQL은 Availability and Consistency를 중요시한다.
* Cassandra architecture
  + Cassandra는 Same time에 Same Platform에서 Same Function을 수행한다.
  + Cassandra의 Node들은 서로 Communicate하며 어떤 Node가 가장 빠른 지 등의 Information을 서로 공유하고 Track 한다. 이 후 결과값을 Back 해준다.
* Cassandra는 rows of information을 빠르게 Access하기에 좋다.
* Cassandra는 Analytics와 Spark integration을 사용하기는 서로 다른 두개의 링을 자기복제 한다.
* CQL(Cassandra Query Language)
  + Cassandra의 API는 CQL이며, CQL은 Applications에서 존재하는 Database drivers를 보기 쉽게 만들어 준다.
  + CQL은 SQL같지만 큰 Limitation을 지니고 있다.

e. g) NO JOIN : Data를 normalization 할 수 없다. 그러므로 Data는 여전히 non-relational 이다

모든 쿼리는 몇가지의 primary key를 가져야만 한다.

* CQLSH는 command line에서 create tables, etc… 등의 작업에 사용되어 질 수 있다.
* 모든 Table은 Keyspace(=Databases)안에 있어야 한다. 즉, Keyspace consist of tables.
* Cassandra and Spark
  + DataStax는 Spark-Cassandra connector를 제공한다.
  + Cassandra tables을 DataFrames으로서 읽고 쓰는 것을 가능하게 해준다.
  + 적절한 수준으로 데이터프레임의 Queries를 passing하는데 좋다.
  + Use cases:
    - Cassandra에 저장된 Data를 Analytics를 위해 Spark를 사용한다.
    - Spark를 사용하여 Data를 변환하고 Cassandra에 저장한다.

**Lecture 6-8**

**MongoDB Overview**

* MongoDB는 document개념이라고 생각하면 된다.
* MongoDB는 Single primary Database, Single Master Node를 가진다.(HBase와 일치)
* JSON blob of data를 MongoDB 안의 document로 저장 가능하다.
* No real schema enforced.
  + 원한다면 서로 다른 fields를 Every document에 가질 수 있다.
  + No single “key” 그러나 원한다면 any fields이던지 indices를 지정 가능하다.
  + 만약 ‘shard’를 원한다면 indices를 shard하면 된다.
* MongoDB terminology
  + Databases
  + Collections
  + Documents
  + MongoDB는 Databases, tables and row대신에 Databases, Collections and Documents를 사용한다.
* Replication Sets
  + MongoDb is a single-master.
  + Database instance 복제본을 백업하여 유지한다.
* Replica Set Quirks
  + 서버들의 대다수는 반드시 primary에 동의하여야 한다. 그러므로 많은 server를 가질 수 없다.
  + ‘arbiter’ node를 설정하여 실패 event 발생시 누가 primary node가 될 것인지 투표한다. 그러나 하나의 Cluster안에서 오직 하나의 ‘arbiter’ node만을 가질 수 있다.
  + Apps must know about enough servers in the replica set to be able to reach one to learn who’s primary
  + Replicas only address durability, not your ability to scale
    - But which generally isn’t recommended
    - And your DB will still go into ready-only mode for a bit while a new primary is elected.
  + Dumb을 이용할 수 있다. Secondaries가 딜레이 될 때.
* Sharding
  + App Server Process는 mongos를 통해 Primary에 접근, Primary는 Index(=Unique Key)를 이용하여 Secondary에 접근 Secondary들은 Key를 통해 Value를 추출하여 App Server Process로 Key and Value Return.
  + 또한 mongos는 Config Server를 가진다. Config Server(=Master node)는 Data가 어떻게 Partition 되어있는지에 대한 Information을 지니고 있다
  + Mongos는 running일 때에 balancer를 Background에 실행 시키는데 balancer는 만약 실제로 value를 가져오지 못할 수도 있기 때문에 실시간으로 replica sets를 rebalance하여 value를 가져올 수 있게 해준다.
* Sharding Quirks
  + Auto-sharding은 때때로 동작하지 않을 수 있다.
    - Split storms and mongos process는 매우 자주 restart된다.
  + 3 config server를 반드시 가져야만 한다.
    - 만약 하나라도 down되면 DB는 다운된다.
    - Config server는 replica sets의 single-master design 위에 있다.
  + MongoDB’s loose document model은 효율적인 sharding엔 맞지 않을 수 있다.
  + Master node는 Zookeeper에 의해 For Availability 관리된다. (It is same process with HBase)
* Neat Things About MongoDB
  + 단지 NoSQL Database가 아닌 매우 유연한 document model이다.
  + Shell은 모두 JavaScript interpreter이다.
  + 많은 indices를 지원한다.
    - 그러나 오직 하나만 sharding을 위해 사용될 수 있다.
    - 2-3개 이상은 여전히 지원하지 않는다.
    - Full-text indices for text searches – MongoDB는 big documents of information을 저장하기에 좋은 선택이다.
  + SQL connector를 이용 가능하다.
    - 그러나 MongoDB는 여전히 join이나 normalized data를 위해 디자인 되어지지는 않았다.

**Lecture 6-12**

**Choosing Database**

1. Scaling requirements
   * Transaction rate : how many request do we intend to get per second?
     + e. g) The Data Comming from Web Server, Web Site
2. Support considerations
   * 보안 관련해서는 MongoDB가 가장 낫다.
3. We don’t need to consider Budget considerations.
4. CAP considerations
5. Simplicity
   1. Think about the minimum requirements that we need for our system and keep it as simple as possible.

Something like external server that maintain its configuration.

1. An example
   1. We’re building an internal phone directory app
      * Scale: limited
      * Consistency: Eventual is fine
      * Availability requirements: not mission critical
      * MySQL is probably already installed on our web server…
      * The Good choice is “MySQL”

Reason)

Already that is installed on the phone.

We don’t need Partition-Tolerance because the scale is limited.

* 1. We want to mine web server logs for interesting patterns. What are the most popular times of day? What’s the average session length?
     + Scale: 매우 크다.
     + HDFS시스템으로 web logs를 저장한다. 후에 Spark, Hive and Pig 등을 이용하여 Data를 분석한다.
     + 이런 경우엔 굳이 Database가 필요하지 않다.
  2. We have a Big Spark job that produces movie recommendations for end users nightly. Something needs to vend this data to our web applications. We work for some huge company with massive scale. Downtime is not tolerated. Must be fast. Eventual consistency OK.
     + Solution : Cassandra

Reason)

We don’t care about the Consistency. But We need to handl