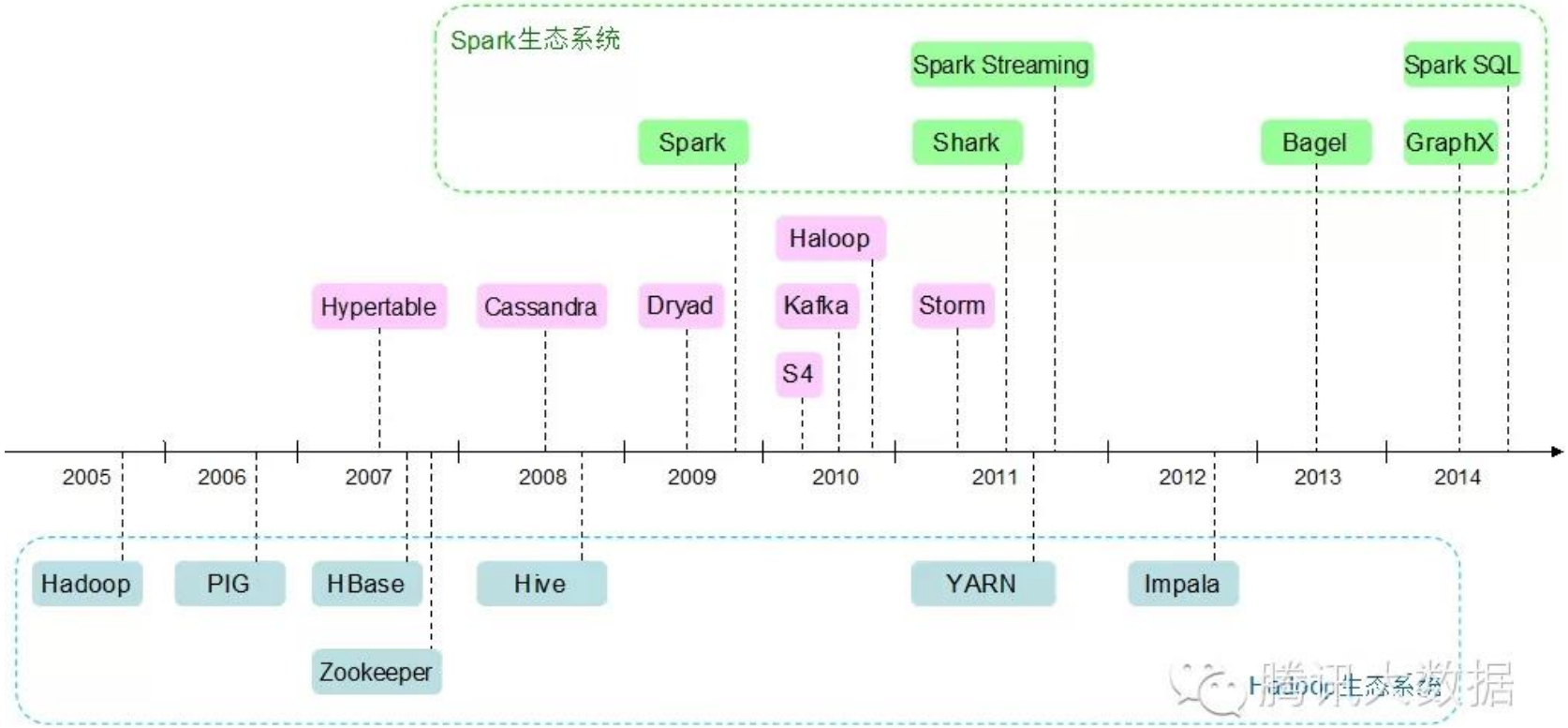
Hadoop/MapReduce 建置與開發實務

國家高速網路中心
莊家雋

Outline

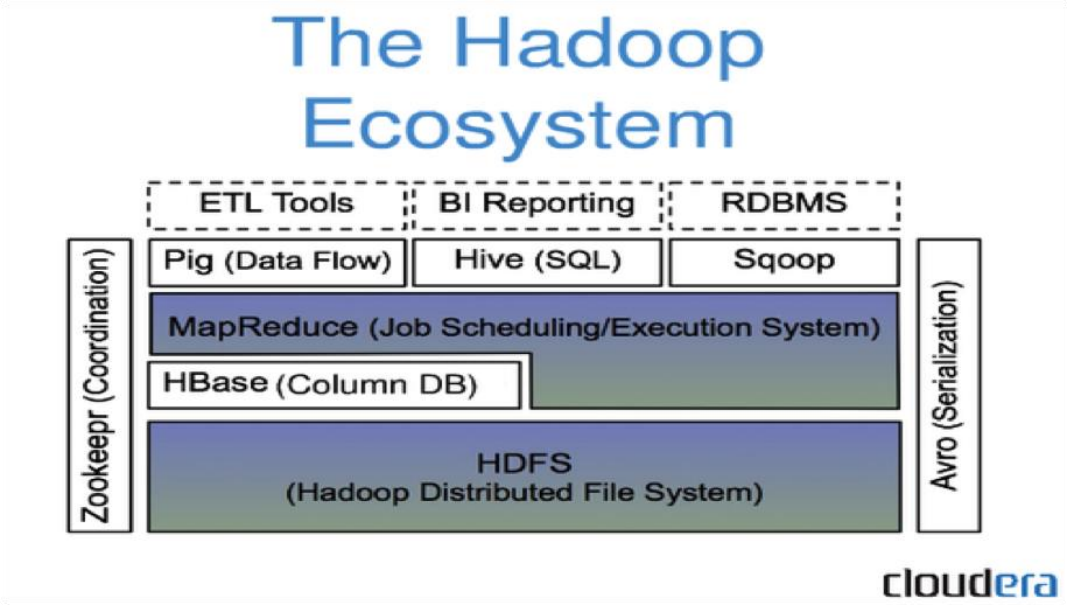
- Hadoop ecosystem 簡介
  - Intro, Version, distribution, OS base installation
- HDFS
  - Intro, install & configure,UI & CLI, Java programing
- Mapreduce
  - YARN, Intro, install & configure, UI,Java programing
- Hbase
  - Intro, install & configure , UI & CLI, Java programing

Bigdata platform evolution



http://www.dataguru.cn/article-6920-1.html

What is Hadoop ecosystem

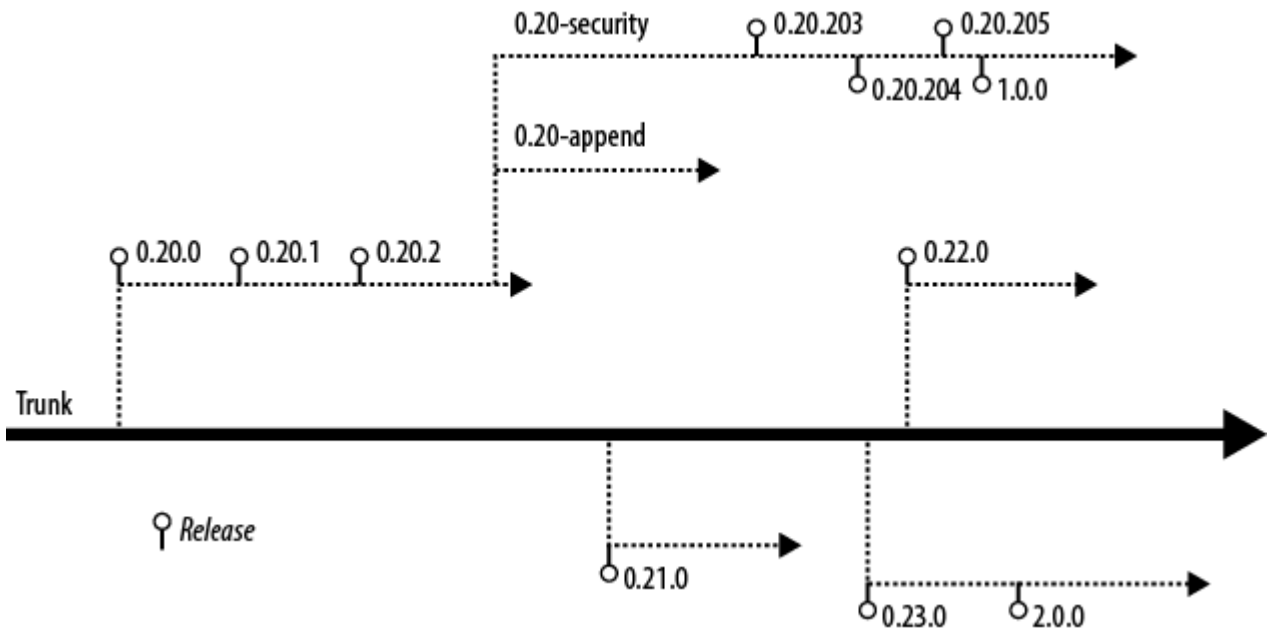


Google	OpenSource
GFS	HDFS
MapReduce	Hadoop MapReduce
BigTable	HBase
Chubby	Zookeeper

Hadoop distribution

- Apache
  - Hadoop 2.7.1 released at 2015/7/6
  - HBase 1.0 released at 2015/2/22
- Cloudera: CDH
  - 今天採用CDH 5.3.2
    - Hadoop-2.5.0-cdh5.3.2
    - Hbase-0.98.6-cdh5.3.2
  - 最新CDH 5.4.4
    - Hadoop-2.6.0-cdh5.4.4
    - HBase-1.0.0-cdh5.4.4
- Hortonworks: HDP
- MapR

Hadoop version



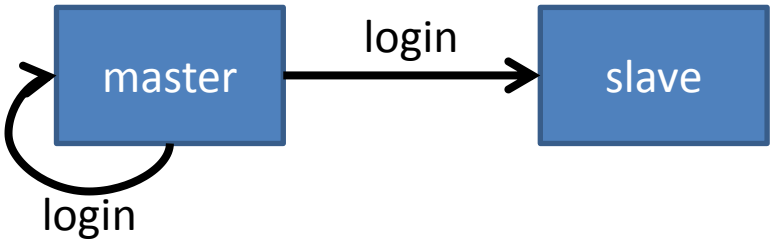
https://www.safaribooksonline.com/library/view/hadoop-operations/9781449327279/ch04.html

Hadoop version feature

Feature	0.2	0.21	0.22	0.23	1	2	CDH3	CDH4
Production quality	X				X		X	X
HDFS append		X	X	X	X	X	X	X
Kerberos security		X[a]	X	X	X	X	X	X
HDFS symlinks		X	X	X		X		X
YARN (MRv2)				X		X		X
MRv1 daemons[b]	X	X	X		X		X	X
Namenode federation				X		X		X
Namenode HA				X		X		X

OS base installation

- 使用 hostname設定主機名稱
  - master / slave
  - 修改 /etc/hosts
- Install JDK (orcale JDK or open JDK)
  - 在.bashrc
  - export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386



1. 在master產生 公鑰(id_rsa.pub) 與 私鑰(id_rsa)
   $ ssh-keygen –t rsa
2. 將公鑰由master送到**master**與**所有slave**上並公開
   hadoop@master $ scp id_rsa.pub hadoop@slave:/home/hadoop/
   hadoop@slave $ cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
   hadoop@slave$ chmod 600 ~/.ssh/authorized_keys

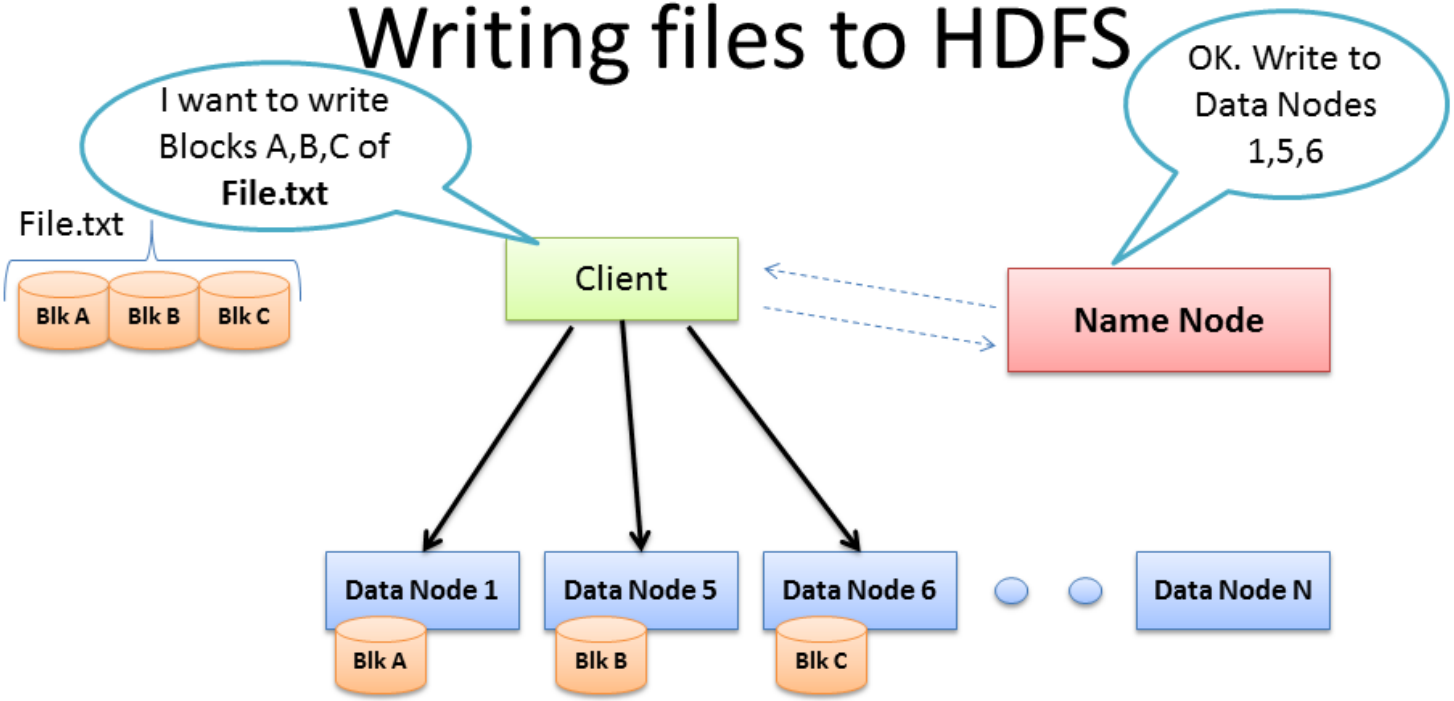
Outline

- Hadoop ecosystem 簡介
  - Intro, Version, distribution, OS base installation
- HDFS
  - Intro, install & configure,CLI, Java programing
- Mapreduce
  - YARN, Intro, install & configure, Java programing
- Hbase
  - Intro, install & configure , CLI, Java programing

HDFS intro

- Master-slave architecture
  - 1 master and MANY slaves
- Master host 上運行NameNode
  - Single point failure of NameNode
- Slave host 上運行 DataNode

Writing files to HDFS

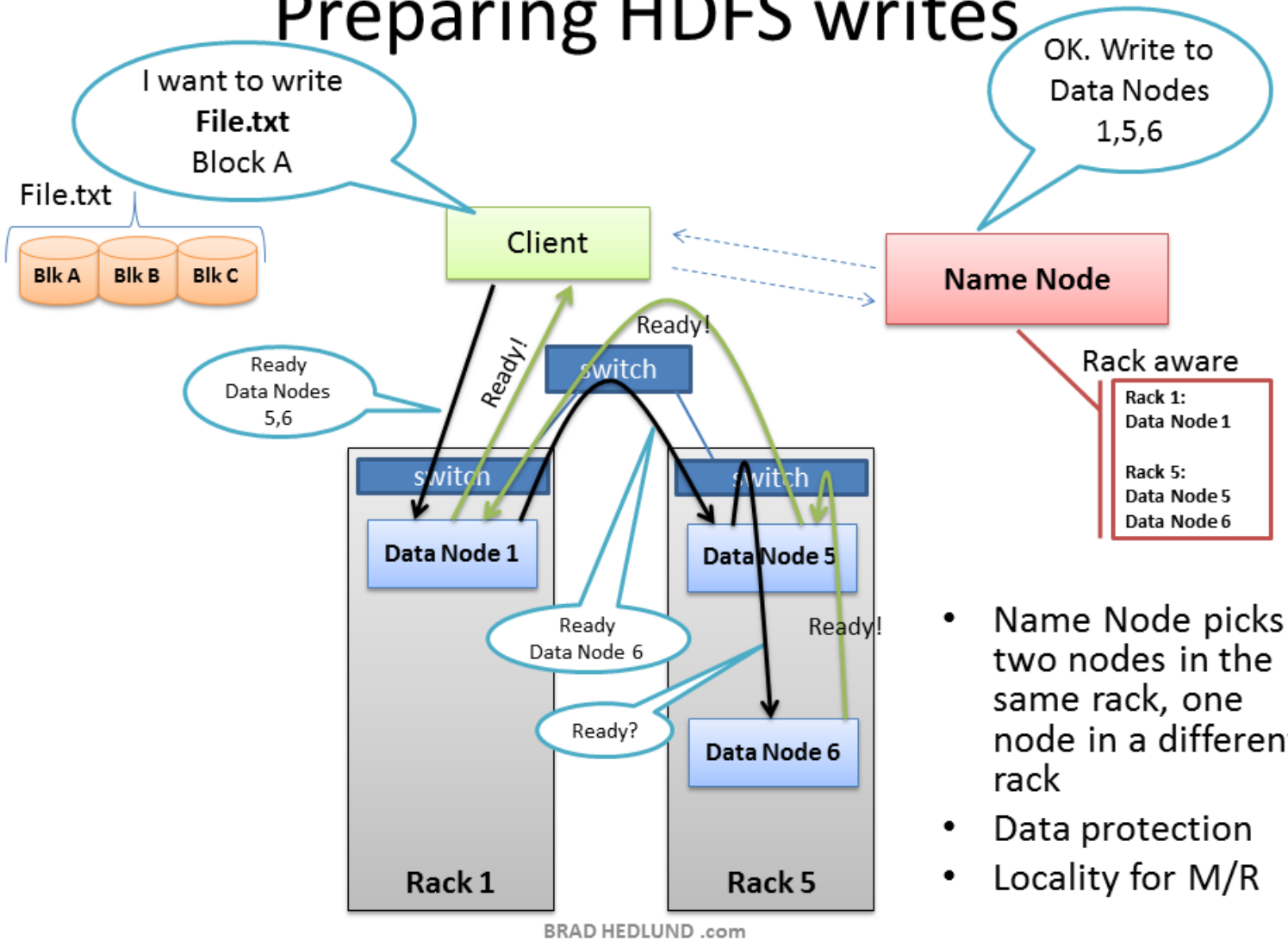


- Client consults Name Node
- Client writes block directly to one Data Node
- Data Nodes replicates block
- Cycle repeats for next block

BRAD HEDLUND .com

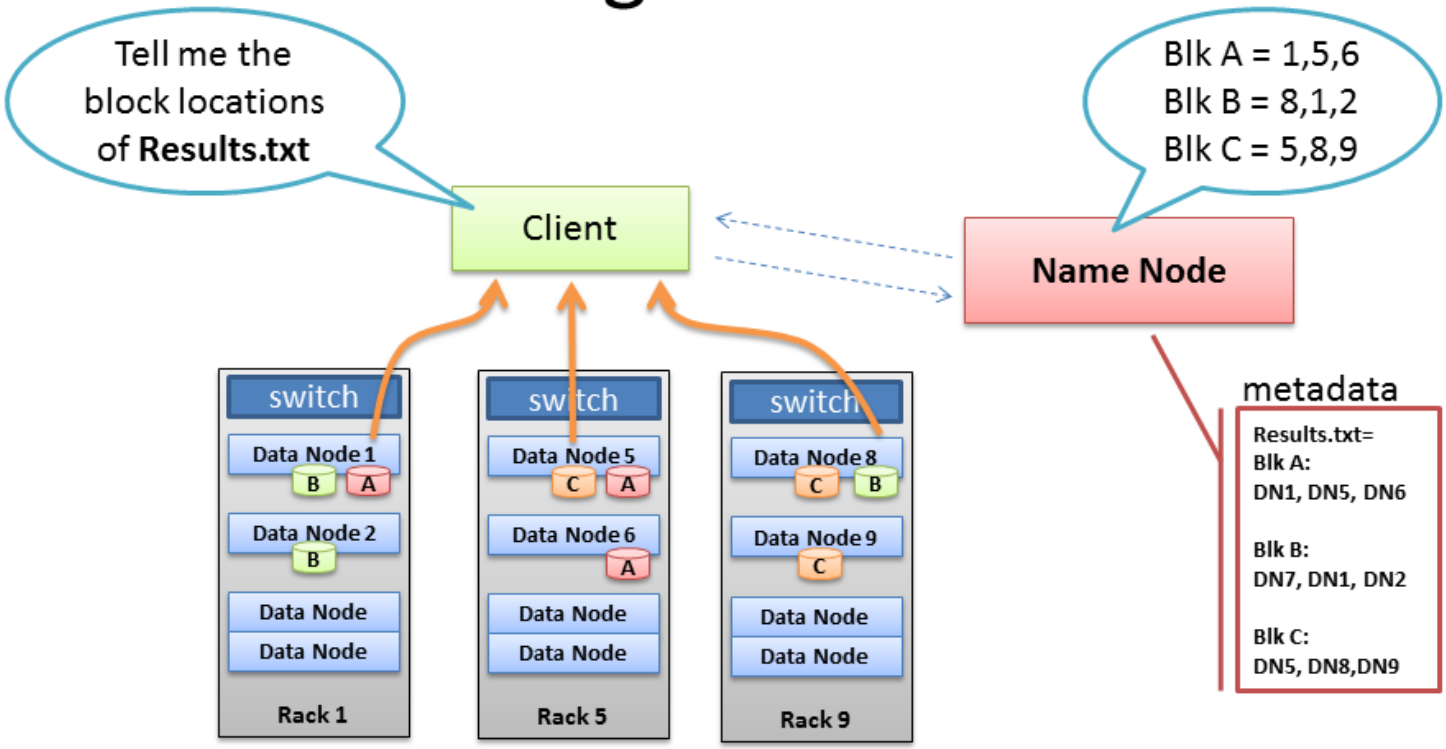
http://www.ewdna.com/2013/04/Hadoop-HDFS-Comics.html
http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

Preparing HDFS writes



- Name Node picks two nodes in the same rack, one node in a different rack
- Data protection
- Locality for M/R

Client reading files from HDFS

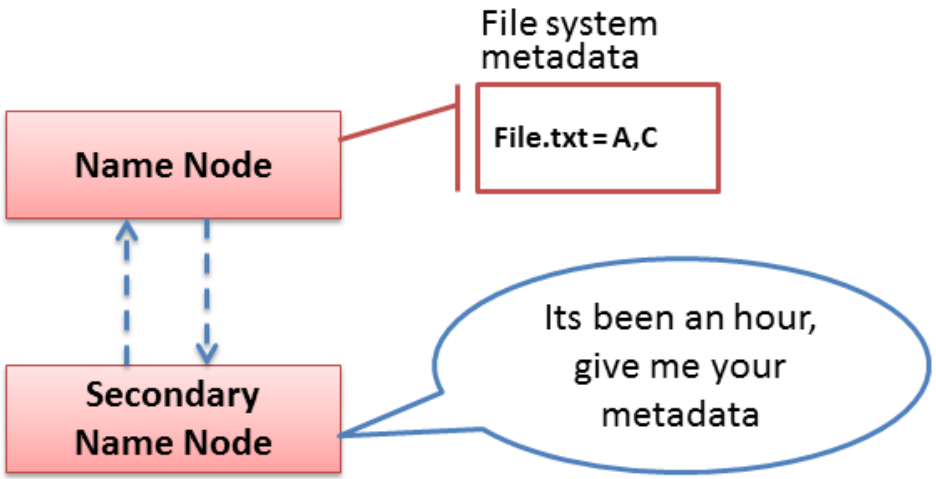


- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

BRAD HEDLUND .com

http://www.ewdna.com/2013/04/Hadoop-HDFS-Comics.html
http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

Secondary Name Node



- Not a hot standby for the Name Node
- Connects to Name Node every hour*
- Housekeeping, backup of Name Node metadata
- Saved metadata can rebuild a failed Name Node

BRAD HEDLUND .com

- NOT standby namenode
- NOT standby namenode
- NOT standby namenode

- 解壓hadoop-2.5.0-cdh5.3.2.tar.gz
  - cp /opt/hadoop-2.5.0-cdh5.3.2.tar.gz /home/hadoop
  - tar zxvf hadoop-2.5.0-cdh5.3.2.tar.gz
  - mv hadoop-2.5.0-cdh5.3.2 hadoop
  - /home/hadoop/hadoop

HDFS configuration

- /home/hadoop/hadoop/libexec/hadoop-config.sh
  - export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
- /home/hadoop/hadoop/etc/hadoop/slaves
  - slave
- /home/hadoop/hadoop/etc/hadoop/hadoop-env.sh
  - export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
- /home/hadoop/hadoop/etc/hadoop/hdfs-site.xml
- /home/hadoop/hadoop/etc/hadoop/core-site.xml
- 環境變數
- 同步所有hadoop設定檔

http://archive.cloudera.com/cdh4/cdh/4/hadoop/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml
http://archive.cloudera.com/cdh4/cdh/4/hadoop/hadoop-project-dist/hadoop-common/core-default.xml

core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://master:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/hadoop/hadoop_dir</value>
  </property>
</configuration>
```

hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.permissions</name>
    <value>false</value>
  </property>
</configuration>
```

- In .bashrc

```
export HADOOP_HOME=/home/hadoop/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop

export YARN_HOME=$HADOOP_HOME
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop

export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

hdadm@master$ scp –r /home/hadoop/hadoop hadoop@slave:/home/hadoop

Hadoop 啟動scripts

```
/usr/hadoop/hadoop/bin
├── container-executor
├── hadoop
├── hdfs
├── mapred
├── rcc
├── test-container-executor
└── yarn
/usr/hadoop/hadoop/sbin
├── distribute-exclude.sh
├── hadoop-daemon.sh
├── hadoop-daemons.sh
├── hdfs-config.sh
├── httpfs.sh
├── kms.sh
├── mr-jobhistory-daemon.sh
├── refresh-namenodes.sh
├── slaves.sh
├── start-all.sh
├── start-balancer.sh
├── start-dfs.sh
├── start-secure-dns.sh
├── start-yarn.sh
├── stop-all.sh
├── stop-balancer.sh
├── stop-dfs.sh
├── stop-secure-dns.sh
├── stop-yarn.sh
├── yarn-daemon.sh
└── yarn-daemons.sh
```

负责启动所有与HDFS相关的服务和命令行工具的脚本。因为hadoop是由java实现的，这个脚本的实质是在拼装一个java命令行来启动服务和工具对应的MainClass它已是最底层的启动脚本。

负责启动所有与MAPRED相关的服务和命令行工具的脚本。因为hadoop是由java实现的，这个脚本的实质是在拼装一个java命令行来启动服务和工具对应的MainClass它已是最底层的启动脚

负责启动所有与YARN相关的服务和命令行工具的脚本。因为hadoop是由java实现的，这个脚本的实质是在拼装一个java命令行来启动服务和工具对应的MainClass它已是最底层的启动脚本。

负责启动Hadoop服务(Daemon进程)的脚本。该脚本启停的服务都是通过调用/bin/hdfs来完成的，它做的主要工作是把启停命令通过nohup .... 的方式包装成了一个后台运行的daemon!

负责在多台目标机器上启动Daemon进程，它的工作完全是委派给slaves.sh去实现的。但在调用slaves.sh之前，调用了hadoop-config.sh完成了一些配置工作，一个重要的地方就是完成了对HADOOP_SLAVE_NAMES变量的赋值工作

这个脚本的命名并不妥当！它是一个向目标机器推送命令的Util脚本。slaves.sh在通过SSH推送命令时，会首先读取$HADOOP_SLAVE_NAMES这个数组中的机器列表作为推送目标，当这个数组为空时才使用slaves文件中给出的机器列表。

负责启动整个HDFS的脚本。该脚本是通过调用hadoop-damons.sh再调用slaves.sh来完成的，该文件中调用hadoop-daemons.sh时传递了一个重要参数：--hostsnames，这是告知命令执行的目标机器！在启动namenode和secondary-namenode的命令行中都通过读取配置显式地给出，但启动datanode的命令行则不会给出，后续脚本会使用slaves文件。

http://blog.csdn.net/bluishglc/article/details/43237289

HDFS CLI

- 格式化NameNode,
  - $hadoop namenode –format
  - 破壞性指令，只需執行一次
- 啟動與關閉HDFS
  - $start-dfs.sh
  - $stop-dfs.sh
- 確認namenode與datanode皆啟動
  - JPS
  - http://master:50070/
  - /home/hadoop/hadoop/logs

HDFS CLI

- 基本指令
  - hadoop fs –ls <file_in_hdfs>
  - hadoop fs –lsr <dir_in_hdfs>
  - hadoop fs –get <file_in_hdfs> <file_in_local>
  - hadoop fs –put <file_in_local> <file_in_hdfs>
  - hadoop fs –rm <file_in_hdfs>
  - hadoop fs –rmr <dir_in_hdfs>
  - hadoop fs -mkdir <dir_in_hdfs>
  - hadoop fs –chmod XXX <file_in_hdfs>
  - hadoop fs –chown XXX <file_in_hdfs>
  - hadoop fs –chgrp XXX <file_in_hdfs>

HDFS namespace

- HDFS default absolute URI
  - hadoop fs –ls /abc.txt
  - 等同 hadoop fs –ls **hdfs://master:9000** /abc.txt

- HDFS default relative URI
  - Hadoop fs –ls abc.txt
  - 等同於hadoop fs –ls hdfs://master:9000/**user/hdadm**/abc.txt
  - hdadm為目前在Linux的使用者帳號

- Quiz1: 如何存取其他HDFS cluster ??
- Quiz2: hadoop如何知道default URI??

HDFS Limitation

- HDFS
  - Good @ 大量大檔案
  - BAD @ 大量小檔案
    - 64MB block
    - 每個在HDFS上的檔案metadata在namenode上都有metadata
  - Sol. 將大量小檔案archive 至hadoop特有的sequential file

Java program access HDFS

- MAVEN
  - 編譯、打包、自動部署、library相依性管理 工具
  - POM.xml
    - repository
    - Dependency

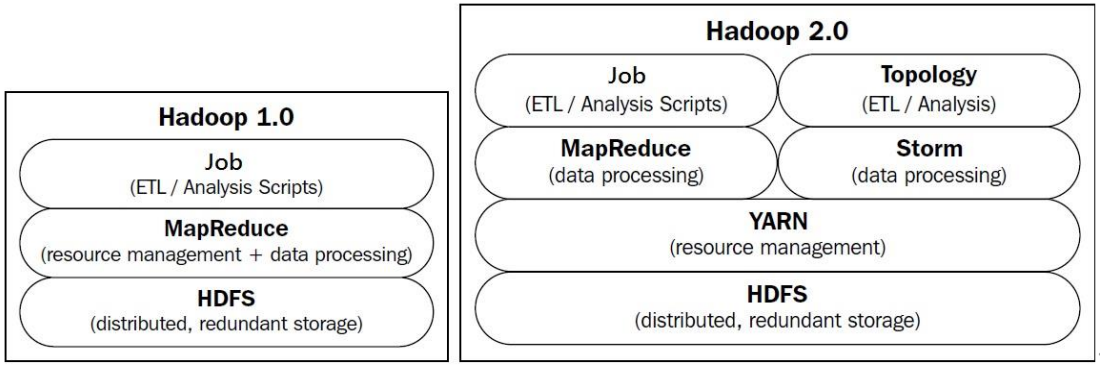
- 使用git 將範例clone下來
  - git clone https://github.com/ogre0403/MR-sample.git
    - org.nchc.train.hdfs.AccessHDFS
    - Copy local file to HDFS
    - Copy HDFS file to local
    - Generate sequence file

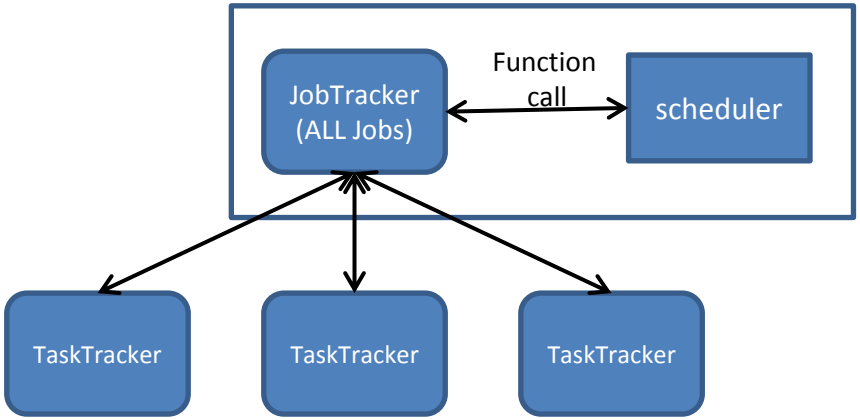
Outline

- Hadoop ecosystem 簡介
  - Intro, Version, distribution, OS base installation
- HDFS
  - Intro, install & configure,CLI, Java programing
- Mapreduce
  - YARN Intro, install & configure,
  - MR intro & Java programing
- Hbase
  - Intro, install & configure , CLI, Java programing

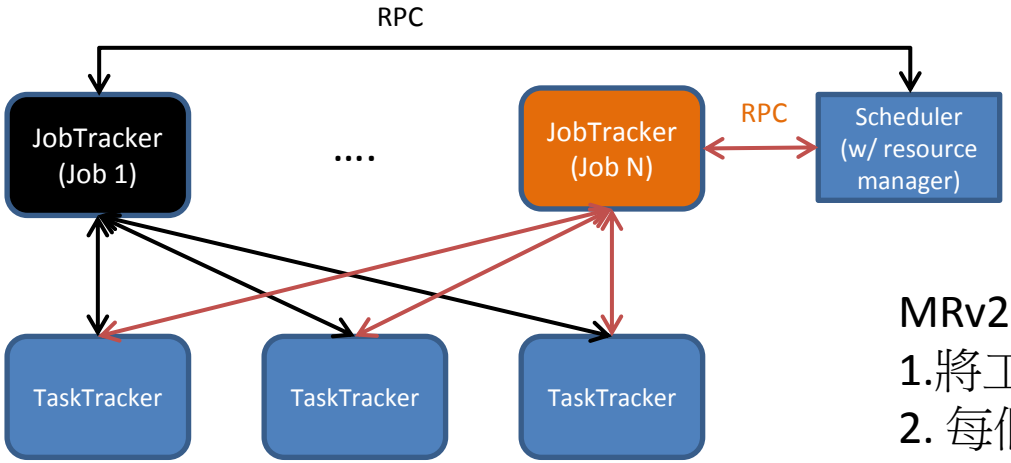
MRv1 v.s. MRv2

- Mapreduce 框架包含
  - 編程模型：map()與reduce()
    - MRv1與MRv2的程式寫法都相同
  - 運行環境：
    - MRv1：JobTracker與TaskTracker，JobTracker同時負責資源管理與所有工作的控制
    - MRv2：由YARN提供

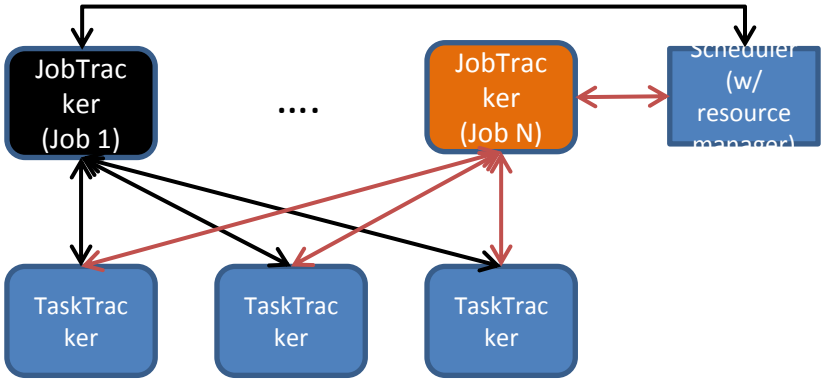




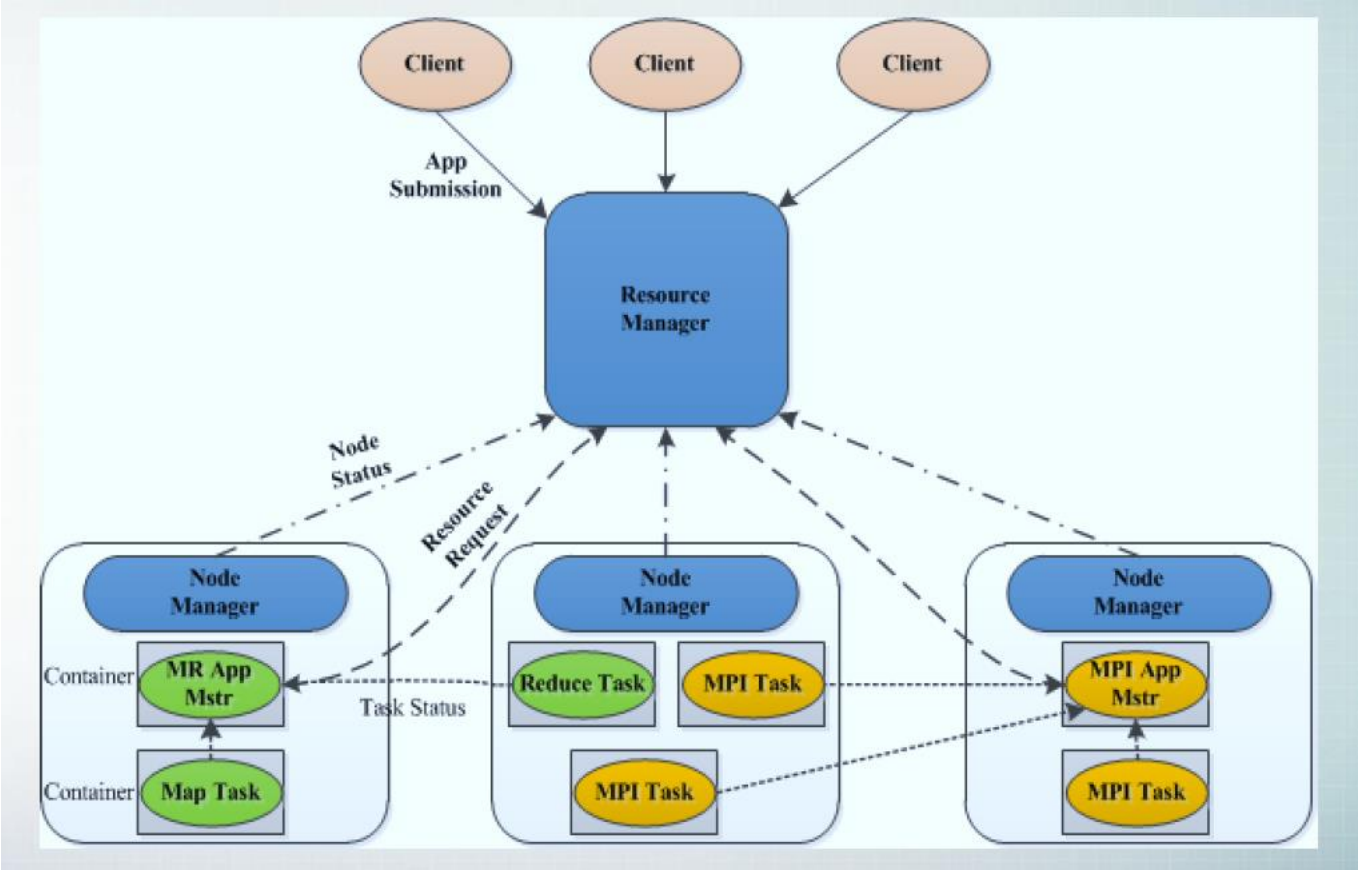
MRv1.
JobTracker負責工作控制與資源管理



MRv2.
1.將工作控制與資源管理分開
2. 每個Job有自己的JobTracker
3.全域的資源管理

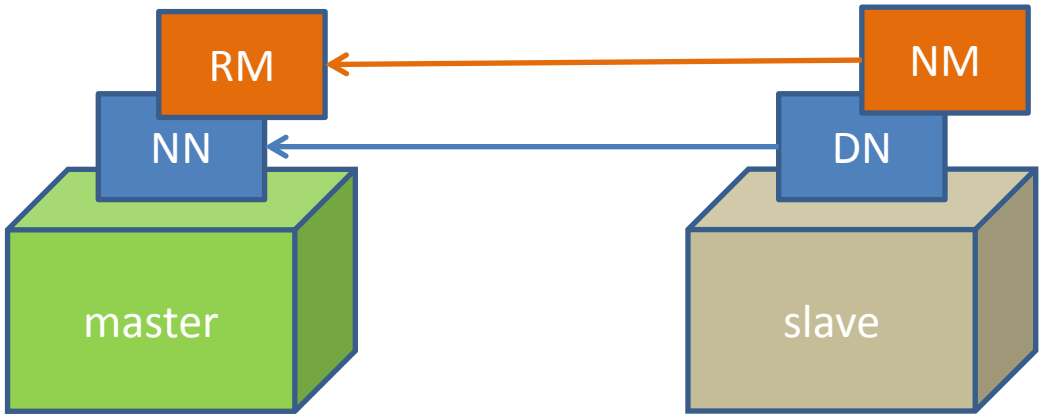


1. 由RM做全局的資源分配
2. NM定時回報目前的資源使用量
3. 每個JOB會有一個負責的AppMaster控制Job
4. 將資源管理與工作控制分開
5. YARN為一通用的資源管理系統
   可達成在YARN上運行多種框架



YARN Configuration

- Master /slave architecture
  - 包括Resource Manager，NodeManager
  - Support RM HA in hadoop 2.6
- 通常將RM與NN裝在一起，DN與NM裝在一起



YARN Configuration

- mapred-site.xml
- yarn-site.xml
- 環境變數
- 同步所有hadoop設定檔

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

mapred-site.xml

```xml
<?xml version="1.0"?>
<configuration>
<property>
   <name>yarn.resourcemanager.scheduler.address</name>
   <value>master:8030</value>
 </property>
<property>
   <name>yarn.resourcemanager.resource-tracker.address</name>
   <value>master:8031</value>
 </property>
<property>
   <name>yarn.resourcemanager.address</name>
   <value>master:8032</value>
 </property>
<property>
   <name>yarn.nodemanager.address</name>
   <value>0.0.0.0:8034</value>
 </property>
<property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce_shuffle</value>
 </property>
<property>
   <name>yarn.nodemanager.local-dirs</name>
   <value>/home/hadoop/hadoop_dir/nm-local-dir</value>
 </property>
<property>
   <name>yarn.nodemanager.log-dirs</name>
   <value>/home/hadoop/hadoop_dir/userlogs</value>
 </property>
</configuration>
```

yarn-site.xml

- In .bashrc

```
export HADOOP_HOME=/home/hadoop/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop

export YARN_HOME=$HADOOP_HOME
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop

export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

hdadm@master$ scp –r /home/hadoop/hadoop slave:/home/hadoop

YARN CLI

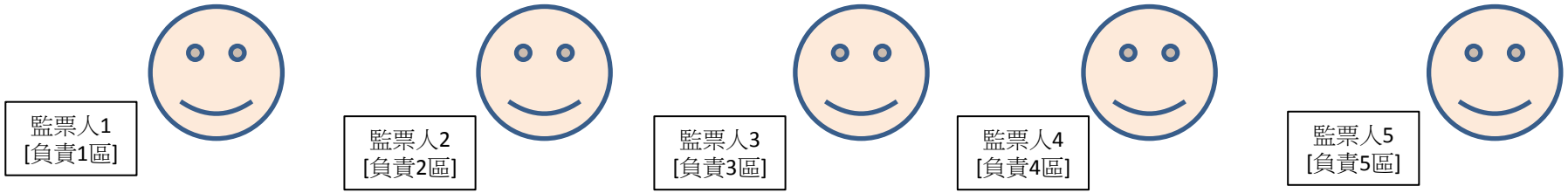
- 啟動與關閉YARN
  - 確認HDFS已啟動
  - $start-yarn.sh
  - $stop-yarn.sh
- 確認ResourceManager與NodeManager皆啟動
  - JPS
  - http://master:8088/cluster
- Run mr example
  - In /home/hadoop/hadoop/share/hadoop/mapreduce
  - hadoop jar hadoop-mapreduce-examples-2.5.0-cdh5.3.2.jar pi 10 1000

Outline

- Hadoop ecosystem 簡介
  - Intro, Version, distribution, OS base installation
- HDFS
  - Intro, install & configure,CLI, Java programing
- Mapreduce
  - YARN Intro, install & configure,
  - MR intro & Java programing
- Hbase
  - Intro, install & configure , CLI, Java programing

選舉到了…

- 台北市10個選區，共100萬票，要算出每個候選人的得票數



號次	票數
2	1
1	1
…	…

號次	票數
1	1
1	1
3	1
…	…

號次	票數
3	1
2	1
1	1

號次	票數
1	1
3	1
3	…

號次	票數
3	1
2	1
3	1

2	1
1	1
…	…

5	1
1	1
7	1
…	…

5	1
2	1
1	1

1	1
5	1
3	…

4	1
2	1
6	1

Shuffle & Sort
由各投開票所送到中選會

號次	票數
1	1
1	1
1	1
1	1
1	…

號次	票數
2	1
2	1
2	1
2	1
2	…

號次	票數
3	1
3	1
3	1
3	1
3	…

中選會
[負責 全部的候選人]

號次	票數
1	1
1	1
1	1
1	1
1	…

號次	票數
2	1
2	1
2	1
2	1
2	…

號次	票數
3	1
3	1
3	1
3	1
3	…

號次	總票數
1	187532

號次	總票數
2	574821

號次	總票數
3	237647

2	1
1	1
…	…

combine

姓別	總分
1	1840
2	1740
3	

1	1
3	1
…	…

combine

姓別	總分
1	1700
2	1520
3	

2	1
1	1
…	…

combine

姓別	總分
1	1700
2	1520
3	

1	1
1	1
3	…

combine

姓別	總分
1	1560
2	1240
3	

2	1
2	1
…	…

combine

姓別	總分
1	1760
2	1660
3	

Shuffle & Sort
由各投開票所送到中選會

號次	票數
1	1840
1	1700
1	1700
1	1560
1	…

號次	票數
2	1740
2	1520
2	1520
2	1240
2	…

號次	票數
3	….
3	…
3	…
3	…
3	…

中選會
[負責 全部的候選人]

號次	總票數
1	187532

號次	總票數
2	574821

號次	總票數
3	237647

算字數 - Mapper

Text file

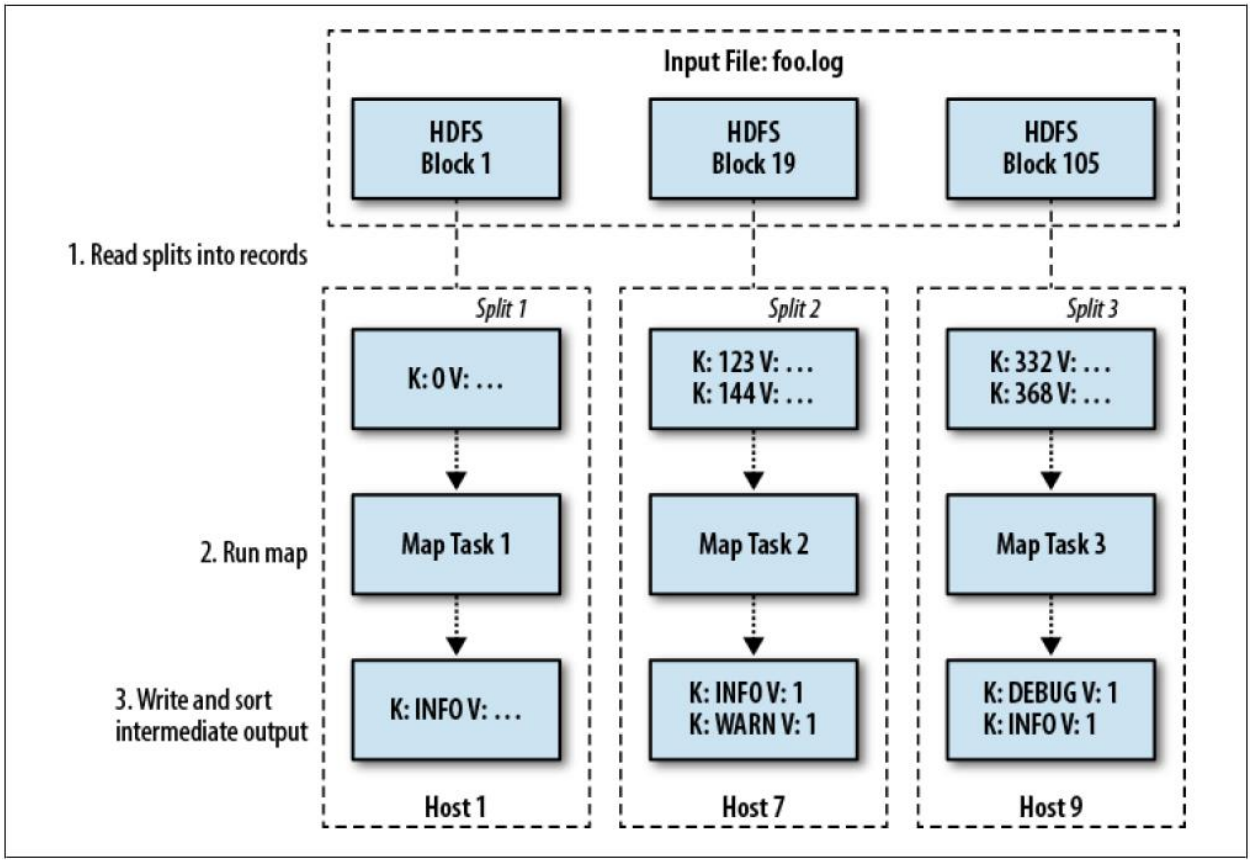
This is a book
Hello American

Visit The official site
Our network of more

The American Broadcasting

3 splits

1. 將輸入的文字檔案切成split
2. Mapper將split中的每一行讀出來
   (由 inputformat做)
3. 將每一行讀到的字都輸出 (字, 1)
   (Mapper 真正做的事)



•(k1, v1) → list(k2, v2)

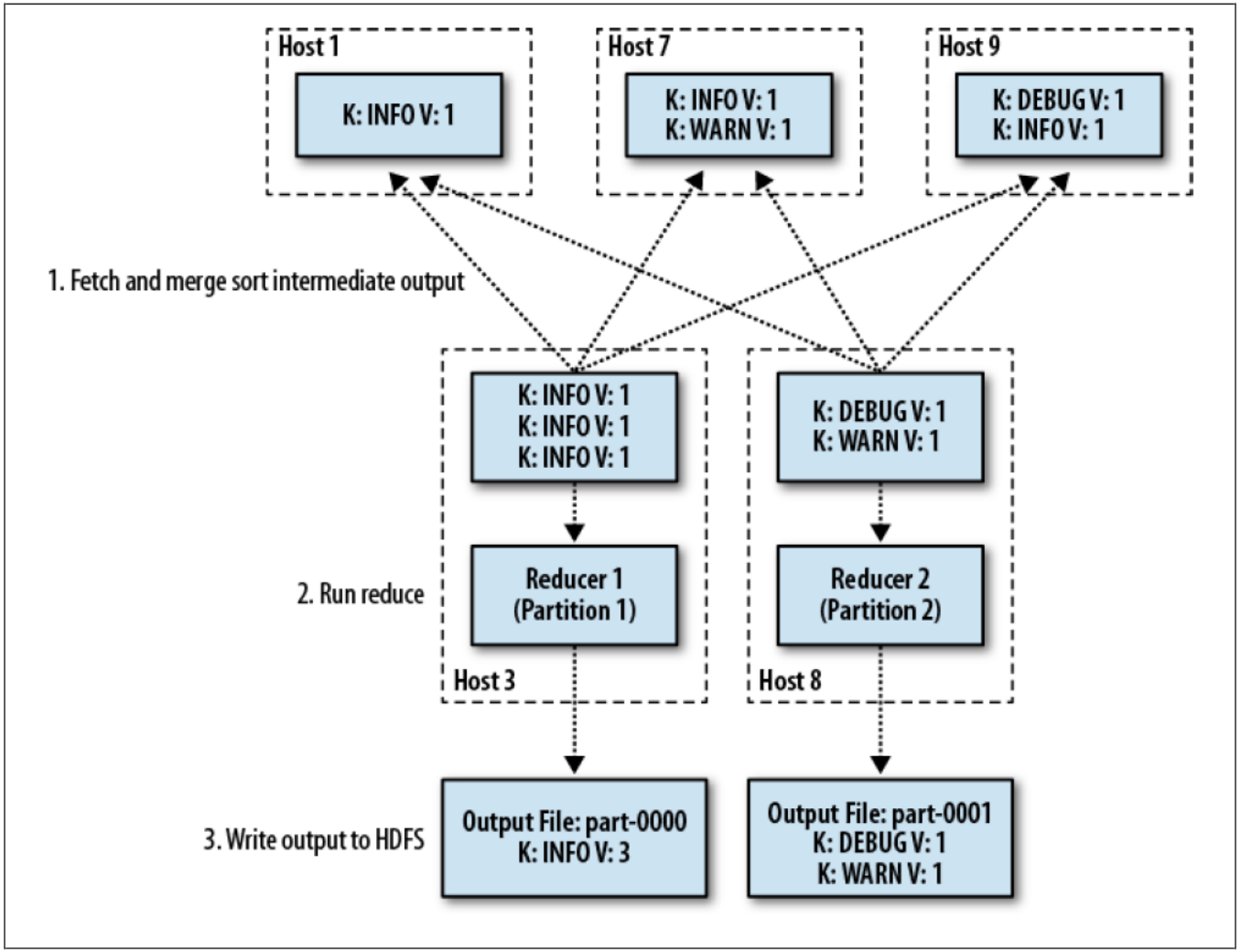
算字數 – Shuffle & Sort

- Black box
  - 開發人員不用煩腦，framework會自行處理
- 在給Reducer之前完成
- 用來保證Reducer得到的資訊有下列三個特性
  - 若Reducer看到某個Key1，會看到相對應的所有value
    - Reducer 1收到 This這個字，會收到很多 1
  - 給定Key1，所有Key1的值都會被同一個Reducer處理
  - 同一個Reducer有可能處理多個Key值

Reducer1 收到 (INFO, [1,1,1])
Reducer 2 收到 (DEBUG, [1]), (WARN, [1])

算字數 - Reducer

Reducer 對每個字的出現次數做加總



•(k2, list(v2)) → (k3,v3)

用正規的語法描述...

- Mapper：
  - (k1, v1) → list(k2, v2)
  - (0 , “This is a book book”) →
    (“This” , 1), (“is” ,1), (“a” ,1), (“book” ,1),(“book” ,1)
  - (0, 第一張選票) → (一號,0),(二號,1),(三號,0)
- Reducer：
  - (k2, list(v2)) → (k3,v3)

  - (“This” , [1]) → (“This” , 1)
  - (“is” ,[1]) → (“is” ,1)
  - (“a” ,[1]) → (“a” ,1)
  - (“book” ,[1, 1]) → (“book” ,2)

  (一號,[1,0,0,1,1,1,0,1,0,1,0]) → (一號, 6)
  (二號, [0,1,1,0,0,0,0,0,1,0,0]) → (二號, 3)
  (三號, [0,0,0,0,0,0,1,0,0,0,1]) → (三號,2)

算字數 - Pseudocode

```
void Map (key, value){
        for each word x in value:
            output.collect(x, 1);
}
```

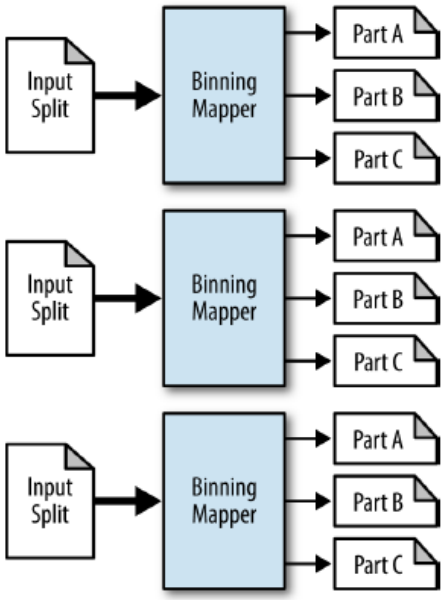
```
void Reduce (keyword, <list of value>){
        for each x in <list of value>:
            sum+=x;
        final_output.collect(keyword, sum);
}
```

算字數 – real code

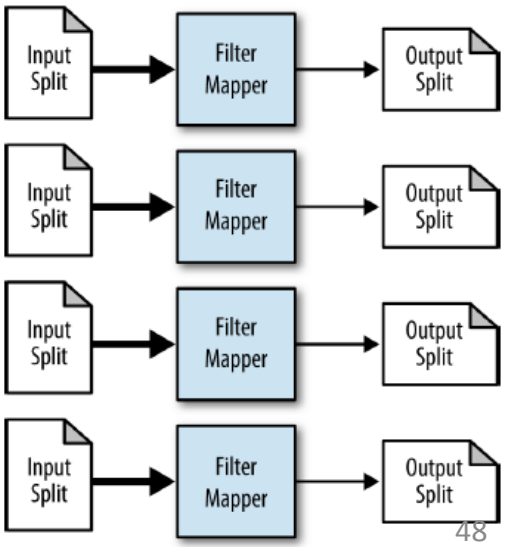
```
public void map(LongWritable key, Text value, Context context
                ) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString()); // line to string token
        while (itr.hasMoreTokens()) {
          word.set(itr.nextToken());    // set word as each input keyword
          context.write(word, one);     // create a pair <keyword, 1>
        }
 }
```

```
public void reduce(Text key, Iterable<IntWritable> values,  Context context
                   ) throws IOException, InterruptedException {
          int sum = 0; // initialize the sum for each keyword
          for (IntWritable val : values) {
            sum += val.get();
          }
          result.set(sum);
          context.write(key, result); // create a pair <keyword, number of occurences>
}
```

- Quiz1. MapReduce一定要有Mapper與Reducer?
- Ans:
  - Map-only job只需要map(), 不需要Reduce()
  - job.setNumReduceTask(0)



- Quiz2. MapReduce只能處理文字資料?
  - MapReduce提供的是一個平行運算的framework
  - 文字資料只是Hadoop本身剛好有提供適合的input處理機制
  - 只要輸入可以分成多個獨立的輸入，就可以透過多個Mapper平行處理
  - EX. 有四個影片 要做轉檔， 如果不平行處理，等第一個轉完完再轉第二個…



Java Programing

- Code skeleton
- Driver code snippet
- Map class snippet
- Reduce class snippet
- git clone https://github.com/ogre0403/MR-sample.git
  - org.nchc.train.mr.wordcount
  - org.nchc.train.mr.seq
  - org.nchc.train.mr.kmeans
  - org.nchc.train.mr.bfs

```
public class MyMR {
  public  class MyMapper extends Mapper<Object, Text, Text, IntWritable> {
          ...
  }

  public  class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
          ...
   }

  public static void main(String[] args) throws Exception {
          ...
   }
}
```

Map code

Reduce code

Driver setup

Driver setup

```
Configuration conf = new Configuration();
Job job = new Job(conf, “New MR job");
job.setJarByClass(MyMR.class);
job.setMapperClass(MyMapper.class);
job.setReducerClass(MyReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

Configuration & Run

```
Configuration conf = new Configuration();
Job job = new Job(conf, “New MR job");
…
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

Set Map/Reduce/Combine Class

```
job.setJarByClass(MyMR.class);
job.setMapperClass(MyMapper.class);
job.setReducerClass(MyReducer.class);
```

Set input/output format

```
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

- Inputformat
  - Hadoop 如何讀取來源資料
  - plain text, DB, or customer source…
  - 預設為TextInputFormat class
    - 每一行為一筆record,
    - key 為在文件中的offset
    - value為整行內容

- Outputformat
  - Hadoop如何將分析完的結果輸出
  - 預設為TextOutputFormat class
  - 每一筆結果為輸出文件中的一行
  - 每一行包含key/value，預設以tab分隔
  - Key/value可為任意class, 但需在Driver中設定

- 若使用預設的TextInputFormat/TextOutputFormat, 無需在Driver中設定

- 若使用非預設的input/output format
  - job.setInputFormatClass(SequenceFileInputFormat.class);
  - job.setOutputFormatClass(NullOutputFormat.class);

public class MyMapper extends

Mapper< Input Key class , Input Value class , Output Key class , Output value class > {

public void map(Input Key class key, Input Value class value, Context context)
throws IOException, InterruptedException{

Output Key class newkey = new Output Key class ()

Output value class newvalue = new Output value class ()

....
Context.write(newkey,newvalue);

}

}

```
public  class WordCountMapper
            extends Mapper< Object, Text , Text, IntWritable>{

            public void map(Object key, Text value, Context context )
                        throws IOException, InterruptedException {

            StringTokenizer itr = new StringTokenizer(value.toString());
             IntWritable one = new IntWritable(1);

            while (itr.hasMoreTokens()) {
                         Text word = new Text(itr.nextToken())
                        context.write(word, one);
            }
   }
}
```

```
public  class MyReducer extends
        Reducer<Input Key class, Input Value class, Output Key class, Output value class> {

        public void reduce(Input Key class key, Iterable<Input Value class> values, Context context)
                                    throws IOException, InterruptedException{

                Output Key class newkey  =  new Output Key class()

                Output value class newvalue = new Output value class()

                        ....
                Context.write(newkey,newvalue);

        }
}
```

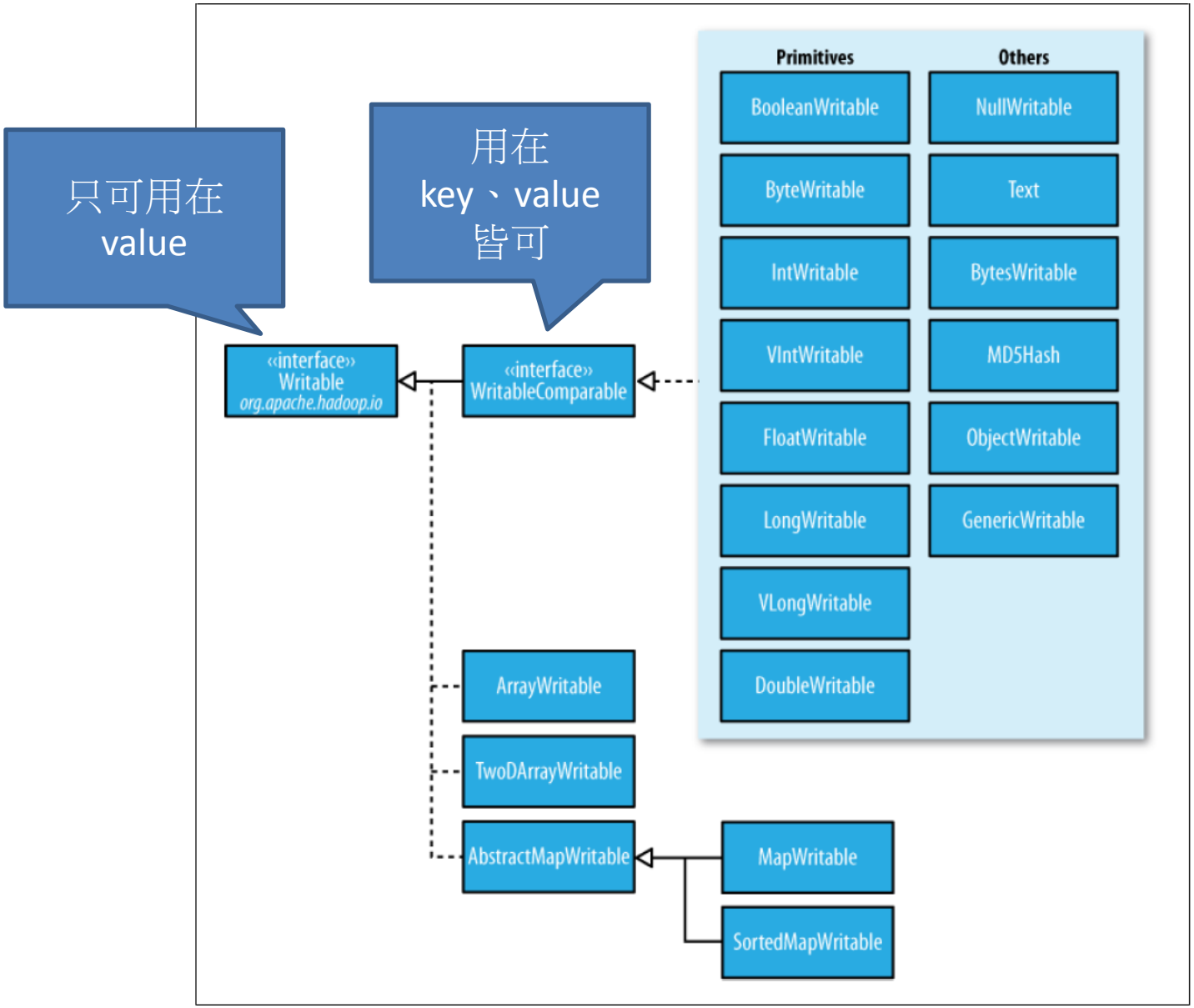
```
public class WordCountReducer
        extends  Reducer< Text, IntWritable , Text , IntWritable > {

        public void reduce( Text key, Iterable< IntWritable > values, Context context)
                throws IOException, InterruptedException {

                int sum = 0;
                IntWritable result = new IntWritable();
                for (IntWritable val : values) {
                        sum += val.get();
                }
                result.set(sum);
                context.write(key, result);
        }
}
```

What is Writable Class

- 什麼是Text類型、什麼是IntWritable類型
  - Text: Wrapper for Java String class
  - IntWritable: Wrapper for Java int
- 序列化框架
  - 物件在網路上傳遞要透過serialize/deserializae
  - Java 本身有Serializable
  - Hadoop自行設計Writable 序列化框架
- 若內建的writable不合需求，需自行定義
  - Implement writable : 用在value
  - Implement writablecomparable: 用在key、value


只可用在
value
用在
key、value
皆可
«interface»
Writable
org.apache.hadoop.io
«interface»
WritableComparable
Primitives
Others
BooleanWritable
ByteWritable
IntWritable
VIntWritable
FloatWritable
LongWritable
VLongWritable
DoubleWritable
NullWritable
Text
BytesWritable
MD5Hash
ObjectWritable
GenericWritable
ArrayWritable
TwoDArrayWritable
AbstractMapWritable
MapWritable
SortedMapWritable

New and Old API

Map 區

Reduce 區

設定區

```
Class MR{
    Class Mapper … {
    }
    Class Reducer …
    }
    main(){
        JobConf conf = new JobConf("MR.class");
        conf.setMapperClass(Mapper.class);
        conf.setReduceClass(Reducer.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]))
        FileOutputFormat.setOutputPath(conf, new Path(args[1
        JobClient.runJob(conf);
}}
```

Map 程式碼

Reduce 程式碼

其他的設定參數程式碼

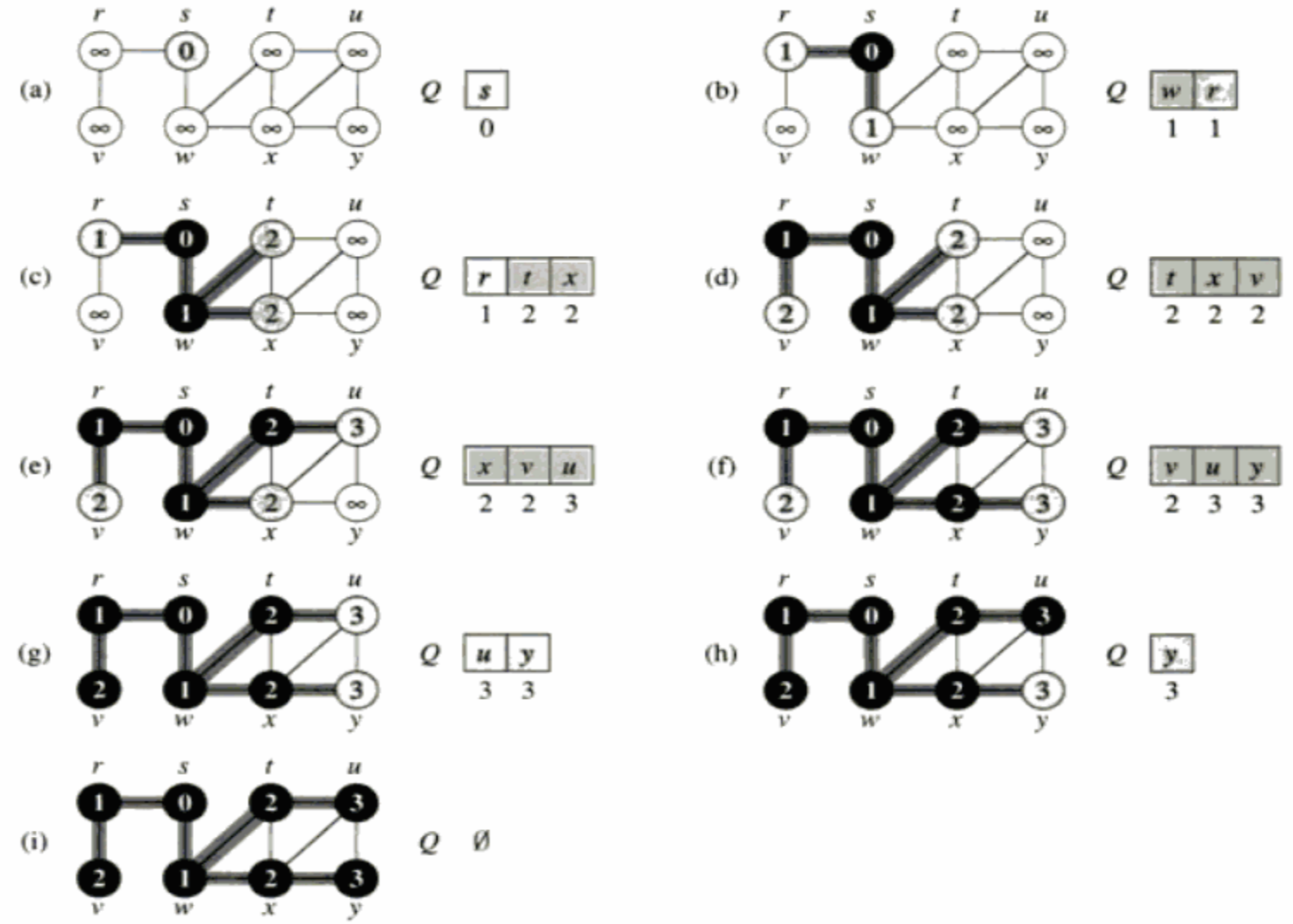
```
import org.apache.hadoop.mapred.*;
class MyMap extends MapReduceBase
implements Mapper < INPUT KEY , INPUT VALUE , OUTPUT KEY , OUTPUT VALUE >
{
// 全域變數區
public void map ( INPUT KEY key, INPUT VALUE value,
        OutputCollector< OUTPUT KEY , OUTPUT VALUE > output,
        Reporter reporter) throws IOException
        {
        // 區域變數與程式邏輯區
        output.collect( NewKey, NewValue);
        }
}
```

```
import org.apache.hadoop.mapred.*;
class MyRed extends MapReduceBase
implements Reducer < INPUT KEY , INPUT VALUE , OUTPUT KEY , OUTPUT VALUE >
{
// 全域變數區
public void reduce ( INPUT KEY key, Iterator< INPUT VALUE > values,
        OutputCollector< OUTPUT KEY , OUTPUT VALUE > output,
        Reporter  reporter) throws IOException
        {
        // 區域變數與程式邏輯區
        output.collect( NewKey, NewValue);
        }
}
```

Put all together

- breadth-first search
- K-means

BFS


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
r s t u
v w x y
Q
s
0
w r
1 1
r t x
1 2 2
t x v
2 2 2
x v u
2 2 3
v u y
2 3 3
u y
3 3
y
3
Ø

BFS Mapper

- Only process gray node, and change color to black
- Just emit same node when color is not gray

Key	value
1	2,5|0|GRAY|
2	1,3,4,5|Integer.MAX_VALUE|WHITE|
3	2,4|Integer.MAX_VALUE|WHITE|
4	2,3,5|Integer.MAX_VALUE|WHITE|
5	1,2,4|Integer.MAX_VALUE|WHITE|

Key	value
1	2,5|0|BLACK|

Key	value
2	NULL|1|GRAY|

Key	value
5	NULL|1|GRAY|

Key	value
2	1,3,4,5|Integer.MAX_VALUE|WHITE

Key	value
3	2,4|Integer.MAX_VALUE|WHITE|

Key	value
4	2,3,5|Integer.MAX_VALUE|WHITE|

Key	value
5	1,2,4|Integer.MAX_VALUE|WHITE|

BFS Reducer

- the reducers job is to take all this data and construct a new node using
  - the non-null list of edges
  - the minimum distance
  - the darkest color

Key	value
1	2,5|0|BLACK|

Key	value
2	NULL|1|GRAY|
2	1,3,4,5|Integer.MAX_VALUE|WHITE|

Key	value
3	2,4|Integer.MAX_VALUE|WHITE|

Key	value
4	2,3,5|Integer.MAX_VALUE|WHITE|

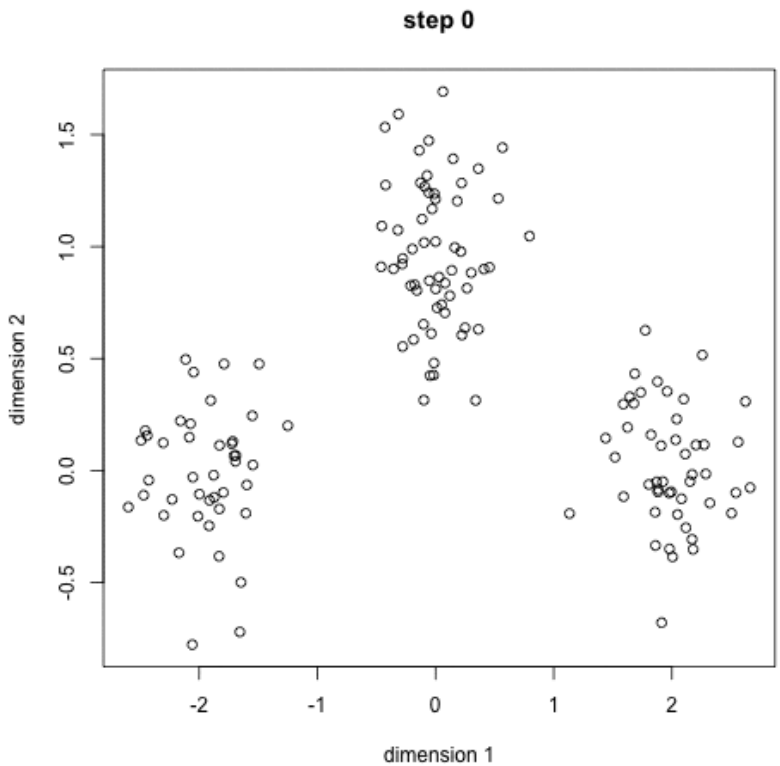
Key	value
5	NULL|1|GRAY|
5	1,2,4|Integer.MAX_VALUE|WHITE|

reducer

Key	value
1	**2,5|0|BLACK|**
2	**1,3,4,5|1|GRAY|**
3	2,4|Integer.MAX_VALUE|WHITE|
4	2,3,5|Integer.MAX_VALUE|WHITE|
5	**1,2,4,|1|GRAY**

K-means

- 隨機選取資料組中的k筆資料當作初始群中心u_1~u_k
- 計算每個資料xi 對應到最短距離的群中心 (固定 ui 求解所屬群 Si)
- 利用目前得到的分類重新計算群中心 (固定 Si 求解群中心 ui)
- 重複step 2,3直到收斂 (達到最大疊代次數 or 群心中移動距離很小)



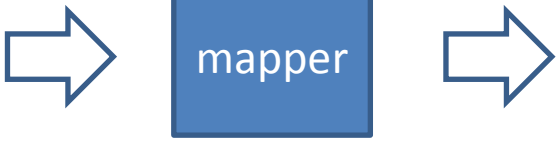
Map

輸入為<目前的中心，point>
求point到每個中心的距離
輸出為<所屬的中心，point>

Read Distributed cache

C1 : (x1,y1)
C2 : (x2,y2)
C3 : (x3,y3)

Key	value
C0	V1(1,2)
C0	V2(7,4)
C0	V3(16,3)
C0	V4(-1,-23)



Key	value
C2	V1(1,2)

Key	value
C2	V2(7,4)

Key	value
C1	V3(16,3)

Key	value
C3	V4(-1,-23)

Reducer

輸入為<中心，屬於該中心的所有point>
對所有的point計算出新的中心
輸出<新的中心，point>做為下一次疊代

Key	value
C1	V3(16,3)

Key	value
C2	V1(1,2)
C2	V2(7,4)

Key	value
C3	V4(-1,-23)

Key	value
C1	V3(16,3)
C2	V1(1,2)
C2	V2(7,4)
C3	V4(-1,-23)

Update Distributed cache

C1 : (x’1,y’1)
C2 : (x’2,y’2)
C3 : (x’3,y’3)

High level tools based on MR

- 什麼都要從MapReduce寫起很麻煩，不需要重復製造輪子
  - Scripts : PIG
  - Data warehouse：HIVE
  - Machine learning framework：Mahout

HIVE short DEMO

- wget http://archive-primary.cloudera.com/cdh5/cdh/5/hive-0.13.1-cdh5.3.2.tar.gz
- tar zxvf hive-0.13.1-cdh5.3.2.tar.gz
- 執行$HIVE_HOME/bin/hive

nm	,	dp	,	Id
劉	,	北	,	A1
李	,	中	,	B1
王	,	中	,	B2

Id	,	dt	,	hr
A1	,	2015-7-7	,	13
A1	,	2015-7-8	,	12
A1	,	2015-7-9	,	4

關聯查詢

A1	劉	北	12.5

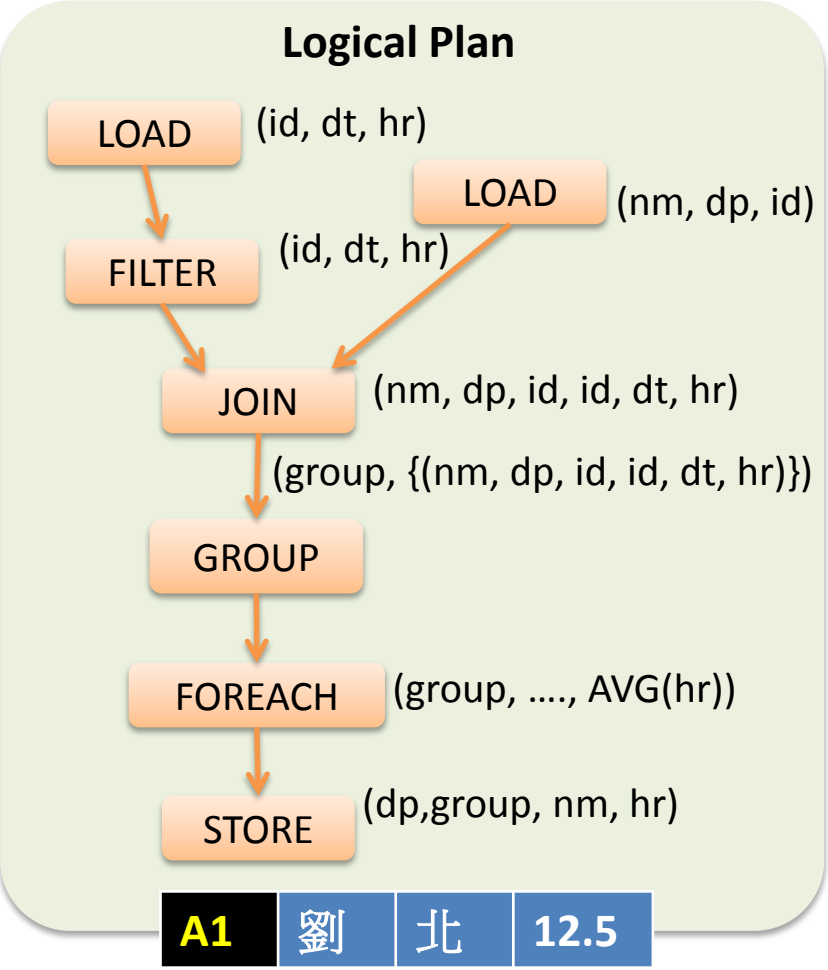
```
hive> create database ogre;
hive> create table ogre.a1 (nm String, dp String, id String) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' tblproperties ("skip.header.line.count"="1");
hive> create table ogre.b1 (id String, dt Date, hr Int) ROW FORMAT DELIMITED FIELDS TERMINATED BY
',' tblproperties ("skip.header.line.count"="1");
hive> create table ogre.result (dp String, id String , nm String, avg Float);
hive> load data inpath "hive.test/file1.txt" OVERWRITE INTO TABLE ogre.a1;
hive> load data inpath "hive.test/file2.txt" OVERWRITE INTO TABLE ogre.b1;
hive> INSERT OVERWRITE TABLE result select a1.id, collect_set(a1.dp),  collect_set(a1.nm), avg(b1.hr)
from a1,b1 where b1.hr > 8 and b1.id = a1.id group by a1.id;
hive> select * from result;
OK
A1	北	劉	12.5
hive> select * from a1,b1 where a1.id = b1.id
Total MapReduce CPU Time Spent: 1 seconds 580 msec
OK
劉	北	A1	A1	2015-07-07	13
劉	北	A1	A1	2015-07-08	12
劉	北	A1	A1	2015-07-09	4
Time taken: 25.296 seconds, Fetched: 3 row(s)
```

Pig Short Demo

- wget http://archive-primary.cloudera.com/cdh5/cdh/5/pig-0.12.0-cdh5.3.2.tar.gz
- tar zxvf pig-0.12.0-cdh5.3.2.tar.gz

nm	,	dp	,	Id
劉	,	北	,	A1
李	,	中	,	B1
王	,	中	,	B2

Id	,	dt	,	hr
A1	,	2015-7-7	,	13
A1	,	2015-7-8	,	12
A1	,	2015-7-9	,	4



```
grunt> A = LOAD '/user/hadoop/pig.test/file1.txt' using PigStorage(',') AS (nm,dp,id);
grunt> B = LOAD '/user/hadoop/pig.test/file2.txt' using PigStorage(',') AS (id, dt, hr);
grunt> C = FILTER B by hr > 8;
grunt> D = JOIN C BY id, A BY id;
grunt> E = GROUP D BY A::id;
grunt> F = FOREACH E GENERATE $1.dp,group,$1.nm, AVG($1.hr);
grunt> STORE F INTO '/user/hadoop/pig.output/';)
```

Mahout Short Demo

- get http://archive-primary.cloudera.com/cdh5/cdh/5/mahout-0.9-cdh5.3.2.tar.gz
- tar zxvf mahout-0.9-cdh5.3.2.tar.gz
  - 分群
  - 推荐

使用Mahout做分群

	國文	數學
ID 1	0	10
ID 2	10	0
ID 3	10	10
ID 4	20	10
ID 5	10	20
ID 6	20	20
ID 7	50	60
ID 8	60	50
ID 9	60	60
ID 10	90	90

Step by Step

```
#vi clustering.data
0 10
10 0
10 10
20 10
10 20
20 20
50 60
60 50
60 60
90 90
```

```
# hadoop fs -mkdir testdata
# hadoop fs -put clustering.data testdata
# hadoop fs -ls -R testdata
-rw-r--r--   3 root hdfs     288374 2014-02-05 21:53 testdata/clustering.data
```

```
# mahout org.apache.mahout.clustering.syntheticcontrol.canopy.Job
-t1 3 -t2 2 -i testdata -o output
...omit...
14/09/08 01:31:07 INFO clustering.ClusterDumper: Wrote 3 clusters
14/09/08 01:31:07 INFO driver.MahoutDriver: Program took 104405
 ms (Minutes: 1.7400833333333334)
```

```
#mahout clusterdump --input output/clusters-0-final --pointsDir output/clusteredPoints
C-0{n=1 c=[9.000, 9.000] r=[]}
    Weight : [props - optional]:  Point:
    1.0: [9.000, 9.000]
C-1{n=2 c=[5.833, 5.583] r=[0.167, 0.083]}
    Weight : [props - optional]:  Point:
    1.0: [5.000, 6.000]
    1.0: [6.000, 5.000]
    1.0: [6.000, 6.000]
C-2{n=4 c=[1.313, 1.333] r=[0.345, 0.527]}
    Weight : [props - optional]:  Point:
    1.0: [1:1.000]
    1.0: [0:1.000]
    1.0: [1.000, 1.000]
    1.0: [2.000, 1.000]
    1.0: [1.000, 2.000]
    1.0: [2.000, 2.000]
```

使用Mahout做推荐

	book-a	book-b	book-c
User 1	★★★★★	★★★★☆	★★★★★
User 2	★★★★☆	★★★★☆	★★★★☆
User 3	★★★★★	★★★★☆	4~5
User 4	★☆☆☆☆	★★☆☆☆	1~2
User 5	★★☆☆☆	★☆☆☆☆	★☆☆☆☆

```
# hadoop fs -ca
3    [3:4.478726
4    [3:1.521273
```

1. 我們預測User4不太喜歡book-c，所以我不會推薦book-c給User4
2. 我們預測User3喜歡book-c，所以我會推薦book-c給User3

```
#vi recom.data
1,1,5
1,2,4
1,3,5
2,1,4
2,2,5
2,3,4
3,1,5
3,2,4
4,1,1
4,2,2
5,1,2
5,2,1
5,3,1
```

```
# hadoop fs -mkdir testdata
# hadoop fs -put recom.data testdata
# hadoop fs -ls -R testdata
-rw-r--r--   3 root hdfs     288374 2014-02-05 21:53 testdata/recom.data
```

```
# mahout recommenditembased -s SIMILARITY_EUCLIDEAN_DISTANCE -i testdata -o output
...omit...
      File Input Format Counters
            Bytes Read=287
       File Output Format Counters
            Bytes Written=32
14/09/04 05:46:56 INFO driver.MahoutDriver:
Program took 434965 ms (Minutes: 7.249416666666667)
```

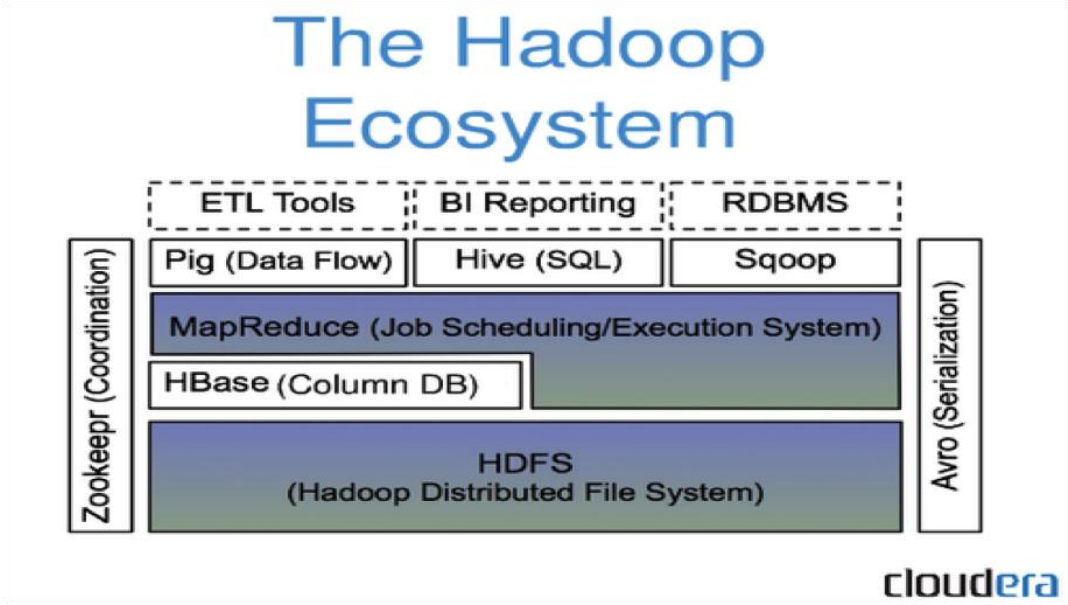
```
# hadoop fs -cat output/part-r-00000
3       [3:4.4787264]
4       [3:1.5212735]
```

Outline

- Hadoop ecosystem 簡介
  - Intro, Version, distribution, OS base installation
- HDFS
  - Intro, install & configure,CLI, Java programing
- Mapreduce
  - YARN Intro, install & configure,
  - MR intro & Java programing
- HBase
  - Intro, install & configure , CLI, Java programing

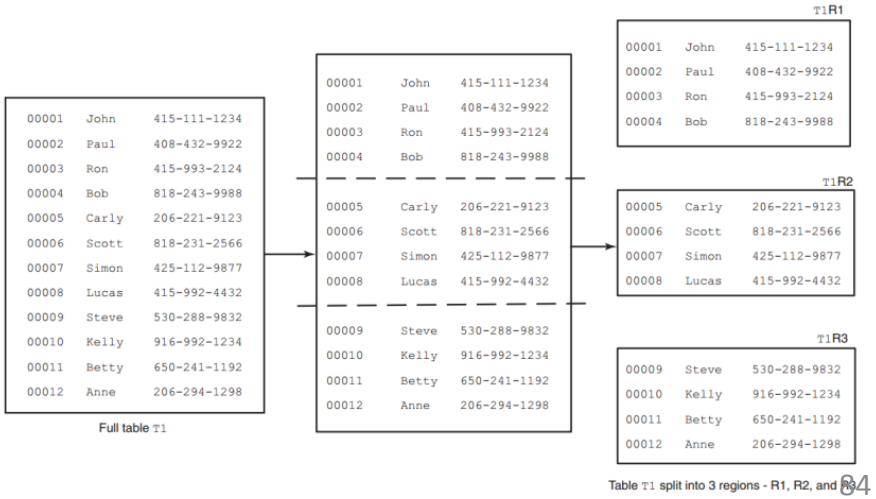
What is HBase

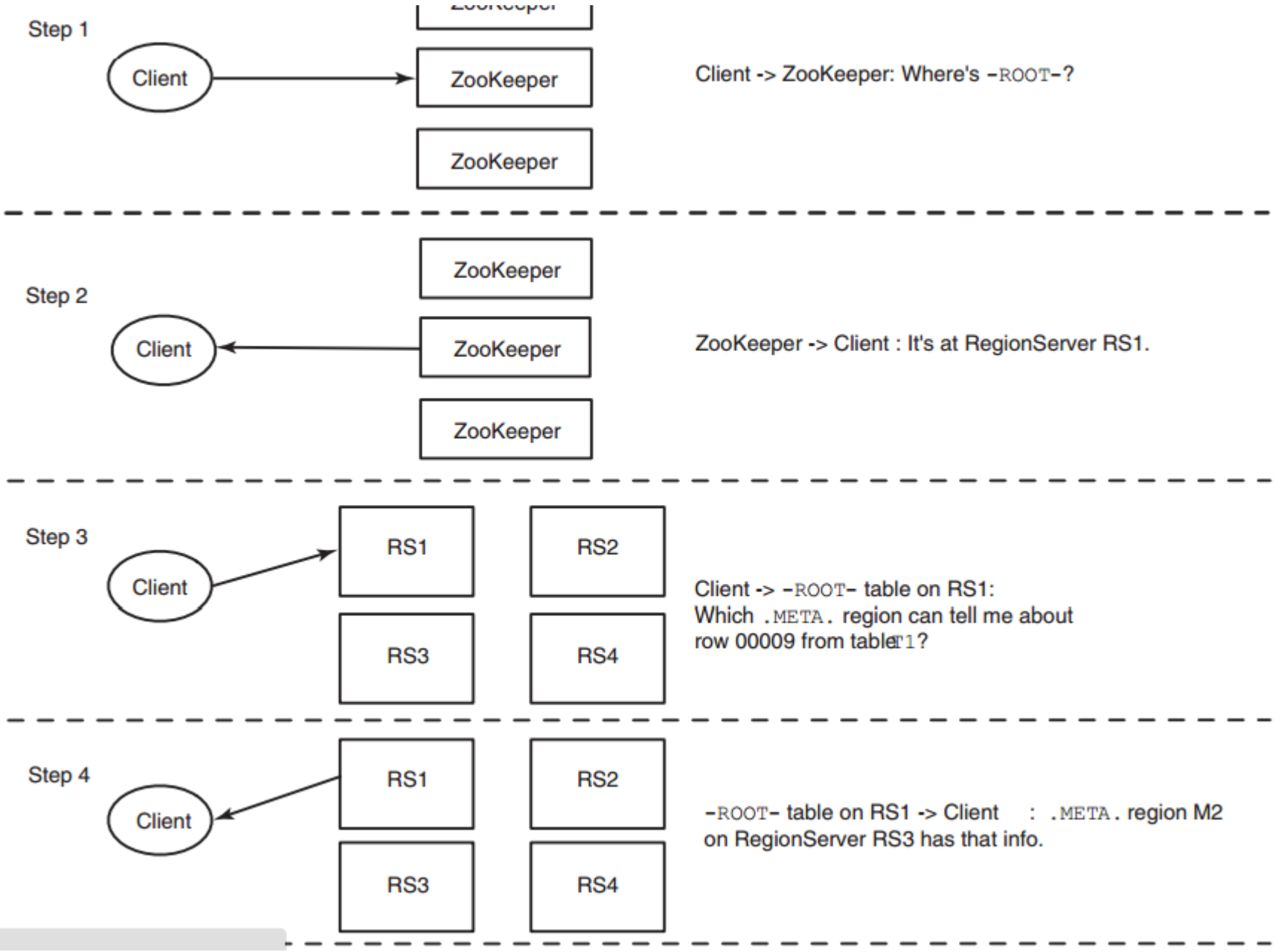
- Hbase是一個高可靠性、高性能、column-orient、 scalability 的分散式儲存系統

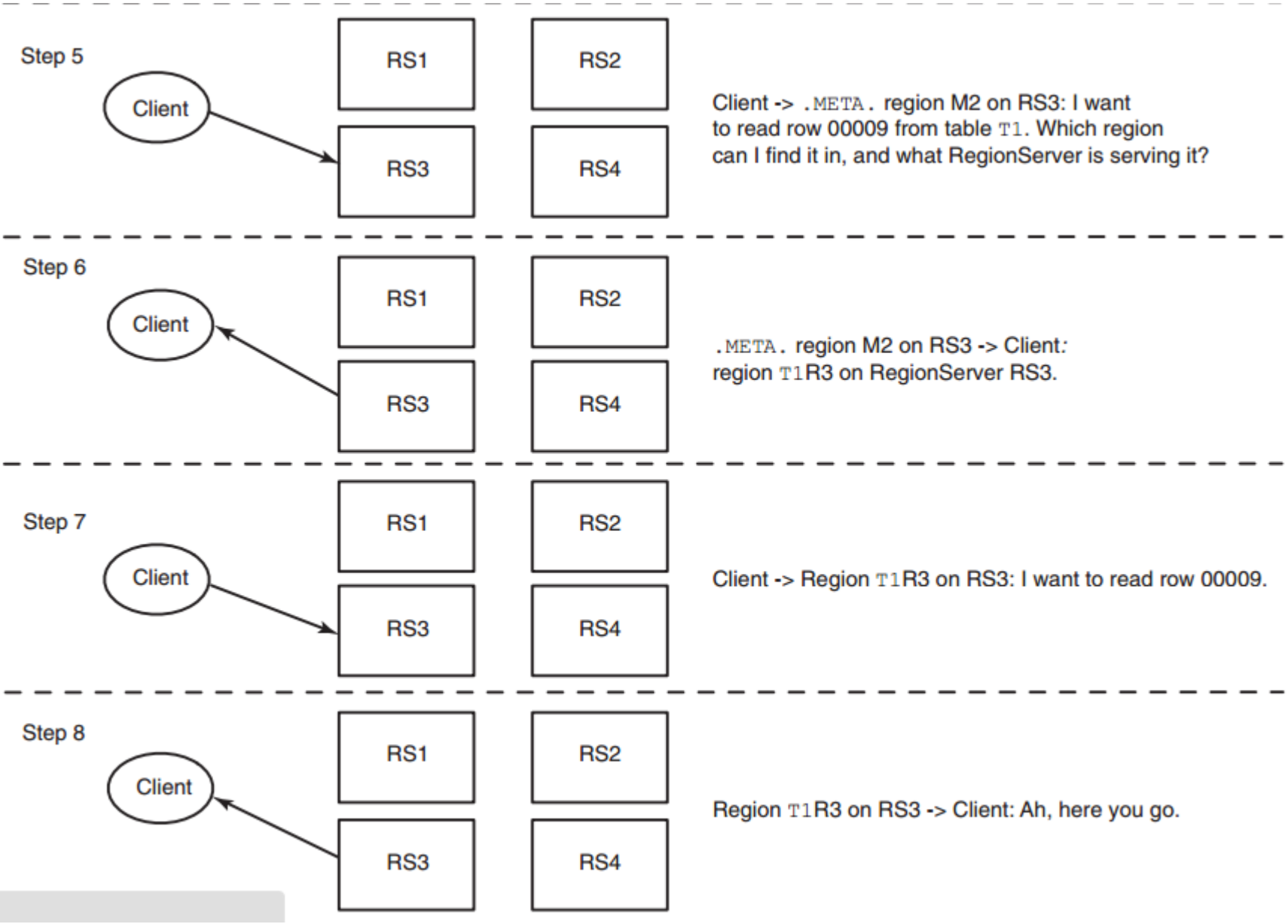


Hbase architecture

- HMaster
  - Responsible for assigning regions to RegionServer
- RegionServer
  - Table can be split into many region
  - Each RegionServer contains many regions
  - Add RS to horizontal scale out

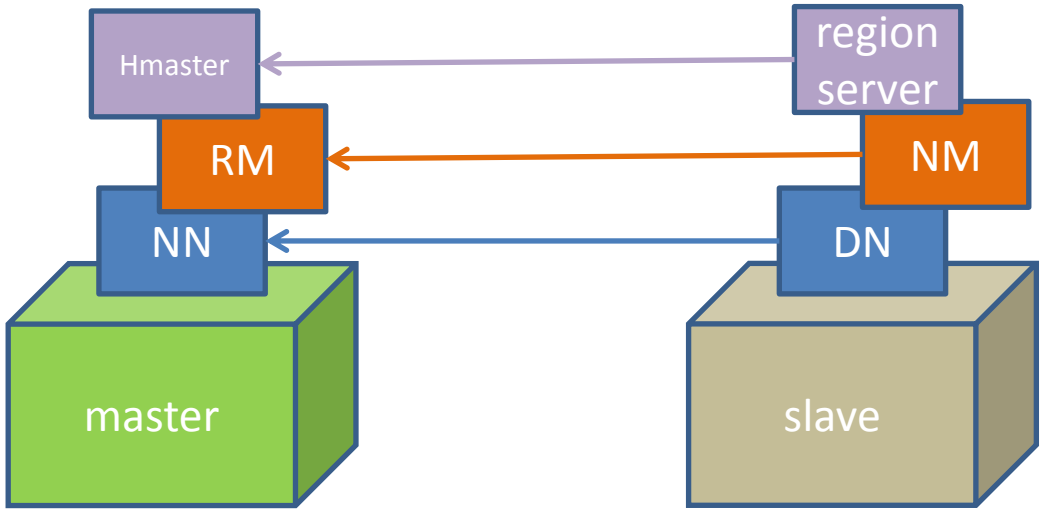



Step 1
Client
ZooKeeper
ZooKeeper
Client -> ZooKeeper: Where's -ROOT-?
Step 2
Client
ZooKeeper
ZooKeeper
ZooKeeper
ZooKeeper -> Client : It's at RegionServer RS1.
Step 3
Client
RS1
RS2
RS3
RS4
Client -> -ROOT- table on RS1:
Which .META. region can tell me about
row 00009 from table T1?
Step 4
Client
RS1
RS2
RS3
RS4
-ROOT- table on RS1 -> Client : .META. region M2
on RegionServer RS3 has that info.


Step 5
Client
RS1
RS2
RS3
RS4
Client -> .META. region M2 on RS3: I want to read row 00009 from table T1. Which region can I find it in, and what RegionServer is serving it?
Step 6
Client
RS1
RS2
RS3
RS4
.META. region M2 on RS3 -> Client: region T1R3 on RegionServer RS3.
Step 7
Client
RS1
RS2
RS3
RS4
Client -> Region T1R3 on RS3: I want to read row 00009.
Step 8
Client
RS1
RS2
RS3
RS4
Region T1R3 on RS3 -> Client: Ah, here you go.

HBase Configuration

- 包括HMaster與RegionServer
- 通常將HMaster與NN裝在一起，RegionServer與DN裝在一起



HBase configuration

- 解壓縮hbase-0.98.6-cdh5.3.2.tar.gz
- /home/hadoop/hbase/conf/regionservers
  - slave
- hbase-env.sh
  - export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
  - export HBASE_MANAGES_ZK=true
- hbase-site.xml
- 環境變數
- 同步所有hbase設定檔

hbase-site.xml

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://master:9000/hbase</value>
  </property>
<property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
<property>
    <name>hbase.zookeeper.quorum</name>
    <value>master</value>
  </property>
</configuration>
```

- In .bashrc

```
export HBASE_HOME=/home/hadoop/hbase
export PATH=$PATH:$HBASE_HOME/bin
```

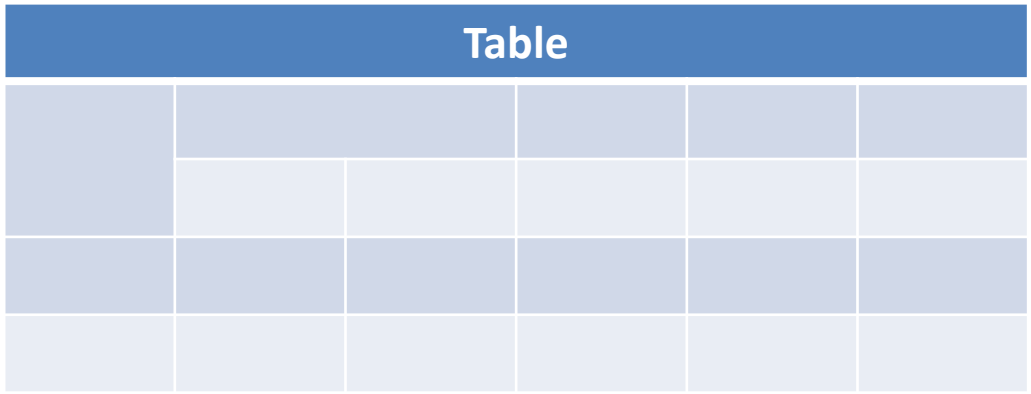
hdadm@master$ scp –r /home/hadoop/hbase hadoop@slave:/home/hadoop

Hbase CLI

- 啟動與關閉HBase
  - 確認HDFS已啟動
  - $start-hbase.sh
  - $stop-hbase.sh
- 確認HMaster與RegionServer皆啟動
  - JPS
  - http://master:60010/master-status

Hbase data model

- Table
  - Hbase organize data into tables



Hbase data model

- Row and rowkey
  - Data is stored to its row
  - Rows are ineditfied uniquely by their rowkey
  - Rowkeys are stored lexicographically
  - Rowkeys are always treated as byte[]

Table					
Rowkey					
R1					
R2					

Hbase data model

- Column family
  - Data within a row is grouped by column family (CF)
  - CF must be declared with table creation
  - CF cannot be add or delete
  - CF names are treated as String

Table					
Rowkey	CF1		CF2	CF3	
R1					
R2					

Hbase data model

- Column qualifier
  - Qualifier is used to address data
  - Qualifier need NOT be specified in advanced
  - Qualifier name is treated as byte[]
  - CF + qualifier can be seen as column in RDBMS
    - Represent by CF:qualifier

Table					
Rowkey	CF1		CF2	CF3	
	q1	q2	q1	q3	q4
R1					
R2					

Hbase data model

- Cell
  - Combination of rowkey, CF, qualifier uniquely identifies a cell
  - Data is stored in a cell, call value
  - Value is treated as byte[]

Table					
Rowkey	CF1		CF2	CF3	
	q1	q2	q1	q3	q4
R1	V1	v2			
R2	v1	v2	v3	v4	v5

Hbase data model

- Version
  - Values within a cell are versioned.
  - Versions are identified by timestamp, treated as long

Table					
Rowkey	CF1		CF2	CF3	
	q1	q2	q1	q3	q4
R1	V1	v2			
R2	v1	v2	v3	v4	V5-1 V5-2

Ver:1329088321289

Ver: 132908818321

Statistics example

- By user
- By time unit
  - Per Day, per month , per year…
- By action type
  - Like, comment, …

Initial Design

Table													
RK (userID)	TIME												
	20140901	20140902	…	20140931	20140900	20140901	…	20140923	201401	201402	…	201412	…
123	3			4									
987					15			9					

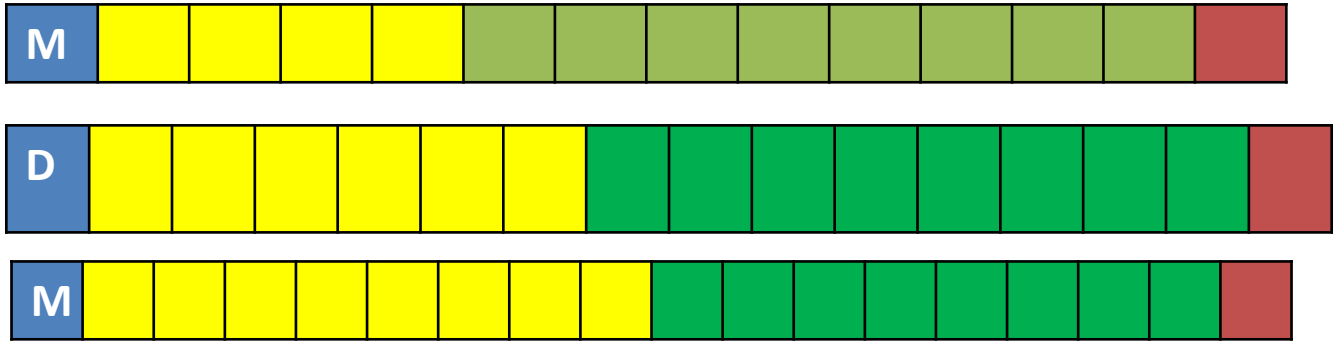
- Bad Design 1
  - Billion column per row is fine
  - Use column filter for query is BAD for performance
  - How about query by action ?

Initial Design

Table												
Rowkey (userID)	Like				Dislike				Comment			
	2014	201401	…	20140914	2014	201401	…	20140914	2014	201401	…	20140914
123	3			4								
987					15			9				

- Bad Design 2
  - CF should be defined first
    - How about add new action type ??
  - Number of CF should NOT be too much

Composition Rowkey



- Unit+TimeBase+UserID+Type
  - Unit : Char, (1 byte, H, D, M)
  - TimeBase: String
    - Length: 4 (Unit = M) or 6 (Unit = D) or 8 (Unit = H)
  - UserID: Long (8 bytes)
  - Type: Short (1 bytes)
    - 1 = Like, 2 = Dislike, 3 = comment

Better Design

qualifiers

Column Family

Table												
rowkey Unit+TimeBase+UserID+Type	Daily				Hour				Monthly			
	1	2	…	31	00	01	…	23	1	2	…	12
D+201409+123+Dislike	3			4								
H+20140908+123+Like					15			9				
M+2014+123+Comment									1	5		

rowkey

value

主鍵查詢
rowkey query

- Get 789's Like action counts from from 2014/9/7 to 2014/9/20
  - Full RK: D + 201409 + DEF + Like

Table										
Rowkey (Unit+TimeBase+UserID+Type)	…	Daily								…
		…	7	8	9	…	19	20	…	
D+201409+123+Dislike			**21**	**14**	**56**	**…**	**21**	**47**		
D+201409+123+Like			**25**	**12**	**78**	**…**	**98**	**112**		
D+201409+123+comment			**27**	**21**	**57**	**…**	**31**	**34**		
D+201409+789+Like			**26**	**41**	**29**	**…**	**7**	**35**		
H+20140908+123+Like										
M+2014+123+Comment										

部分主鍵查詢
Partial rowkey query

- Get 123's each action counts from from 2014/9/7 to 2014/9/20
  - Start RK: D+ 201409 + 123
  - END RK : D+ 201409 + 124

Table										
Rowkey (Unit+TimeBase+UserID+Type)	...	Daily								...
		...	7	8	9	...	19	20	...	
D+201409+123+Dislike			**21**	**14**	**56**	**...**	**21**	**47**		
D+201409+123+Like			**25**	**12**	**78**	**...**	**98**	**112**		
D+201409+123+comment			**27**	**21**	**57**	**...**	**31**	**34**		
D+201409+789+Like			**26**	**41**	**29**	**...**	**7**	**35**		
H+20140908+123+Like										
M+2014+123+Comment										

Hbase Shell

- $ hbase shell # start hbase shell
- In Hbase(main):
- create 't1','info'
- put 't1','GrandpaD','info:name', 'mark Twain'
- put 't1','GrandpaD','info:email','samuel@clemens.org‘
- put 't1','GrandpaD','info:password', 'ABC456‘
- put 't1','GrandpaD','info:password', ‘abc123‘
- put 't1','GrandpaD',‘**INFO**:password', ‘abc123‘ **FAIL**

- get 't1','GrandpaD‘
- get 't1', 'GrandpaD', {COLUMN => 'info:password'}
- get 't1', 'GrandpaD', {COLUMN => 'info:password', VERSIONS => 3}

Java program access HBase

- git clone https://github.com/ogre0403/MR-sample.git
  - org.nchc.train.hbase.accessHBase
    - Create HTable
    - Put into HTable
    - Get from Htable
    - Scan HBase

- Hadoop Basic
  - Hadoop: The Definitive Guide
  - Hadoop in action
  - Pro Apache Hadoop, 2ed
  - Hadoop in Practice, 2ed
- System maintenance
  - Hadoop Operations
- MapReduce algorithm
  - MapReduce Design Patterns
- HBase Basic
  - HBase: The Definitive Guide
  - HBase in Action
- Ecosystem
  - PIG: Programming Pig
  - Hive: Programming Hive
  - Mahout: Mahout in Action
- 系統實踐
  - Hadoop技术内幕：深入解析Hadoop Common和HDFS架构设计与实现原理
  - Hadoop技术内幕：深入解析MapReduce架构设计与实现原理
  - Hadoop技术内幕：深入解析YARN架构设计与实现原理