Hadoop/MapReduce 建置與開發實務

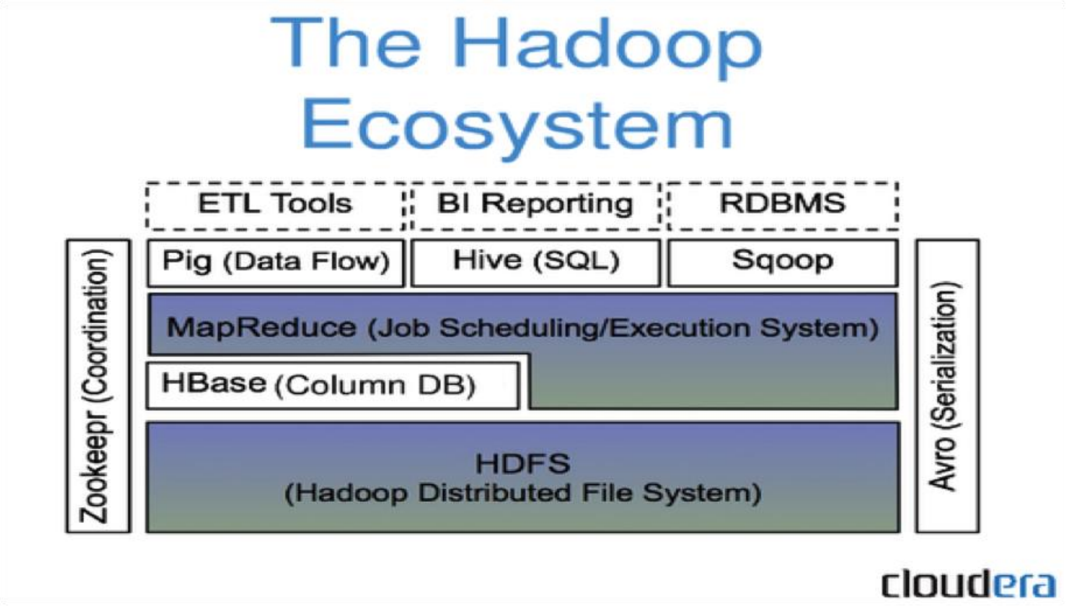
國家高速網路中心

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

What is Hadoop ecosystem

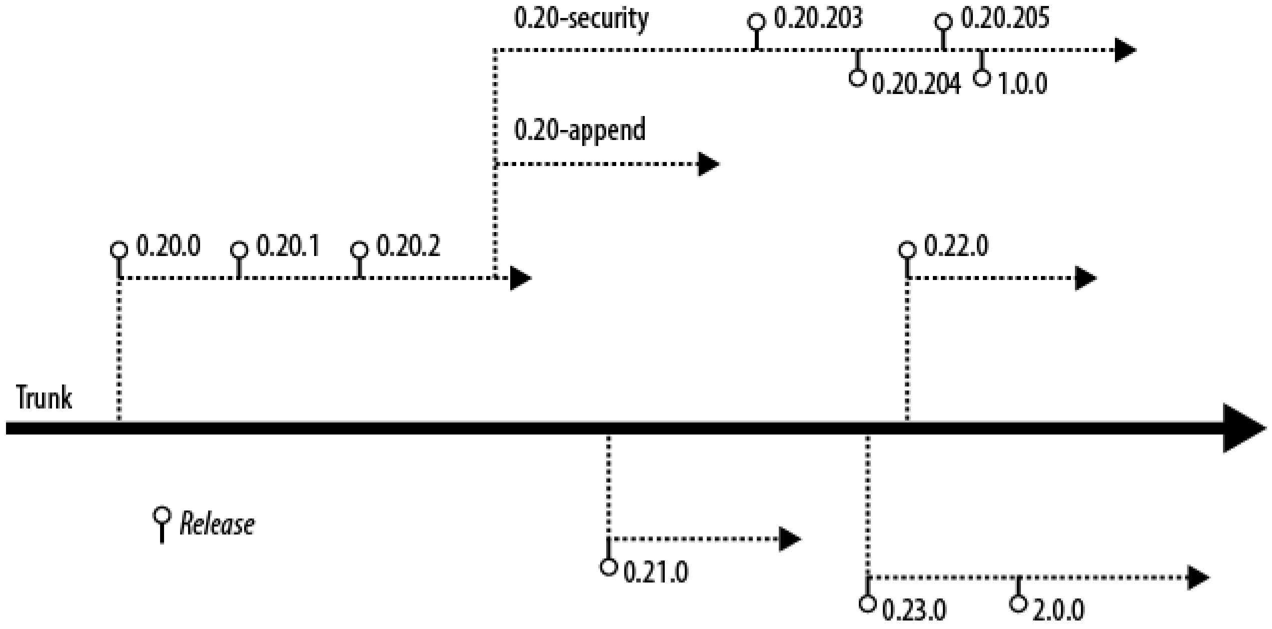


Google	OpenSource
GFS	HDFS
MapReduce	Hadoop MapReduce
BigTable	HBase
Chubby	Zookeeper

Hadoop distribution

- Apache
  - Hadoop 2.6 released at 2014/11/14
  - HBase 1.0 released at 2015/2/22
- Cloudera: CDH
  - 今天採用CDH 4.7
    - Hadoop-2.0.0-cdh4.7.0
    - Hbase-0.94.15-cdh4.7.0
  - 最新CDH 5.3.2
    - Hadoop-2.5.0-cdh5.3.2
    - HBase-0.98.6-cdh5.3.2
- Hortonworks: HDP
- MapR

Hadoop version



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

Hadoop version feature

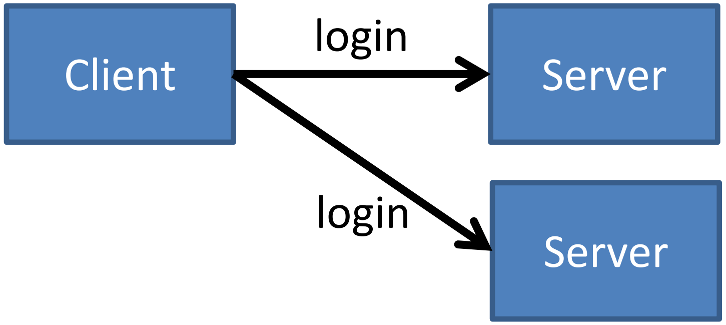
Feature	0.2	0.21	0.22	0.23	1	2	CDH3	CDH4
Production quality	X				X		X	X
HDFS append		X	X	X	X	X	X	X
Kerberos security		X[a]	X	X	X	X	X	X
HDFS symlinks		X	X	X		X		X
YARN (MRv2)				X		X		X
MRv1 daemons[b]	X	X	X		X		X	X
Namenode federation				X		X		X
Namenode HA				X		X		X

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

- 設定 ssh key登入
  - 在master 和slave
  - cd .ssh
  - rm ./*

- 解壓hadoop-2.0.0-cdh4.7.0.tar.gz
  - cp /opt/ hadoop-2.0.0-cdh4.7.0.tar.gz /home/hadoop
  - tar zxvf hadoop-2.0.0-cdh4.7.0.tar.gz
  - mv hadoop-2.0.0-cdh4.7.0 hadoop
  - /home/hadoop/hadoop



Client

login

Server

login

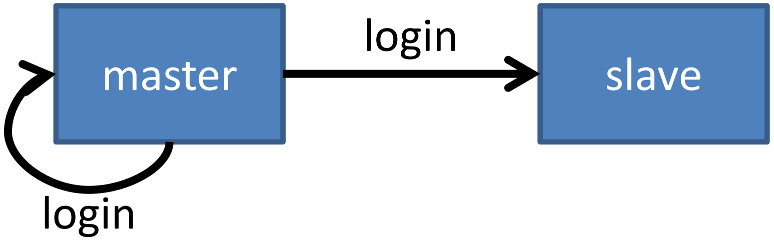
Server

1. 在Client 上 產生 公鑰(id_rsa.pub) 與 私鑰(id_rsa)

```
$ ssh-keygen –t rsa
```

2. 將公鑰由Client送到Server上並公開

```
user@client  $ scp id_rsa.pub hdadm@server:/home/user/
user@server $ cat ~/id_rsa.pub  >>  ~/.ssh/authorized_keys
user@server$ chmod 600 ~/.ssh/authorized_keys
```



master

login

slave

login

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

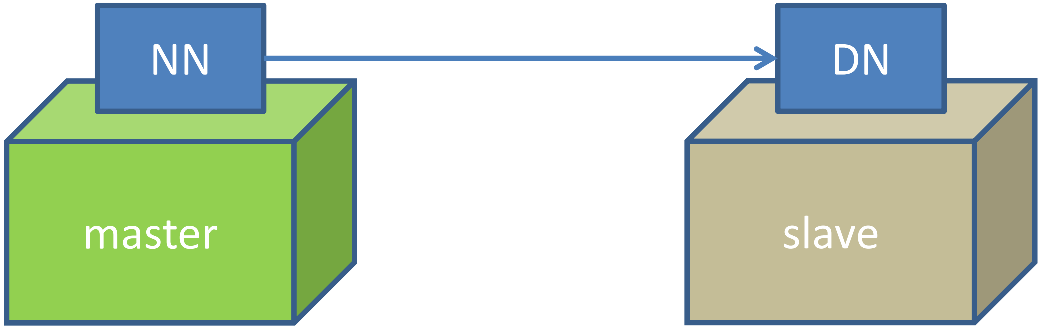
```
hadoop@master  $ scp id_rsa.pub hadoop@slave:/home/hadoop/
hadoop@slave $ cat ~/id_rsa.pub  >>  ~/.ssh/authorized_keys
hadoop@slave$ chmod 600 ~/.ssh/authorized_keys
```

Outline

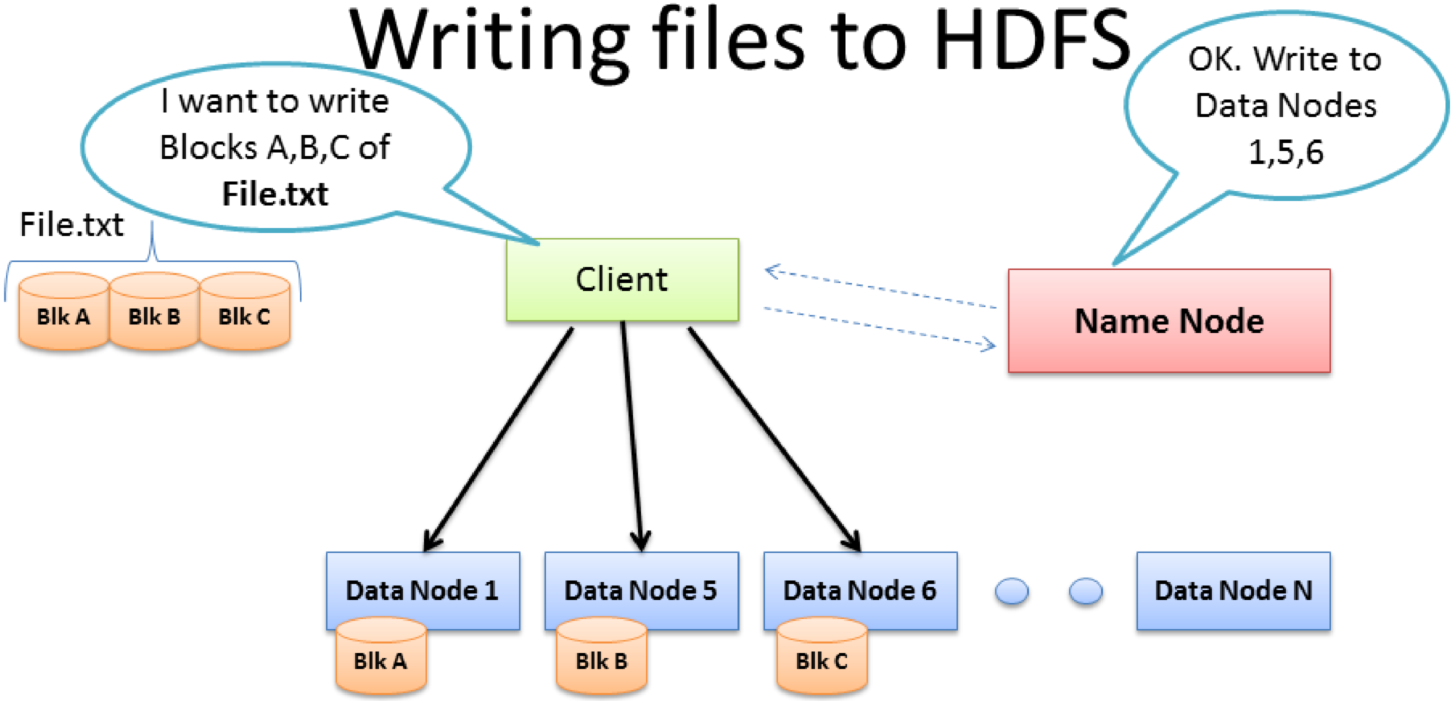
- Hadoop ecosystem 簡介
  - Intro, Version, distribution, OS base installation
- HDFS
  - Intro, install & configure,CLI, Java programing
- Mapreduce
  - YARN, Intro, install & configure, Java programing
- Hbase
  - Intro, install & configure , CLI, Java programing

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

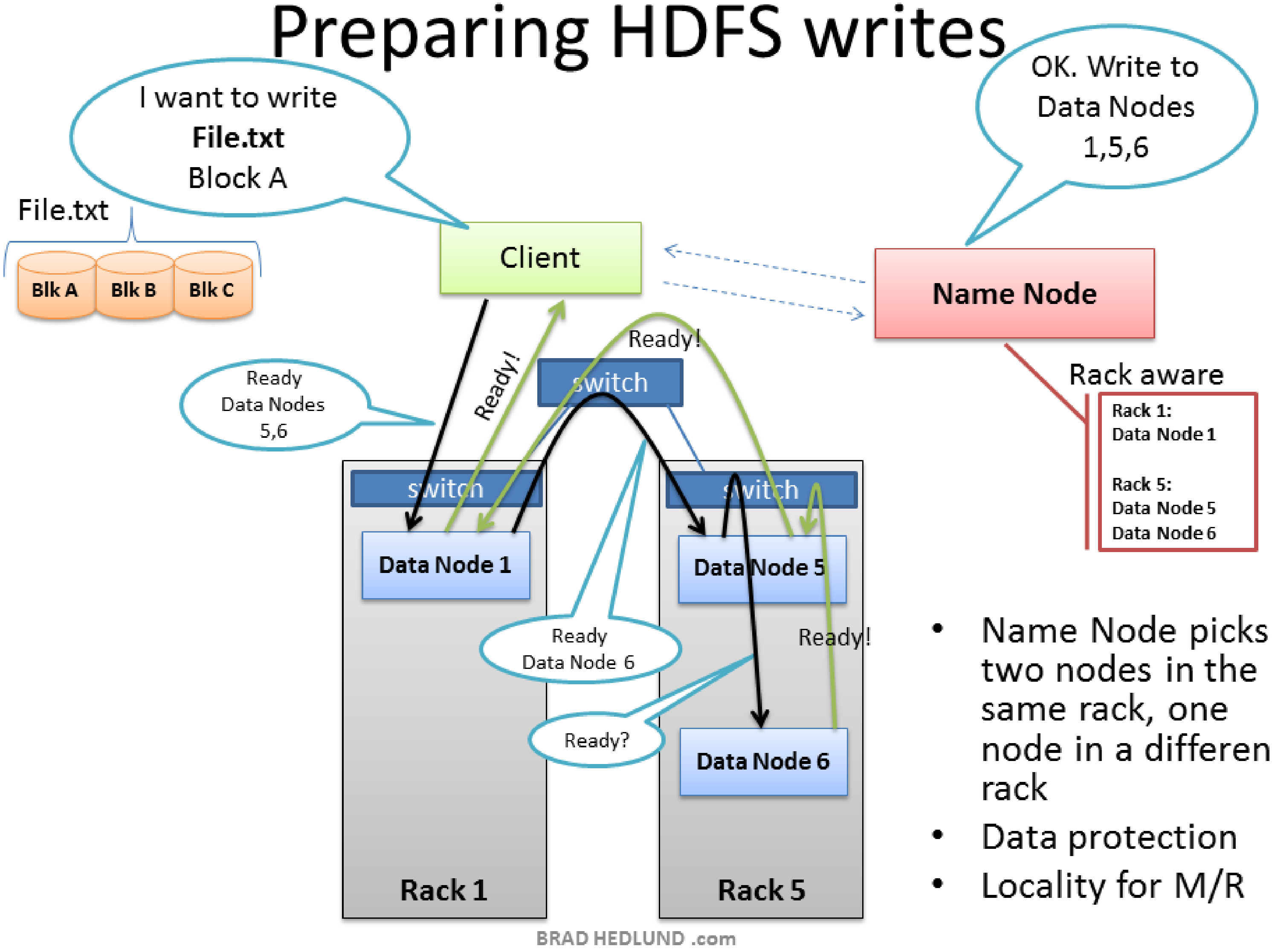


- Client consults Name Node
- Client writes block directly to one Data Node
- Data Nodes replicates block
- Cycle repeats for next block

BRAD HEDLUND .com

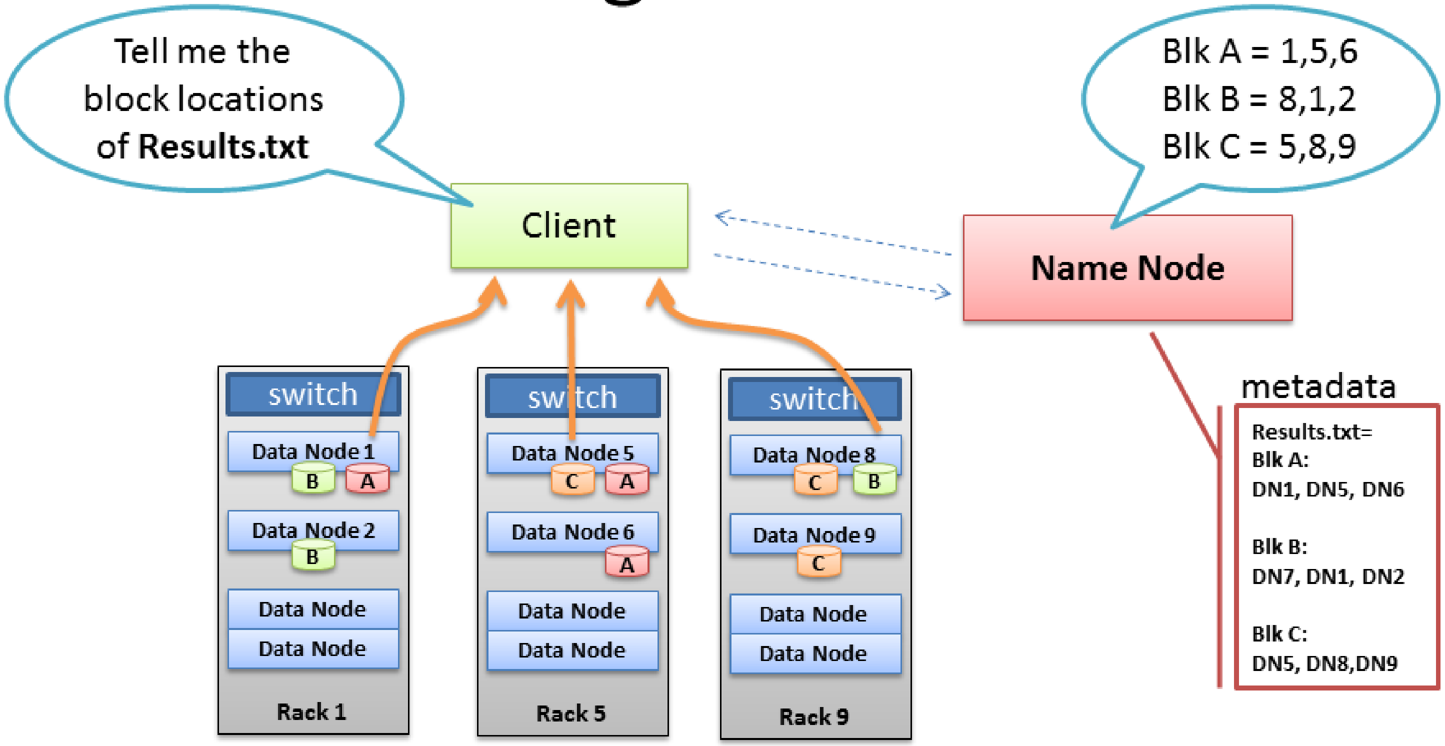
http://www.ewdna.com/2013/04/Hadoop-HDFS-Comics.html
http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

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

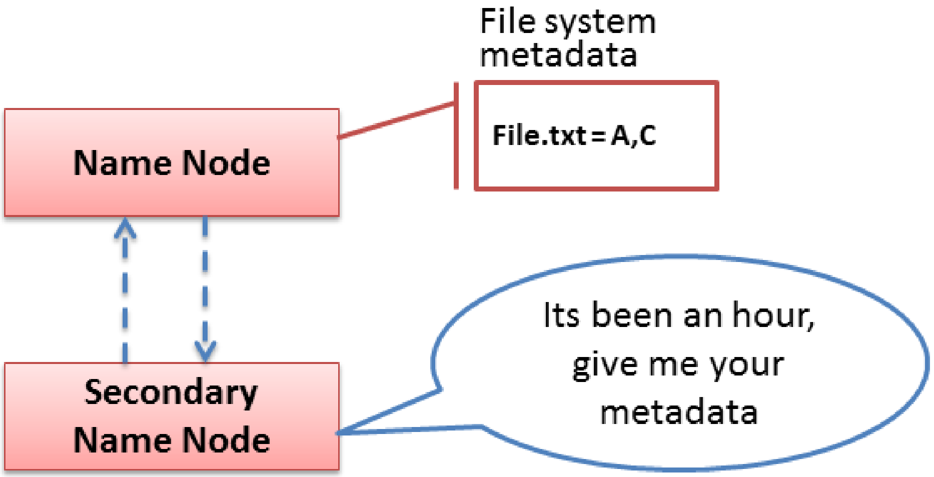


- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

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-2.0.0-cdh4.7.0.tar.gz
  - cp /opt/ hadoop-2.0.0-cdh4.7.0.tar.gz /home/hadoop
  - tar zxvf hadoop-2.0.0-cdh4.7.0.tar.gz
  - mv hadoop-2.0.0-cdh4.7.0 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

```
<?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>
```

core-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>
```

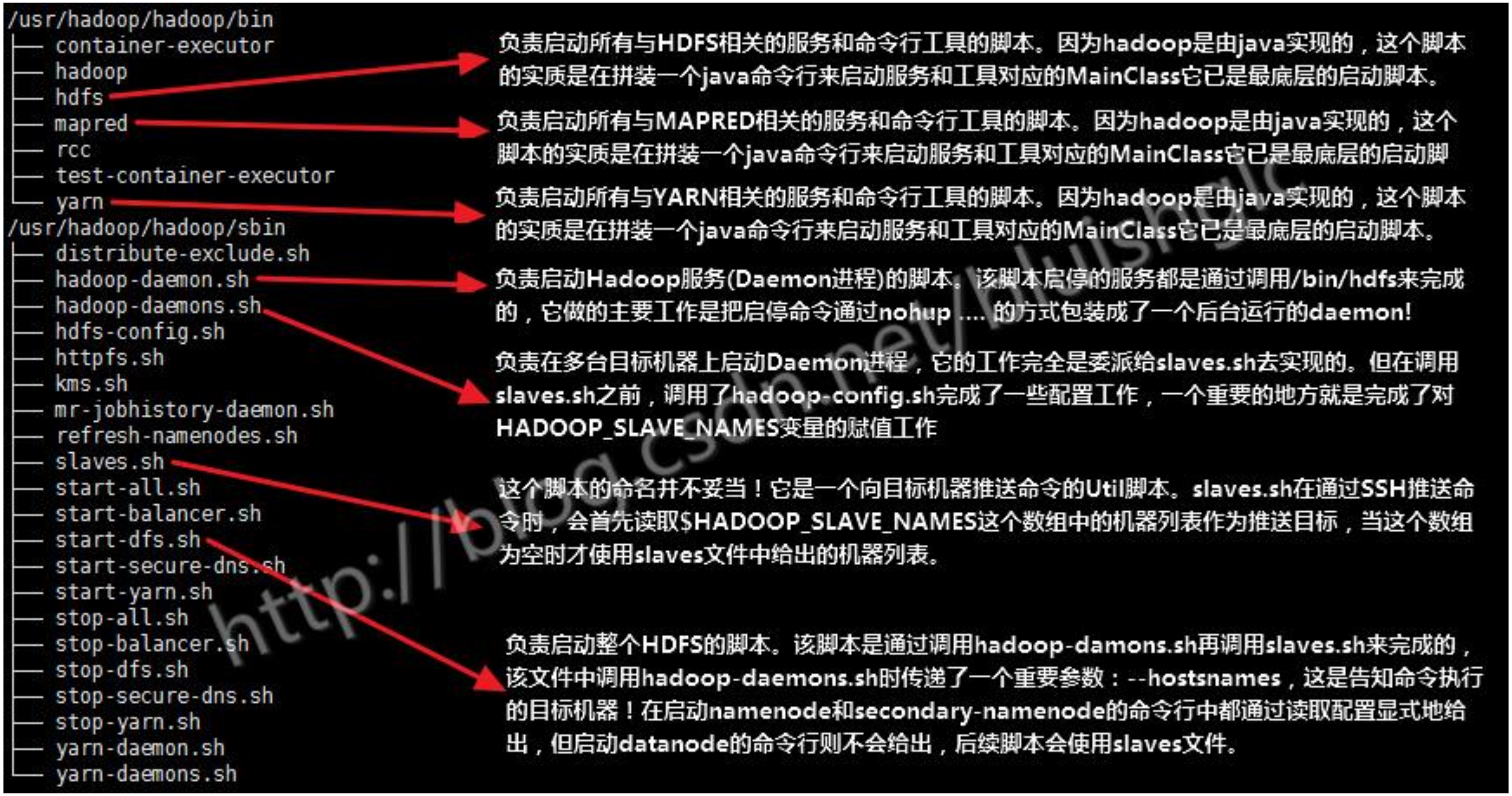
hdfs-site.xml

- 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 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

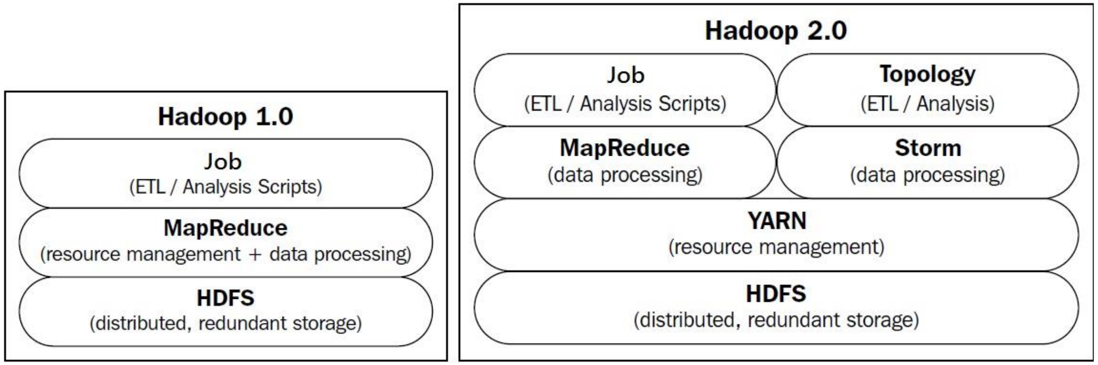
- 使用git 將範例clone下來
  - git clone https://github.com/ogre0403/MR-sample.git
  - git checkout NE-2015-CH01
  - org.nchc.train.hdfs.AccessHDFS
    - Copy local file to HDFS
    - Copy HDFS file to local
    - Generate sequence file

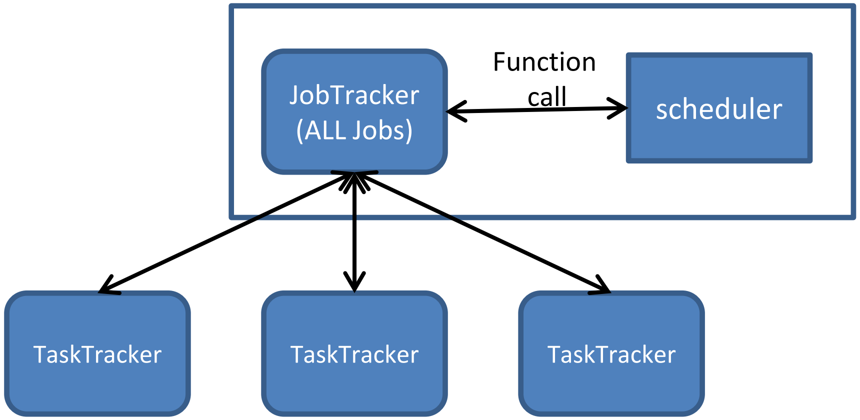
Outline

- 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

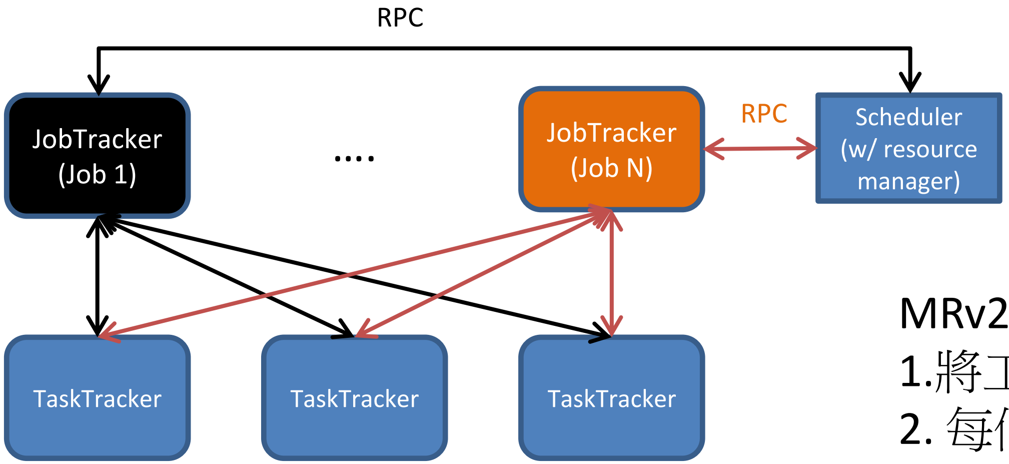
MRv1 v.s. MRv2

- 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

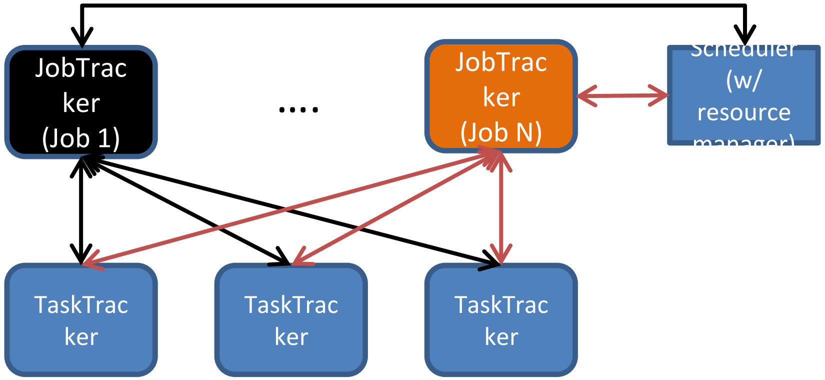




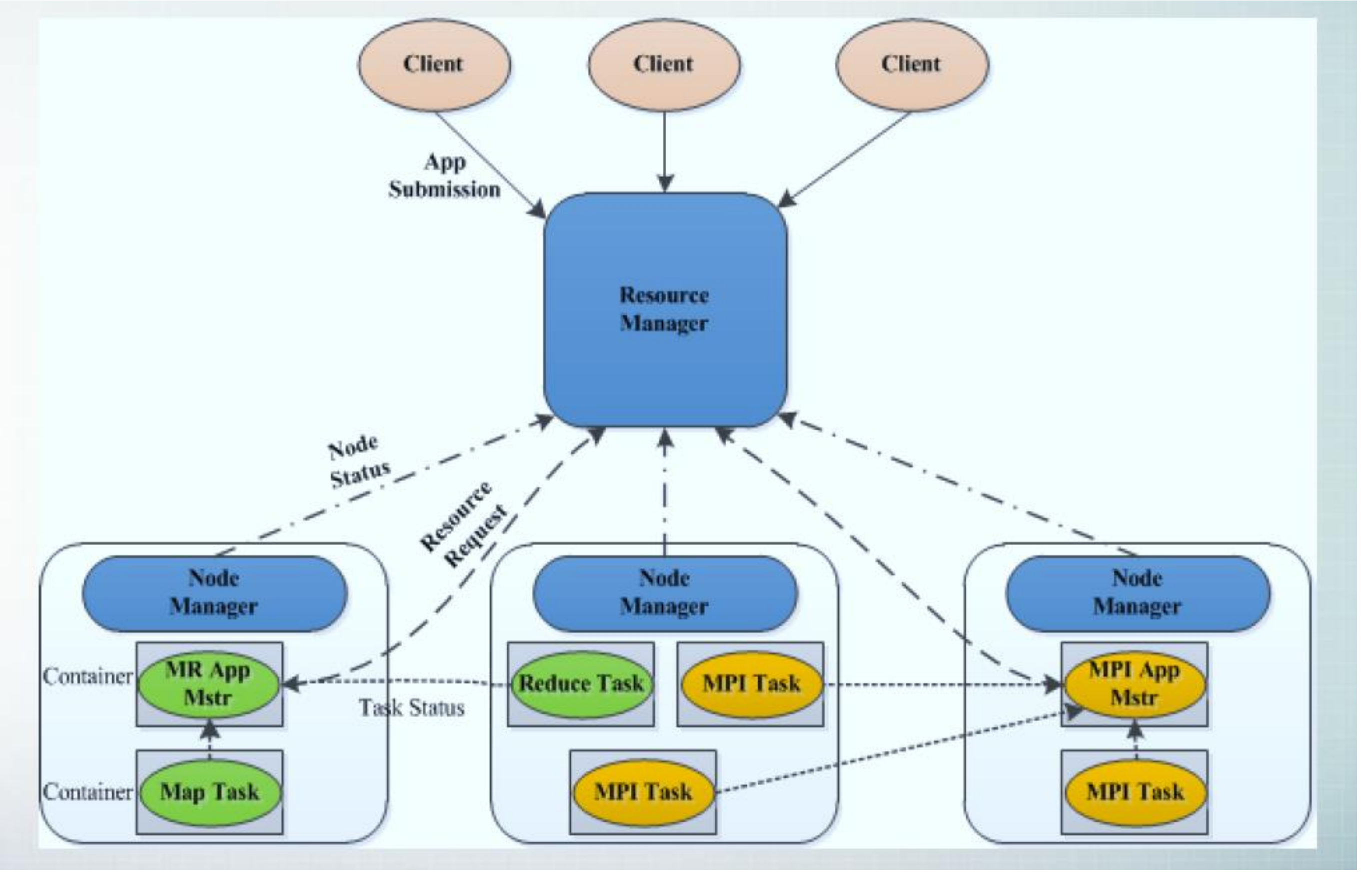
MRv1.
JobTracker負責工作控制與資源管理



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

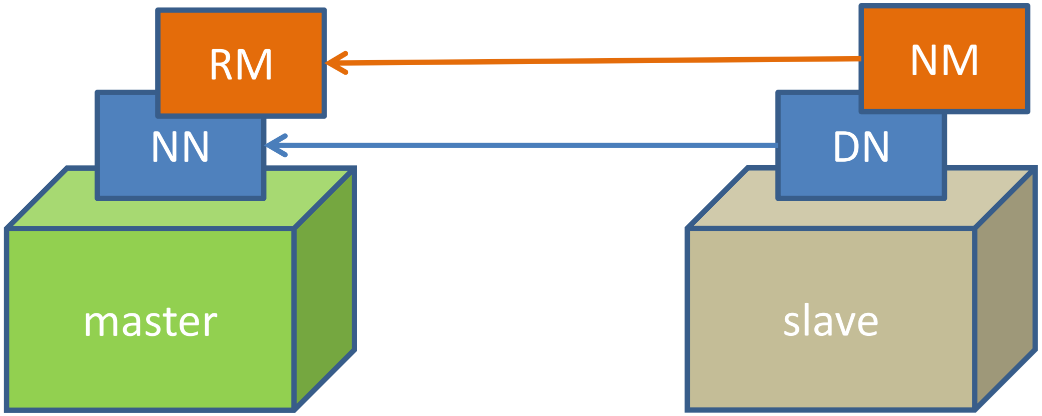


1. 由RM做全局的資源分配
2. NM定時回報目前的資源使用量
3. 每個JOB會有一個負責的AppMaster控制Job
4. 將資源管理與工作控制分開
5. YARN為一通用的資源管理系統
   可達成在YARN上運行多種框架



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

yarn-site.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>
```

- 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-2.0.0-cdh4.7.0.jar pi 10 1000

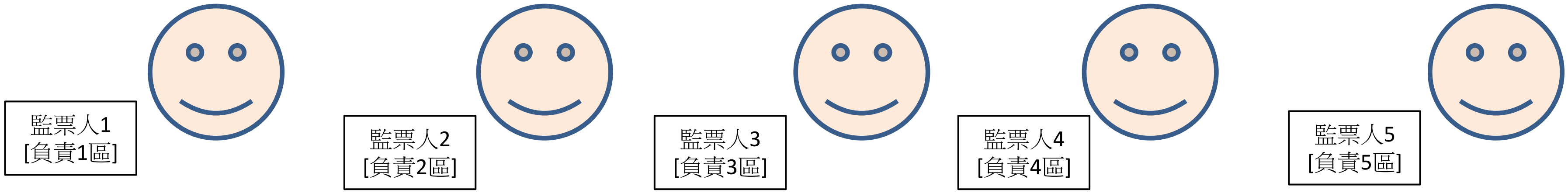
Outline

- 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

Id：A151
選2號
Id：B257
選5號



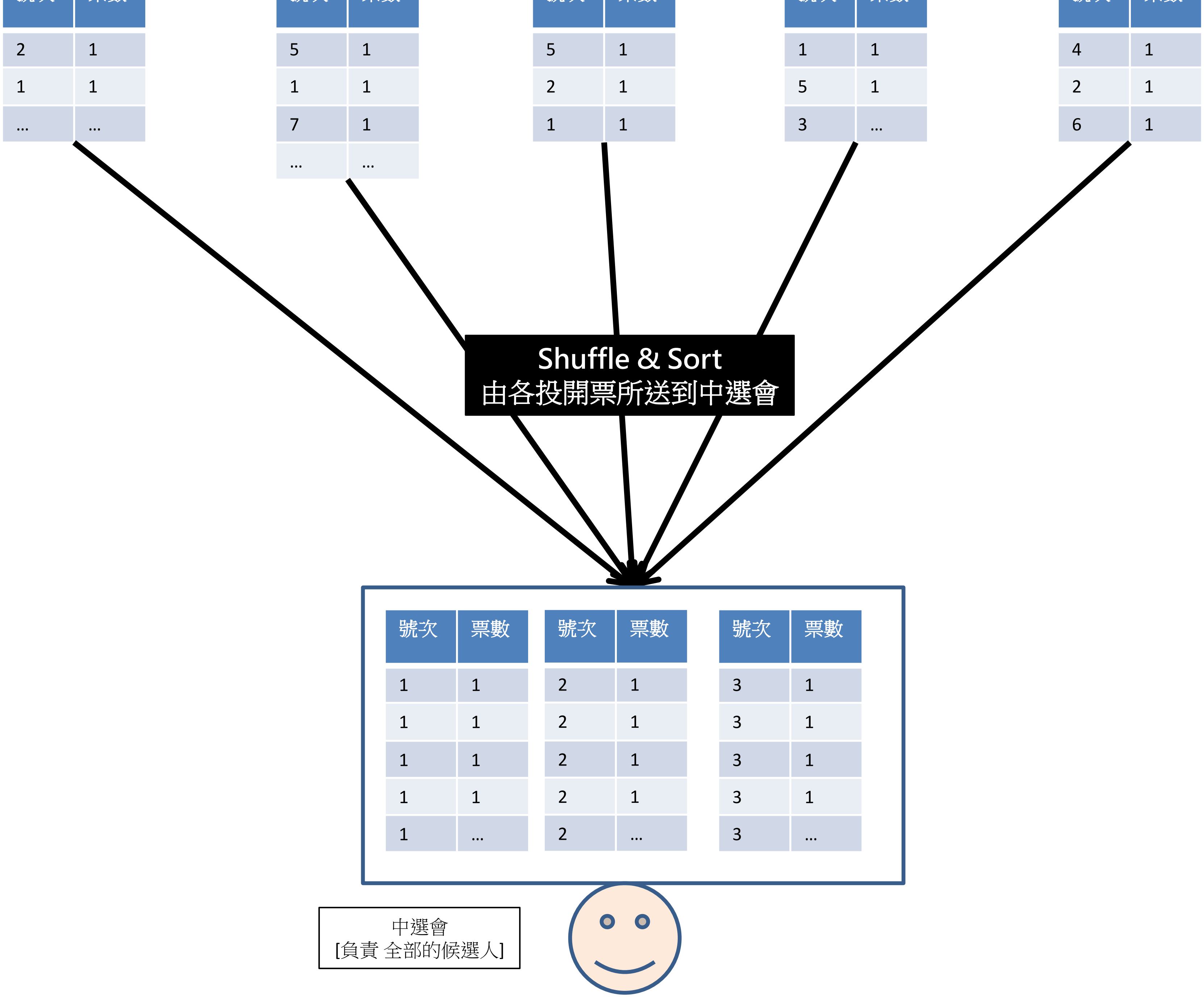
號次	票數
2	1
1	1
…	…

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	…

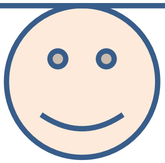
號次	票數
3	1
3	1
3	1
3	1
3	…

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
…	…

1	1
3	1
…	…

2	1
1	1
…	…

1	1
1	1
3	…

2	1
2	1
…	…

combine　combine　combine　combine　combine

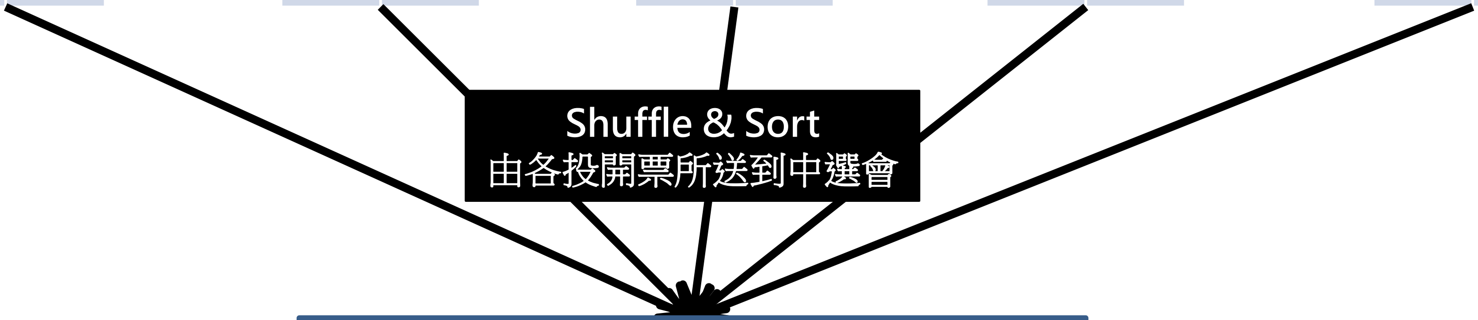
姓別	總分
1	1840
2	1740
3	

姓別	總分
1	1700
2	1520
3	

姓別	總分
1	1700
2	1520
3	

姓別	總分
1	1560
2	1240
3	

姓別	總分
1	1760
2	1660
3	



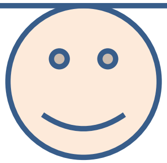
Shuffle & Sort
由各投開票所送到中選會

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	…

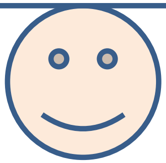
中選會
[負責 全部的候選人]



號次	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

算字數 - Mapper

Text file

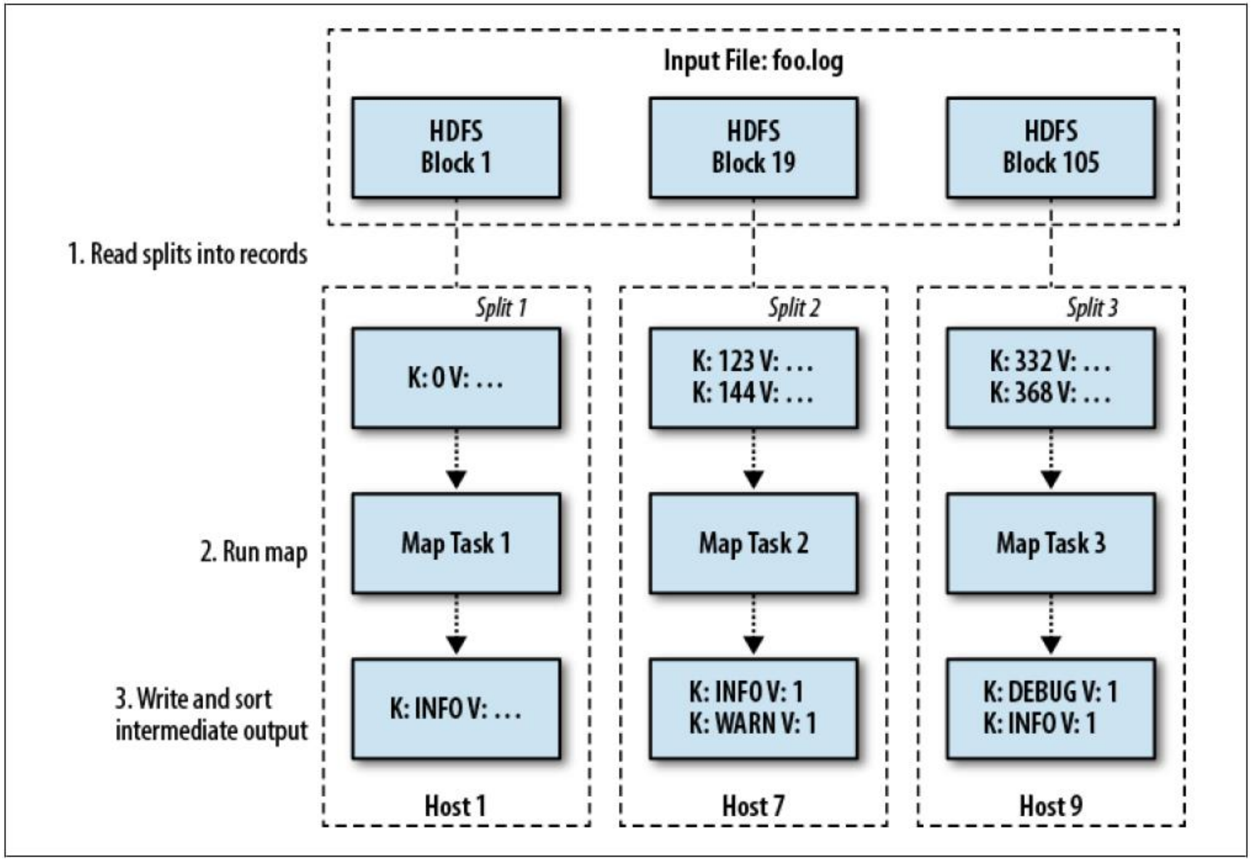
This is a book
Hello American

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

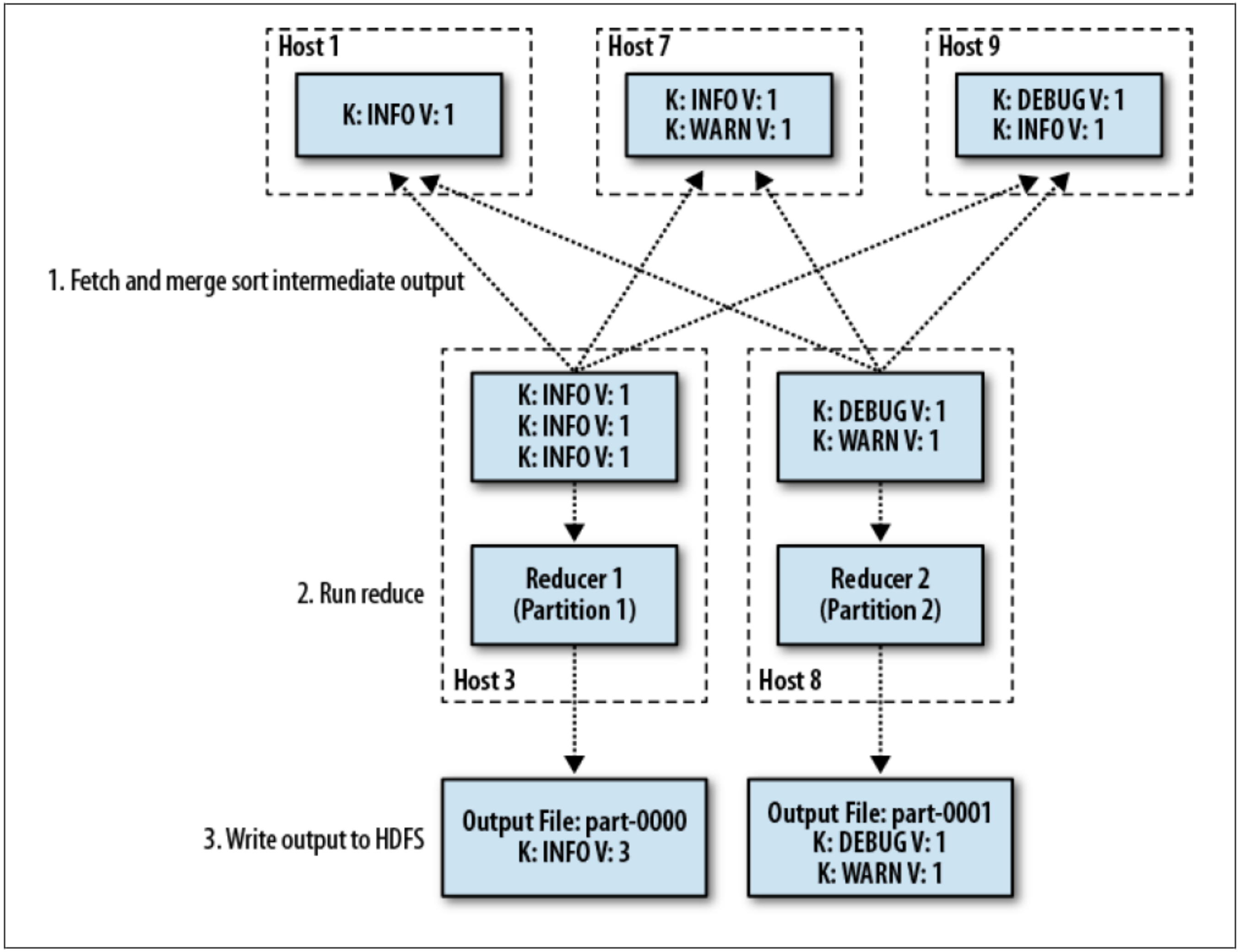
- 用來保證Reducer得到的資訊有下列三個特性
  - 若Reducer看到某個Key1，會看到相對應的所有value
    - Reducer 1收到 This這個字，會收到很多 1
  - 給定Key1，所有Key1的值都會被同一個Reducer處理
  - 同一個Reducer有可能處理多個Key值

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);
}
```

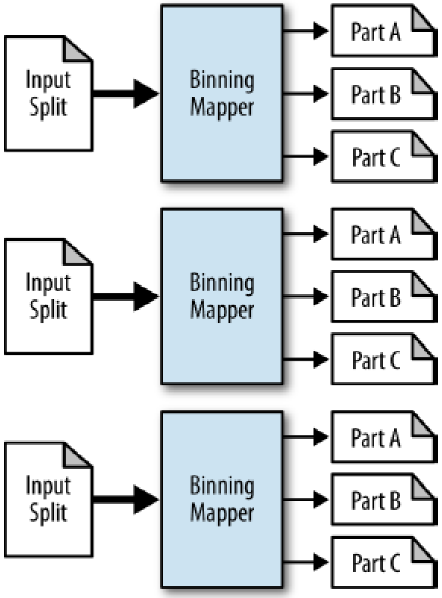
```
void Reduce (keyword, <list of value>){
        for each x in <list of value>:
            sum+=x;
        final_output.collect(keyword, sum);
}
```

算字數 – real code

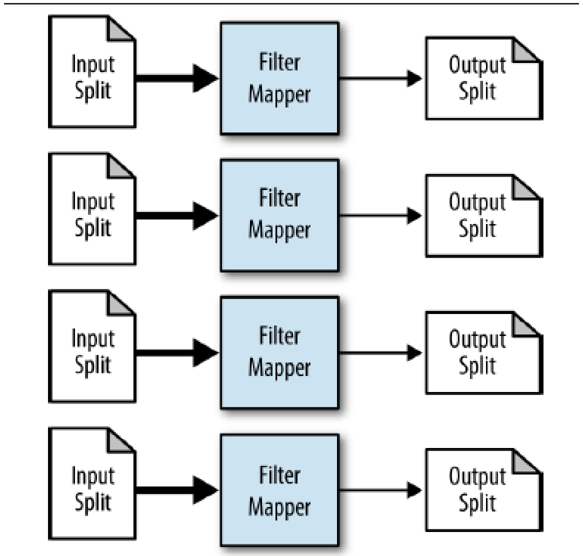
```
public void map(LongWritable key, Text value, Context context
                ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString()); // line to string token
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());    // set word 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
- git checkout NE-2015-CH01
  - org.nchc.train.mr
    - Word count
    - Read sequential file
  - org.nchc.train.mr.kmeans

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

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

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

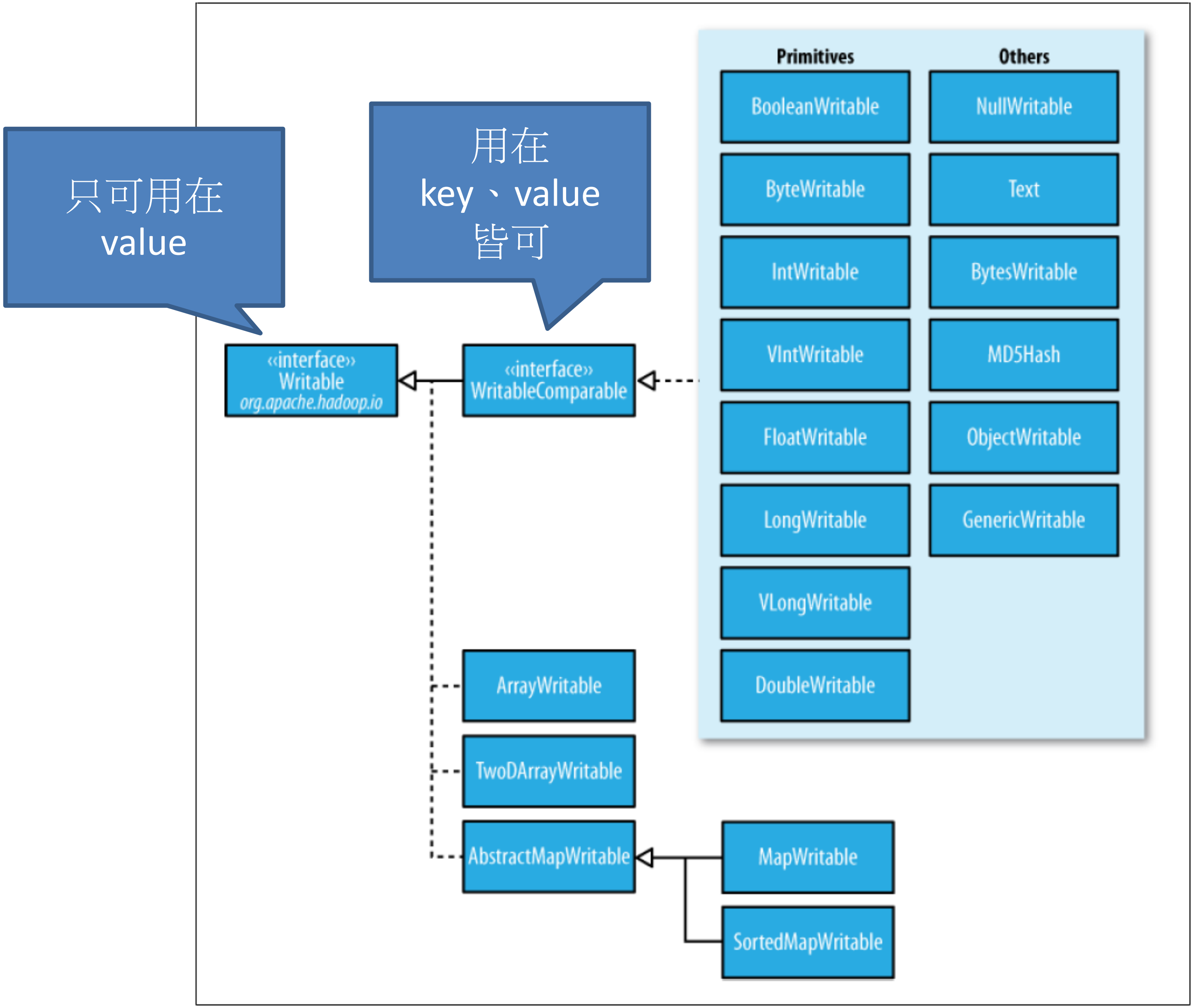
```
public  class WordCountMapper
            extends Mapper< Object, Text , Text, IntWritable>{

            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

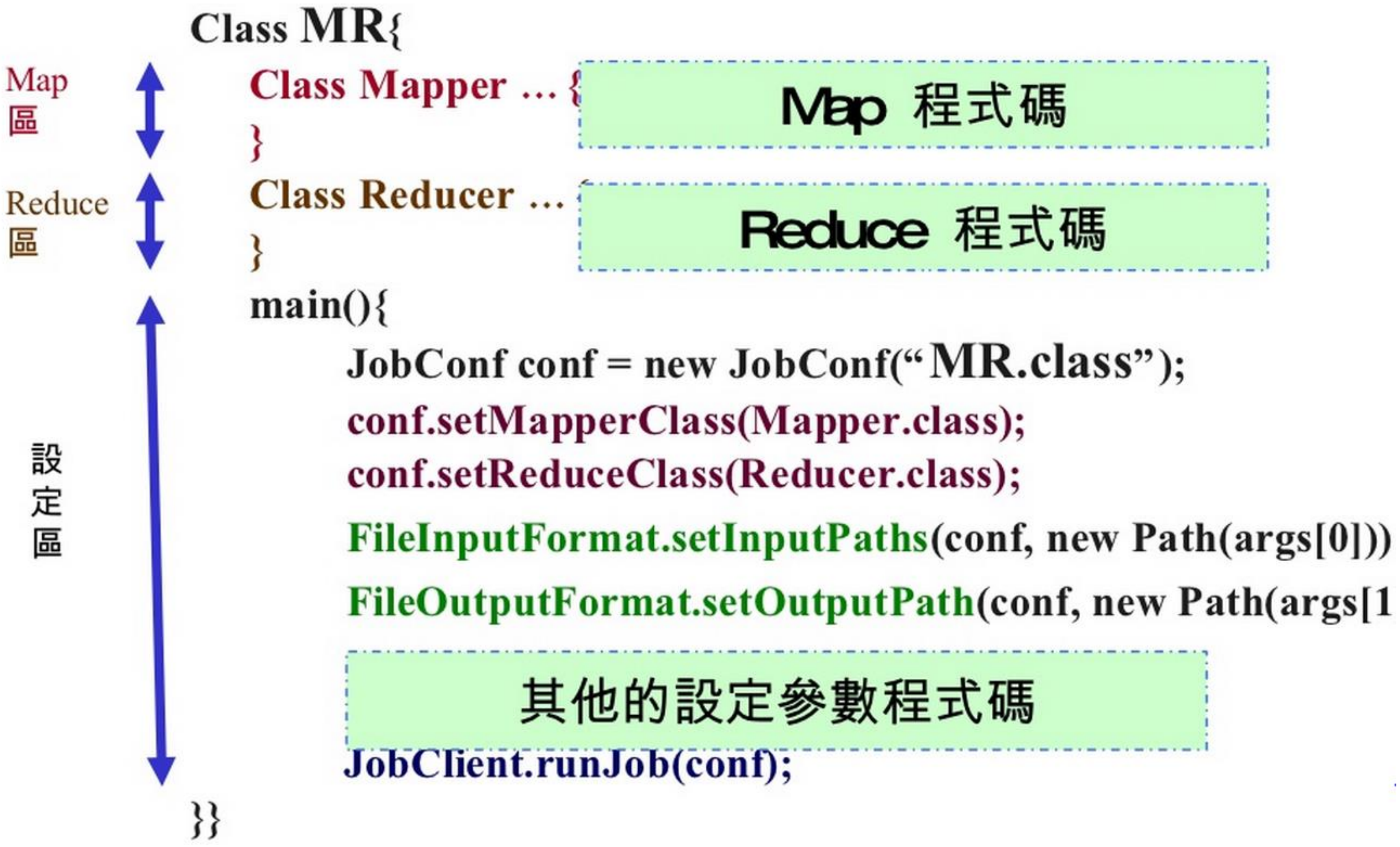
- 什麼是Text類型、什麼是IntWritable類型
  - Text: Wrapper for Java String class
  - IntWritable: Wrapper for Java int
- 序列化框架
  - 物件在網路上傳遞要透過serialize/deserializae
  - Java 本身有Serializable
  - Hadoop自行設計Writable 序列化框架
- 若內建的writable不合需求，需自行定義
  - Implement writable : 用在value
  - Implement writablecomparable: 用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



只可用在
value

用在
key、value
皆可

New and Old API



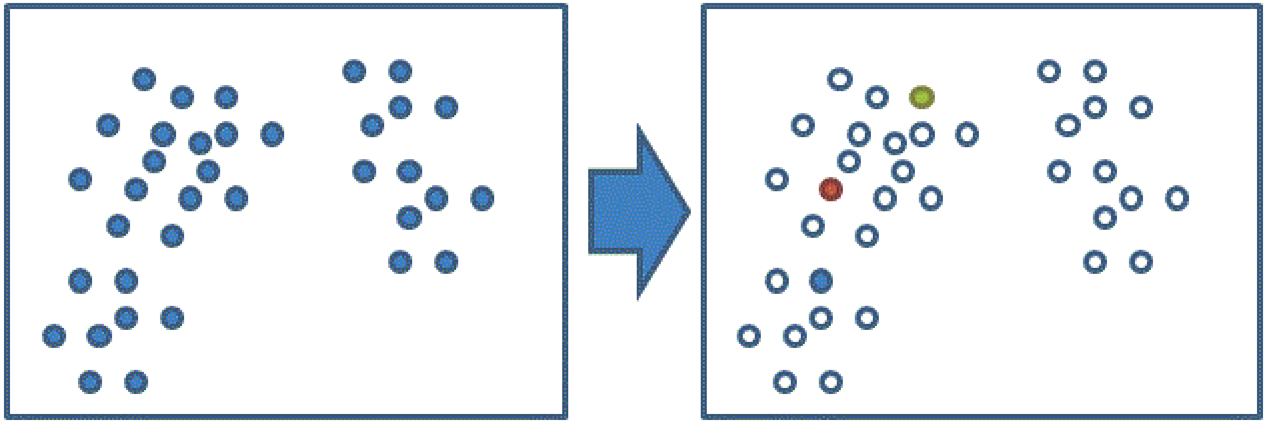
```
Class MR{
    Class Mapper … {
    }
    Class Reducer … 
    }
    main(){
        JobConf conf = new JobConf("MR.class");
        conf.setMapperClass(Mapper.class);
        conf.setReduceClass(Reducer.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]))
        FileOutputFormat.setOutputPath(conf,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

```
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);
        }
}
```

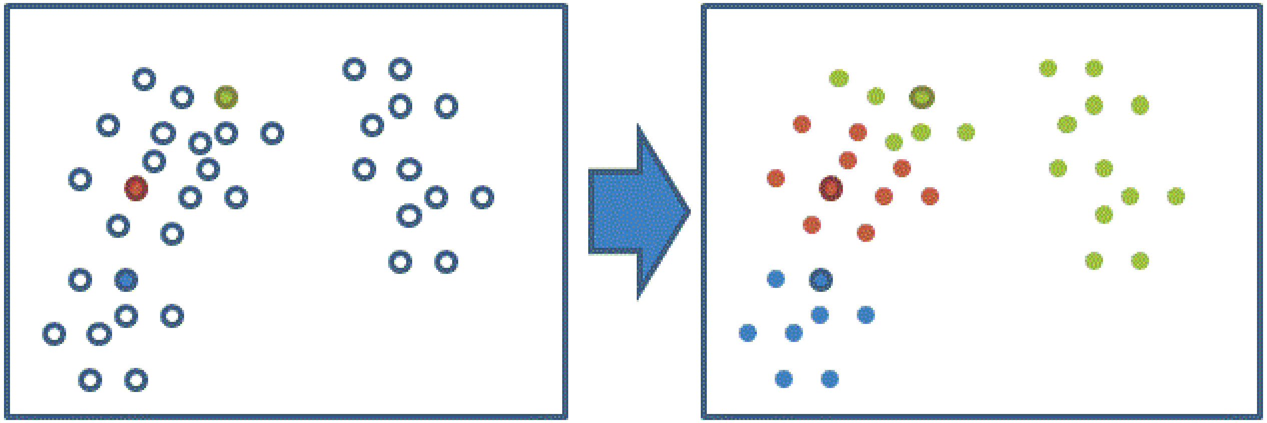
```
import org.apache.hadoop.mapred.*;

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

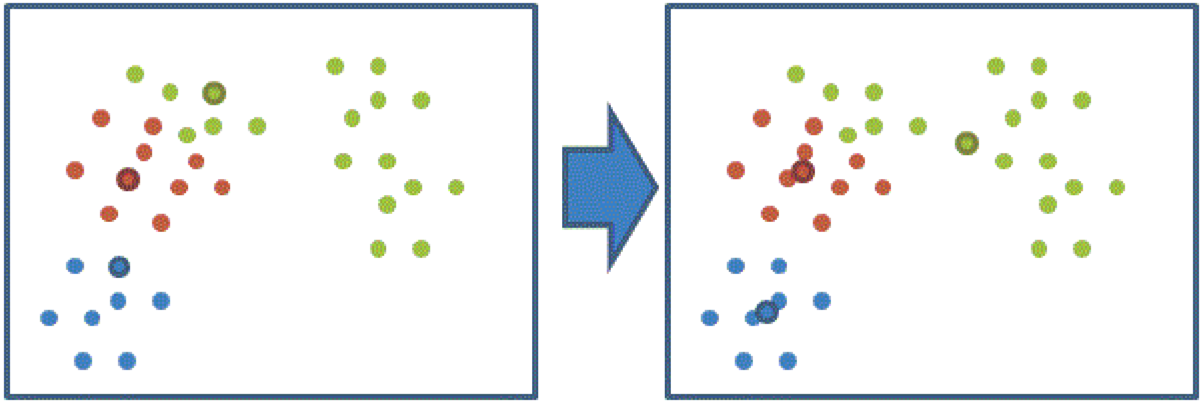
- 一個比較複雜且實際的演算法：K-means
  - 隨機選取資料組中的k筆資料當作初始群中心u_1~u_k



- 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)

MapReduce Version

- 用MR跑k-means的好處
  - 當dataset很大時，求距離很花計算資源
  - 將dataset分成許多小集合，把求距離做平行運算
  - 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>
  - 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
- 每次疊代就是一次MapReduce Job

High level tools based on MR

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

HIVE short DEMO

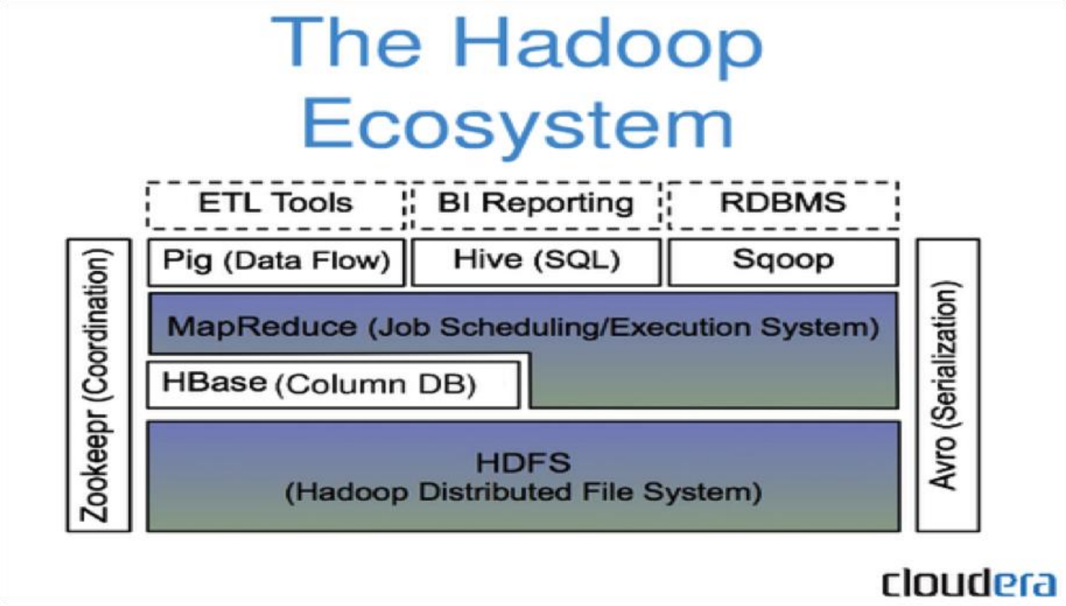
- wget http://archive.cloudera.com/cdh4/cdh/4/hive-0.10.0-cdh4.7.0.tar.gz
- tar zxvf hive-0.10.0-cdh4.7.0.tar.gz
- http://hive.3du.me/Lab-009.html#匯入-csv-資料到-hive-資料表

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

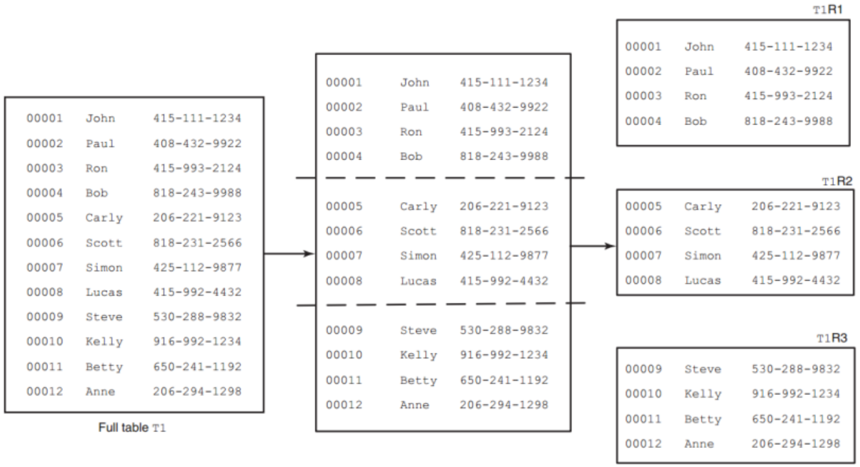
What is HBase

- 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



00001	John	415-111-1234
00002	Paul	408-432-9922
00003	Ron	415-993-2124
00004	Bob	818-243-9988
00005	Carly	206-221-9123
00006	Scott	818-231-2566
00007	Simon	425-112-9877
00008	Lucas	415-992-4432
00009	Steve	530-288-9832
00010	Kelly	916-992-1234
00011	Betty	650-241-1192
00012	Anne	206-294-1298

Full table T1

T1R1

00001	John	415-111-1234
00002	Paul	408-432-9922
00003	Ron	415-993-2124
00004	Bob	818-243-9988

T1R2

00005	Carly	206-221-9123
00006	Scott	818-231-2566
00007	Simon	425-112-9877
00008	Lucas	415-992-4432

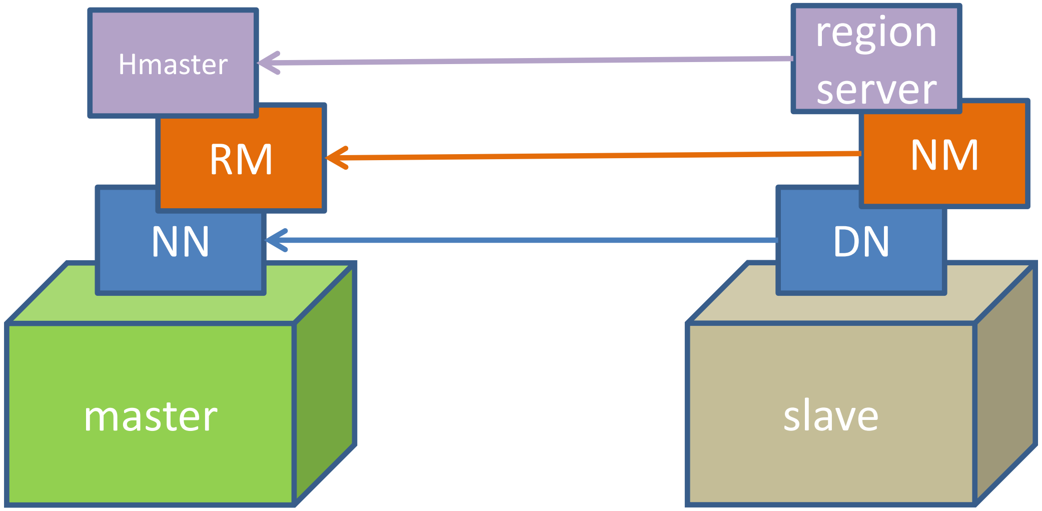
T1R3

00009	Steve	530-288-9832
00010	Kelly	916-992-1234
00011	Betty	650-241-1192
00012	Anne	206-294-1298

Table T1 split into 3 regions - R1, R2, and R3

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-0.94.15-cdh4.7.0.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
```

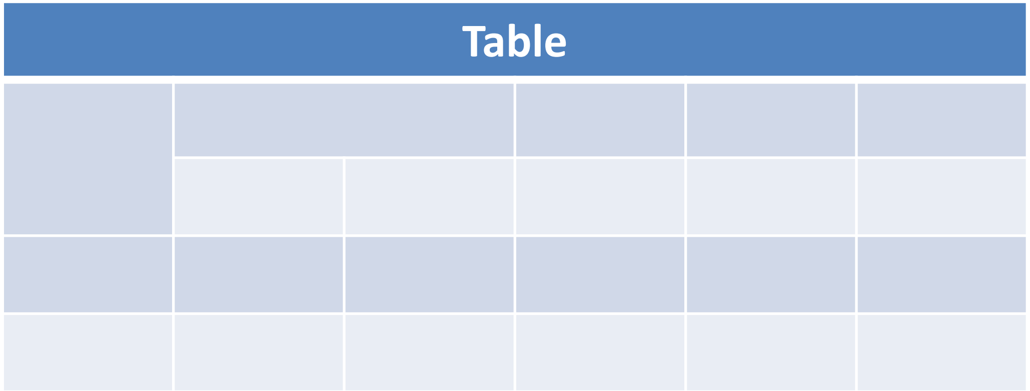
```
hdadm@master$ scp –r /home/hadoop/hbase
slave:/home/hadoop
```

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

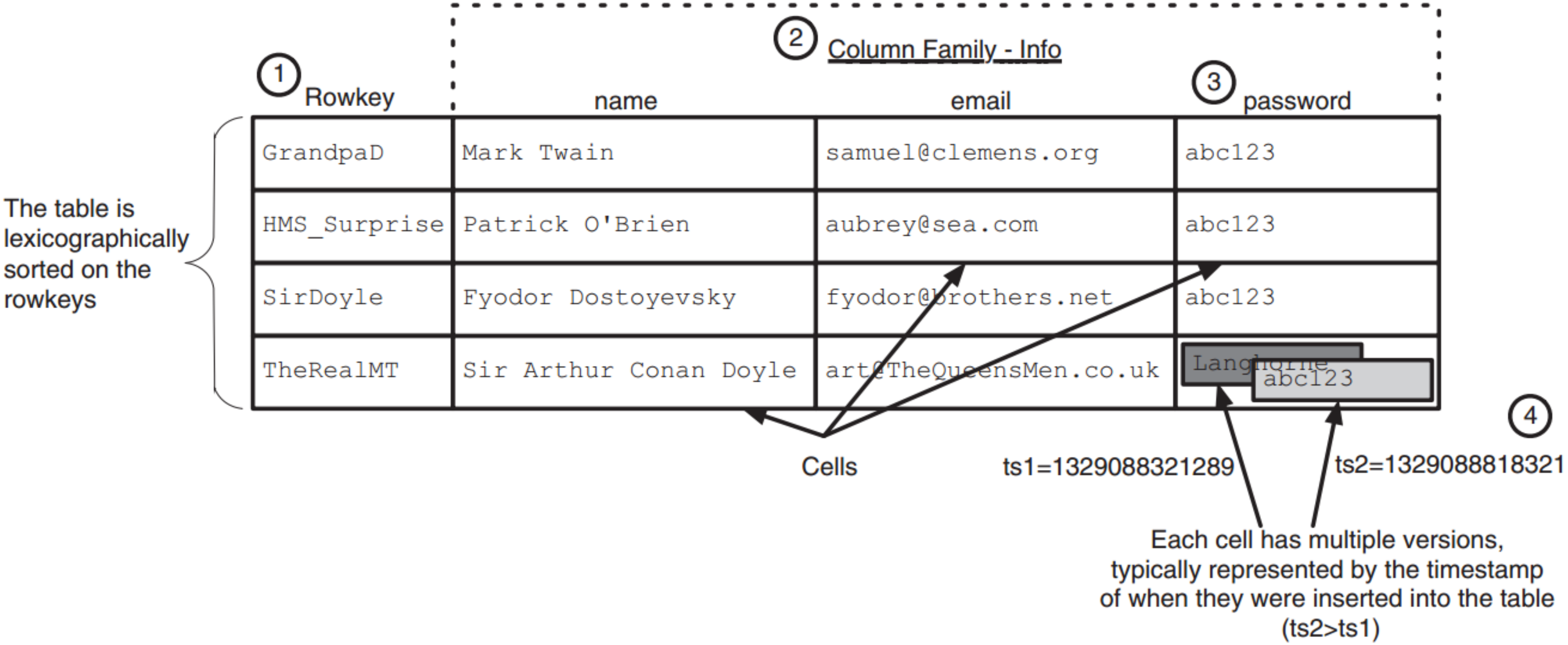
Table					
Rowkey	CF1		CF2	CF3	
	q1	q2	q1	q3	q4
R1	V1	v2			
R2	v1	v2	v3	v4	V5-1 V5-2

Ver:1329088321289

Ver: 132908818321

Hbase data model

- A example put all together



Column Family - Info

Rowkey	name	email	password
GrandpaD	Mark Twain	samuel@clemens.org	abc123
HMS_Surprise	Patrick O'Brien	aubrey@sea.com	abc123
SirDoyle	Fyodor Dostoyevsky	fyodor@brothers.net	abc123
TheRealMT	Sir Arthur Conan Doyle	art@TheQueensMen.co.uk	Langhorne abc123

The table is lexicographically sorted on the rowkeys

Cells

ts1=1329088321289

ts2=1329088818321

Each cell has multiple versions, typically represented by the timestamp of when they were inserted into the table (ts2>ts1)

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
- git checkout NE-2015-CH01
  - org.nchc.train.hbase.accessHBase
    - Create HTable
    - Put into HTable
    - Get from Htable
    - Scan HBase