

# Hadoop/MapReduce

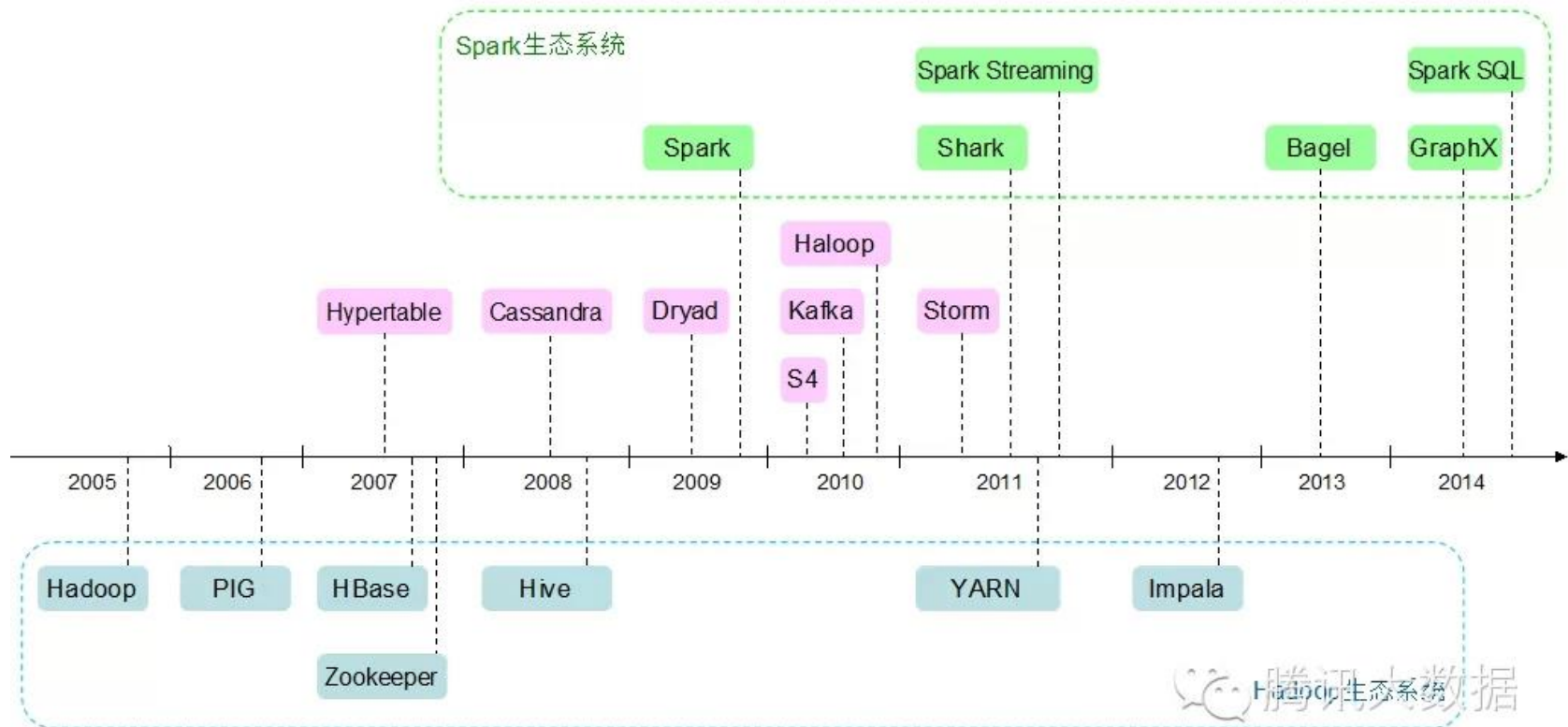
## 建置與開發實務

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 programming
- Mapreduce
  - YARN, Intro, install & configure, UI, Java programming
- Hbase
  - Intro, install & configure , UI & CLI, Java programming

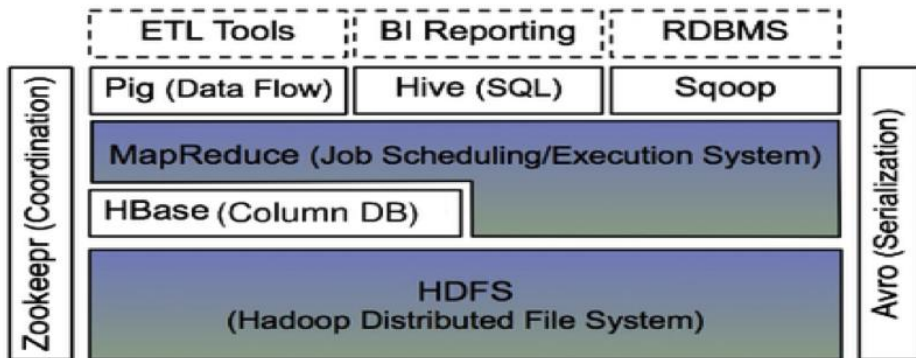
# Bigdata platform evolution



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

# What is Hadoop ecosystem

## The Hadoop Ecosystem



cloudera

Google	OpenSource
GFS	HDFS
MapReduce	Hadoop MapReduce
BigTable	HBase
Chubby	Zookeeper

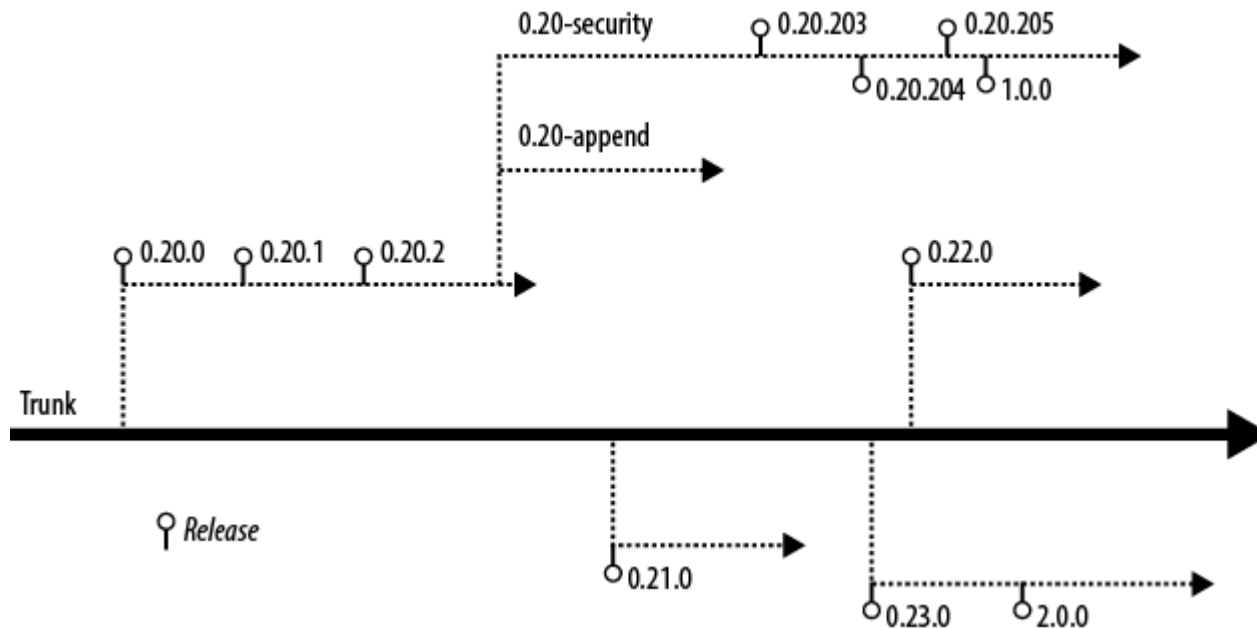
# Hadoop distribution

- Apache
  - Hadoop 2.7 released at 2015/4/23
  - 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 5.4
    - Hadoop-2.6.0-cdh5.4.0
    - HBase-1.0.0-cdh5.4.0
- Hortonworks: HDP
- MapR

cloudera



# Hadoop version



# 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		<a href="#">X[a]</a>	X	X	X	X	X	X
HDFS symlinks		X	X	X		X		X
YARN (MRv2)				X		X		X
<a href="#">MRv1 daemons[b]</a>	X	X	X		X		X	X
Namenode federation				X		X		X
Namenode HA				X		X		X

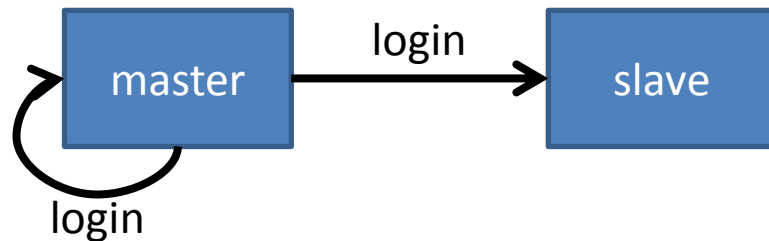
# OS base installation



- 使用 `hostname` 設定主機名稱
  - master / slave
  - 修改 `/etc/hosts`
- Install JDK (oracle 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



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

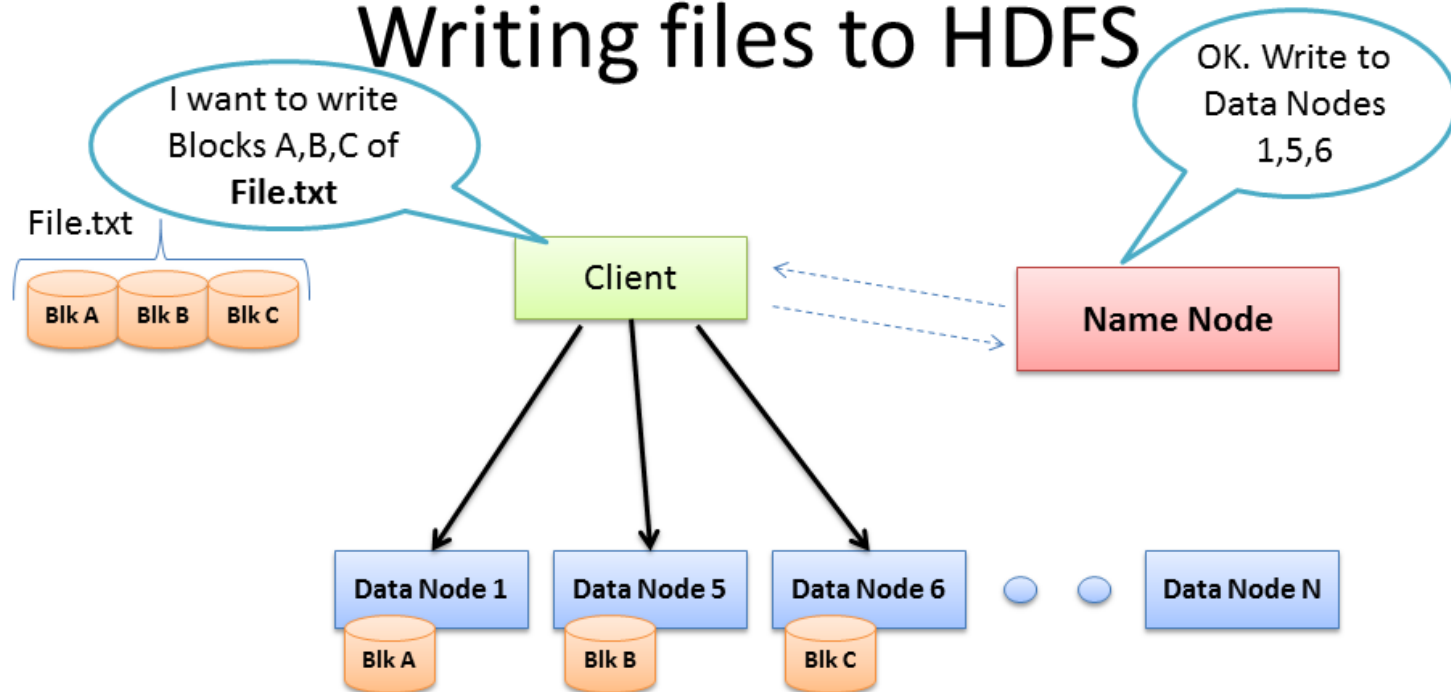
- Hadoop ecosystem 簡介
  - Intro, Version, distribution, OS base installation
- HDFS
  - Intro, install & configure, CLI, Java programming
- Mapreduce
  - YARN, Intro, install & configure, Java programming
- Hbase
  - Intro, install & configure , CLI, Java programming

# 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



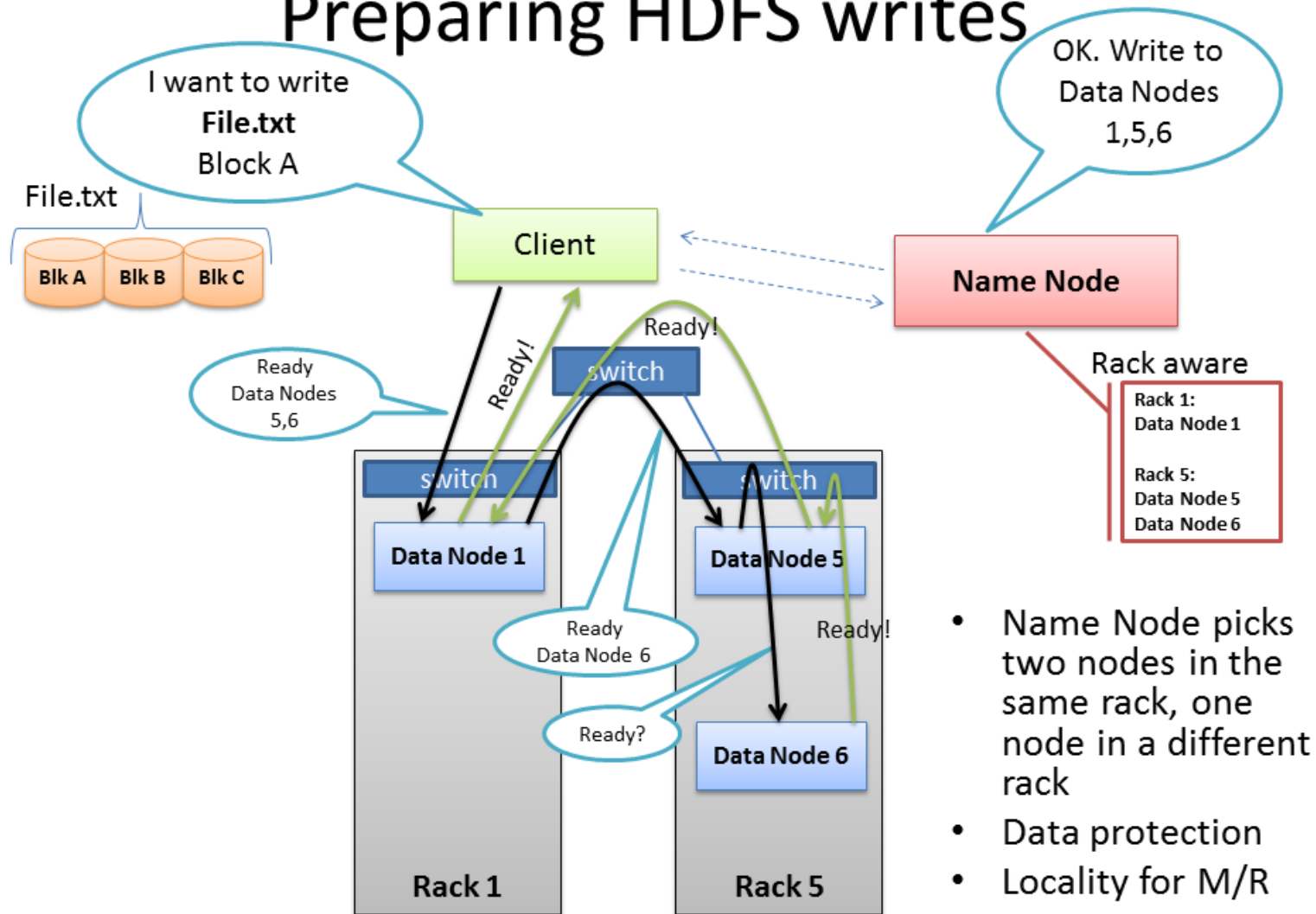
- Client consults Name Node
- Client writes block directly to one Data Node
- Data Nodes replicates block
- Cycle repeats for next block

BRAD HEDLUND .com

<http://www.ewdna.com/2013/04/Hadoop-HDFS-Comics.html>

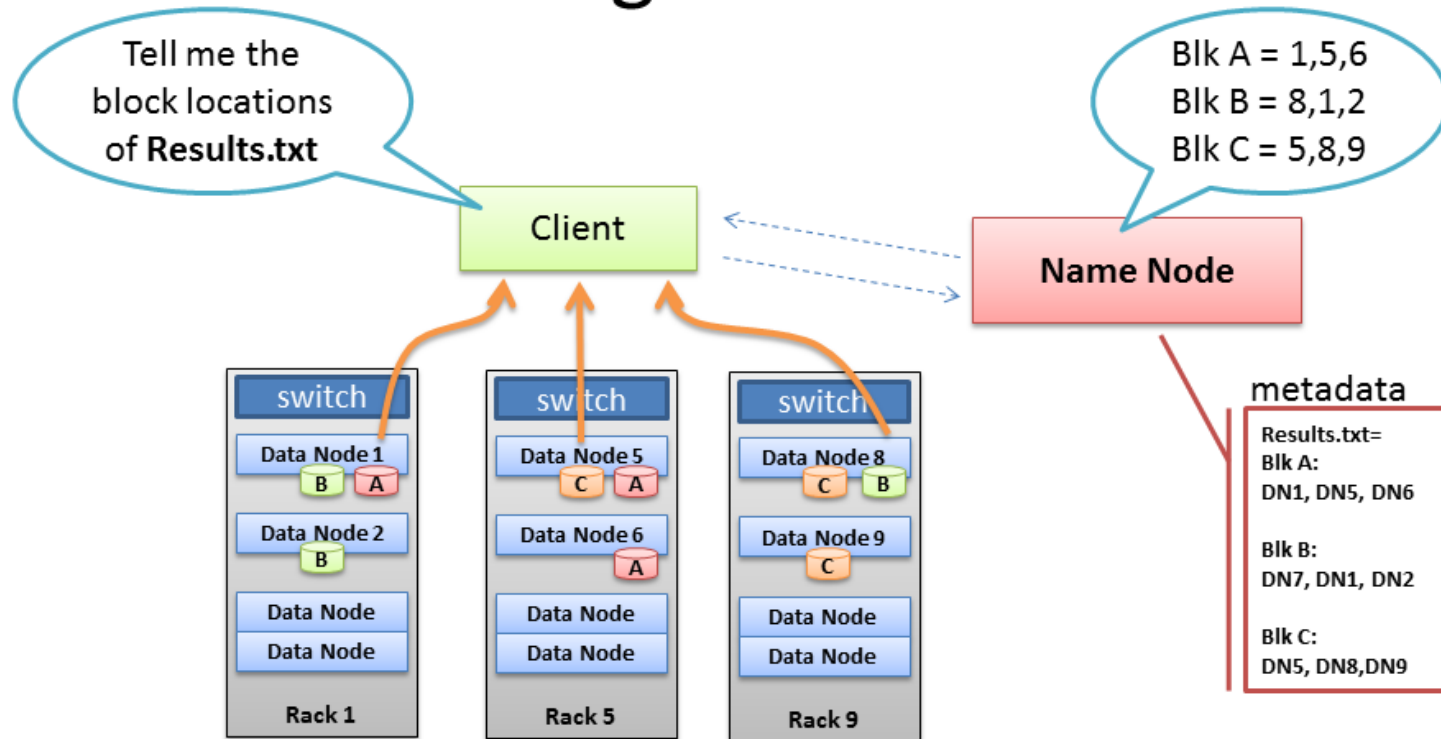
<http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/>

# 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



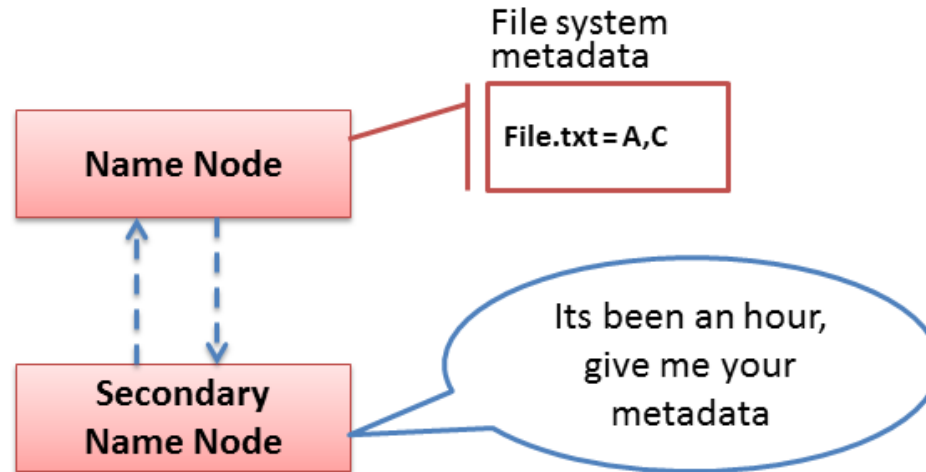
- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

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

<code>/usr/hadoop/hadoop/bin</code>	
<code>container-executor</code>	
<code>hadoop</code>	
<code>hdfs</code>	负责启动所有与HDFS相关的服务和命令行工具脚本。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。
<code>mapred</code>	负责启动所有与MAPRED相关的服务和命令行工具脚本。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。
<code>rcc</code>	
<code>test-container-executor</code>	
<code>yarn</code>	负责启动所有与YARN相关的服务和命令行工具脚本。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。
<code>/usr/hadoop/hadoop/sbin</code>	
<code>distribute-exclude.sh</code>	
<code>hadoop-daemon.sh</code>	负责启动Hadoop服务(Daemon进程)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!
<code>hadoop-daemons.sh</code>	
<code>hdfs-config.sh</code>	
<code>httpfs.sh</code>	
<code>kms.sh</code>	
<code>mr-jobhistory-daemon.sh</code>	
<code>refresh-namenodes.sh</code>	
<code>slaves.sh</code>	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
<code>start-all.sh</code>	
<code>start-balancer.sh</code>	
<code>start-dfs.sh</code>	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。
<code>start-secure-dns.sh</code>	
<code>start-yarn.sh</code>	
<code>stop-all.sh</code>	
<code>stop-balancer.sh</code>	
<code>stop-dfs.sh</code>	负责启动整个HDFS脚本。该脚本是通过调用hadoop-daemons.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：--hostnames，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。
<code>stop-secure-dns.sh</code>	
<code>stop-yarn.sh</code>	
<code>yarn-daemon.sh</code>	
<code>yarn-daemons.sh</code>	

# 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

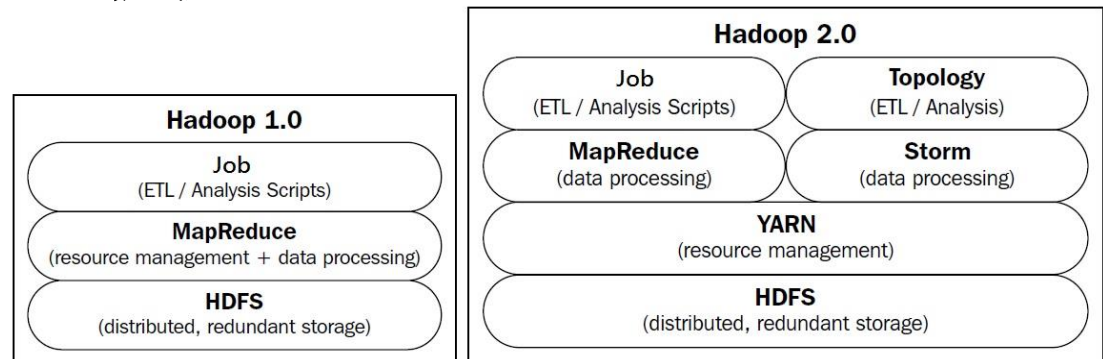
- MAVEN
  - 編譯、打包、自動部署、library相依性管理 工具
  - POM.xml
    - repository
    - Dependency
- 使用git 將範例clone下來
  - git clone <https://github.com/ogre0403/MR-sample.git>
  - git checkout NE-2015-TH03
  - org.nchc.train.hdfs.AccessHDFS
    - Copy local file to HDFS
    - Copy HDFS file to local
    - Generate sequence file

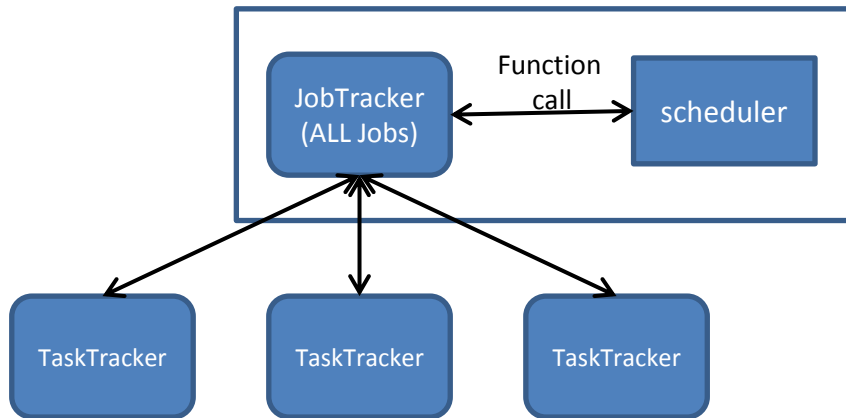
# Outline

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

# MRv1 v.s. MRv2

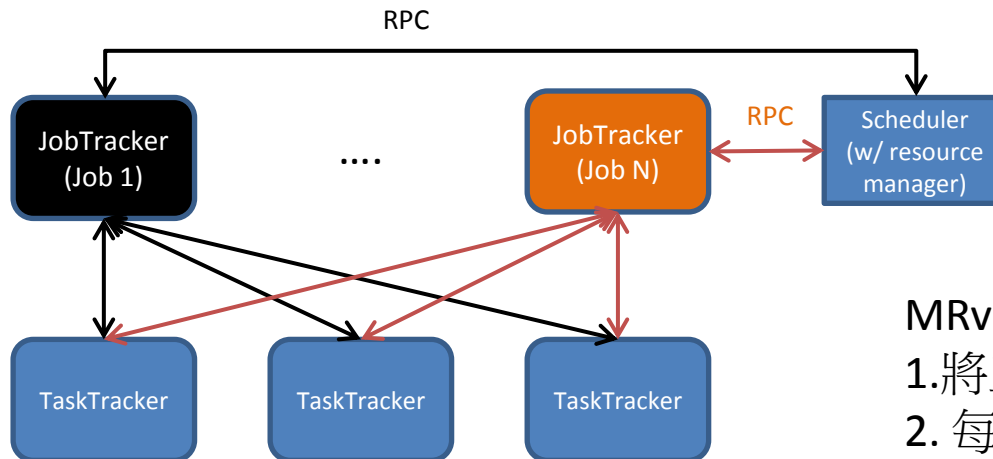
- Mapreduce 框架包含
  - 編程模型：map()與reduce()
    - MRv1與MRv2的程式寫法都相同
  - 運行環境：
    - MRv1：JobTracker與TaskTracker，JobTracker同時負責資源管理與所有工作的控制
    - MRv2：由YARN提供





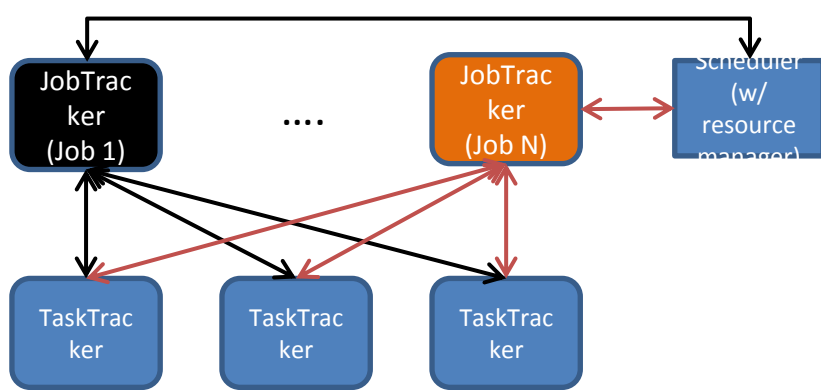
MRv1.

JobTracker負責工作控制與資源管理

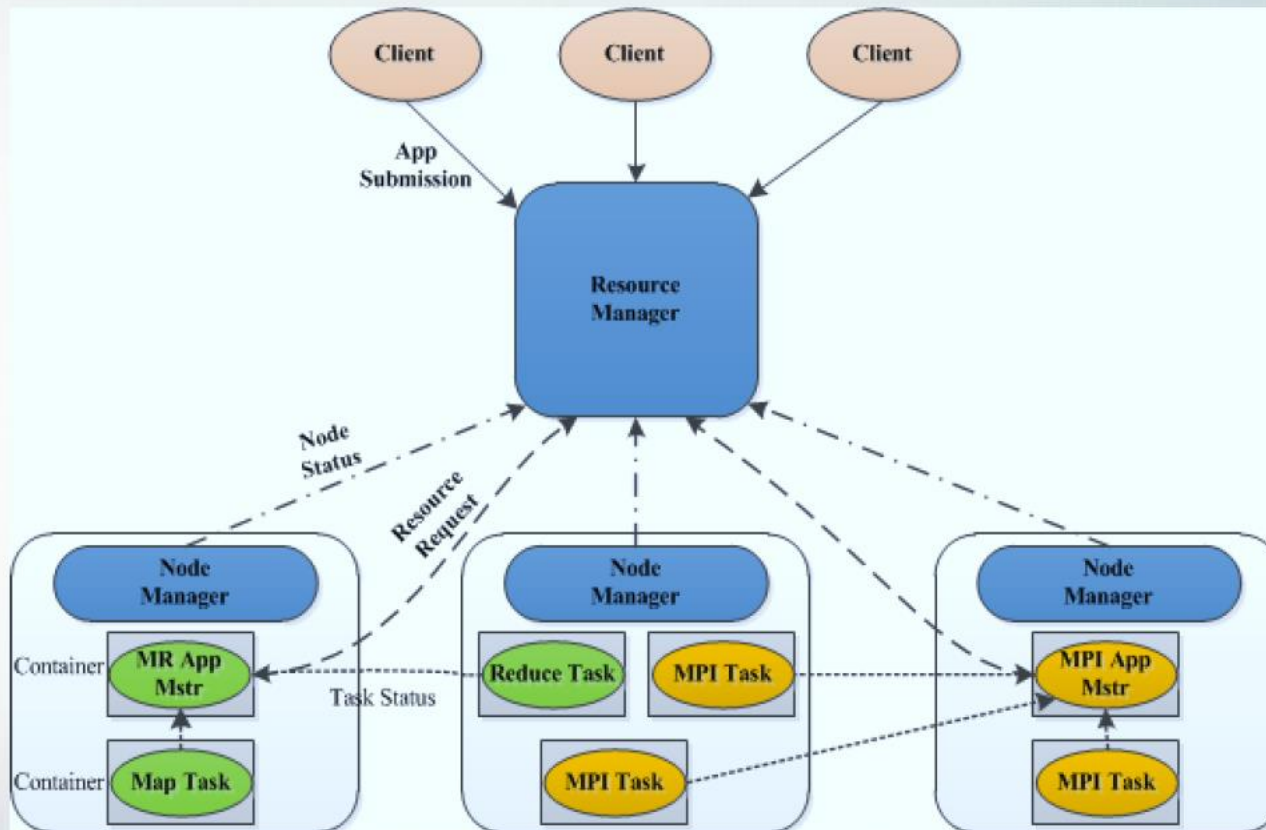


MRv2.

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

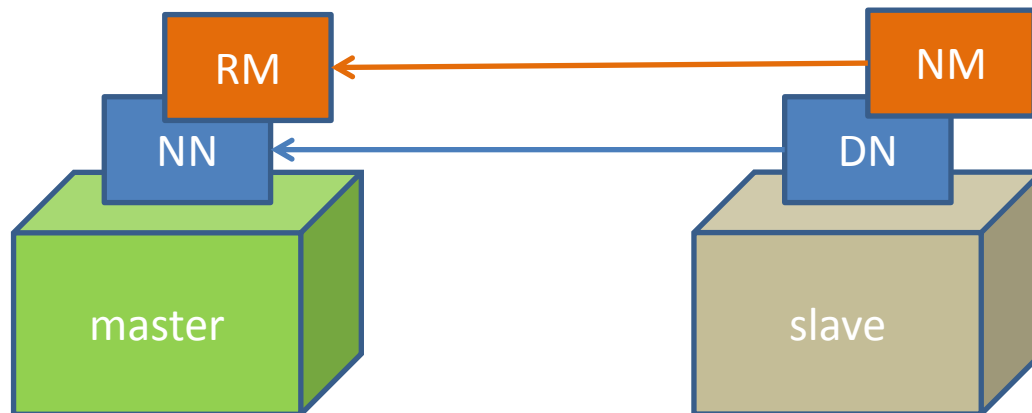


1. 由RM做全局的資源分配
2. NM定時回報目前的資源使用量
3. 每個JOB會有一個負責的AppMaster控制Job
4. 將資源管理與工作控制分開
5. YARN為一通用的資源管理系統  
可達成在YARN上運行多種框架



# 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 programming
- Mapreduce
  - YARN Intro, install & configure,
  - MR intro & Java programming
- Hbase
  - Intro, install & configure , CLI, Java programming

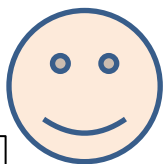
# 選舉到了...

- 台北市10個選區，共100萬票，要算出每個候選人的得票數



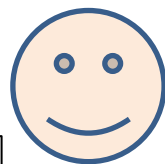
監票人1  
[負責1區]

號次	票數
2	1
1	1
...	...



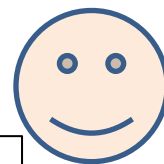
監票人2  
[負責2區]

號次	票數
1	1
1	1
3	1
...	...



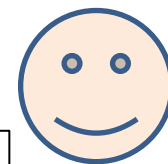
監票人3  
[負責3區]

號次	票數
3	1
2	1
1	1



監票人4  
[負責4區]

號次	票數
1	1
3	1
3	...



監票人5  
[負責5區]

號次	票數
3	1
2	1
3	1

號次	票數
2	1
1	1
...	...

號次	票數
5	1
1	1
7	1
...	...

號次	票數
5	1
2	1
1	1

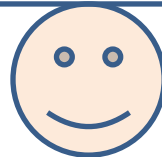
號次	票數
1	1
5	1
3	...

號次	票數
4	1
2	1
6	1

**Shuffle & Sort**  
由各投開票所送到中選會

號次	票數	號次	票數	號次	票數
1	1	2	1	3	1
1	1	2	1	3	1
1	1	2	1	3	1
1	1	2	1	3	1
1	...	2	...	3	...

中選會  
[負責全部的候選人]



號次	票數	號次	票數	號次	票數
1	1	2	1	3	1
1	1	2	1	3	1
1	1	2	1	3	1
1	1	2	1	3	1
1	...	2	...	3	...



號次	總票數
1	187532

號次	總票數
2	574821

號次	總票數
3	237647

選別	得票
2	1
1	1
...	...

選別	得票
1	1
3	1
...	...

選別	得票
2	1
1	1
...	...

選別	得票
1	1
1	1
3	...

選別	得票
2	1
2	1
...	...

combine

combine

combine

combine

combine

性別	總分
1	1840
2	1740
3	

性別	總分
1	1700
2	1520
3	

性別	總分
1	1700
2	1520
3	

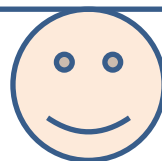
性別	總分
1	1560
2	1240
3	

性別	總分
1	1760
2	1660
3	

Shuffle & Sort  
由各投開票所送到中選會

號次	票數	號次	票數	號次	票數
1	1840	2	1740	3	...
1	1700	2	1520	3	...
1	1700	2	1520	3	...
1	1560	2	1240	3	...
1	...	2	...	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

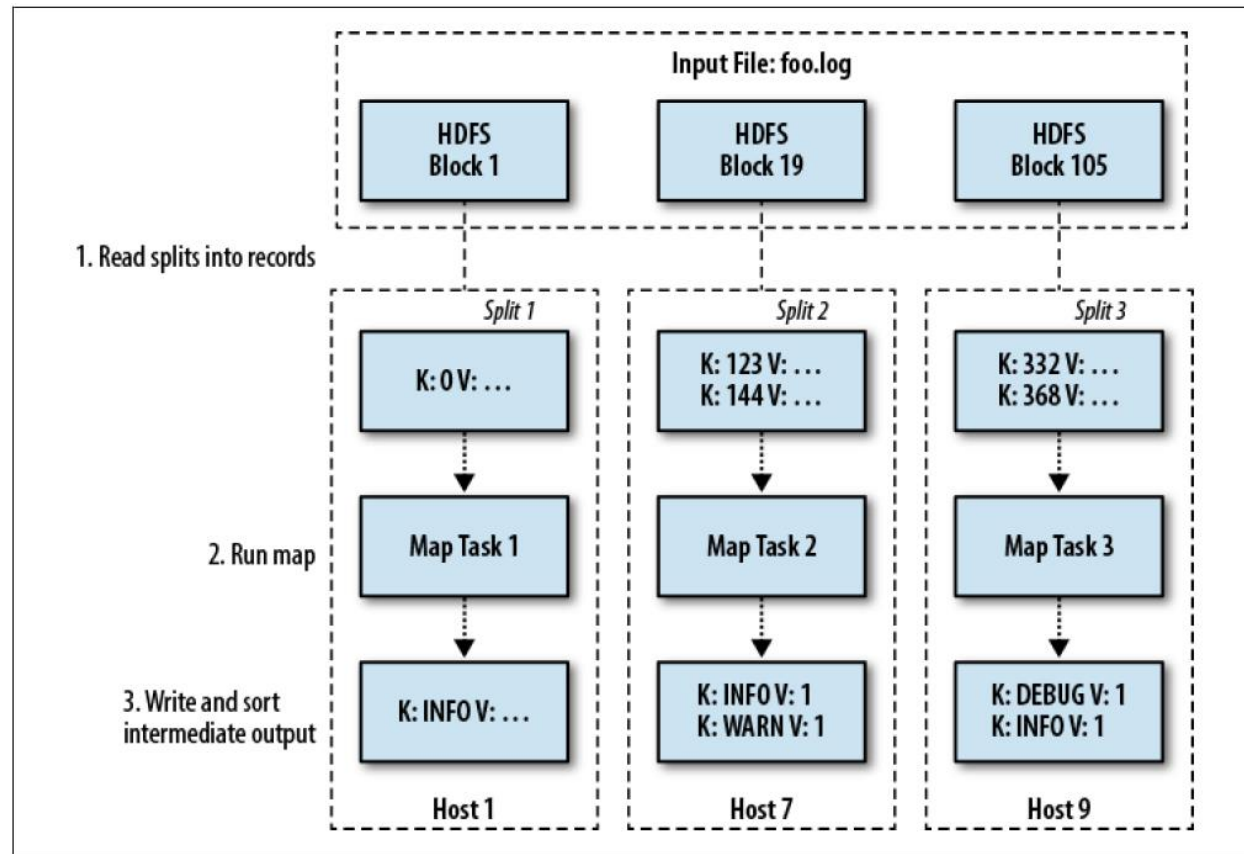
# 算字數 - Mapper

Text file

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)

This is a book  
Hello American  
Visit The official site  
Our network of more  
The American Broadcasting

3 splits



•(k1, v1) → list(k2, v2)

# 算字數 – Shuffle & Sort

- Black box

- 開發人員不用煩惱，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

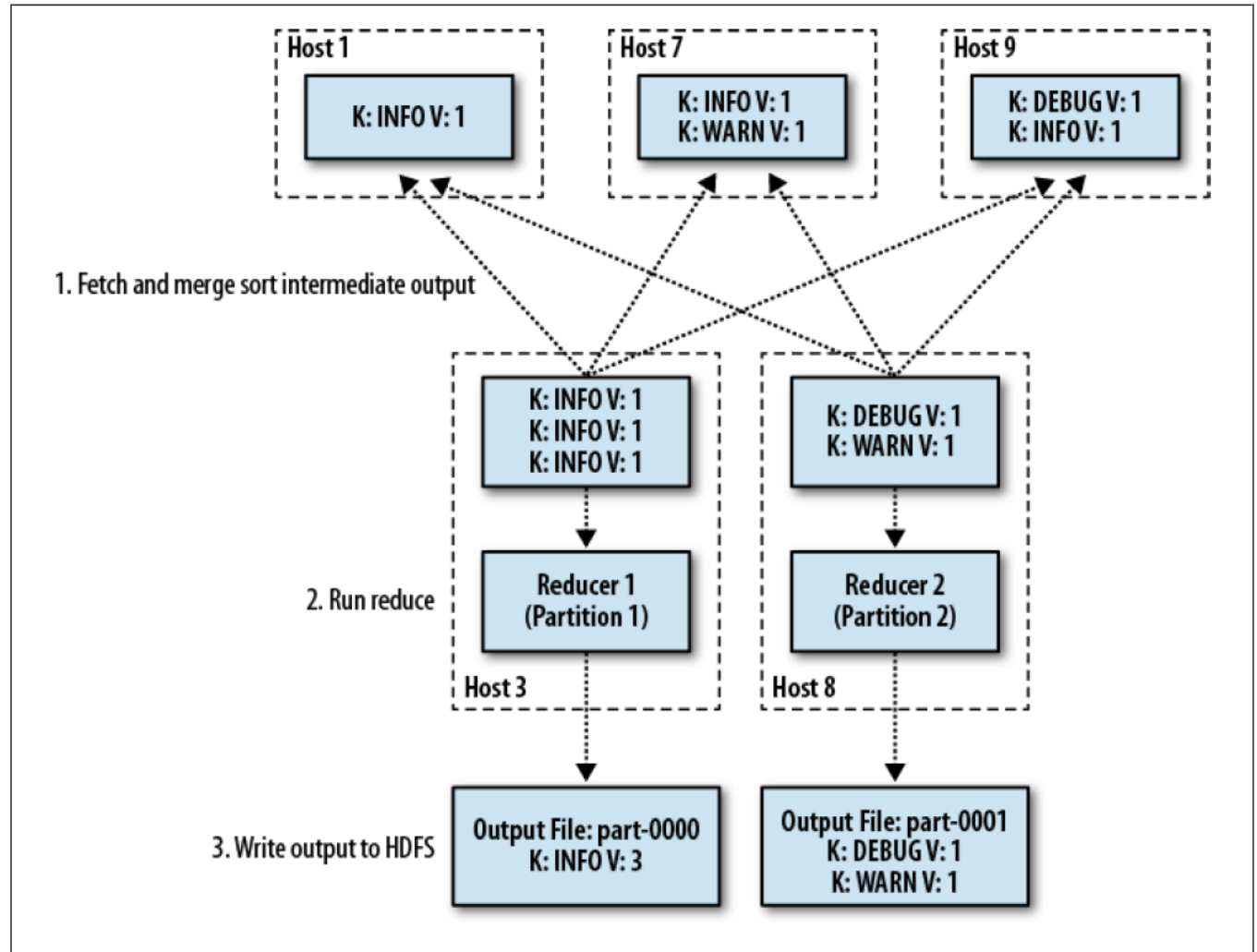
- 若Reducer看到某個Key1，會看到相對應的所有value
    - Reducer 1收到 This這個字，會收到很多 1
  - 給定Key1，所有Key1的值都會被同一個Reducer處理
  - 同一個Reducer有可能處理多個Key值

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 :

- $(k1, v1) \rightarrow \text{list}(k2, v2)$
- $(0, \text{"This is a book book"}) \rightarrow$   
 $(\text{"This"}, 1), (\text{"is"}, 1), (\text{"a"}, 1), (\text{"book"}, 1), (\text{"book"}, 1)$
- $(0, \text{第一張選票}) \rightarrow (\text{一號}, 0), (\text{二號}, 1), (\text{三號}, 0)$

## •Reducer :

- $(k2, \text{list}(v2)) \rightarrow (k3, v3)$
- $(\text{"This"}, [1]) \rightarrow (\text{"This"}, 1)$
- $(\text{"is"}, [1]) \rightarrow (\text{"is"}, 1)$
- $(\text{"a"}, [1]) \rightarrow (\text{"a"}, 1)$
- $(\text{"book"}, [1, 1]) \rightarrow (\text{"book"}, 2)$

$(\text{一號}, [1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0]) \rightarrow (\text{一號}, 6)$

$(\text{二號}, [0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0]) \rightarrow (\text{二號}, 3)$

$(\text{三號}, [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]) \rightarrow (\text{三號}, 2)$

# 算字數 - Pseudocode

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

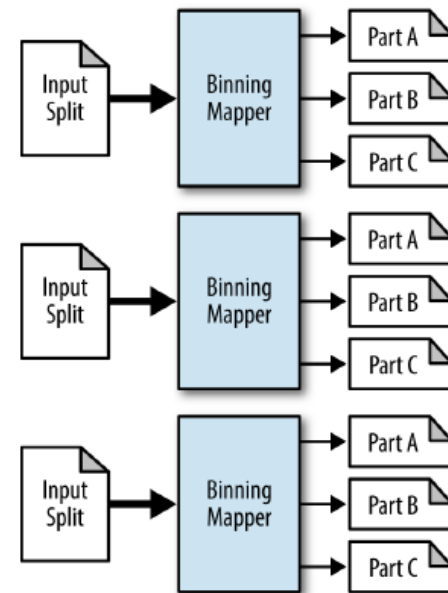
```
void Reduce (keyword, <list of value>){  
    for each x in <list of value>:  
        sum+=x;  
    final_output.collect(keyword, sum);  
}
```

## 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>
    }
}
```

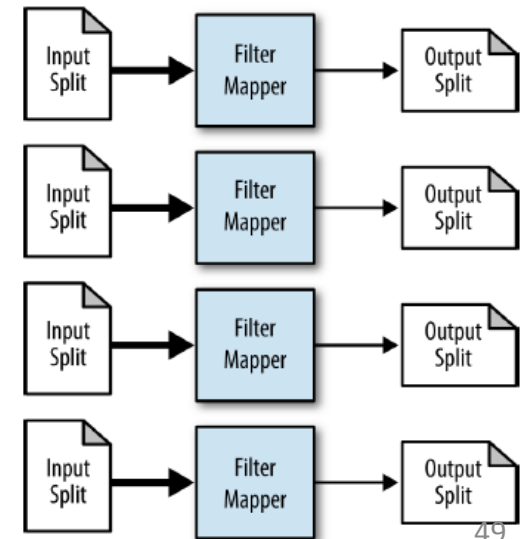
```
public void reduce(Text key, Iterable<IntWritable> values, Context context
    ) throws IOException, InterruptedException {
    int sum = 0; // initialize the sum for each keyword
    for (IntWritable val : values) {
        sum += val.get();
    }
    result.set(sum);
    context.write(key, result);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

- 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-TH03
  - 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> {
```

```
        ...
```

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);
```



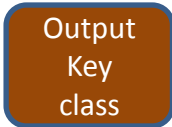
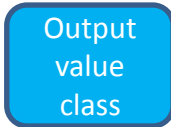
- Inputformat



- 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

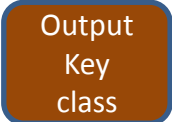
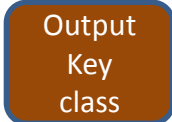
- **Outputformat**
  - Hadoop如何將分析完的結果輸出
  - 預設為TextOutputFormat class
  - 每一筆結果為輸出文件中的一行
  - 每一行包含key/value，預設以tab分隔
  - Key/value可為任意class, 但需在Driver中設定
- 若使用預設的TextInputFormat/TextOutputFormat, 無需在Driver中設定
- 若使用非預設的input/output format
  - job.setInputFormatClass(SequenceFileInputFormat.class);
  - job.setOutputFormatClass(NullOutputFormat.class);

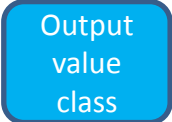
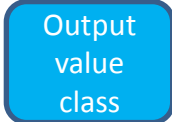


# public class MyMapper extends

Mapper< , , ,  > {

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

 newkey = new  ()

 newvalue = new  ()

....

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 {

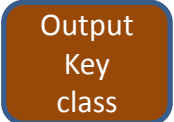
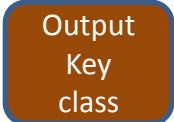
        StringTokenizer itr = new StringTokenizer(value.toString());
        IntWritable one = new IntWritable(1);


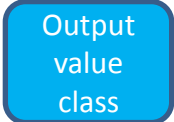
        while (itr.hasMoreTokens()) {
            Text word = new Text(itr.nextToken())
            context.write(word, one);
        }
    }
}
```

public class MyReducer extends

Reducer<  ,  ,  ,  > {

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

 newkey = new  ()

 newvalue = new  ()

....

Context.write(newkey,newvalue);

}

}

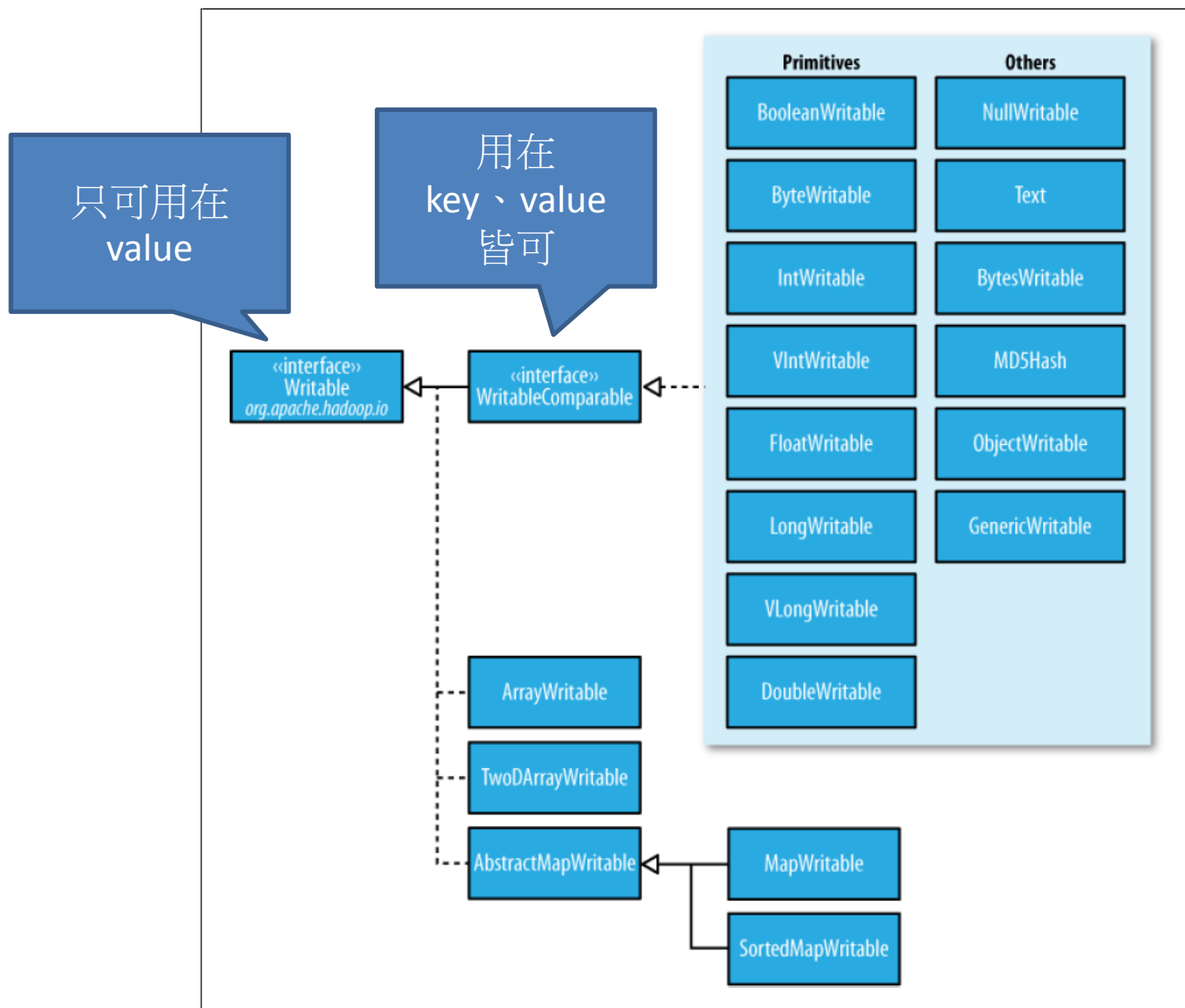
```
public class WordCountReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    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/deserialize
  - 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



# New and Old API



```

import org.apache.hadoop.mapred.*;

1 class MyMap extends MapReduceBase
  implements Mapper < INPUT KEY , INPUT VALUE , OUTPUT KEY , OUTPUT VALUE >
2 {
3   // 全域變數區
4   public void map ( INPUT KEY key, INPUT VALUE value,
                      OutputCollector< OUTPUT KEY , OUTPUT VALUE > output,
                      Reporter reporter) throws IOException
5   {
6     // 區域變數與程式邏輯區
7     output.collect( NewKey, NewValue);
8   }
9 }

```



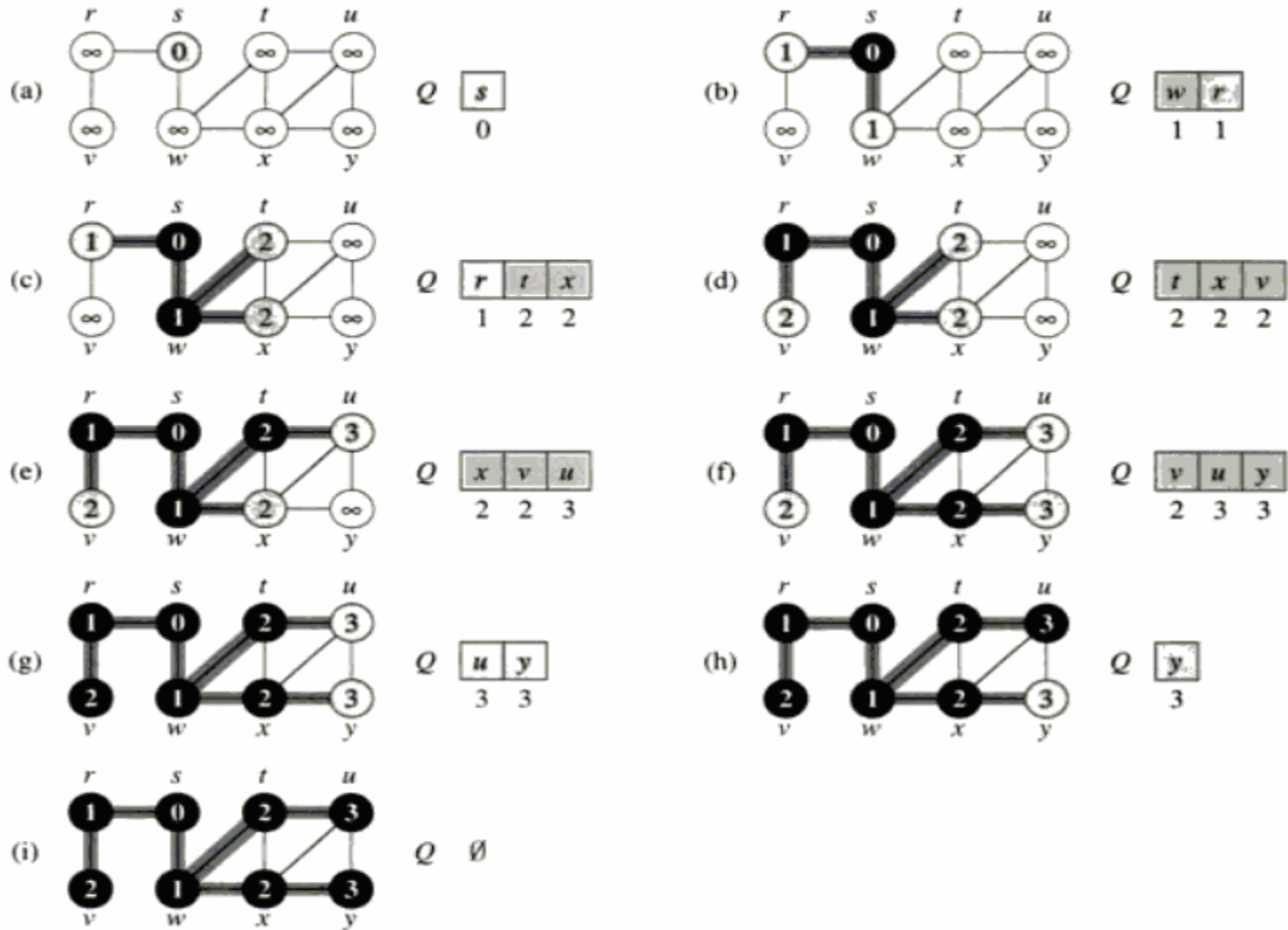
```
import org.apache.hadoop.mapred.*;
```

```
1  class MyRed extends MapReduceBase
   implements Reducer < INPUT KEY , INPUT VALUE , OUTPUT KEY , OUTPUT VALUE >
2  {
3  // 全域變數區
4  public void reduce ( INPUT KEY key, Iterator< INPUT VALUE > values,
   OutputCollector< OUTPUT KEY , OUTPUT VALUE > output,
   Reporter reporter) throws IOException
5  {
6  // 區域變數與程式邏輯區
7  output.collect( NewKey, NewValue);
8  }
9  }
```

# Put all together

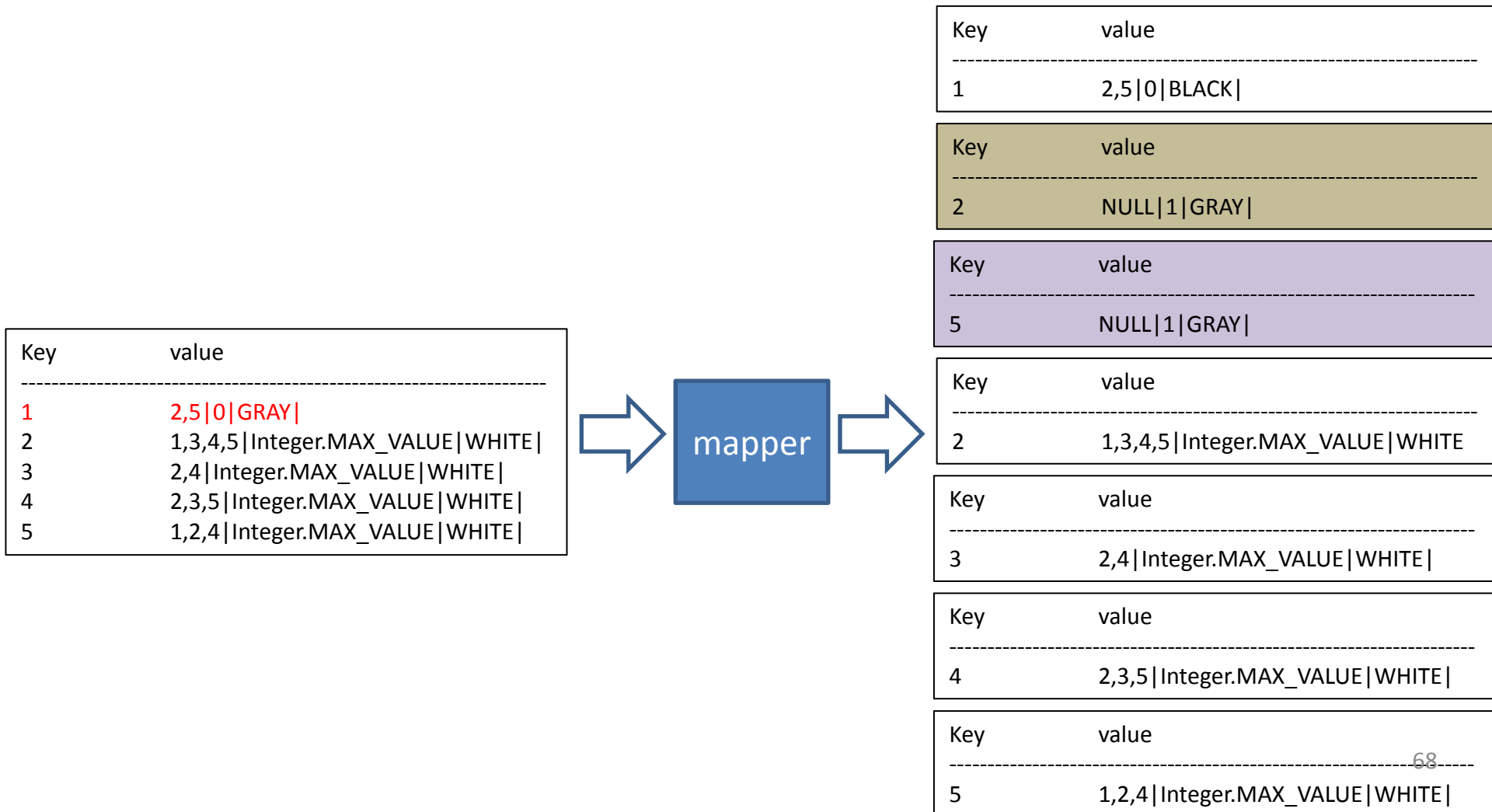
- breadth-first search
- K-means

# BFS



# BFS Mapper

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



# 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

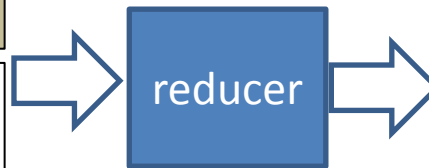
Key	value
1	2,5 0 BLACK

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

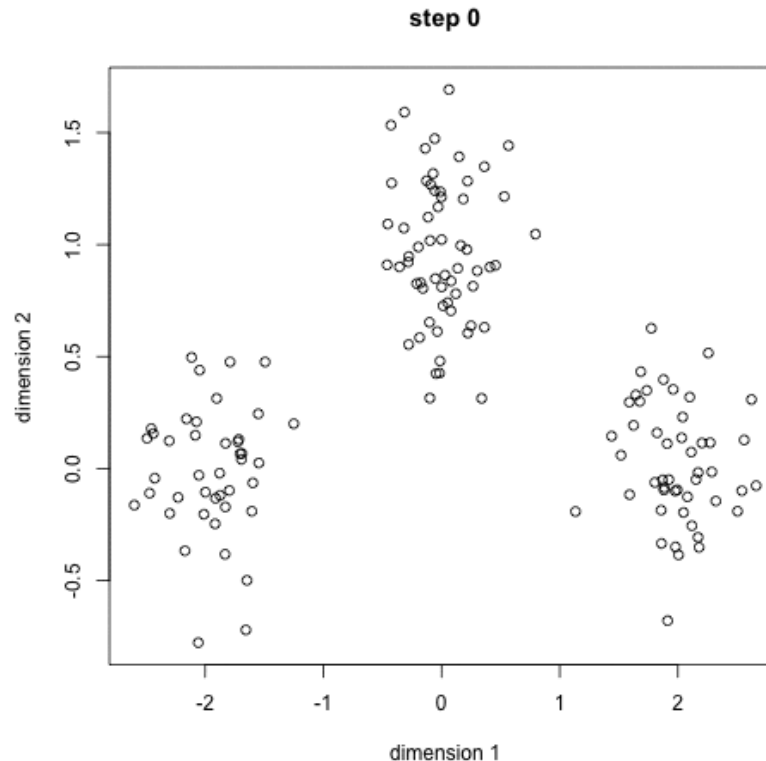
Key	value
5	NULL 1 GRAY
5	1,2,4 Integer.MAX_VALUE WHITE



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 $u_1 \sim u_k$
- 計算每個資料 $x_i$  對應到最短距離的群中心 (固定  $u_i$  求解所屬群  $S_i$ )
- 利用目前得到的分類重新計算群中心 (固定  $S_i$  求解群中心  $u_i$ )
- 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>

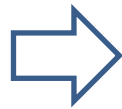
Read Distributed cache

C1 : (x1,y1)

C2 : (x2,y2)

C3 : (x3,y3)

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



mapper



Key	value
<hr/>	
C2	V1(1,2)

Key	value
<hr/>	
C2	V2(7,4)

Key	value
<hr/>	
C1	V3(16,3)

Key	value
<hr/>	
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
<hr/>	
C1	V3(16,3)

Key	value
<hr/>	
C2	V1(1,2)
C2	V2(7,4)

Key	value
<hr/>	
C3	V4(-1,-23)



reducer



Key	value
<hr/>	
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寫起很煩麻，不需要重複製造輪子
  - Data warehouse : HIVE
  - Scripts : PIG
  - Machine learning framework : Mahout

# HIVE short DEMO

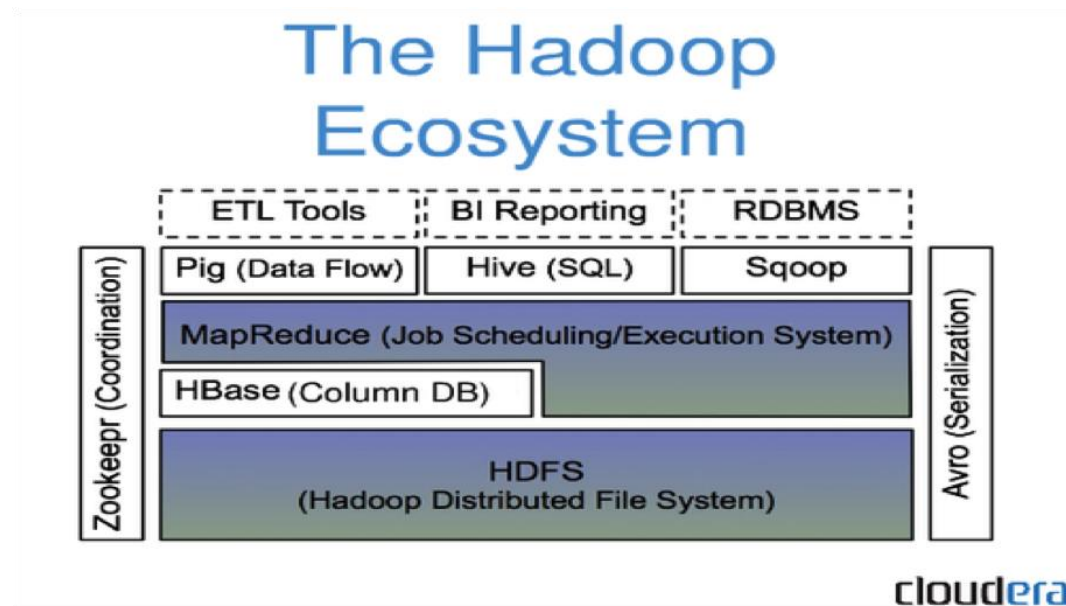
- `wget http://archive.cloudera.com/cdh4/cdh/4/hive-0.10.0-cdh4.7.0.tar.gz`
- `tar zxvf hive-0.10.0-cdh4.7.0.tar.gz`
- `http://hive.3du.me/Lab-009.html#匯入-csv-資料到-hive-資料表`

# Outline

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

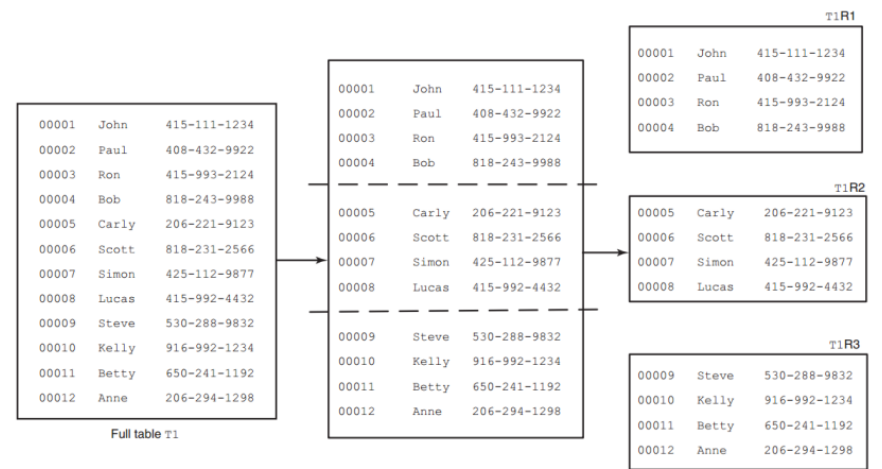
# What is HBase

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

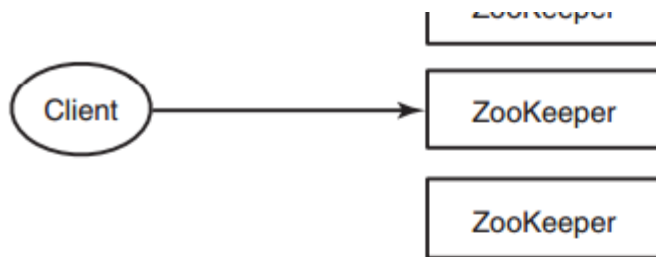


# Hbase architecture

- 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

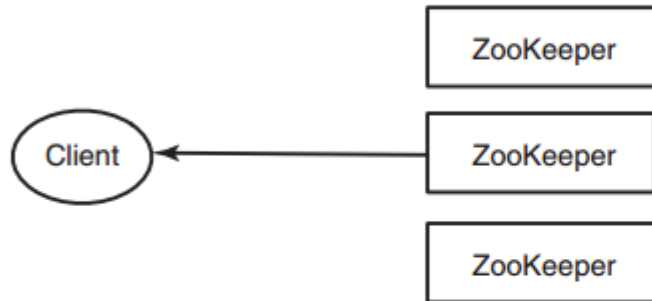


Step 1



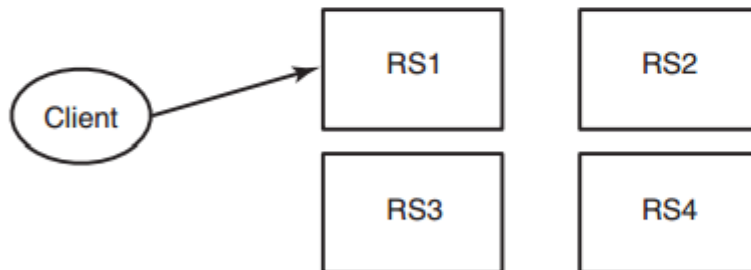
Client -> ZooKeeper: Where's `-ROOT-`?

Step 2



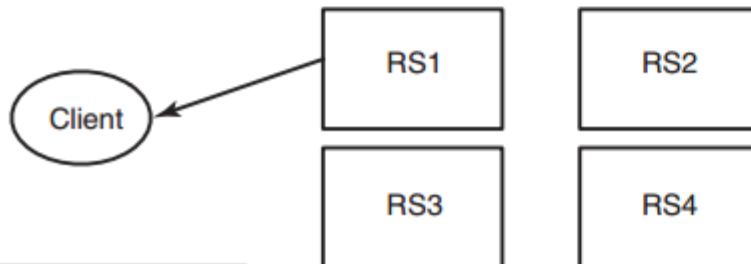
ZooKeeper -> Client : It's at RegionServer RS1.

Step 3



Client -> `-ROOT-` table on RS1:  
Which `.META.` region can tell me about  
row 00009 from table 1?

Step 4



`-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

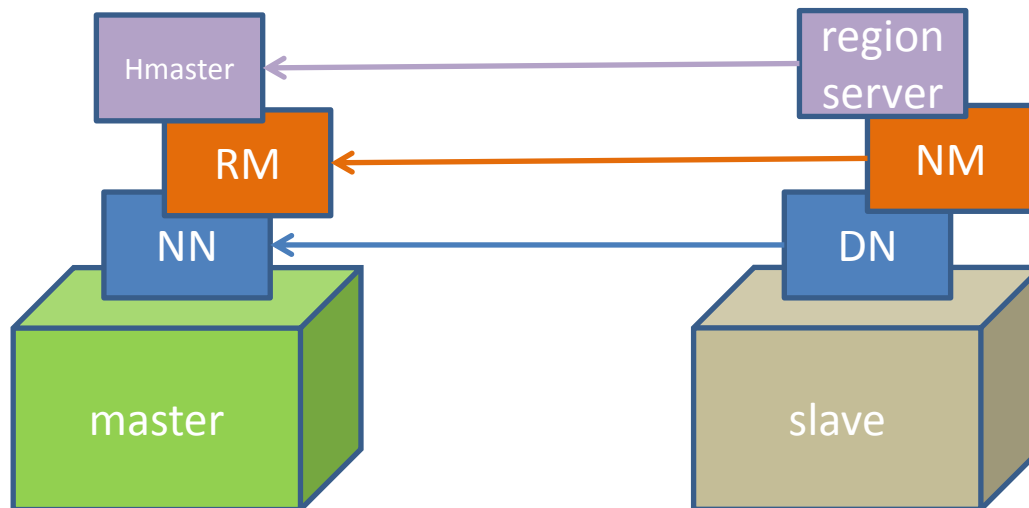
RS3

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-0.94.15-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  
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

Table					

# Hbase data model

- Row and rowkey
  - Data is stored to its row
  - Rows are identified 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 ?

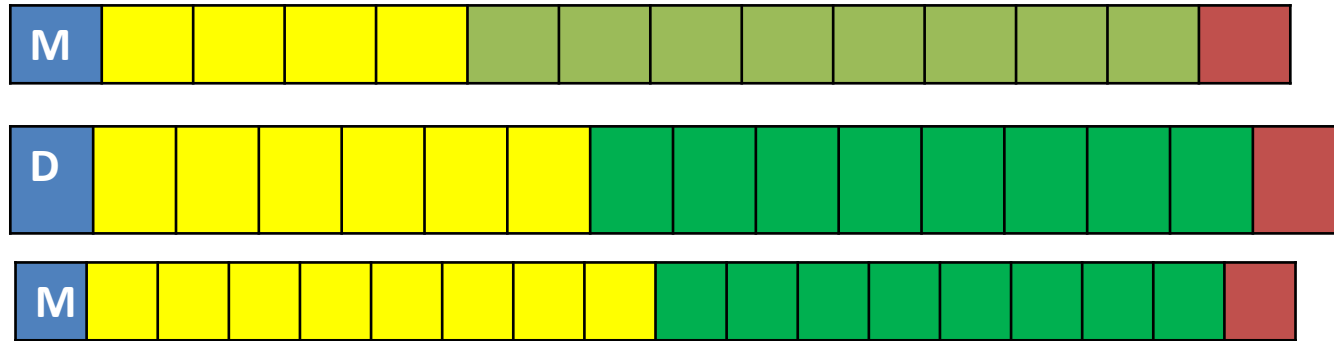
# Initial Design

Table

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

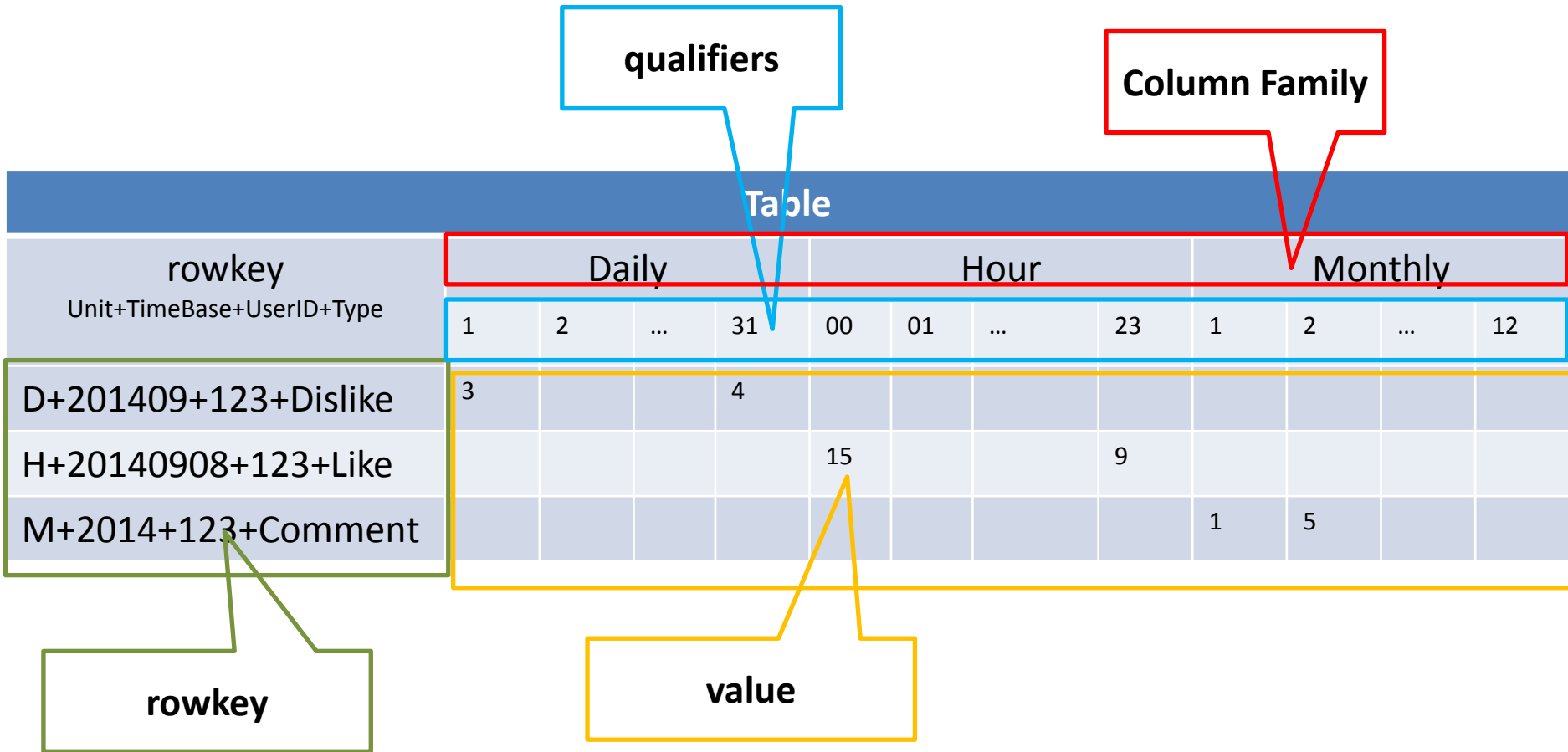
- Bad Design 2
  - CF should be defined first
    - How about add new action type ??
  - Number of CF should NOT be too much

# 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



# 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 + 12**4**

Table										
Rowkey ( Unit+TimeBase+UserID+Type )	...	Daily								...
		...	7	8	9	...	19	20	...	
<b>D+201409+123</b> +Dislike			<b>21</b>	<b>14</b>	<b>56</b>	...	<b>21</b>	<b>47</b>		
<b>D+201409+123</b> +Like			<b>25</b>	<b>12</b>	<b>78</b>	...	<b>98</b>	<b>112</b>		
<b>D+201409+123</b> +comment			<b>27</b>	<b>21</b>	<b>57</b>	...	<b>31</b>	<b>34</b>		
<b>D+201409+789</b> +Like			<b>26</b>	<b>41</b>	<b>29</b>	...	<b>7</b>	<b>35</b>		
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>
- git checkout NE-2015-TH03
  - org.nchc.train.hbase.accessHBase
    - Create HTable
    - Put into HTable
    - Get from Htable
    - Scan HBase

# Reference

- 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
- 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

