進階課程

Hadoop 進階程式設計與 HBase 資料庫整合實作

王耀聰　陳威宇

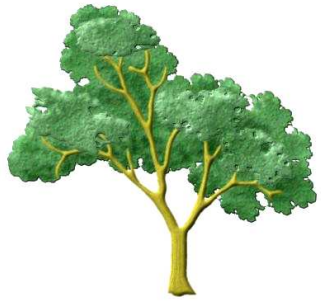
jazz@nchc.org.tw

waue@nchc.org.tw

財團法人國家實驗研究院

國家高速網路與計算中心

NATIONAL CENTER FOR HIGH-PERFORMANCE COMPUTING

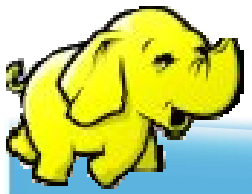


課程大綱 （1）

第一天	
09:30~10:20	介紹課程 與 Hadoop簡介
10:20~10:30	休息
10:30~12:00	Hadoop生態系簡介
	實作一：Hadoop Streaming 範例操作
12:00~13:00	午餐
13:00~15:00	開發輔助工具 Eclipse
	Map Reduce 程式架構
15:00~15:10	休息
15:10~16:30	程式設計I- HDFS 操作
	程式設計II-範例程式

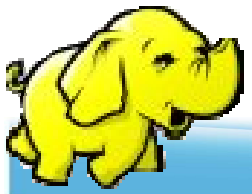
課程大綱 （2）

第二天	
09:30~10:20	HBase 簡介與架構
10:20~10:30	休息
10:30~12:00	HBase 安裝操作說明
12:00~13:00	午餐
13:00~15:00	HBase 程式架構與範例
15:00~15:10	休息
15:10~16:00	Hadoop + HBase + PHP 案例實務
16:00~16:30	hadoop ＋ 關聯式資料庫



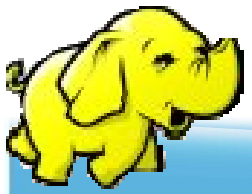
學員背景調查

- Java 語言 ??
- PHP 語言 ?? MySQL 資料庫 ??
- Linux 操作 ?? 電腦叢集維護 ??
- 安裝過 Hadoop ??
- 參加過 Hadoop 基礎課程 ??



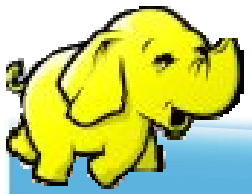
It’s Show Time

- 名稱
- 服務公司/就讀學校
- 報名原因
- 預期收穫



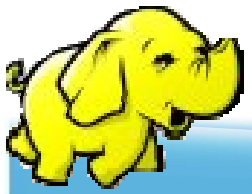
引言

雲端運算這個名詞雖然紅，
但我一定需要雲端運算嗎？
他用在什麼場合？又或非它不可嗎？



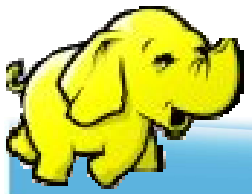
Computing with big datasets

is a fundamentally different challenge than doing “big compute” over a small dataset



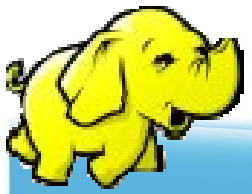
平行分散式運算

- 格網運算(網格運算, Grid computing)
  - MPI, PVM, Condor…
- 著重於: 分散工作量
- 目前的問題在於：如何分散資料量
  - Reading 100 GB off a single filer would leave nodes starved – just store data locally



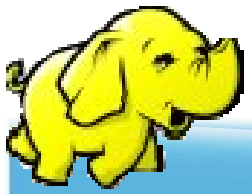
分散大量資料： Slow and Tricky

- 交換資料需同步處理
  - **Deadlock** becomes a problem
- 有限的頻寬
  - Failovers can cause **cascading failure**



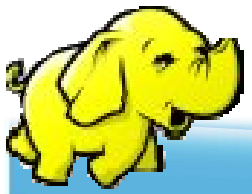
數字會說話

- Data processed by Google every month: 400 PB … in 2007
  - ◆Max data in memory: 32 GB
  - ◆Max data per computer: 12 TB
  - ◆Average job size: 180 GB
- 光一個device的讀取時間= 45 minutes



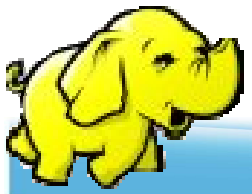
所以 ...

- 運算資料可以很快速，但瓶頸在於硬碟的 I/O
  - 1 HDD = 75 MB/sec
- 解法: parallel reads
  - 1000 HDDs = 75 GB/sec



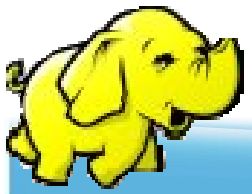
MapReduce 的動機

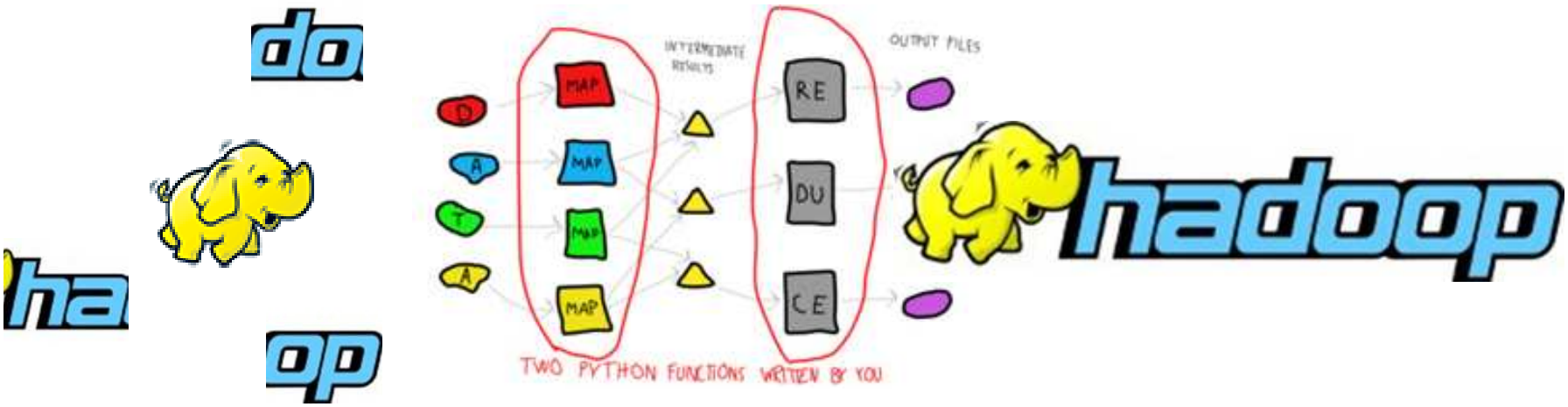
- Data > 1 TB
- 交互運算於大量的CPU
- 容易開發與使用
  - High-level applications written in MapReduce
  - Programmers don’t worry about socket(), etc.



Conclusions

- “大資料集的運算”與“小資料集的高速運算”，兩者是迥然不同的挑戰。
- 大資料量的解決方法將需要：
  - 提供囊括所有解決之道的新工具
  - MapReduce 與 HDFS 等工具是其中之一



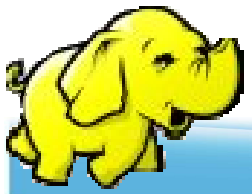


一、Hadoop 簡介

Hadoop 是一套儲存並處理
petabytes 等級資訊的
雲端運算技術

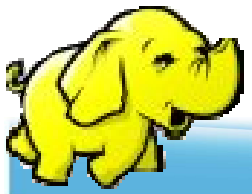
Hadoop

- 以Java開發
- 自由軟體
- 上千個節點
- Petabyte等級的資料量
- 創始者 Doug Cutting
- 為Apache 軟體基金會的 top level project



特色

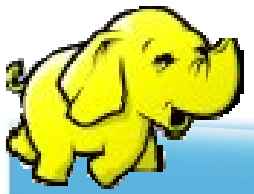
- 巨量
  - 擁有儲存與處理大量資料的能力
- 經濟
  - 由一般個人電腦所架設的叢集環境
- 效率
  - 藉由平行分散檔案以致得到快速的回應
- 可靠
  - 當某個節點發生錯誤，系統能即時自動的取得備份資料以及佈署運算資源



實用案例

Hadoop於Yahoo的運作資訊

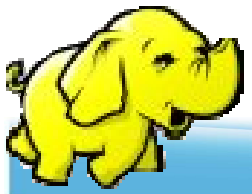
年份	日期	節點數	耗時（小時）
2006	四月	188	47.9
2006	五月	500	42
2006	十一月	20	1.8
2006	十一月	100	3.3
2006	十一月	500	5.2
2006	十一月	900	7.8
2007	七月	20	1.2
2007	七月	100	1.3
2007	七月	500	2
2007	七月	900	2.5



Sort benchmark, every nodes with terabytes data.

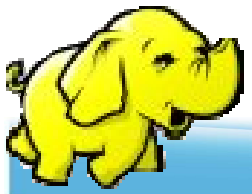
誰在用Hadoop ？(1)

- Facebook
  - 處理 internal log and dimension data sources
  - for reporting/analytics and machine learning.
- IBM
  - Blue Cloud Computing Clusters
- Journey Dynamics
  - 用 Hadoop MapReduce 分析 billions of lines of GPS data 並產生交通路線資訊.
- Krugle
  - 用 Hadoop and Nutch 建構 原始碼搜尋引擎



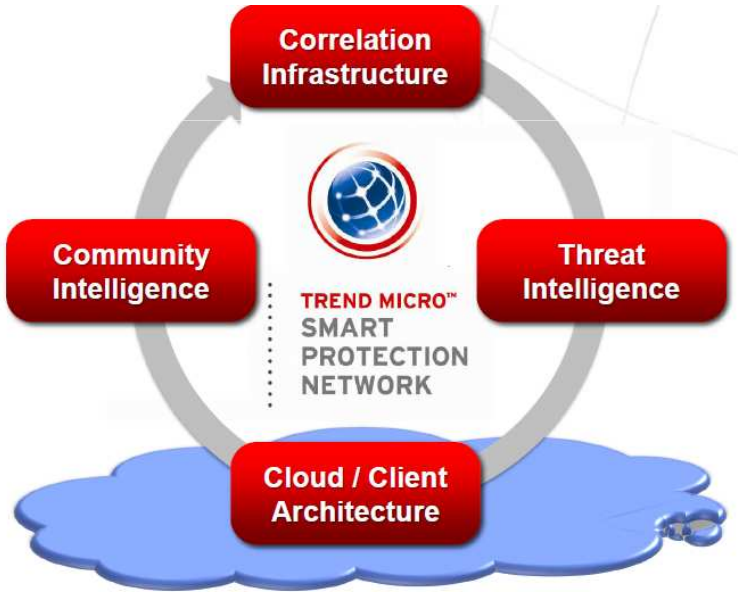
誰在用Hadoop？(2)

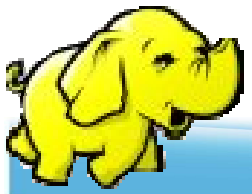
- SEDNS - Security Enhanced DNS Group
  - 收集全世界的 DNS 以探索網路分散式內容.
- Technical analysis and Stock Research
  - 分析股票資訊
- University of Nebraska Lincoln, Research Computing Facility
  - 用Hadoop跑約200TB的CMS經驗分析
  - **緊湊渺子線圈**（**CMS**，**C**ompact **M**uon **S**olenoid）為瑞士歐洲核子研究組織CERN的大型強子對撞器計劃的兩大通用型粒子偵測器中的一個。



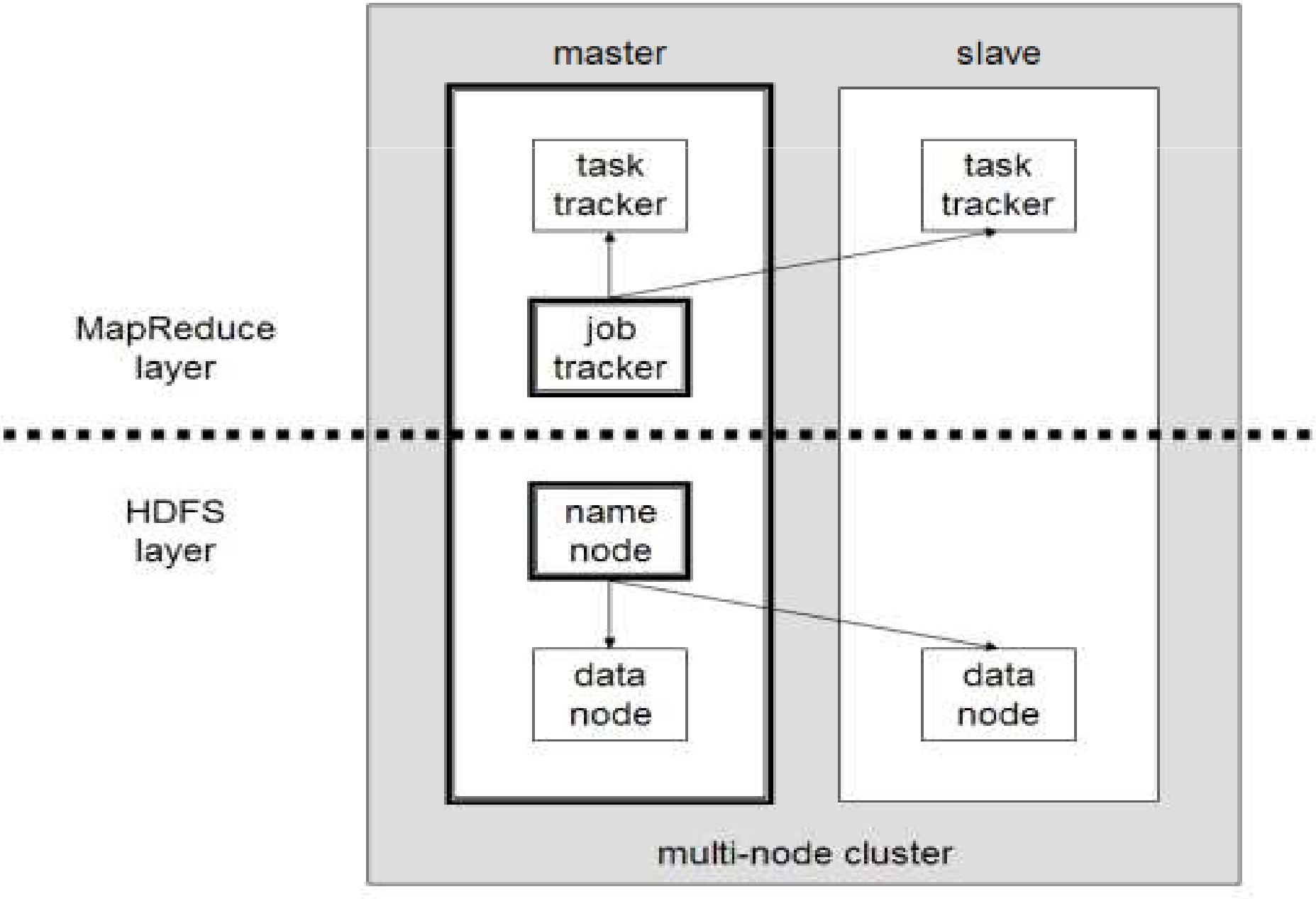
誰在用Hadoop？(3)

- Yahoo!
  - Used to support research for Ad Systems and Web Search
  - 使用Hadoop平台來發現發送垃圾郵件的殭屍網絡
- 趨勢科技
  - 過濾像是釣魚網站或惡意連結的網頁內容

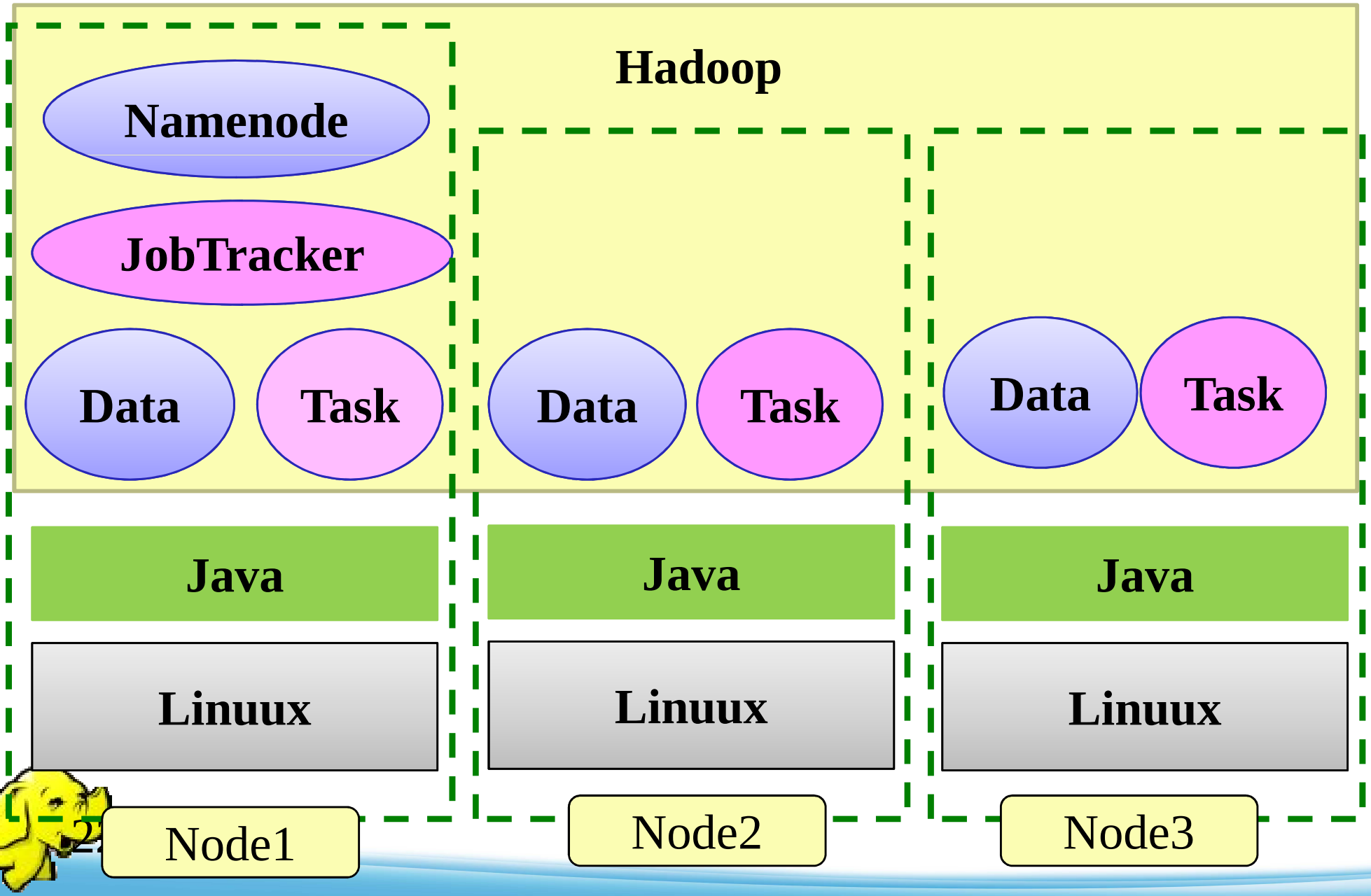




Hadoop 的主要架構



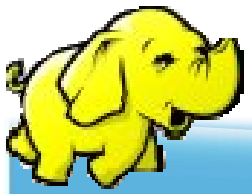
Building Hadoop



名詞

- Job
  - 任務
- Task
  - 小工作
- JobTracker
  - 任務分派者
- TaskTracker
  - 小工作的執行者
- Client
  - 發起任務的客戶端
- Map
  - 應對
- Reduce
  - 總和

- Namenode
  - 名稱節點
- Datanode
  - 資料節點
- Namespace
  - 名稱空間
- Replication
  - 副本
- Blocks
  - 檔案區塊 (64M)
- Metadata
  - 屬性資料



管理資料

Namenode

- Master
- 管理HDFS的名稱空間
- 控制對檔案的讀/寫
- 配置副本策略
- 對名稱空間作檢查及紀錄
- 只能有一個

Datanode

- Workers
- 執行讀/寫動作
- 執行Namenode的副本策略
- 可多個

分派程序

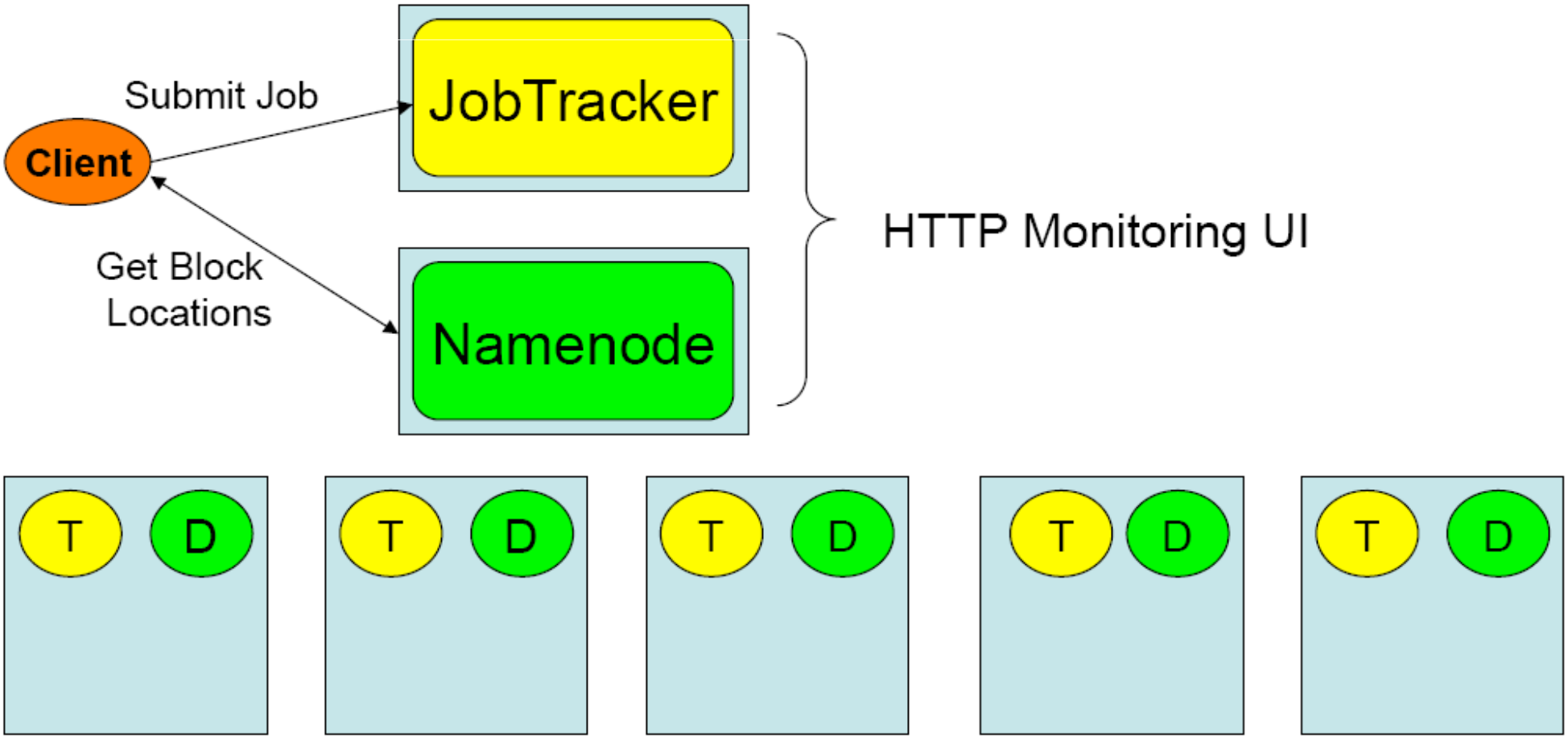
Jobtracker

- Master
- 使用者發起工作
- 指派工作給 Tasktrackers
- 排程決策、工作分配、錯誤處理
- 只能有一個

Tasktrackers

- Workers
- 運作Map 與 Reduce 的工作
- 管理儲存、回覆運算結果
- 可多個

不在雲裡的 Client

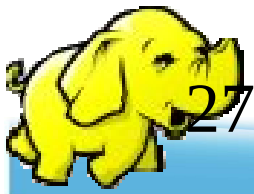


其他的 Open Source 專案:

Sector	The National Center for Data Mining (NCDM)	http://sector.sourceforge.net/

其他不同語言實作的 MapReduce 函式庫

http://trac.nchc.org.tw/grid/wiki/jazz/09-04-14#MapReduce



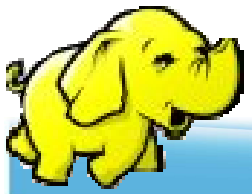
27

關於 Sector / Sphere

- http://sector.sourceforge.net/
- 由美國資料探勘中心(National Center for Data Mining)研發的自由軟體專案。
- 採用C/C++語言撰寫，因此效能較 Hadoop 更好。
- 提供「類似」Google File System與MapReduce的機制
- 基於UDT高效率網路協定來加速資料傳輸效率
- Open Cloud Consortium的Open Cloud Testbed，有提供測試環境，並開發了MalStone效能評比軟體。

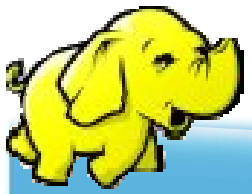
National Center for Data Mining
University of Illinois at Chicago

Open Data Group
http://www.opendatagroup.com/



Conclusions

- 所有工作都由JobTracker統一分派，由眾多TaskTracker 執行，每個TaskTracker又可以執行多個Task threads
- 所有名稱空間與檔案的metadata都由一個Namenode統籌，檔案空間為所有Datanode的集合，hdfs的基本單位為block
- Client 只需要丟工作或存取在“雲”的資料

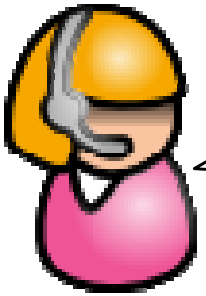


Hadoop只支援用Java開發嘛？

Is Hadoop only support Java ?

總不能全部都重新設計吧？如何與舊系統相容？

Can Hadoop work with existing software ?

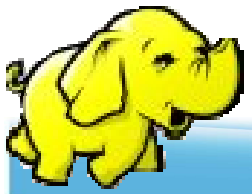


可以跟資料庫結合嘛？

Can Hadoop work with Databases ?

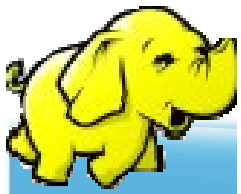
開發者們有聽到大家的需求.....

Yes, we hear the feedback of developers ...



二、Hadoop 相關子專案

- Hadoop Common: The common utilities that support the other Hadoop subprojects.
- HDFS: A distributed file system that provides high throughput access to application data.
- MapReduce: A software framework for distributed processing of large data sets on compute clusters.



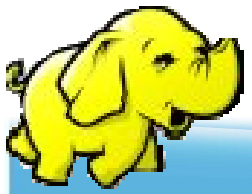
Hadoop 相關子專案

- Chukwa: A data collection system for managing large distributed systems.
- HBase: A scalable, distributed database that supports structured data storage for large tables.
- Hive: A data warehouse infrastructure that provides data summarization and ad hoc querying.
- Pig: A high-level data-flow language and execution framework for parallel computation.
- ZooKeeper: A high-performance coordination service for distributed applications.

Hadoop 生態系(Ecosystem)

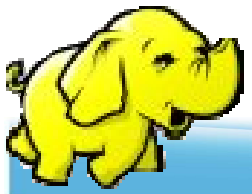
Pig	**Chukwa**	**Hive**	**HBase**
MapReduce	**HDFS**		**ZooKeeper**
Hadoop Core (Hadoop Common)		**Avro**	

Source: *Hadoop: The Definitive Guide*



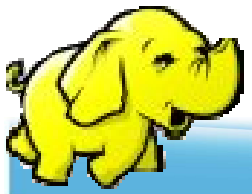
Avro

- Avro is a data serialization system.
- It provides:
  - *Rich data structures.*
  - *A compact, fast, binary data format.*
  - *A container file, to store persistent data.*
  - *Remote procedure call (RPC).*
  - *Simple integration with dynamic languages.*
- For more detail, please check the official document: http://avro.apache.org/docs/current/



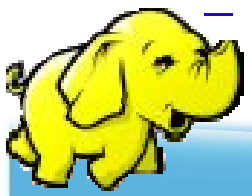
Zoo Keeper

- http://hadoop.apache.org/zookeeper/
- ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications.



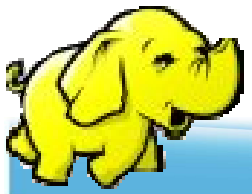
Pig

- http://hadoop.apache.org/pig/
- Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs.
- Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs
- Pig's language layer currently consists of a textual language called Pig Latin, which has the following key properties:
  - Ease of programming
  - Optimization opportunities
  - Extensibility



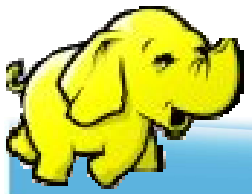
Hive

- http://hadoop.apache.org/hive/
- Hive is a data warehouse infrastructure built on top of Hadoop that provides tools to enable easy data summarization, adhoc querying and analysis of large datasets data stored in Hadoop files.
- Hive QL is based on SQL and enables users familiar with SQL to query this data.



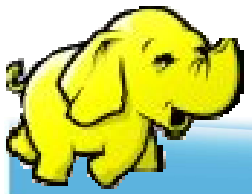
HBase

- HBase is a distributed column-oriented database built on top of HDFS.
- A distributed data store that can scale horizontally to 1,000s of commodity servers and petabytes of indexed storage.
- Designed to operate on top of the Hadoop distributed file system (HDFS) or Kosmos File System (KFS, aka Cloudstore) for scalability, fault tolerance, and high availability.
- Integrated into the Hadoop map-reduce platform and paradigm.



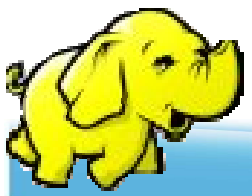
Chukwa

- http://hadoop.apache.org/chukwa/
- Chukwa is an open source data collection system for monitoring large distributed systems.
- built on top of HDFS and Map/Reduce framework
- includes a flexible and powerful toolkit for displaying, monitoring and analyzing res make the best use of the collected data.



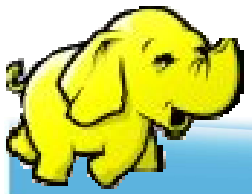
Mahout

- http://mahout.apache.org/
- Mahout is a scalable machine learning libraries.
- implemented on top of Apache Hadoop using the map/reduce paradigm.
- Mahout currently has
  - Collaborative Filtering
  - User and Item based recommenders
  - K-Means, Fuzzy K-Means
  - Mean Shift clustering
  - More ...



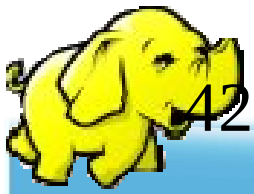
Hadoop 只支援 Java 嗎？

- Although the Hadoop framework is implemented in Java™, Map/Reduce applications need not be written in Java.
- Hadoop Streaming is a utility which allows users to create and run jobs with any executables (e.g. shell utilities) as the mapper and/or the reducer.
- Hadoop Pipes is a SWIG-compatible C++ API to implement Map/Reduce applications (non JNI™ based).



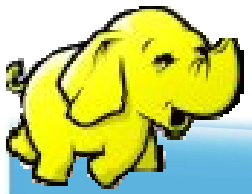
Hadoop Pipes (C++, Python)

- Hadoop Pipes allows C++ code to use Hadoop DFS and map/reduce.
- The C++ interface is "swigable" so that interfaces can be generated for python and other scripting languages.
- For more detail, check the API Document of org.apache.hadoop.mapred.pipes
- You can also find example code at
- hadoop-*/src/examples/pipes
- About the pipes C++ WordCount example code:
- http://wiki.apache.org/hadoop/C++WordCount



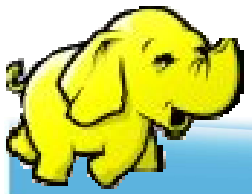
Hadoop Streaming

- Hadoop Streaming is a utility which allows users to create and run Map-Reduce jobs with any executables (e.g. Unix shell utilities) as the mapper and/or the reducer.
- It's useful when you need to run existing program written in shell script, perl script or even PHP.
- Note: both the mapper and the reducer are executables that read the input from STDIN (line by line) and emit the output to STDOUT.
- For more detail, check the official document of Hadoop Streaming



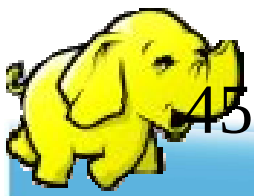
Running Hadoop Streaming

```
jazz@hadoop:~$ hadoop jar hadoop-streaming.jar –help
Usage: $HADOOP_HOME/bin/hadoop [--config dir] jar \
          $HADOOP_HOME/hadoop-streaming.jar [options]
Options:
  -input    <path>     DFS input file(s) for the Map step
  -output   <path>     DFS output directory for the Reduce step
  -mapper   <cmd|JavaClassName>      The streaming command to run
  -combiner <JavaClassName> Combiner has to be a Java class
  -reducer  <cmd|JavaClassName>      The streaming command to run
  -file     <file>     File/dir to be shipped in the Job jar file
  -dfs    <h:p>|local  Optional. Override DFS configuration
  -jt     <h:p>|local  Optional. Override JobTracker configuration
  -additionalconfspec specfile  Optional.
  -inputformat
     TextInputFormat(default)|SequenceFileAsTextInputFormat|JavaClassNam
     e Optional.
  -outputformat TextOutputFormat(default)|JavaClassName  Optional.

… More …
```



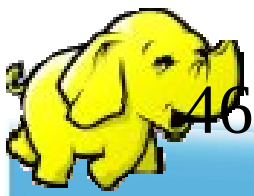
Hadoop Streaming 範例 (1)

```
hadoop:~$ hadoop fs -rmr input output
hadoop:~$ hadoop fs -put /etc/hadoop/conf
    input
hadoop:~$ hadoop jar hadoop-streaming.jar
    -input input -output output -mapper
    /bin/cat -reducer /usr/bin/wc
```



Hadoop Streaming 範例 (2)

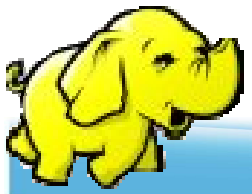
```
hadoop:~$ echo "sed -e \"s/ /\n/g\" | grep ." >
    streamingMapper.sh
hadoop:~$ echo "uniq -c | awk '{print \$2 \"\t\" \$1}'"
    > streamingReducer.sh
hadoop:~$ chmod a+x streamingMapper.sh
hadoop:~$ chmod a+x streamingReducer.sh
hadoop:~$ hadoop fs -put /etc/hadoop/conf input
hadoop:~$ hadoop jar hadoop-streaming.jar -input input
    -output output -mapper streamingMapper.sh -reducer
    streamingReducer.sh -file streamingMapper.sh
    -file streamingReducer.sh
```



三、Map Reduce 與HDFS

3.A：HDFS 檔案系統

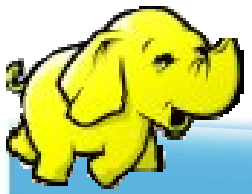
3.B：MapReduce 演算法



三、Map Reduce 與HDFS

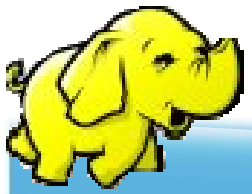
3.A：HDFS - 檔案系統

HDFS = Hadoop Distributed File System，是Hadoop用來存放資料的分散式檔案系統，因此若要讓Hadoop運算資料，就要將待運算的資料放到這個空間才可以；同理可知運算後的資料...。



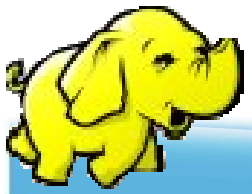
HDFS: 開發動機

- 基於 Google File System (GFS)
- 用便宜而不可靠的電腦，打造互為備援的儲存媒介，拿來存放海量資料。
- 為何不使用現存的檔案系統呢?
  - 不同的工作負載與設計優先權
  - 須處理比其他檔案系統還要更大的資料集



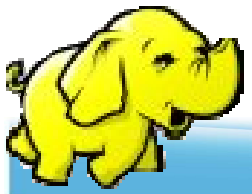
基本假設

- 高單元故障率
  - 便宜的個人電腦商品總是容易故障
- 海量檔案的『客觀』參考數字
  - 就只是幾百萬個檔案而已...
  - 每個大小約100MB或更大; 通常以數GB的檔案最常見
- 這些檔案都只寫一次(write-once)，多數是附加(append)
- 大量的串流讀取需求
- 持續而大量的資料通量(throughput)遠比較短的反應延遲(latency)來得重要

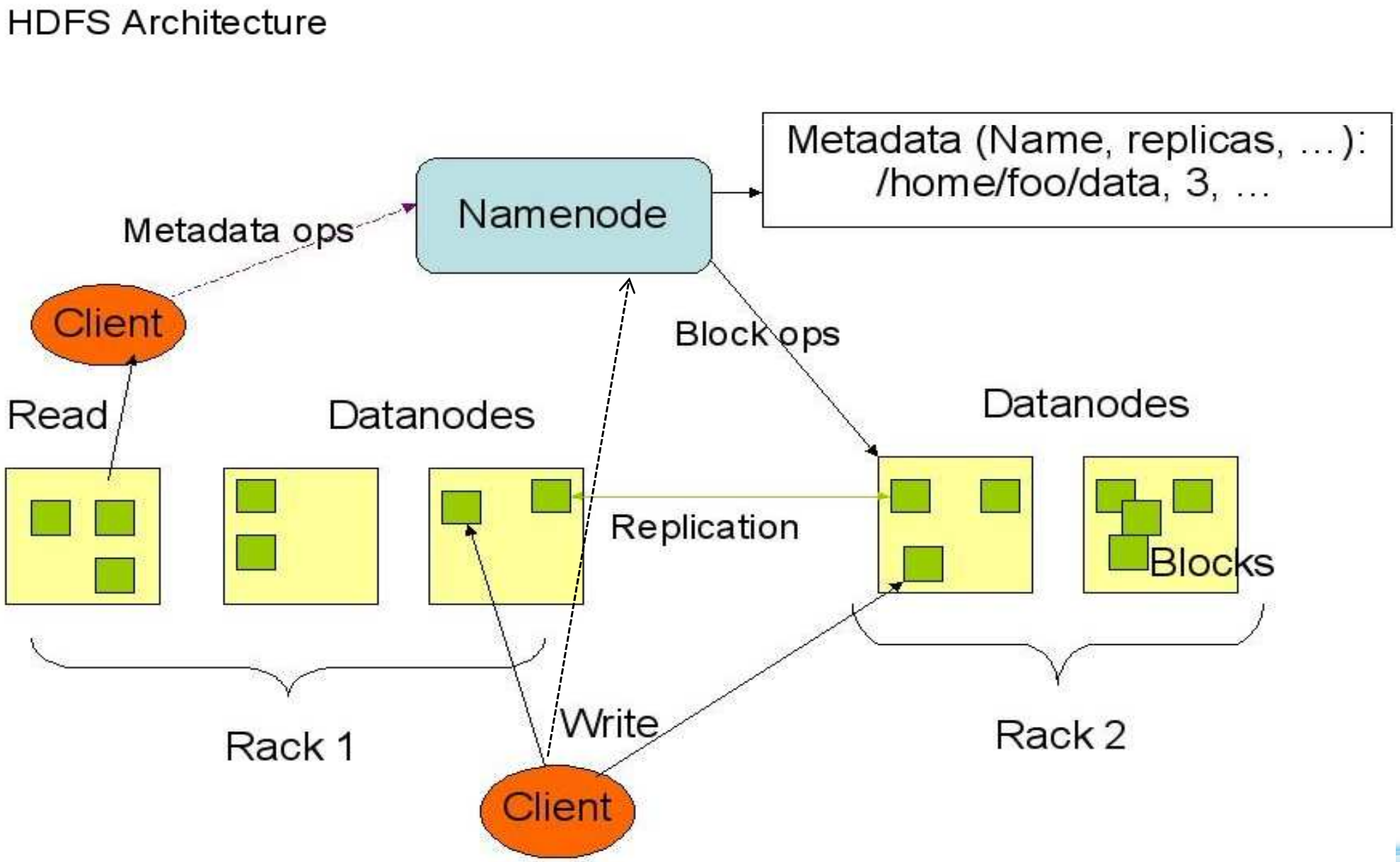


HDFS 設計準則

- 檔案以區塊(block)方式儲存
  - 每個區塊大小遠比多數檔案系統都來得大(預設值為64MB)
- 透過複本機制來提高可靠度
  - 每個區塊至少備分到三台以上的DataNode
- 單一master (NameNode) 來協調存取及屬性資料(metadata)
  - 簡易的集中控管機制
- 沒有資料快取機制(No data caching)
  - 快取對於大資料集與串流讀取沒太大幫助
- 熟悉的介面，但客制化的API
  - 簡化問題；專注於分散式應用



管理資料

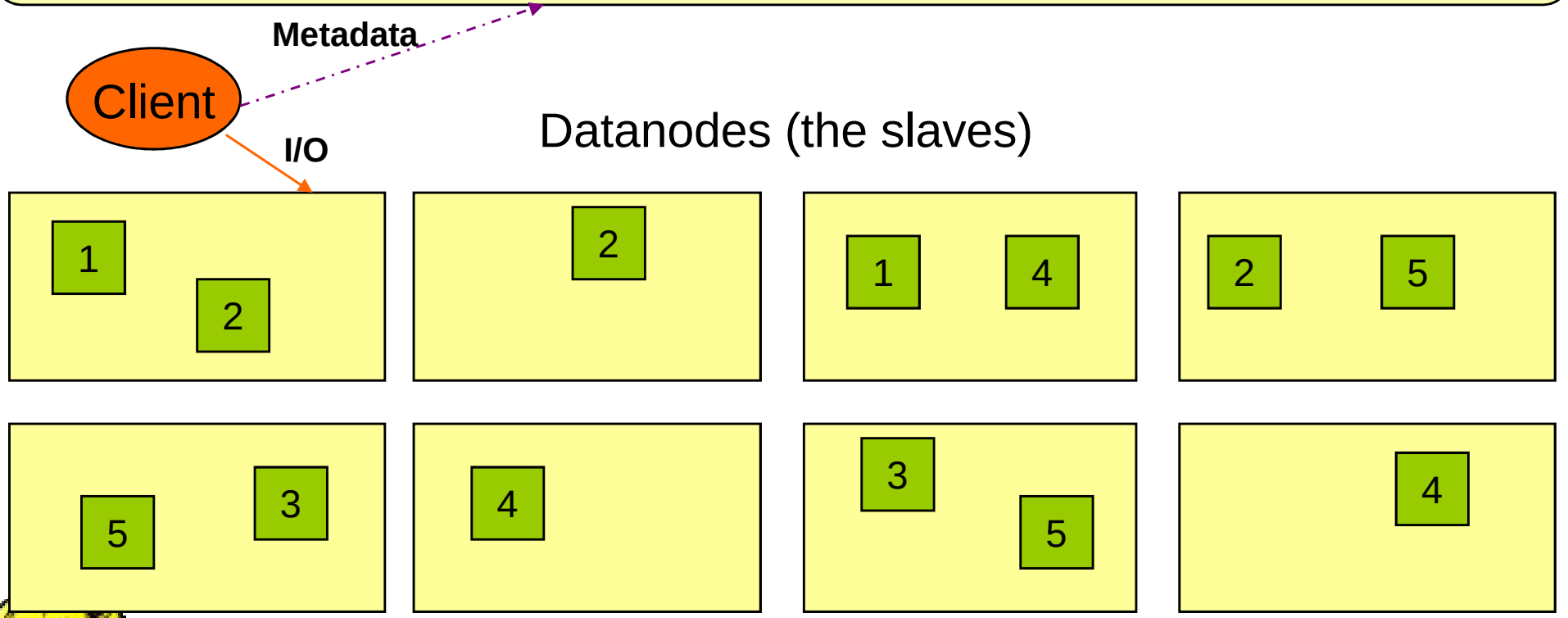

HDFS Architecture
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Namenode
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

HDFS 運作

Namenode (the master)

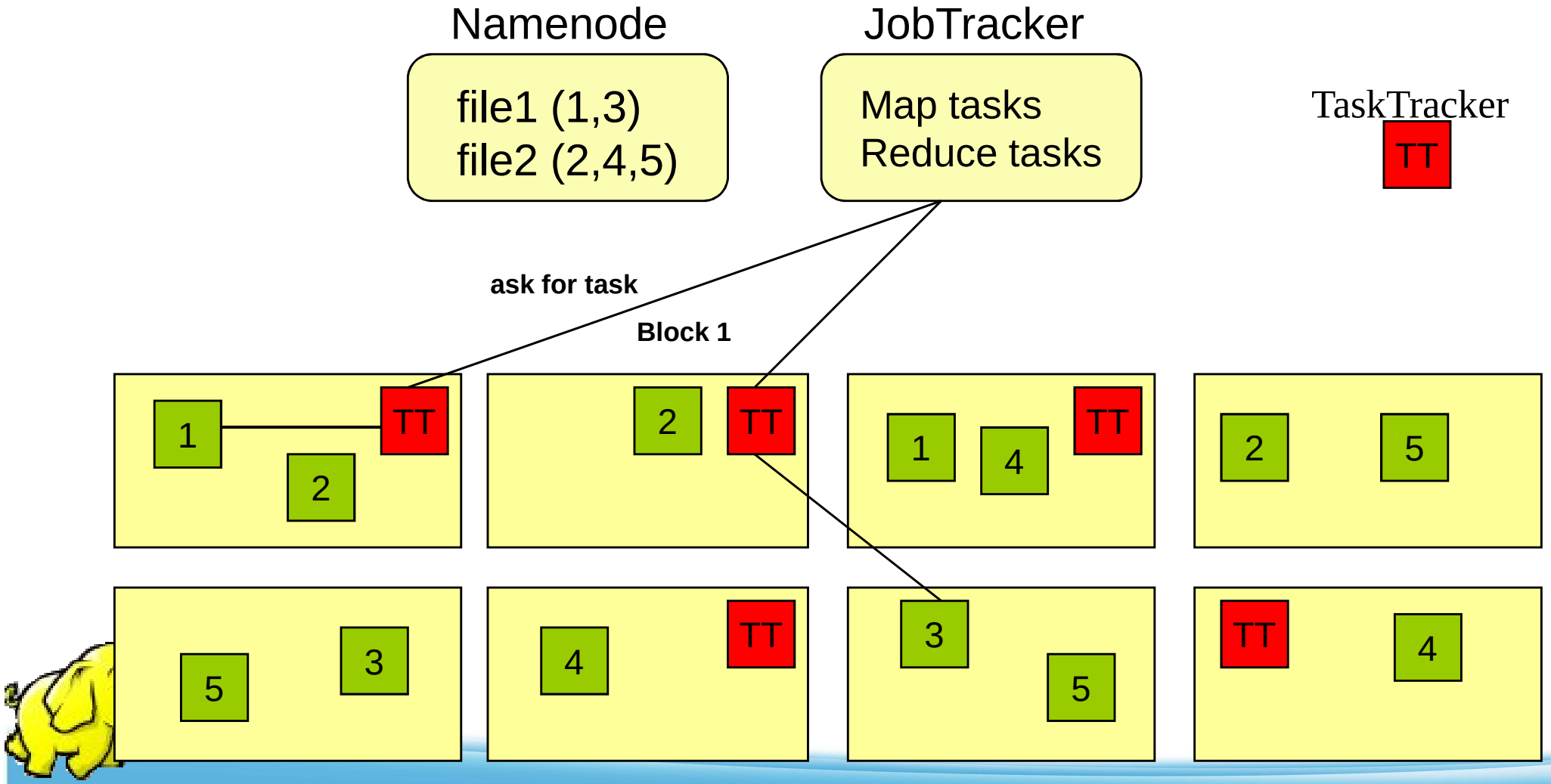
檔案路徑 – 副本數 ， 由哪幾個block組成

name:/users/joeYahoo/myFile - copies:2, blocks:{1,3}
name:/users/bobYahoo/someData.gzip, copies:3, blocks:{2,4,5}

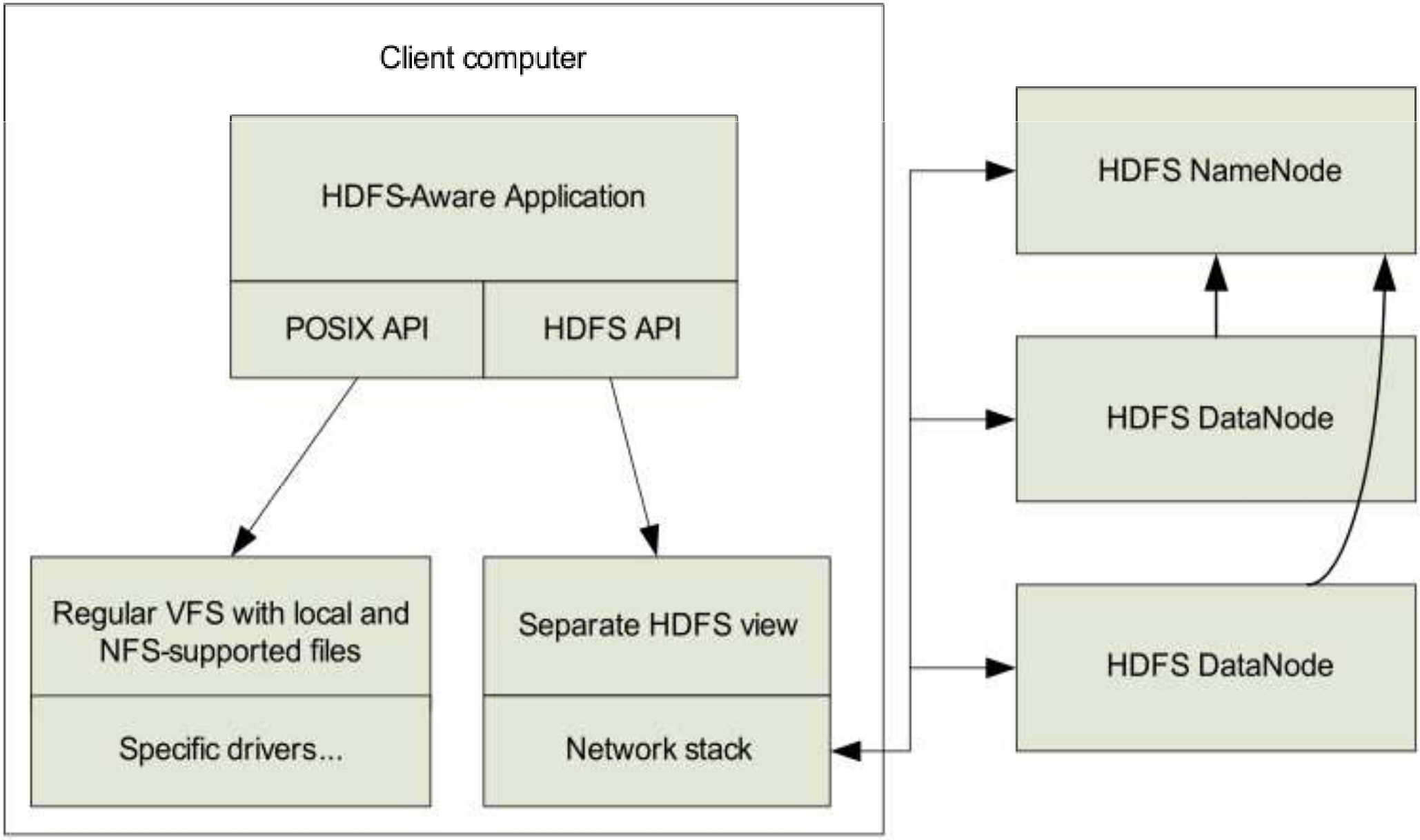


HDFS 運作

- 目的：提高系統的可靠性與讀取的效率
  - 可靠性：節點失效時讀取副本已維持正常運作
  - 讀取效率：分散讀取流量（但增加寫入時效能瓶頸）



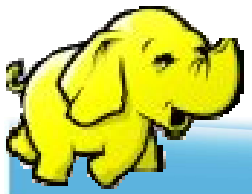
HDFS 系統流程圖


Client computer
HDFS-Aware Application
POSIX API
HDFS API
Regular VFS with local and NFS-supported files
Specific drivers...
Separate HDFS view
Network stack
HDFS NameNode
HDFS DataNode
HDFS DataNode

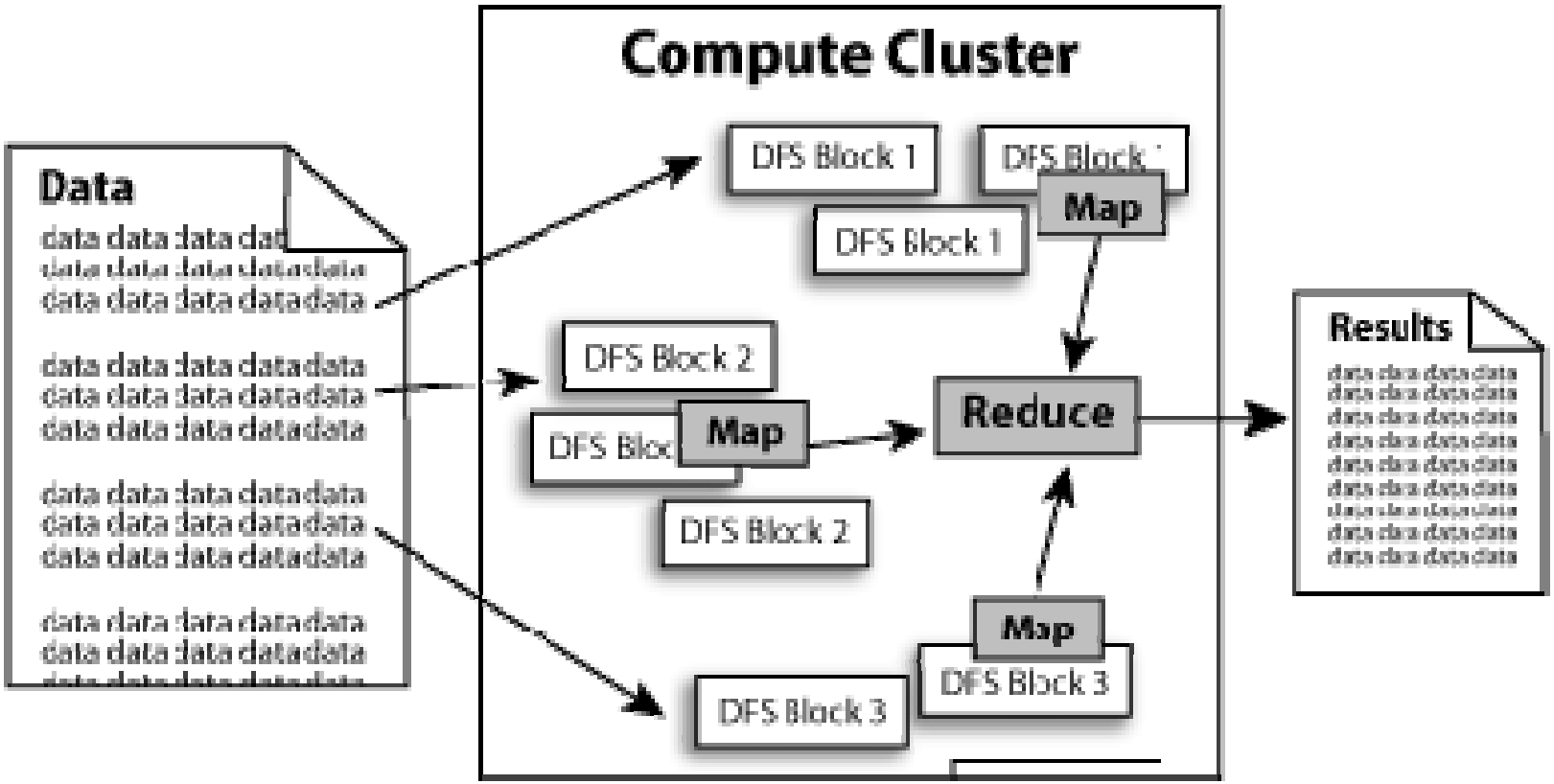
三、Map Reduce 與HDFS

3.B：MapReduce 演算法

Hadoop 的運算方式是透過Map/Reduce 演算法構成，也意謂著，要透過Hadoop API 撰寫平行分散式程式，則一定要懂 Map / Reduce



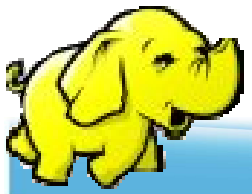
Map / Reduce 定義



MapReduce is a software framework to support distributed computing on large data sets on clusters of computers.

演算法

- Functional Programming : Map Reduce
  - map(...) :
    - [1,2,3,4] – (*2) -> [2,4,6,8]
  - reduce(...):
    - [1,2,3,4] - (sum) -> 10
  - 對應演算法中的Divide and conquer
  - 將問題分解成很多個小問題之後，再做總和



58

Map

- One-to-one Mapper

```
let map(k, v) =
Emit(k.toUpper(),
v.toUpper())
```

(“Foo”, “other”) → (“FOO”, “OTHER”)
(“key2”, “data”) → (“KEY2”, “DATA”)

- Explode Mapper

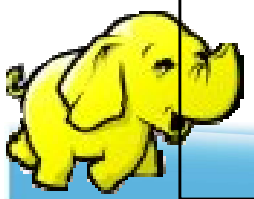
```
let map(k, v) =
foreach char c in v:
  emit(k, c)
```

(“A”, “cats”) → (“A”, “c”), (“A”, “a”),
(“A”, “t”), (“A”, “s”)

- Filter Mapper

```
let map(k, v) =
if (isPrime(v))
then emit(k, v)
```

(“foo”, 7) → (“foo”, 7)
(“test”, 10) → (nothing)

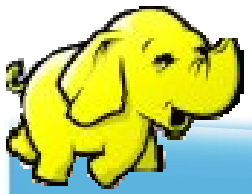


Reduce

Example: Sum Reducer

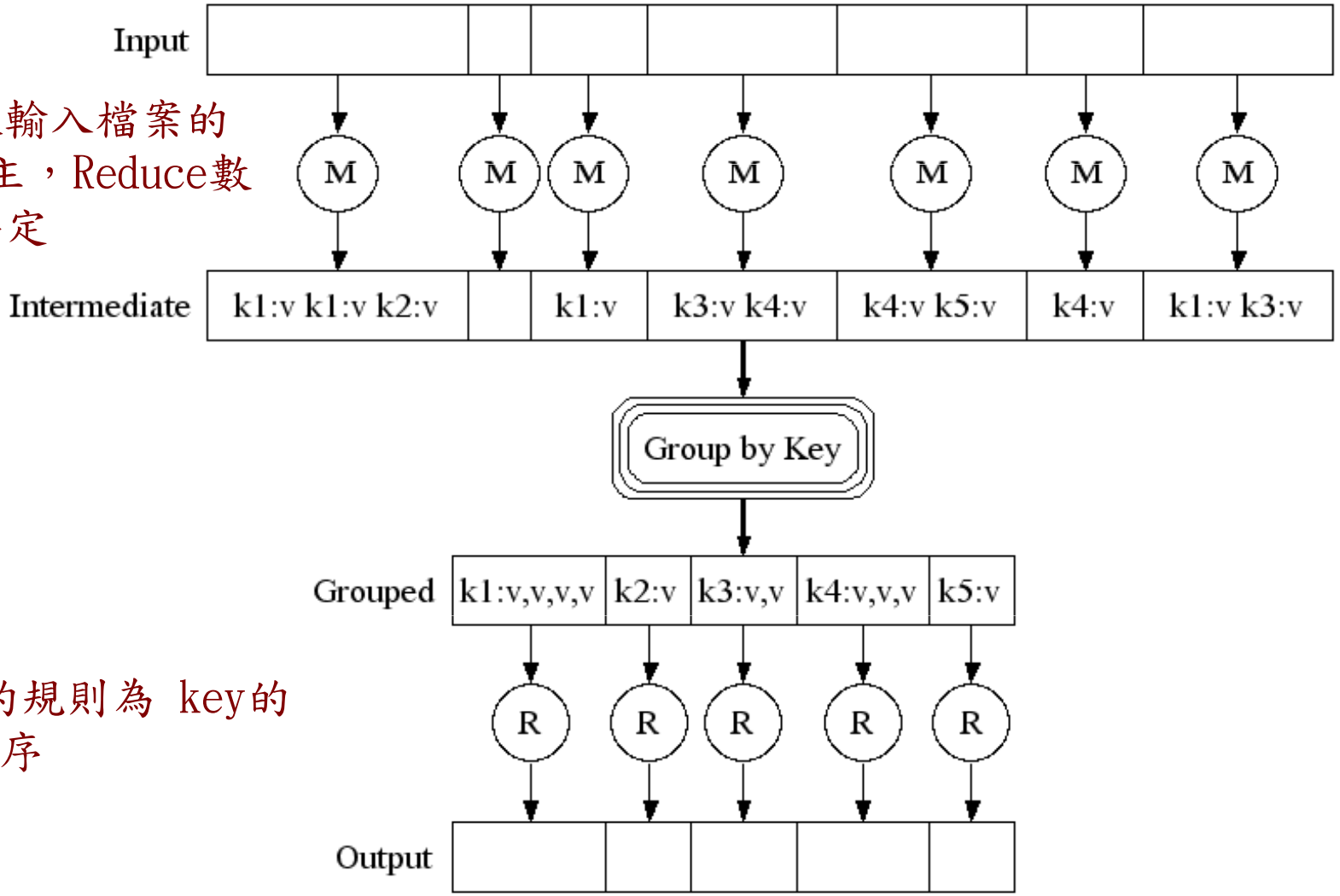
```
let reduce(k, vals) =
    sum = 0
    foreach int v in vals:
        sum += v
    emit(k, sum)
```

("A", [42, 100, 312]) → ("A", 454)
("B", [12, 6, -2]) → ("B", 16)



MapReduce 單台

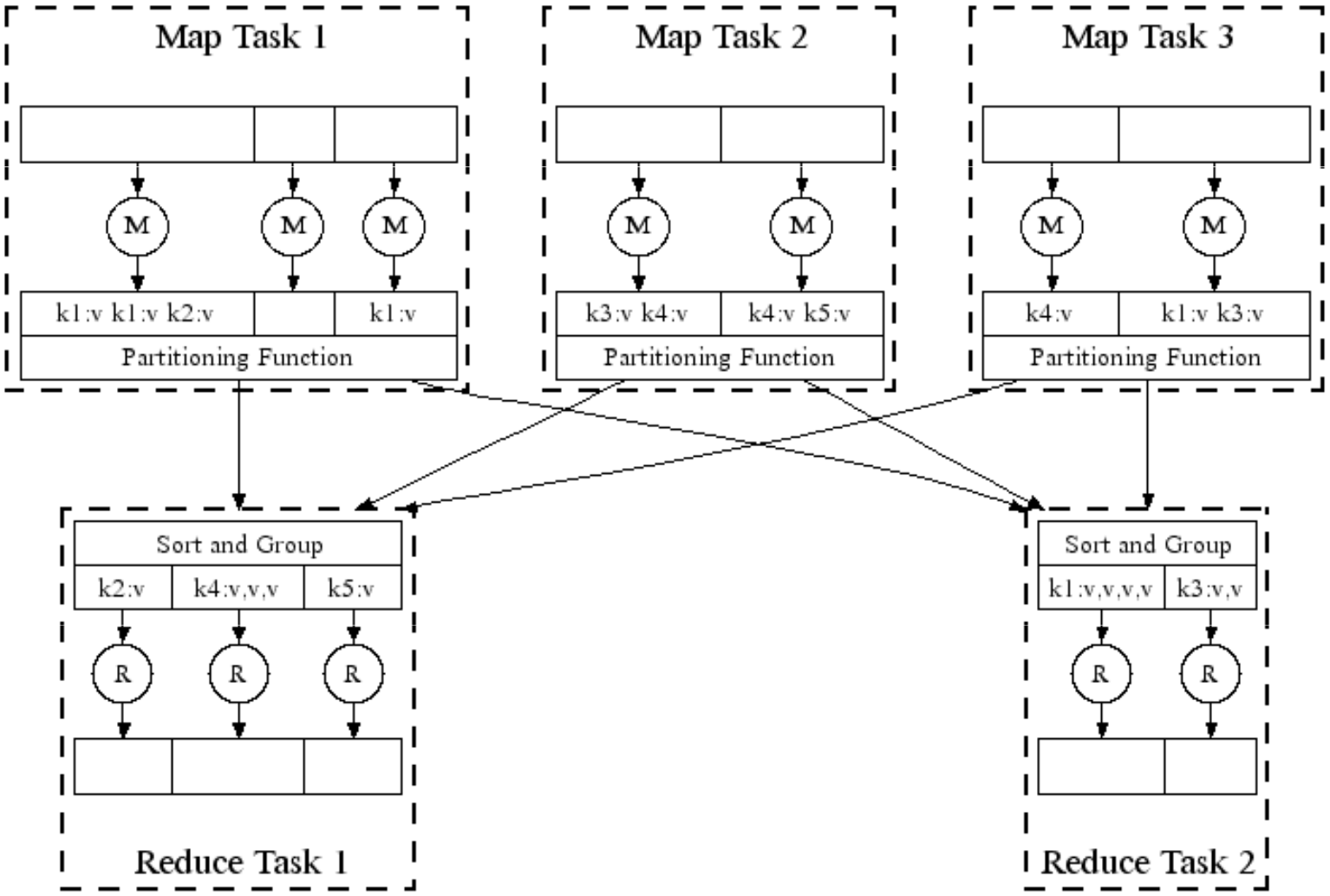
Map 數量依輸入檔案的block數為主，Reduce數量由系統決定

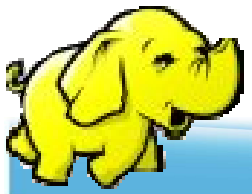


Sort 的規則為 key的字母順序

61

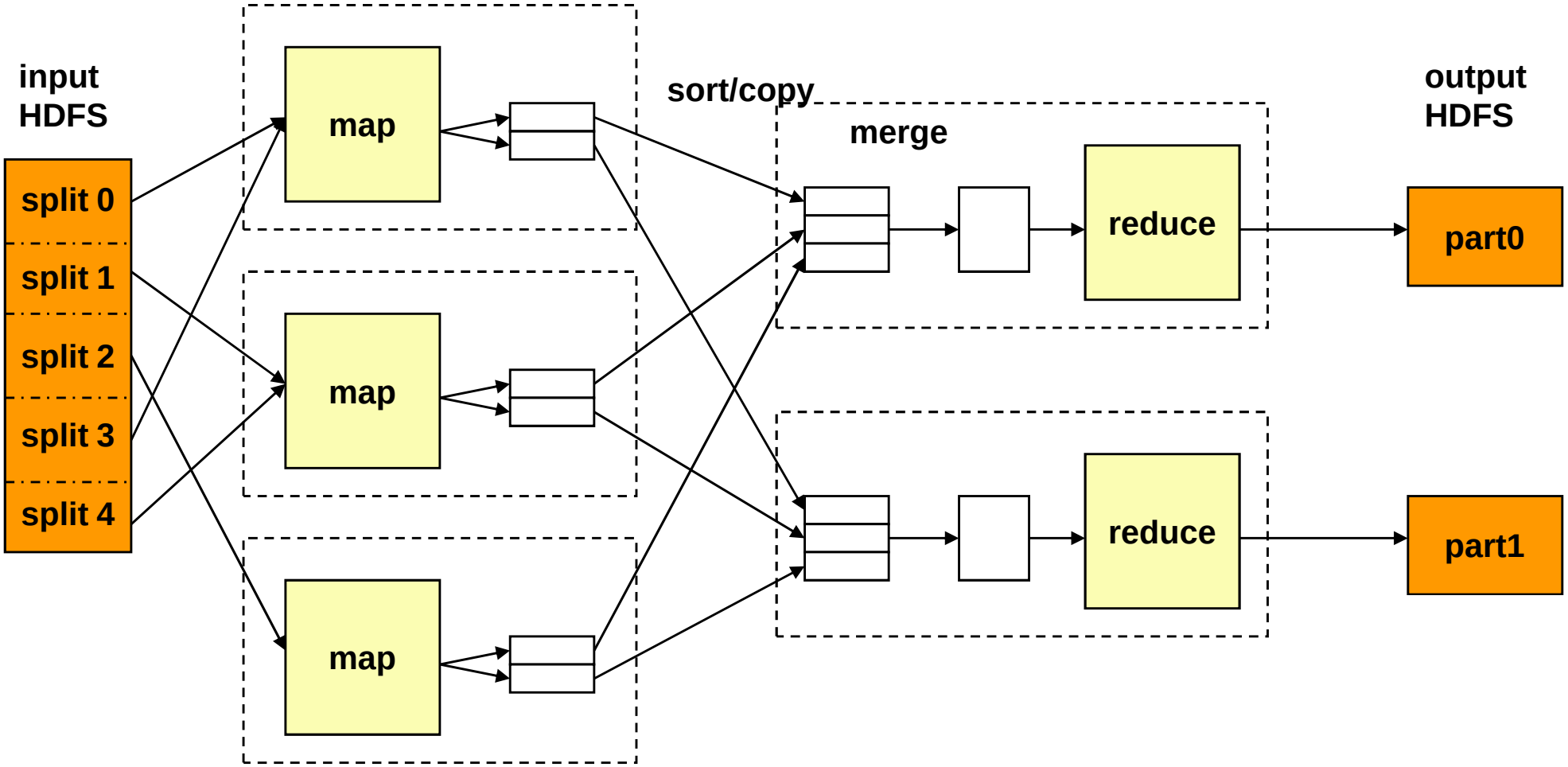
MapReduce 多台





62

MapReduce 運作流程



JobTracker跟NameNode取得需要運算的blocks

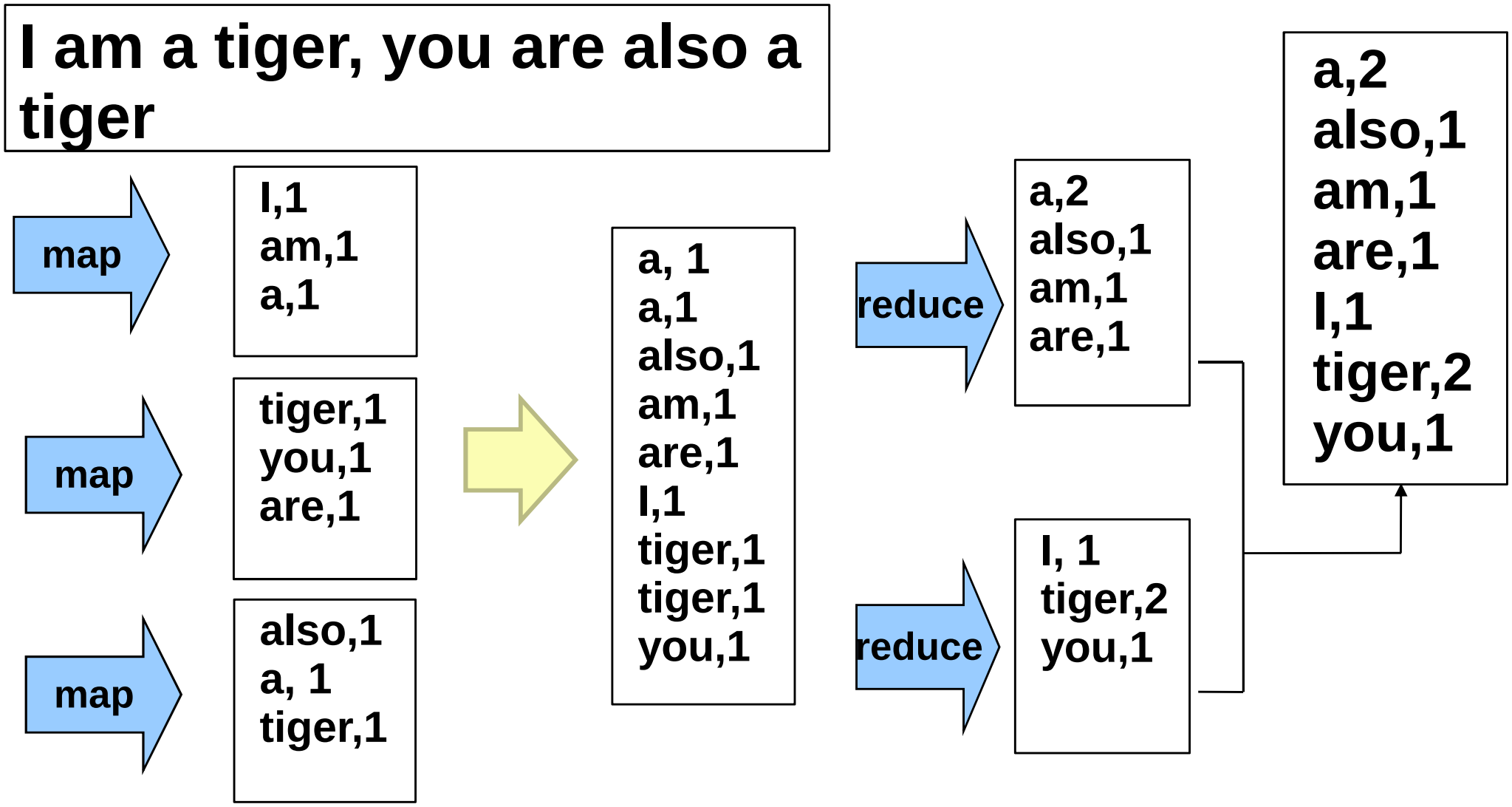
JobTracker選數個TaskTracker來作Map運算，產生些中間檔案

JobTracker將中間檔案整合排序後，複製到需要的TaskTracker去

JobTracker派遣TaskTracker作reduce

reduce完後通知JobTracker與Namenode以產生output

實例範例



JobTracker先選了三個Tracker做map

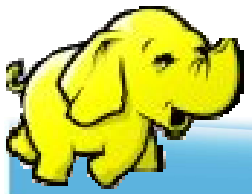
Map結束後，hadoop進行中間資料的整理與排序

JobTracker再選兩個TaskTracker作reduce

四、寫Code 環境準備

4.A : Console 端編譯與執行

4.B : 透過 Eclipse 開發與測試運算



Java 之編譯與執行

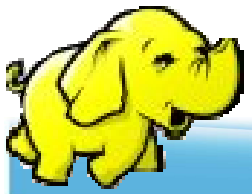
1. 編譯
   - javac △ -classpath △ hadoop-*-core.jar △ -d △ MyJava △ MyCode.java
2. 封裝
   - jar △ -cvf △ MyJar.jar △ -C △ MyJava △ .
3. 執行
   - bin/hadoop △ jar △ MyJar.jar △ MyCode △ HDFS_Input/ △ HDFS_Output/

- 所在的執行目錄為Hadoop_Home
- ./MyJava = 編譯後程式碼目錄
- My jar. jar = 封裝後的編譯檔
- 先放些文件檔到HDFS上的input目錄
- ./input; ./ouput = hdfs的輸入、輸出目錄

WordCount1 練習 (I)

1. cd $HADOOP_HOME; mkdir input_local
2. echo "I like NCHC Cloud Course." > input_local/input1
3. echo "I like nchc Cloud Course, and we enjoy this crouse." > input_local/input2
4. bin/hadoop dfs -put input_local input
5. bin/hadoop dfs -ls input

```
waue@vPro:/opt/hadoop$ bin/hadoop dfs -ls input
Found 2 items
-rw-r--r--   1 waue supergroup         26 2009-03-22 12:15 /user/waue/input/input1
-rw-r--r--   1 waue supergroup         52 2009-03-22 12:15 /user/waue/input/input2
waue@vPro:/opt/hadoop$
```



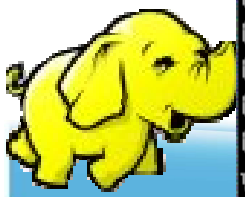
WordCount1 練習 (II)

1. 編輯WordCount.java
   http://trac.nchc.org.tw/cloud/attachment/wiki/jazz/Hadoop_Lab6/WordCount.java?format=raw
2. mkdir MyJava
3. javac -classpath hadoop-*-core.jar -d MyJava WordCount.java
4. jar -cvf wordcount.jar -C MyJava .
5. bin/hadoop jar wordcount.jar WordCount input/ output/

- 所在的執行目錄為Hadoop_Home（因為hadoop-*-core.jar）
- javac編譯時需要classpath, 但hadoop jar時不用
- wordcount.jar = 封裝後的編譯檔，但執行時需告知class name
- Hadoop進行運算時，只有 input 檔要放到hdfs上，以便hadoop分析運算，執行檔（wordcount.jar）不需上傳，也不需每個node都放，程式的載入交由java處理

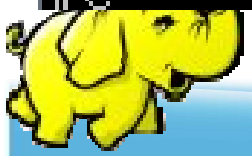
WordCount1 練習(III)

```
waue@vPro:/opt/hadoop$ bin/hadoop dfs -put input input
waue@vPro:/opt/hadoop$ mkdir MyJava
waue@vPro:/opt/hadoop$ javac -classpath hadoop-*-core.jar -d MyJava WordCount.java
waue@vPro:/opt/hadoop$ jar -cvf wordcount.jar -C MyJava .
新增 manifest
新增：WordCount.class (讀=1516)(寫=740)(壓縮 51%)
新增：WordCount$Reduce.class (讀=1591)(寫=642)(壓縮 59%)
新增：WordCount$Map.class (讀=1918)(寫=795)(壓縮 58%)
waue@vPro:/opt/hadoop$ bin/hadoop jar wordcount.jar WordCount input/ output/
09/03/22 11:39:01 WARN mapred.JobClient: Use GenericOptionsParser for parsing the argu
ments. Applications should implement Tool for the same.
09/03/22 11:39:01 INFO mapred.FileInputFormat: Total input paths to process : 1
09/03/22 11:39:01 INFO mapred.FileInputFormat: Total input paths to process : 1
09/03/22 11:39:02 INFO mapred.JobClient: Running job: job_200903201526_0007
09/03/22 11:39:03 INFO mapred.JobClient:  map 0% reduce 0%
09/03/22 11:39:08 INFO mapred.JobClient:  map 100% reduce 0%
09/03/22 11:39:15 INFO mapred.JobClient: Job complete: job_200903201526_0007
09/03/22 11:39:15 INFO mapred.JobClient: Counters: 16
09/03/22 11:39:15 INFO mapred.JobClient:   File Systems
09/03/22 11:39:15 INFO mapred.JobClient:     HDFS bytes read=320950
09/03/22 11:39:15 INFO mapred.JobClient:     HDFS bytes written=130568
09/03/22 11:39:15 INFO mapred.JobClient:     Local bytes read=168448
09/03/22 11:39:15 INFO mapred.JobClient:     Local bytes written=336932
09/03/22 11:39:15 INFO mapred.JobClient:   Job Counters
09/03/22 11:39:15 INFO mapred.JobClient:     Launched reduce tasks=1
09/03/22 11:39:15 INFO mapred.JobClient:     Launched map tasks=1
09/03/22 11:39:15 INFO mapred.JobClient:     Data-local map tasks=1
09/03/22 11:39:15 INFO mapred.JobClient:   Map-Reduce Framework
09/03/22 11:39:15 INFO mapred.JobClient:     Reduce input groups=9284
09/03/22 11:39:15 INFO mapred.JobClient:     Combine output records=18568
09/03/22 11:39:15 INFO mapred.JobClient:     Map input records=7868
09/03/22 11:39:15 INFO mapred.JobClient:     Reduce output records=9284
09/03/22 11:39:15 INFO mapred.JobClient:     Map output bytes=445846
09/03/22 11:39:15 INFO mapred.JobClient:     Map input bytes=320950
09/03/22 11:39:15 INFO mapred.JobClient:     Combine input records=47227
09/03/22 11:39:15 INFO mapred.JobClient:     Map output records=37943
09/03/22 11:39:15 INFO mapred.JobClient:     Reduce input records=9284
waue@vPro:/opt/hadoop$
```



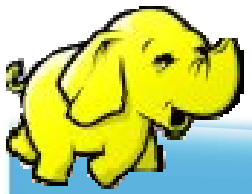
WordCount1 練習(IV)

```
waue@vPro:/opt/hadoop$ bin/hadoop dfs -cat output/part-00000
Cloud   2
Course, 1
Course. 1
I       2
NCHC    1
and     1
course. 1
enjoy   1
like    2
nchc    1
this    1
we      1
```



BTW …

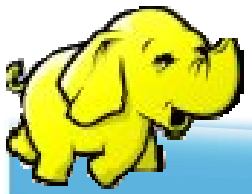
- 雖然Hadoop框架是用Java實作，但Map/Reduce應用程序則不一定要用 Java來寫
- Hadoop Streaming ：
  - 執行作業的工具，使用者可以用其他語言（如：PHP）套用到Hadoop的mapper和reducer
- Hadoop Pipes：C++ API



四、寫Code 環境準備

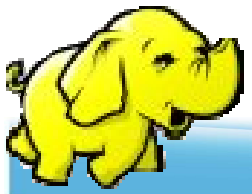
4.A : Console 端編譯與執行

4.B：透過 Eclipse 開發與測試運算



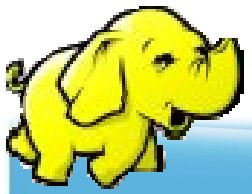
Requirements

- Hadoop 0.20.0 up
- Java 1.6
- Eclipse 3.3 up
- Hadoop Eclipse Plugin 0.20.0 up

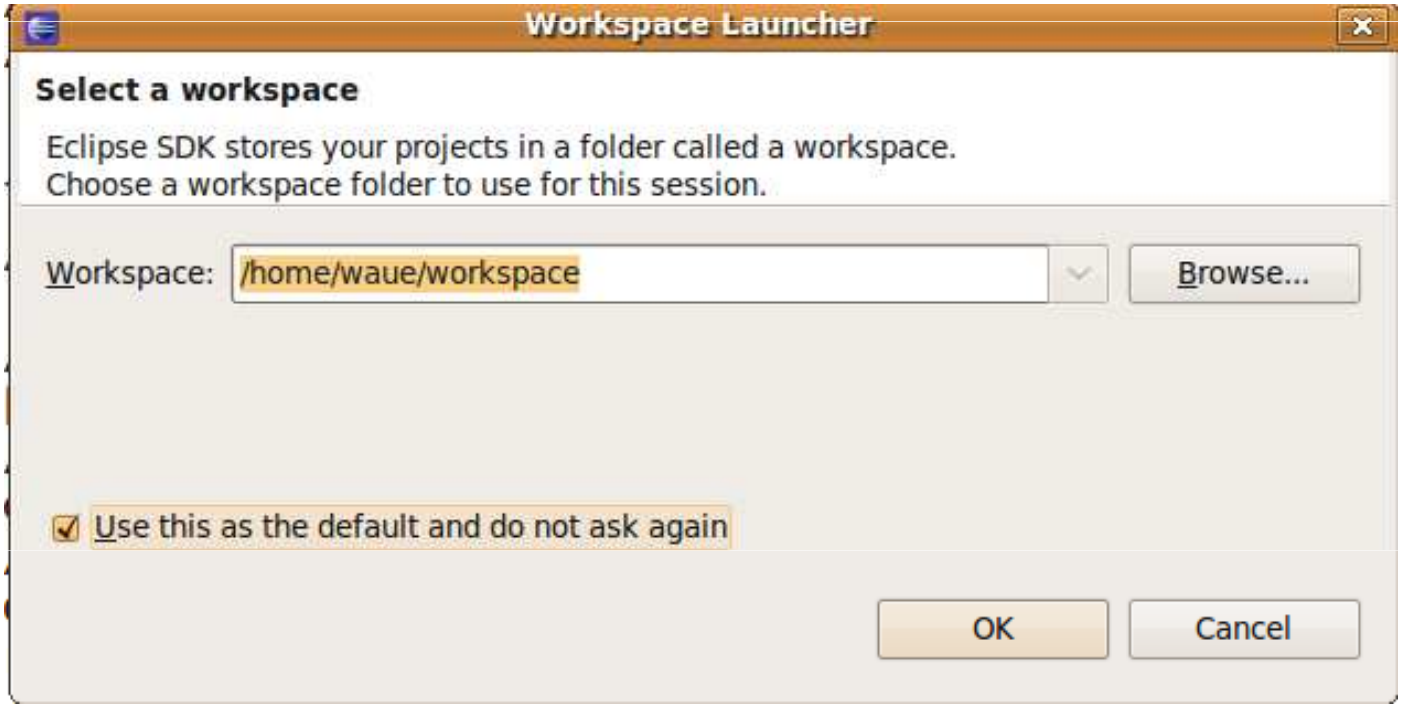


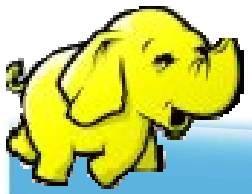
安裝Hadoop Eclipse Plugin

- Hadoop Eclipse Plugin 0.20.0
  - From $Hadoop_0.20.0_home/contrib/eclipse-plugin/hadoop-0.20.0-eclipse-plugin.jar
- Hadoop Eclipse Plugin 0.20.1
  - Compiler needed
  - Or download from http://hadoop-eclipse-plugin.googlecode.com/files/hadoop-0.20.1-eclipse-plugin.jar
- copy to $Eclipse_home/plugins/

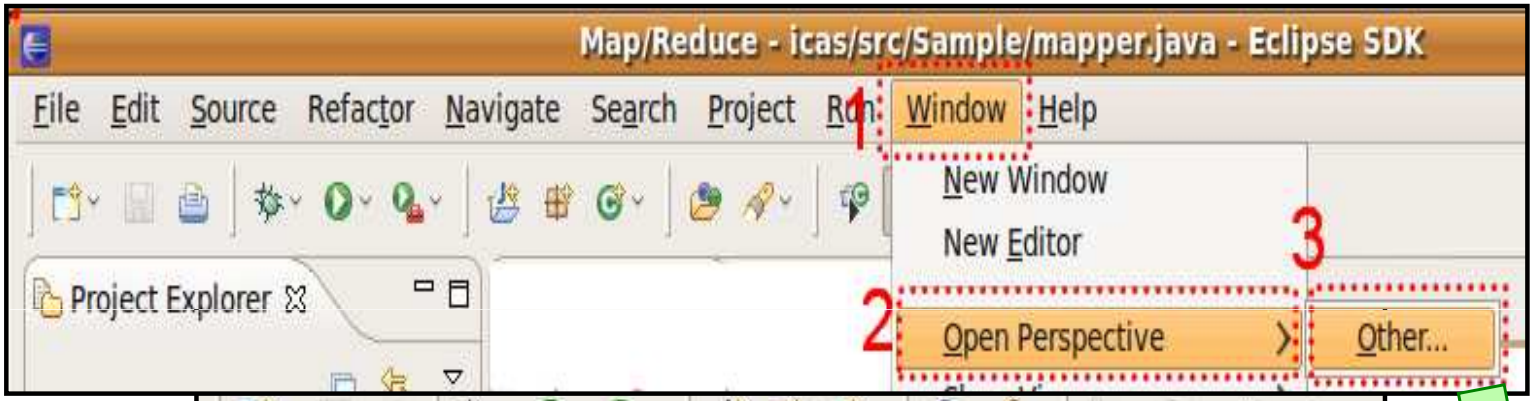


1 打開Eclipse, 設定專案目錄

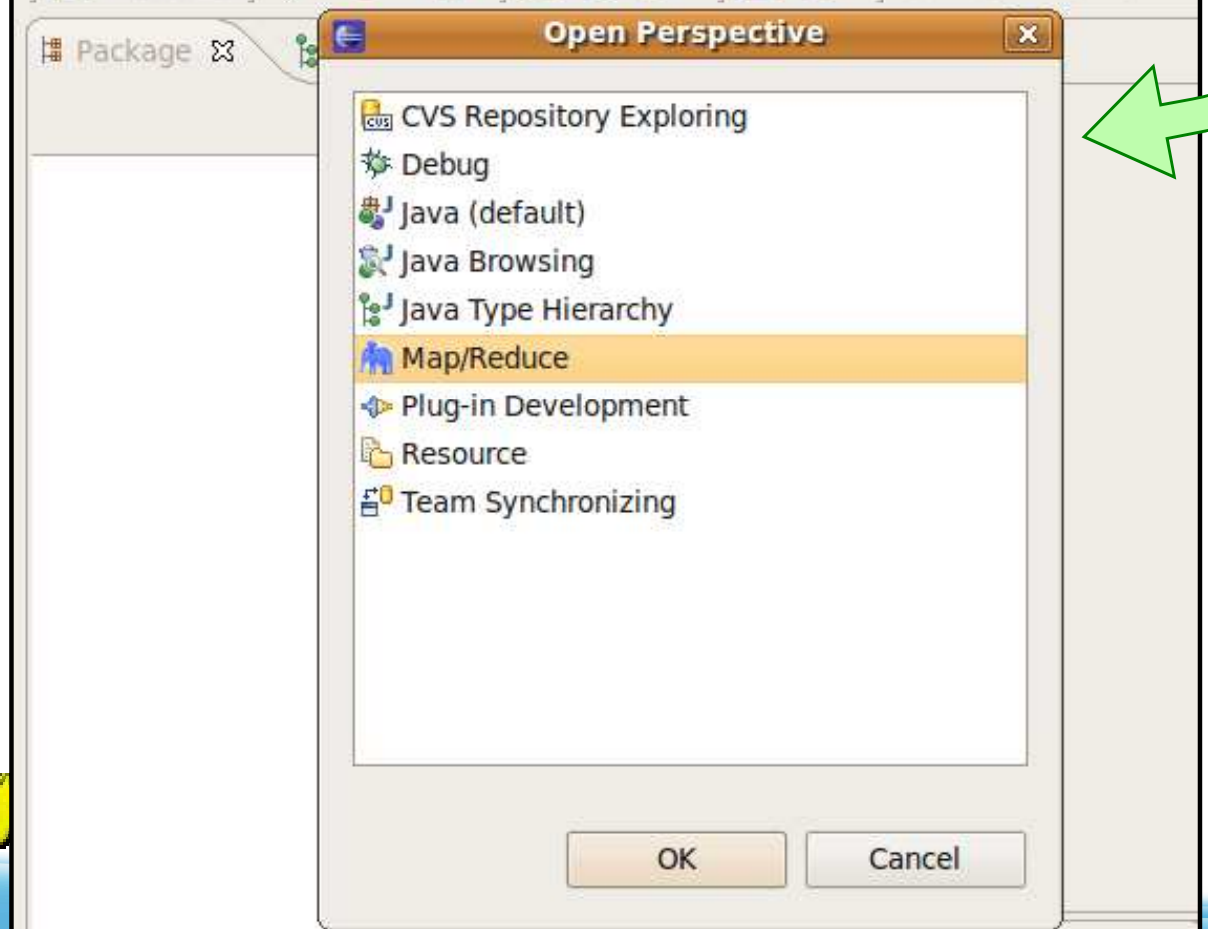




2. 使用Hadoop mode視野

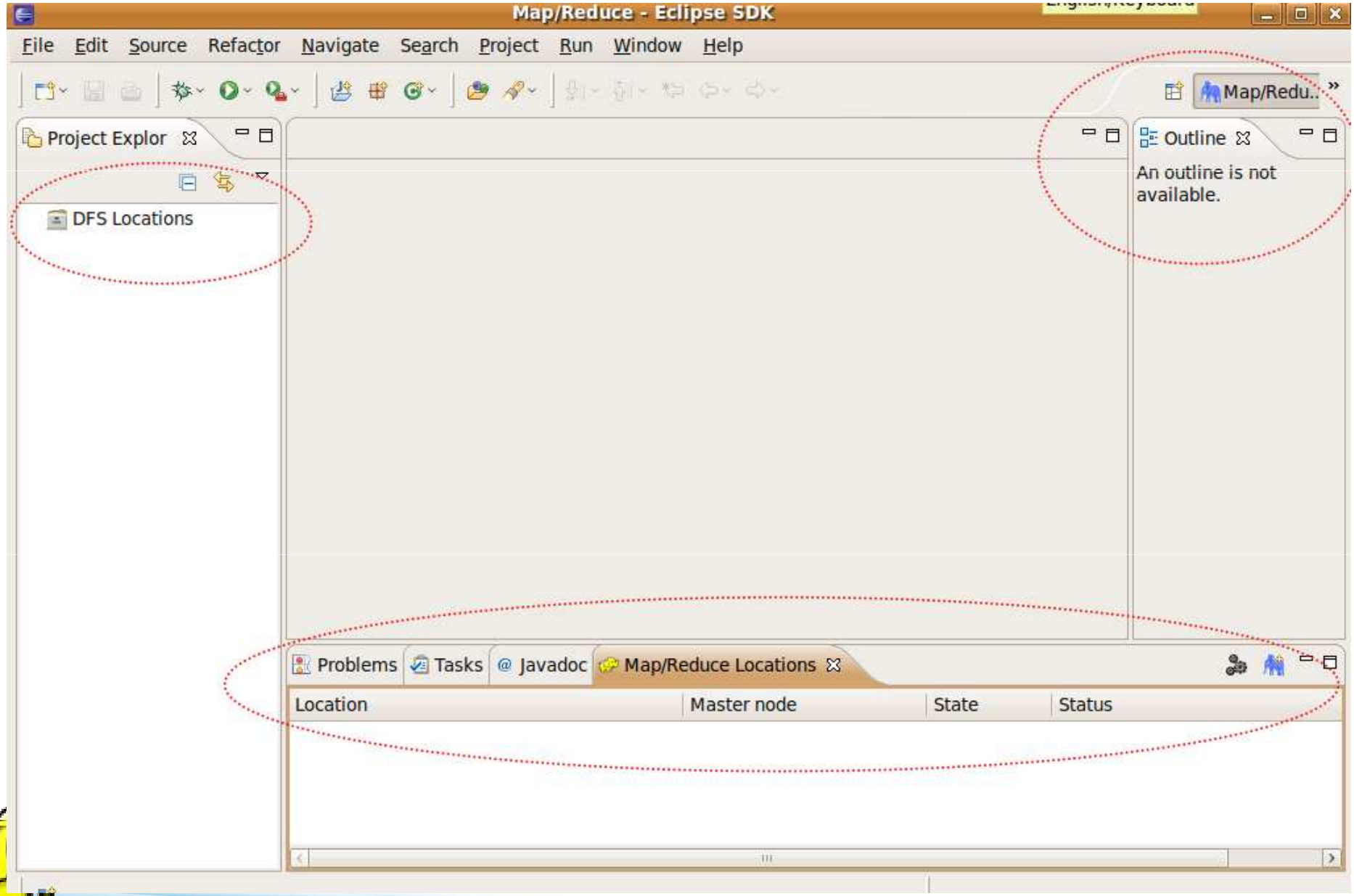


1. Window
2. Open Perspective
3. Other

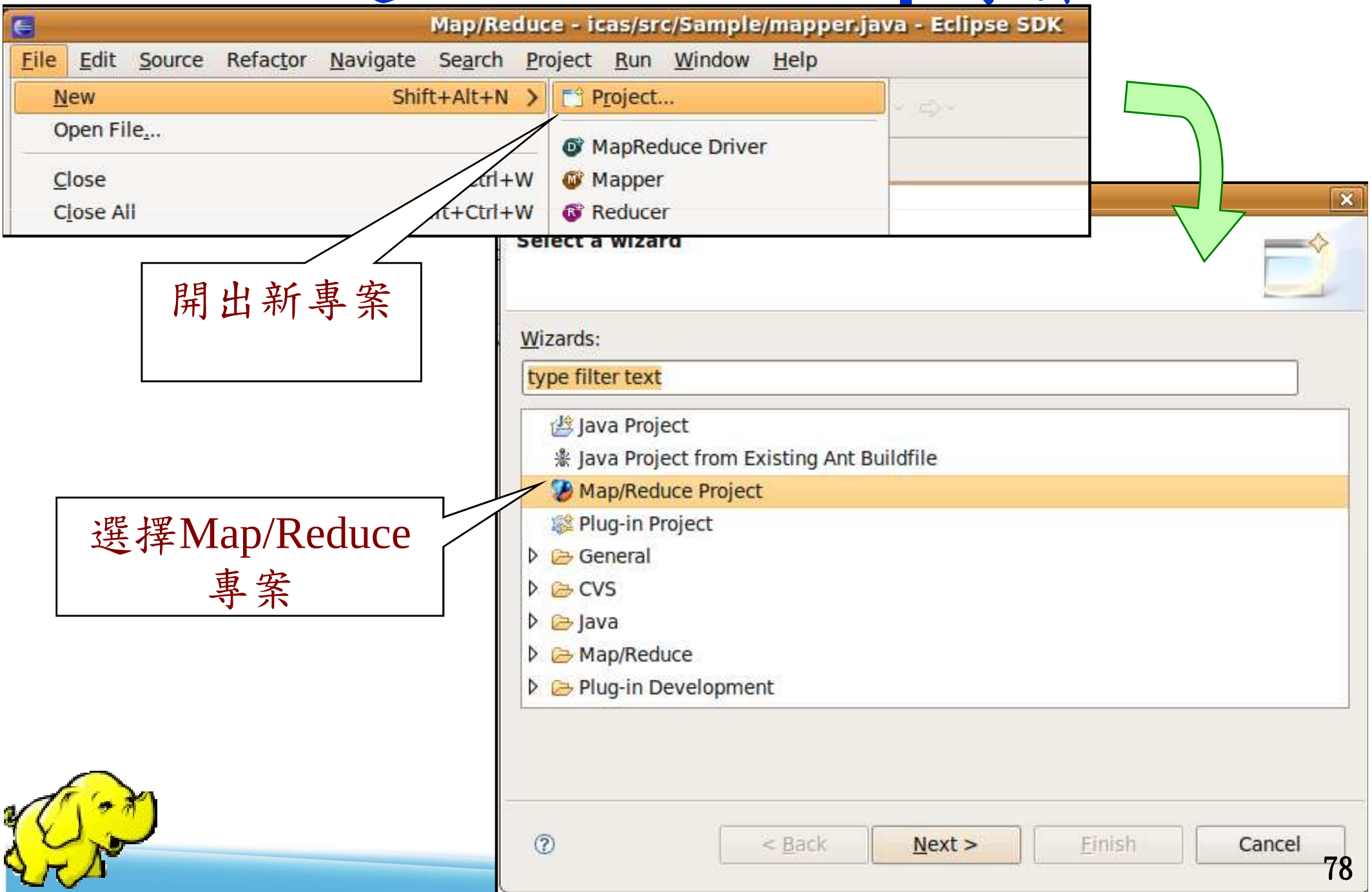


若有看到MapReduce的大象圖示代表Hadoop Eclipse plugin有安裝成功， 若沒有請檢查是否有安之裝正確

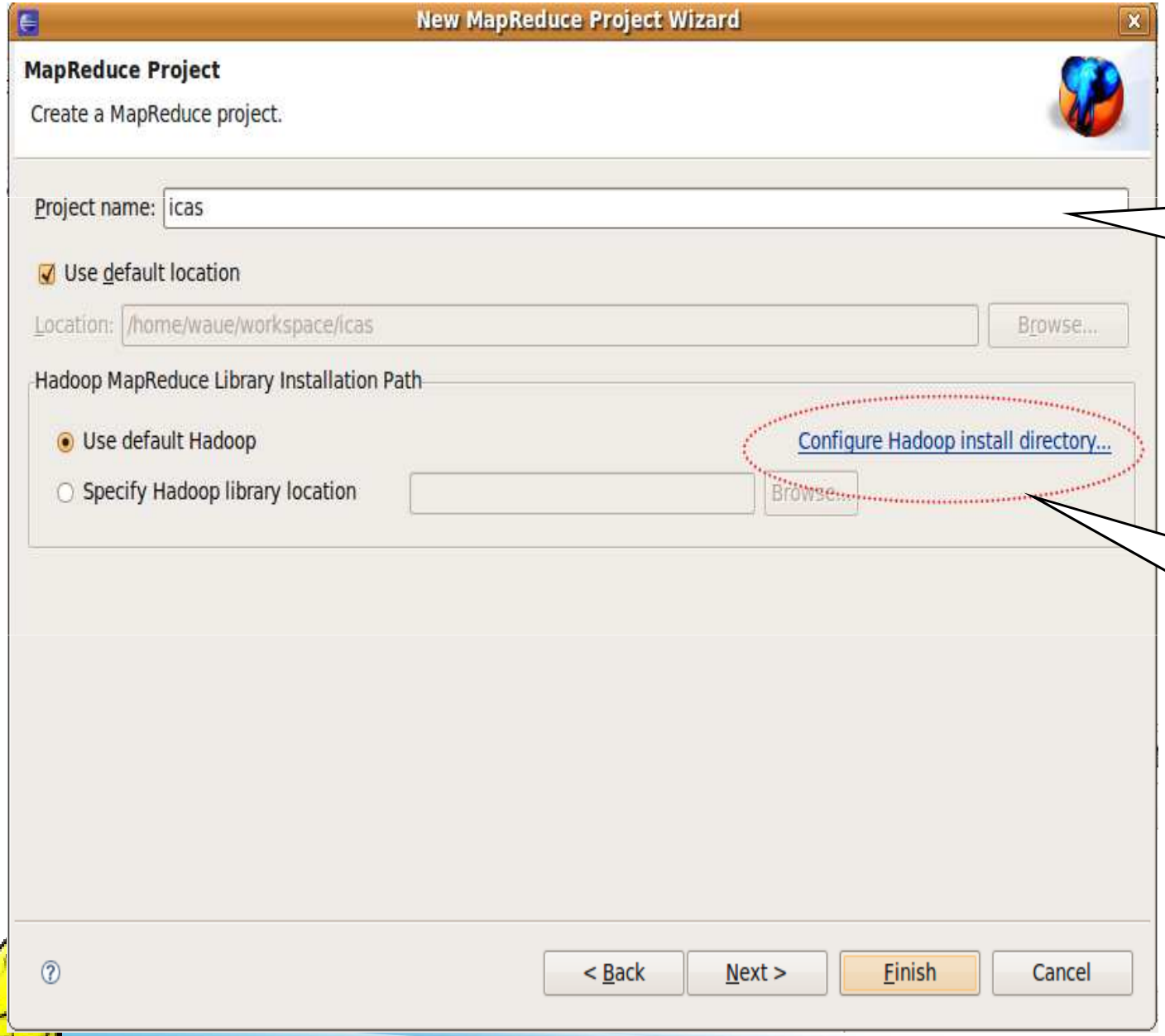
3. 使用Hadoop視野，主畫面將出現三個功能



4.建立一個Hadoop專案



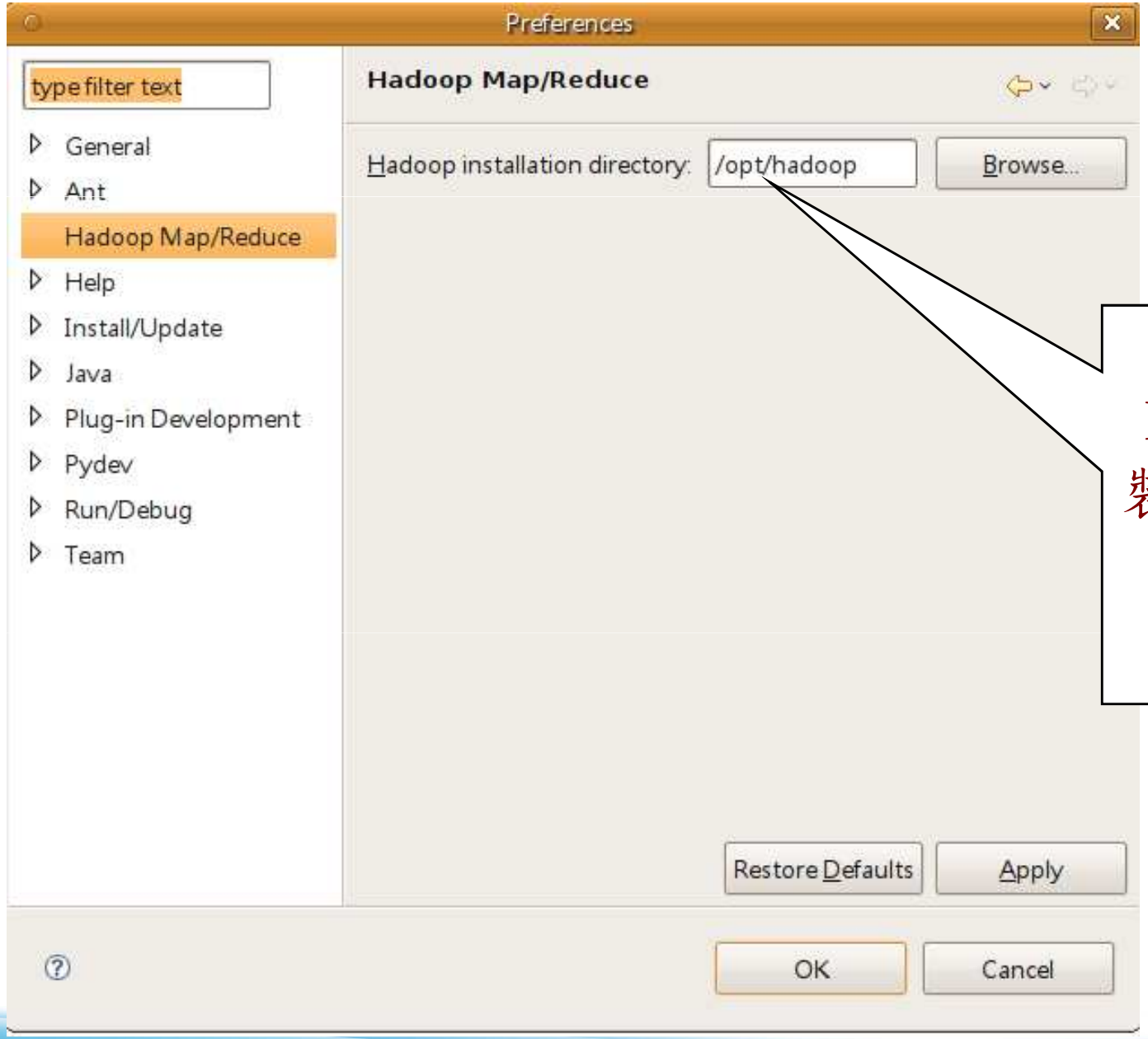
4-1. 輸入專案名稱並點選設定 Hadoop安裝路徑



由此設定專案名稱

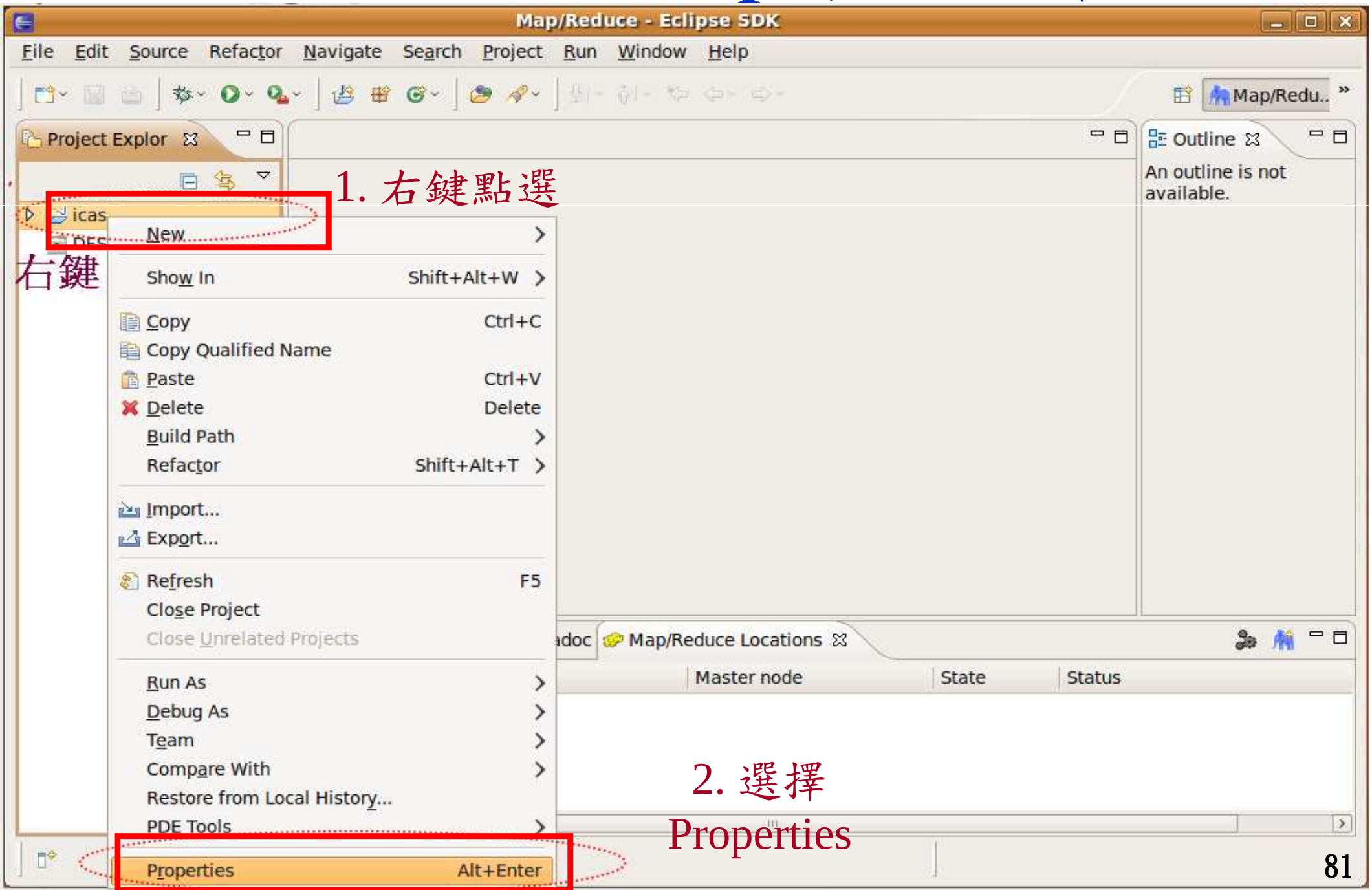
由此設定Hadoop的安裝路徑

4-1-1. 填入Hadoop安裝路徑



於此輸入您Hadoop的安裝路徑，之後選擇 ok

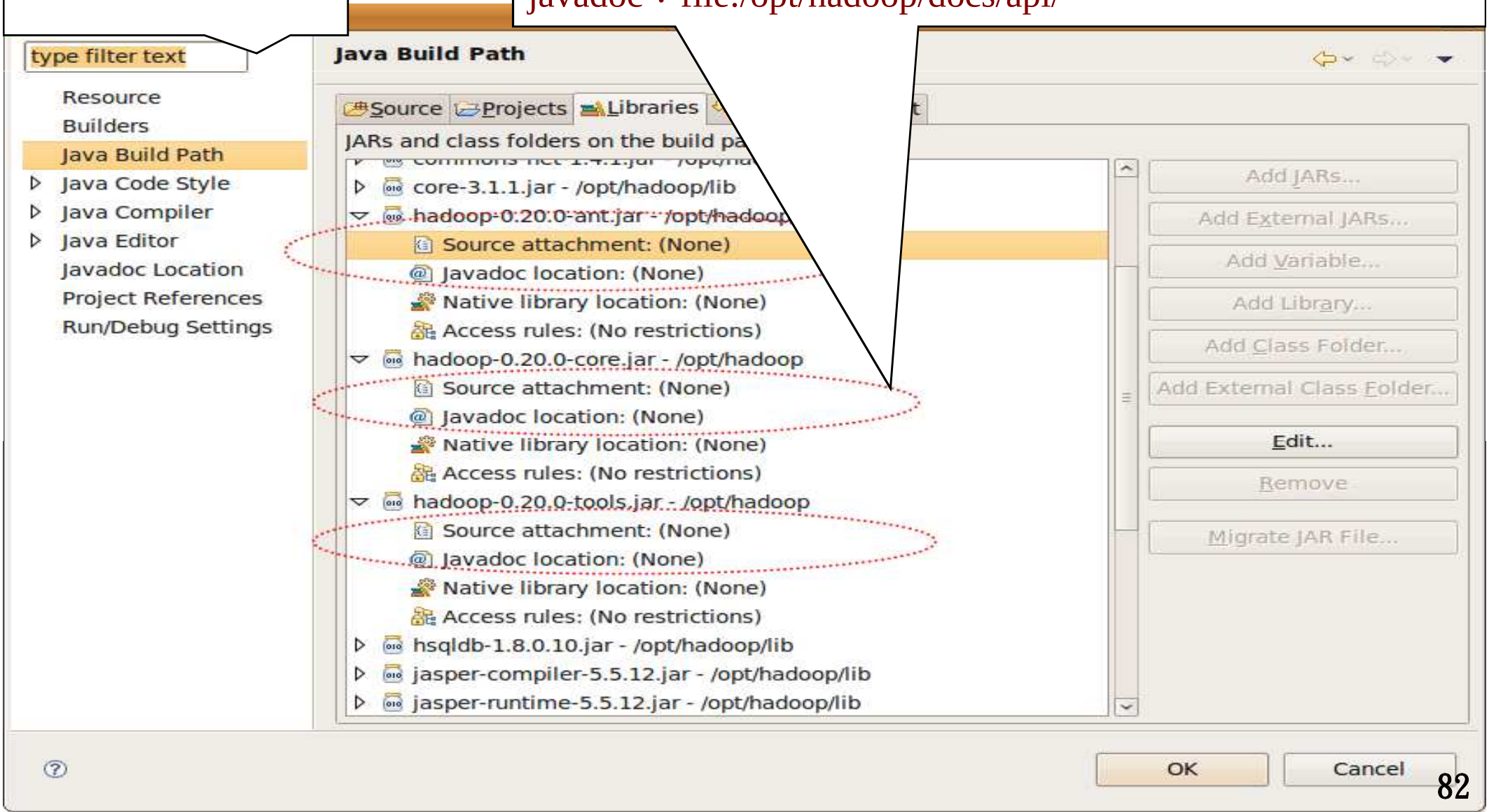
5. 設定Hadoop專案細節



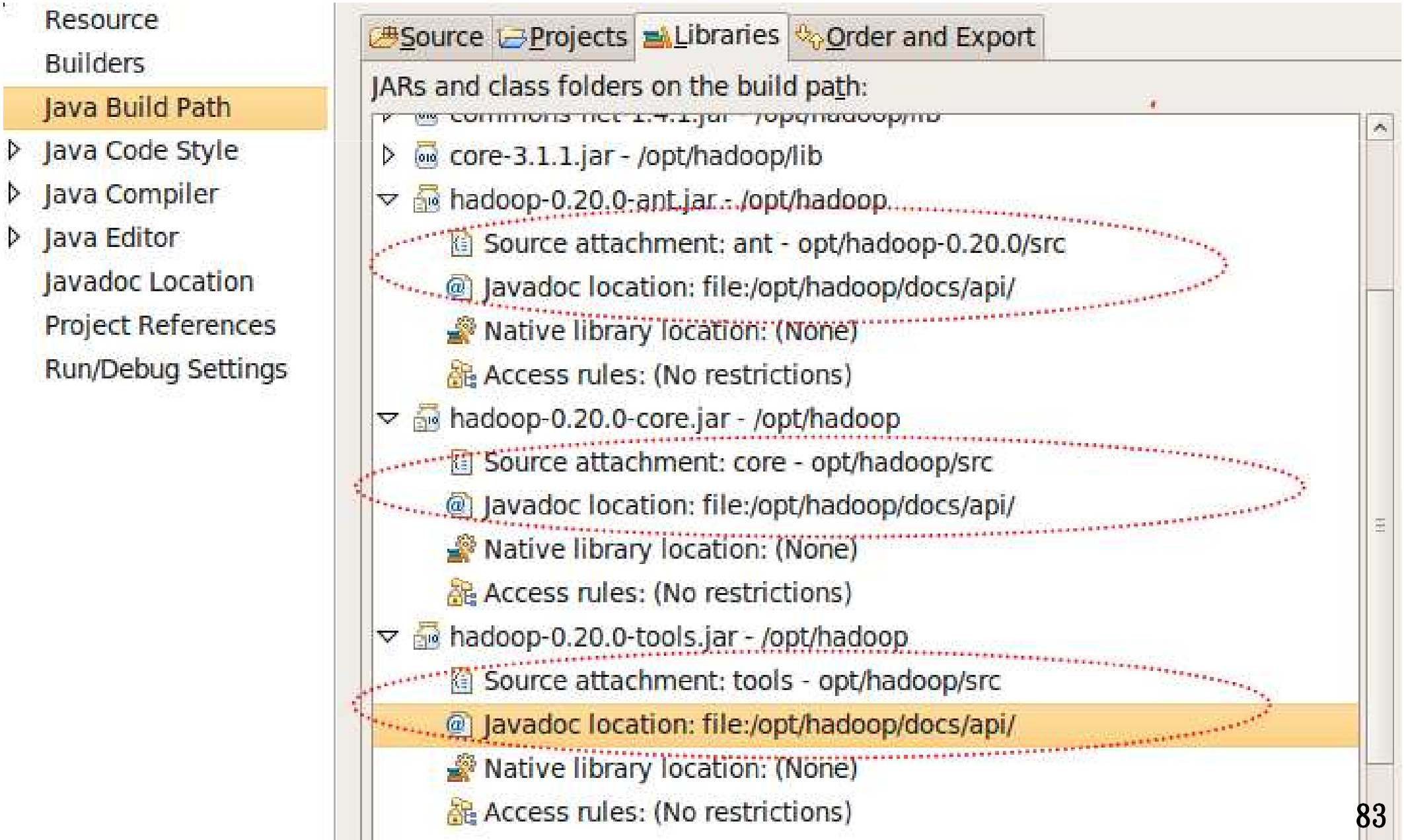
5-1. 設定原始碼與文件路徑

選擇 Java Build Path

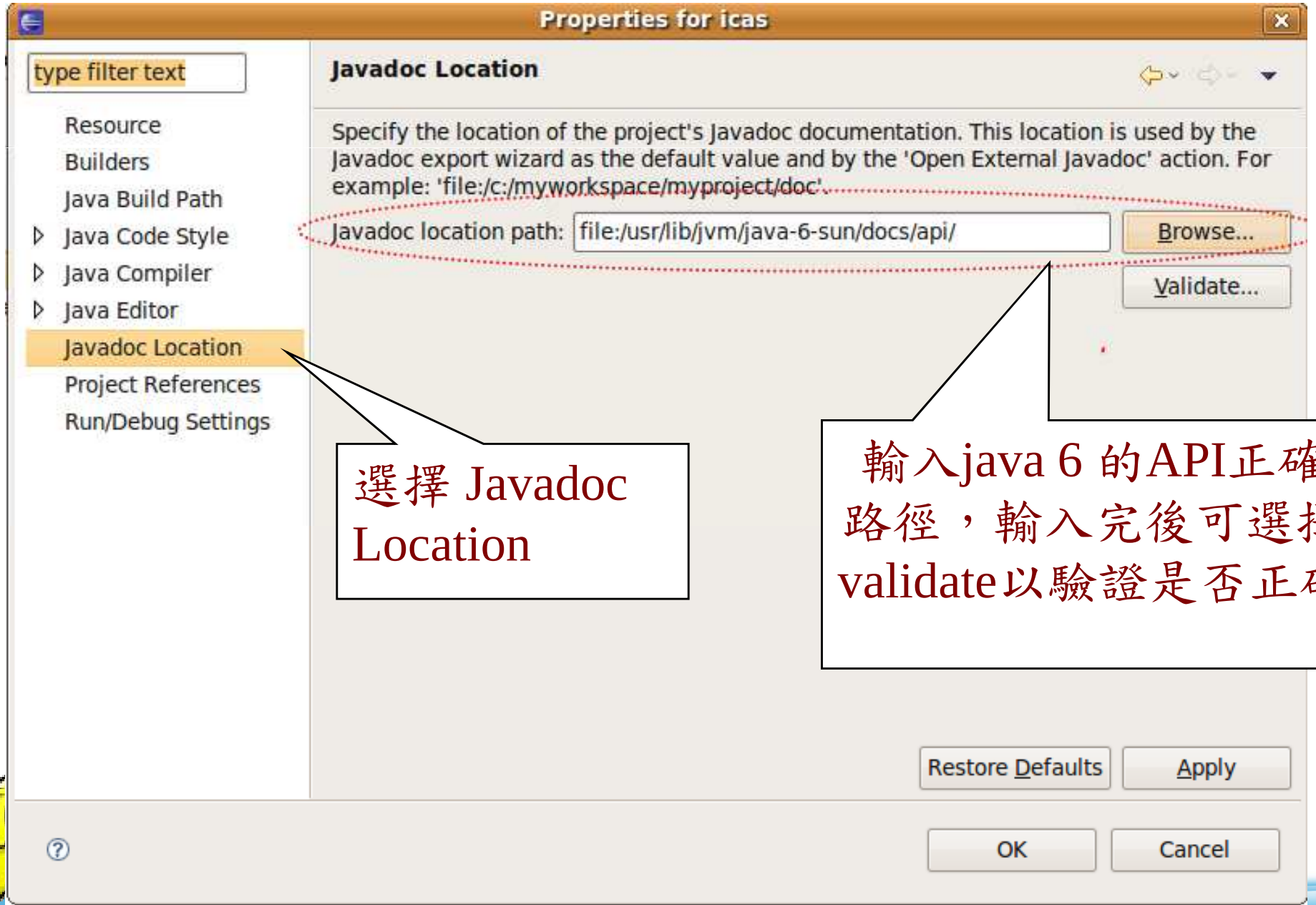
以下請輸入正確的Hadoop原始碼與API文件檔路徑，如
source ：/opt/hadoop/src/
javadoc：file:/opt/hadoop/docs/api/



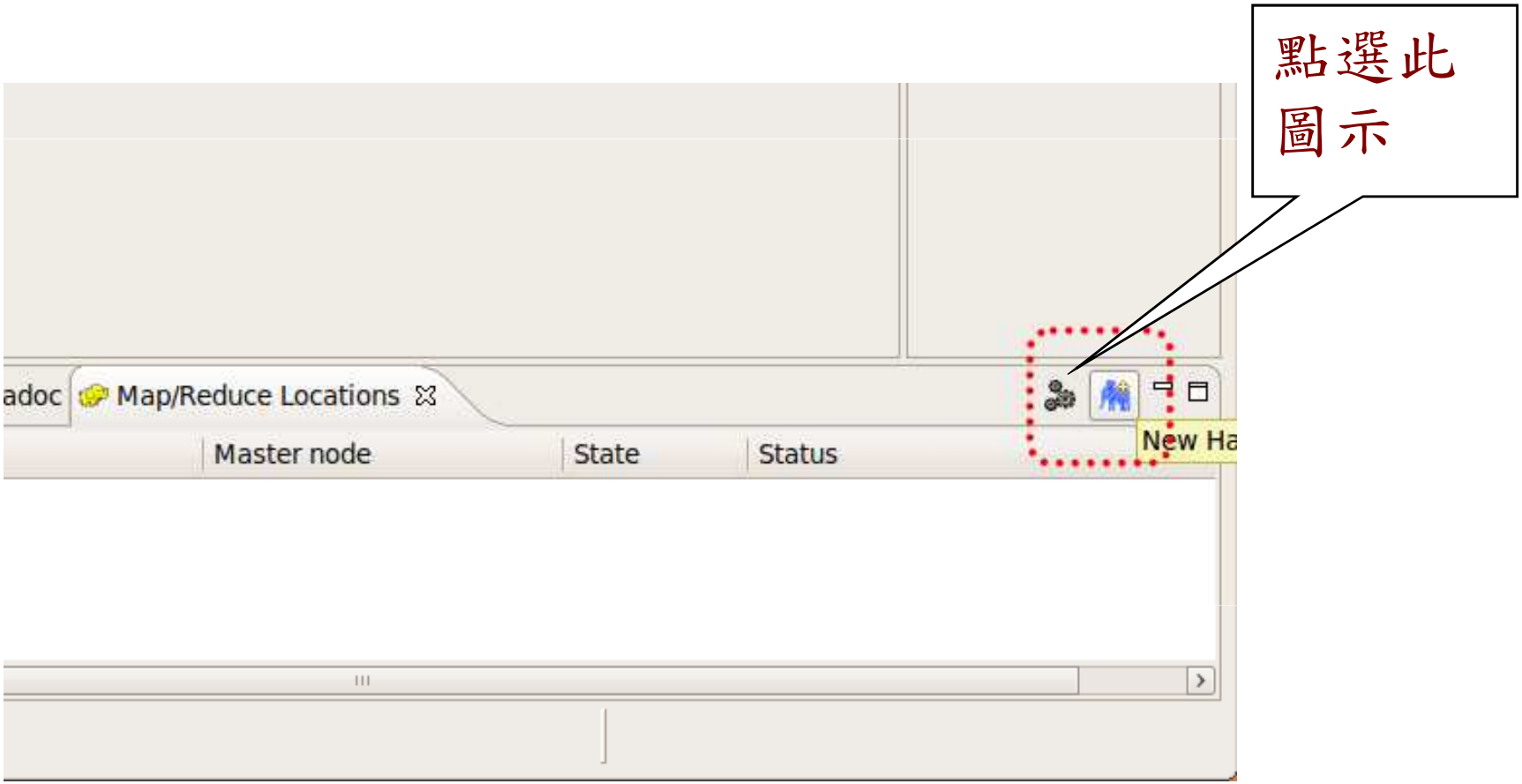
5-1-1. 完成圖

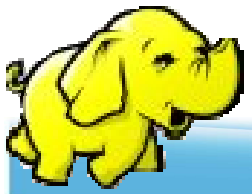

Resource
Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project References
Run/Debug Settings
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
core-3.1.1.jar - /opt/hadoop/lib
hadoop-0.20.0-ant.jar - /opt/hadoop
Source attachment: ant - opt/hadoop-0.20.0/src
Javadoc location: file:/opt/hadoop/docs/api/
Native library location: (None)
Access rules: (No restrictions)
hadoop-0.20.0-core.jar - /opt/hadoop
Source attachment: core - opt/hadoop/src
Javadoc location: file:/opt/hadoop/docs/api/
Native library location: (None)
Access rules: (No restrictions)
hadoop-0.20.0-tools.jar - /opt/hadoop
Source attachment: tools - opt/hadoop/src
Javadoc location: file:/opt/hadoop/docs/api/
Native library location: (None)
Access rules: (No restrictions)

5-2. 設定java doc的完整路徑



6. 連結Hadoop Server與Eclipse



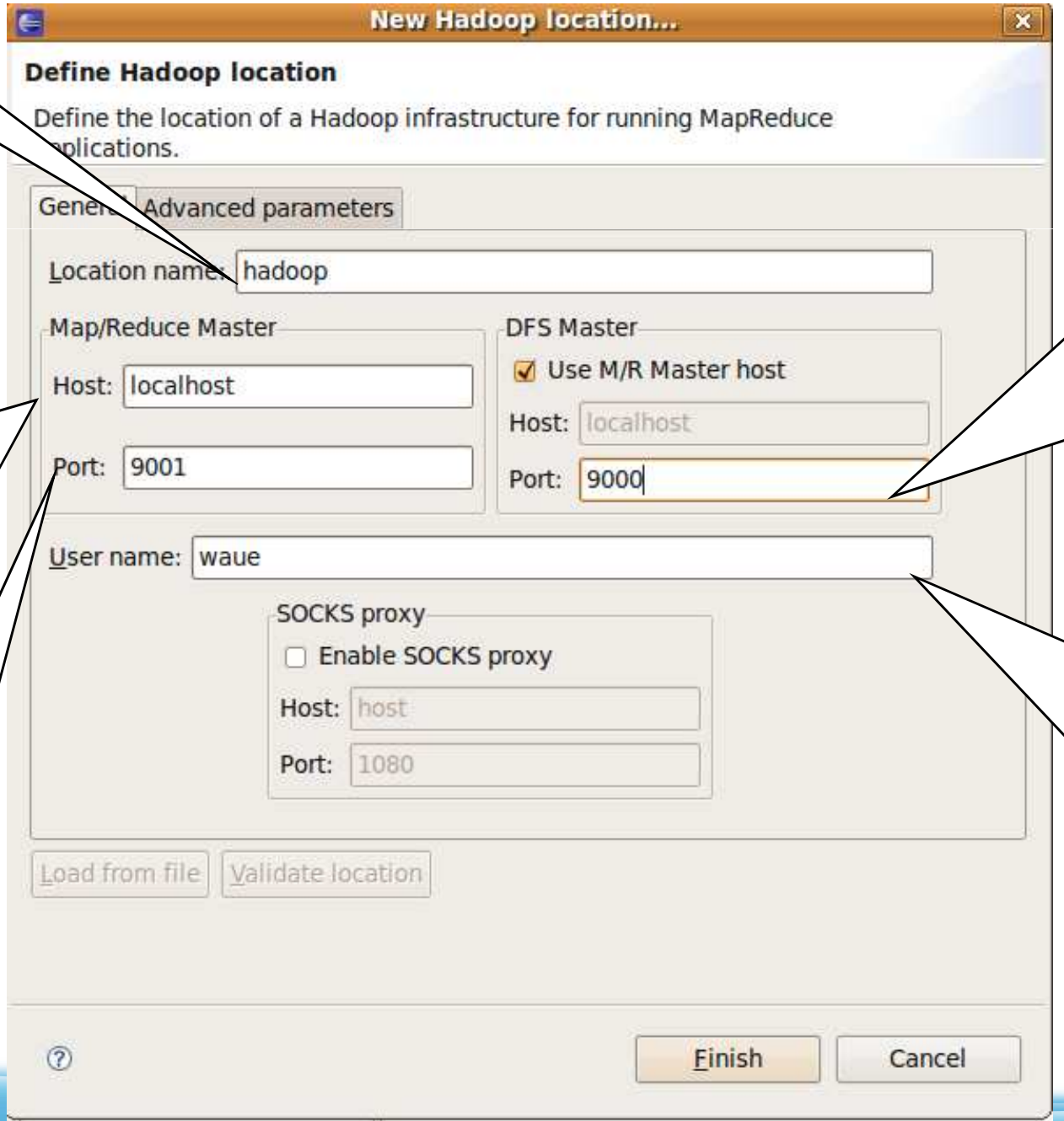


6-1. 設定你要連接的Hadoop主機

任意填一個名稱

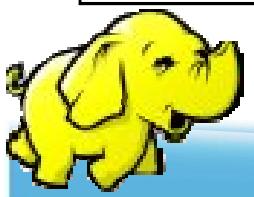
輸入主機位址或domain name

MapReduce 監聽的Port (設定於mapred-site.xml)



HDFS監聽的Port (設定於core-site.xml)

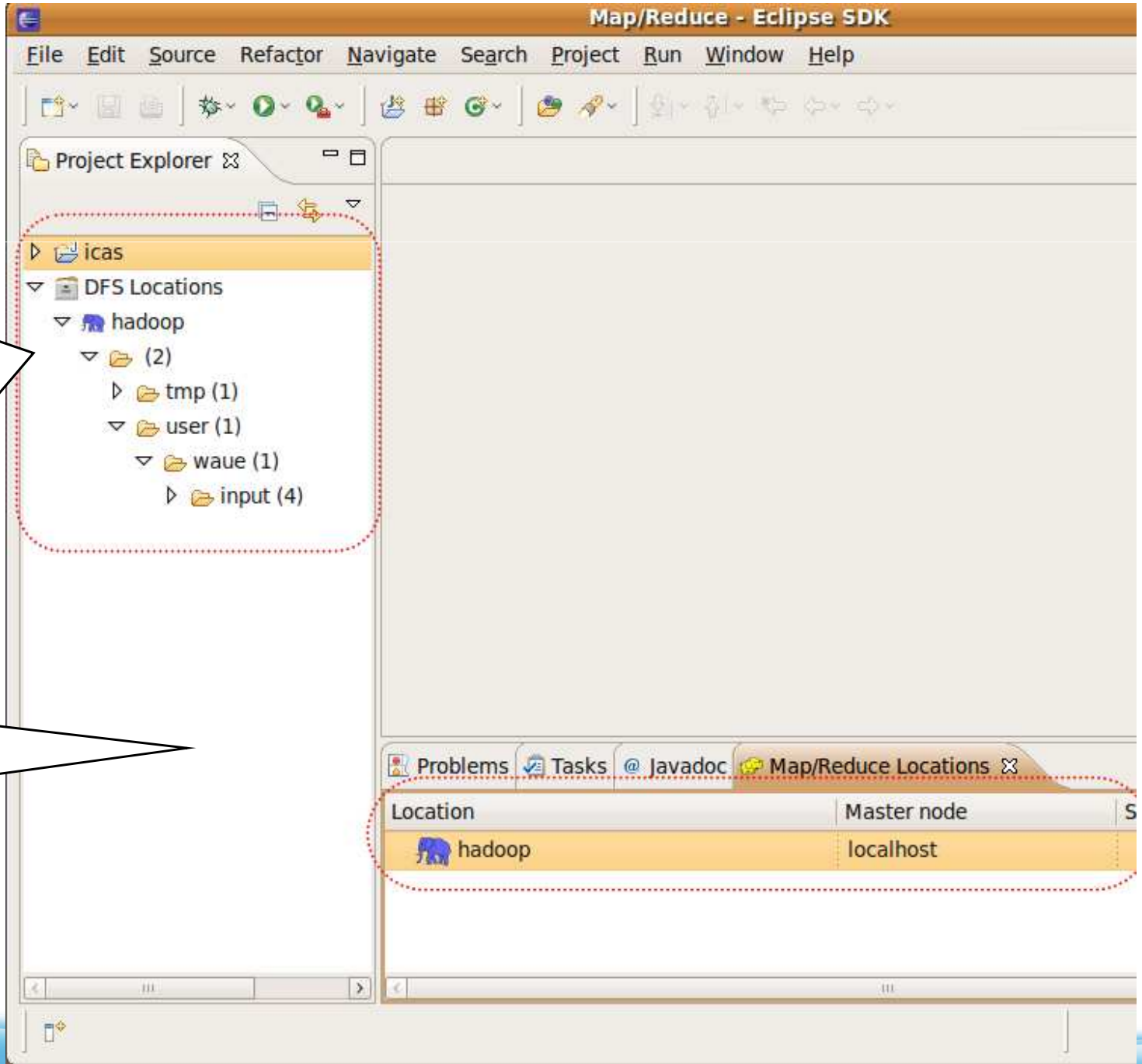
你在此Hadoop Server上的Username



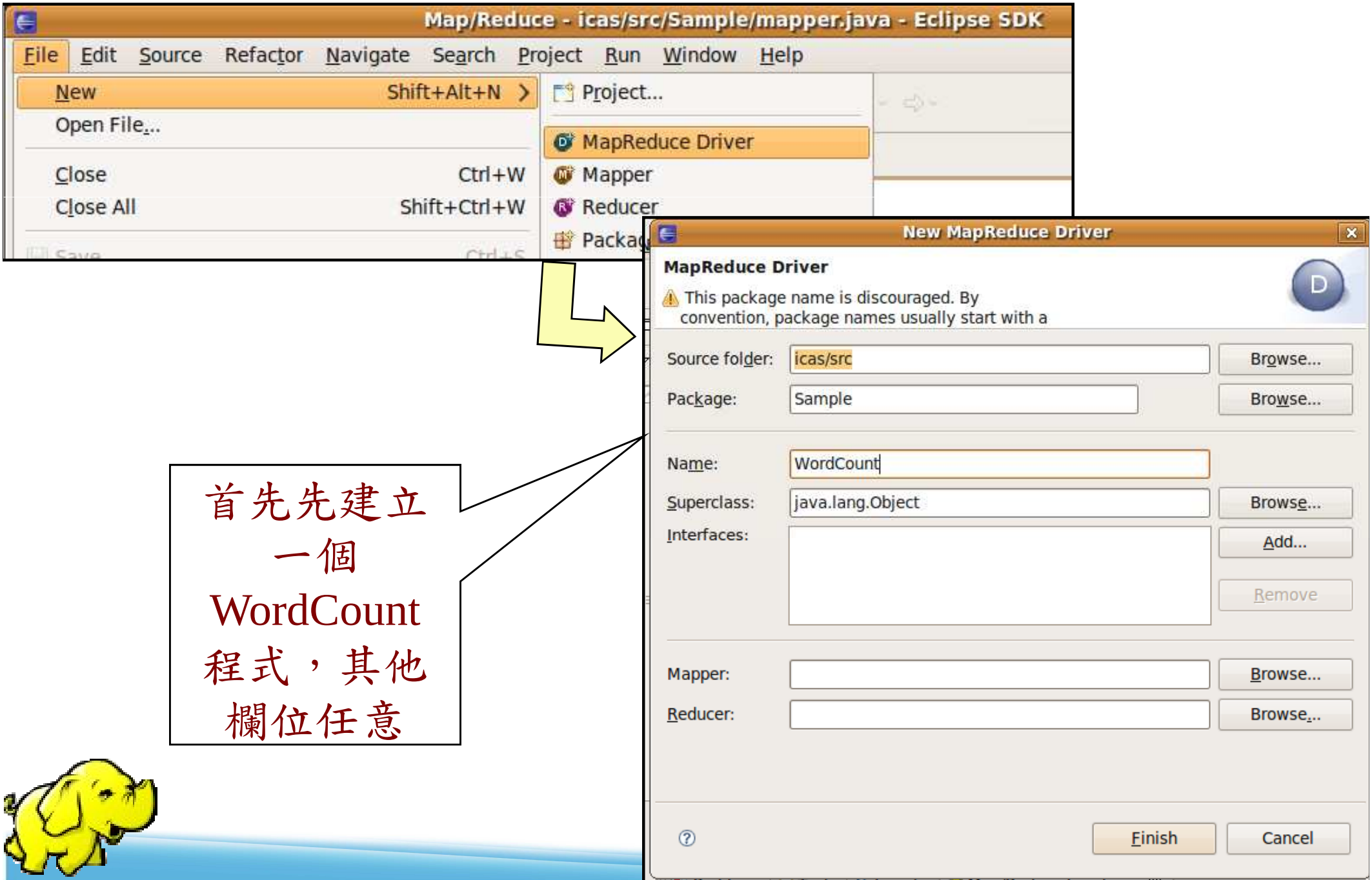
6-2 若正確設定則可得到以下畫面

HDFS的資訊，可直接於此操作檢視、新增、上傳、刪除等命令

若有Job運作，可於此視窗檢視

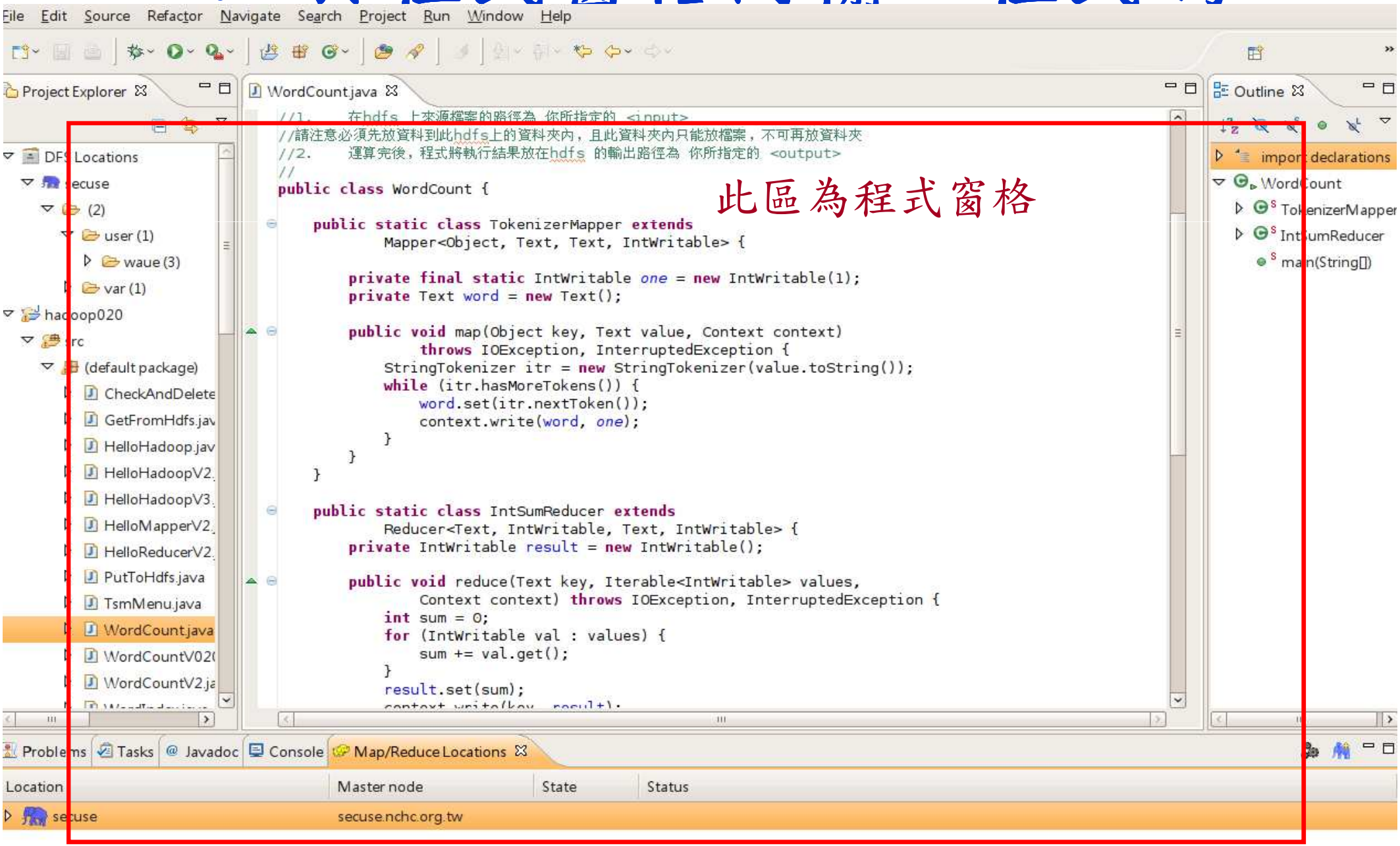


7. 新增一個Hadoop程式



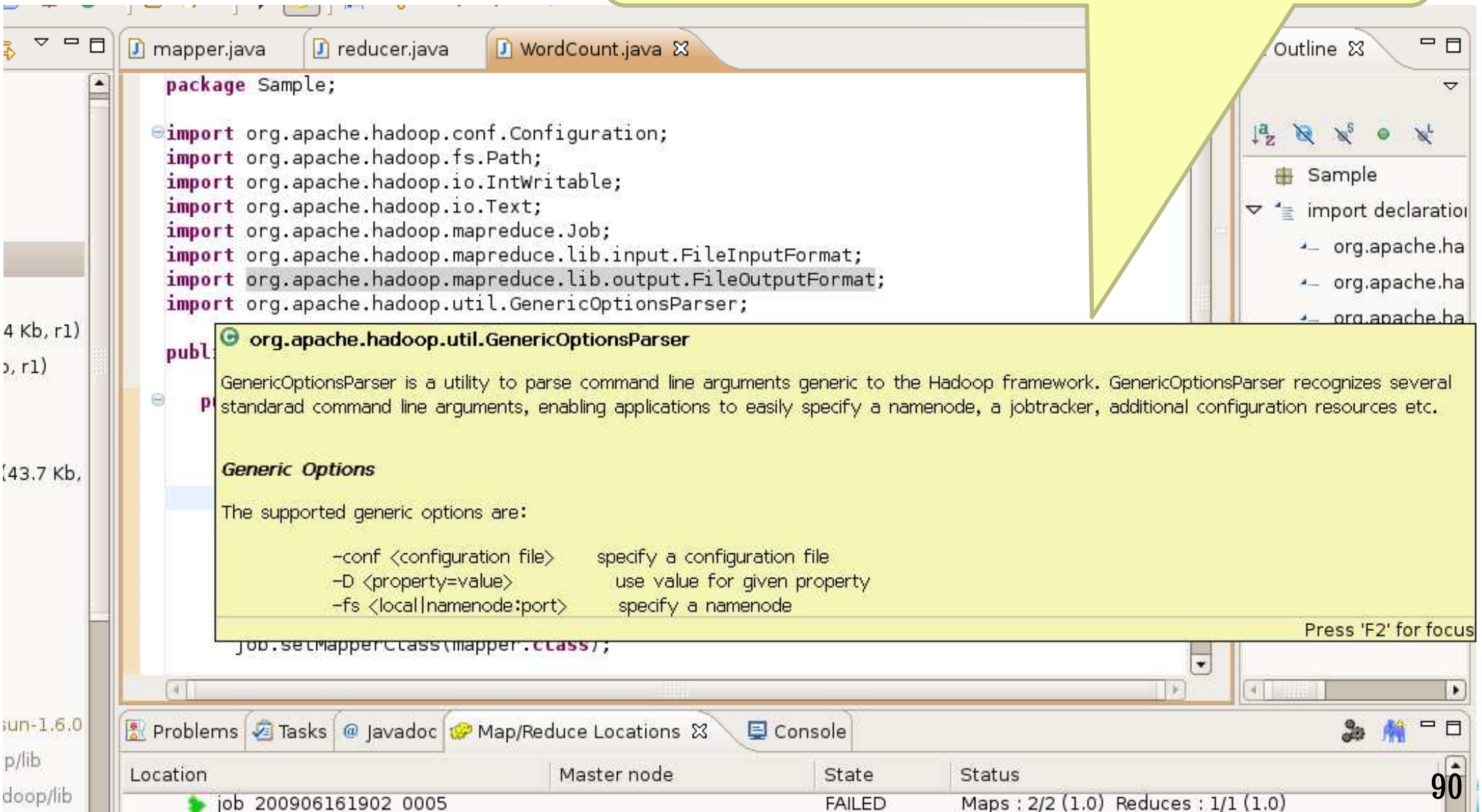
首先先建立一個WordCount程式，其他欄位任意

7.1 於程式窗格內輸入程式碼



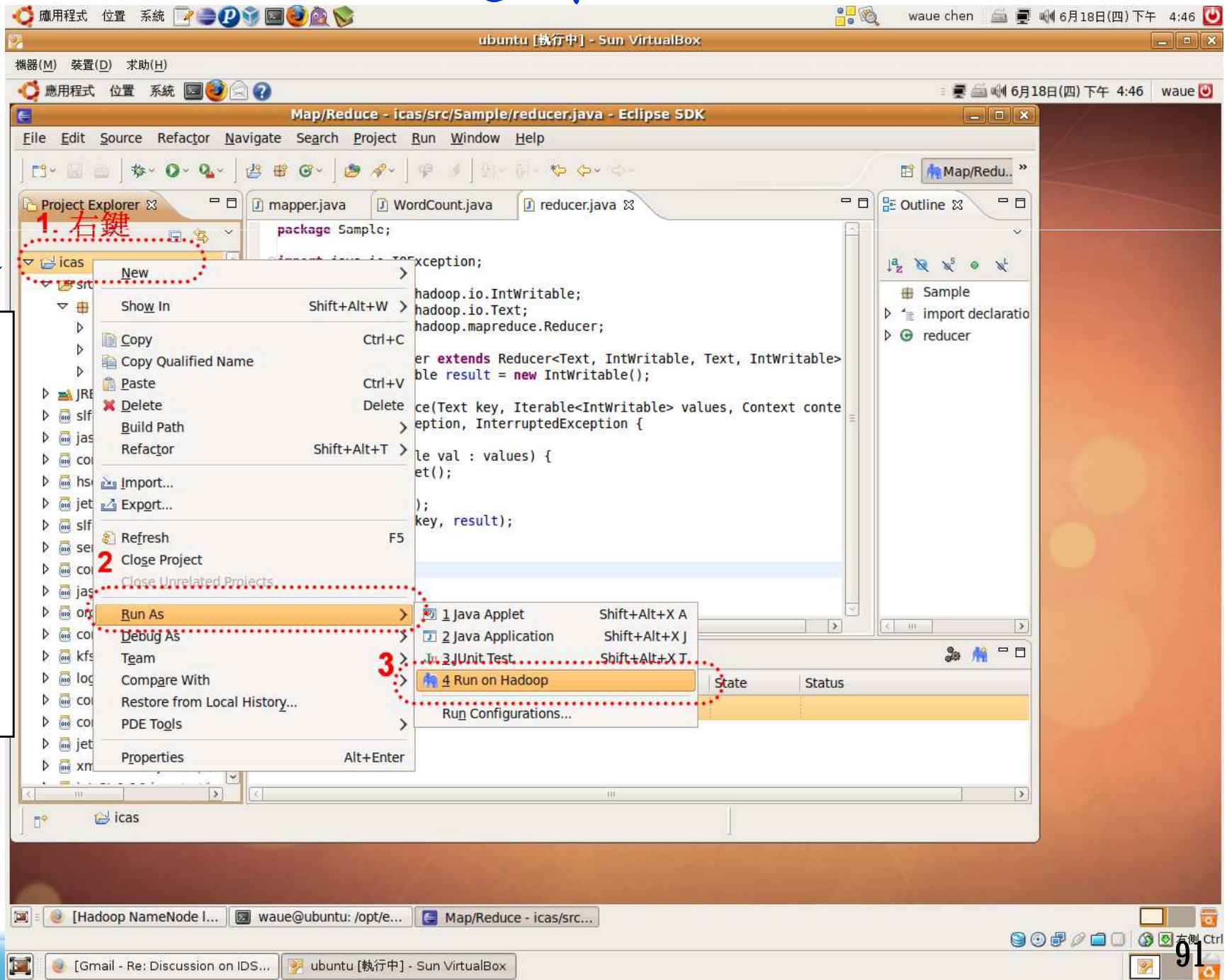
7.2 補充

若之前doc部份設定正確，則滑鼠移至程式碼可取得API完整說明

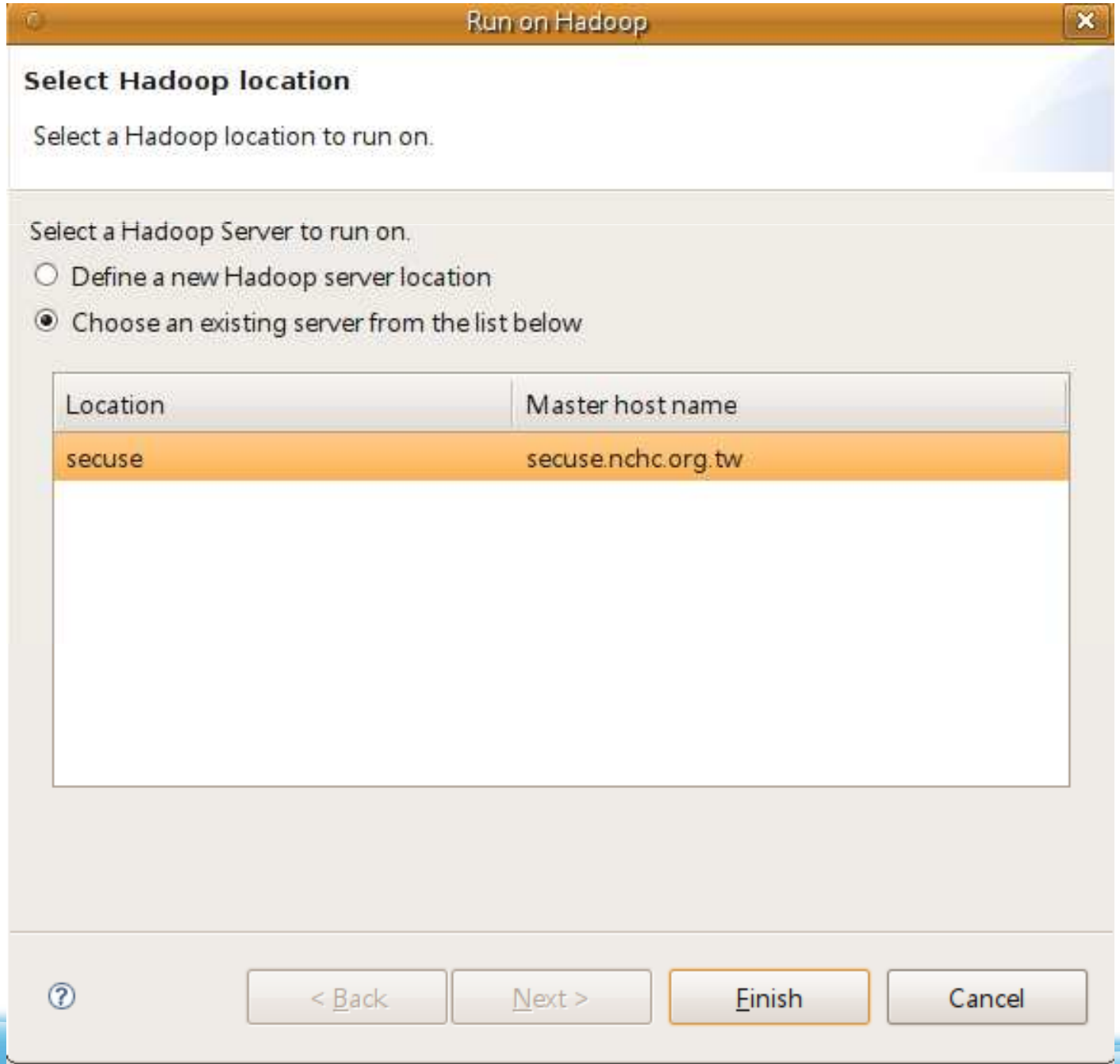


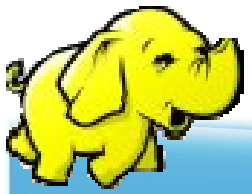
8. 運作

於欲運算的程式碼處點選右鍵 → Run As → **Run on Hadoop**

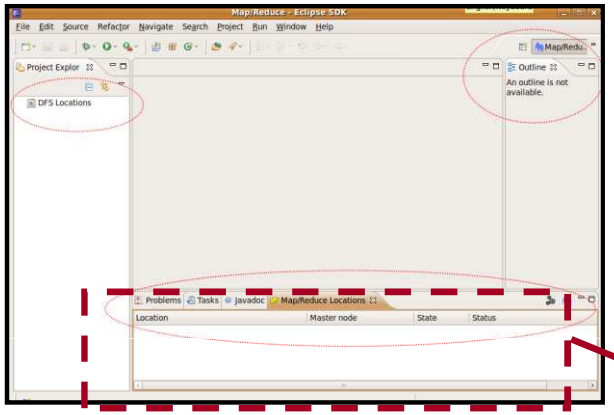


8-1 選擇之前設定好所要運算的主機





8.2 運算資訊出現於Eclipse 右下方的Console 視窗



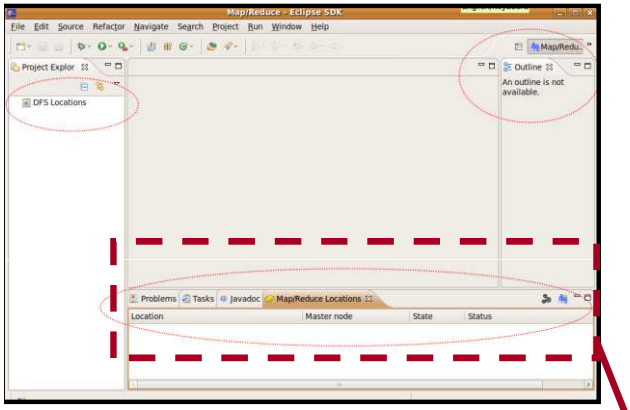
放大

Problems | Tasks | Javadoc | Console | Map/Reduce Locations

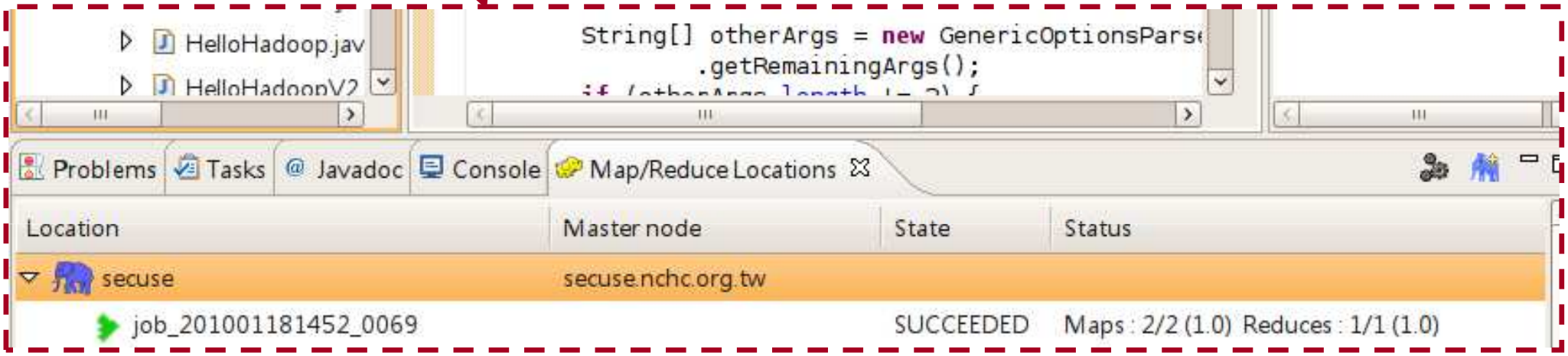
<terminated> WordCount (1) [Java Application] /usr/lib/jvm/java-6-sun-1.6.0.16/bin/java (2010/1/20 下午 6:15:07)

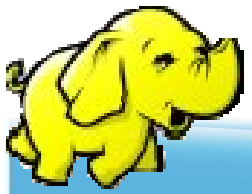
```
10/01/20 18:15:08 WARN conf.Configuration: DEPRECATED: hadoop-site.xml found in the classpath. Usage of hadc
10/01/20 18:15:08 WARN mapred.JobConf: The variable mapred.task.maxvmem is no longer used. Instead use mapre
10/01/20 18:15:08 WARN mapred.JobConf: The variable mapred.task.maxvmem is no longer used. Instead use mapre
10/01/20 18:15:08 INFO input.FileInputFormat: Total input paths to process : 2
10/01/20 18:15:08 INFO mapred.JobClient: Running job: job_201001181452_0078
10/01/20 18:15:09 INFO mapred.JobClient:  map 0% reduce 0%
10/01/20 18:15:16 INFO mapred.JobClient:  map 100% reduce 0%
10/01/20 18:15:28 INFO mapred.JobClient:  map 100% reduce 100%
10/01/20 18:15:30 INFO mapred.JobClient: Job complete: job_201001181452_0078
10/01/20 18:15:30 INFO mapred.JobClient: Counters: 17
10/01/20 18:15:30 INFO mapred.JobClient:   Job Counters
10/01/20 18:15:30 INFO mapred.JobClient:     Launched reduce tasks=1
10/01/20 18:15:30 INFO mapred.JobClient:     Launched map tasks=2
10/01/20 18:15:30 INFO mapred.JobClient:     Data-local map tasks=2
10/01/20 18:15:30 INFO mapred.JobClient:   FileSystemCounters
10/01/20 18:15:30 INFO mapred.JobClient:     FILE_BYTES_READ=153
```

8.3 剛剛運算的結果出現如下圖



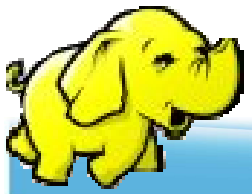
放大





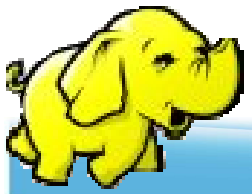
Conclusions

- 優點
  - 快速開發程式
  - 易於除錯
  - 智慧尋找函式庫
  - 自動鍊結API
  - 直接操控 HDFS 與 JobTracker
  - …
- 缺點
  - Plugin 並會因Eclipse 版本而有不同的狀況



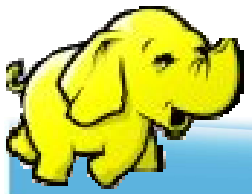
五、開發Hadoop Map/Reduce

程式設計者只需要解決”真實的”問題，
架構面留給MapReduce

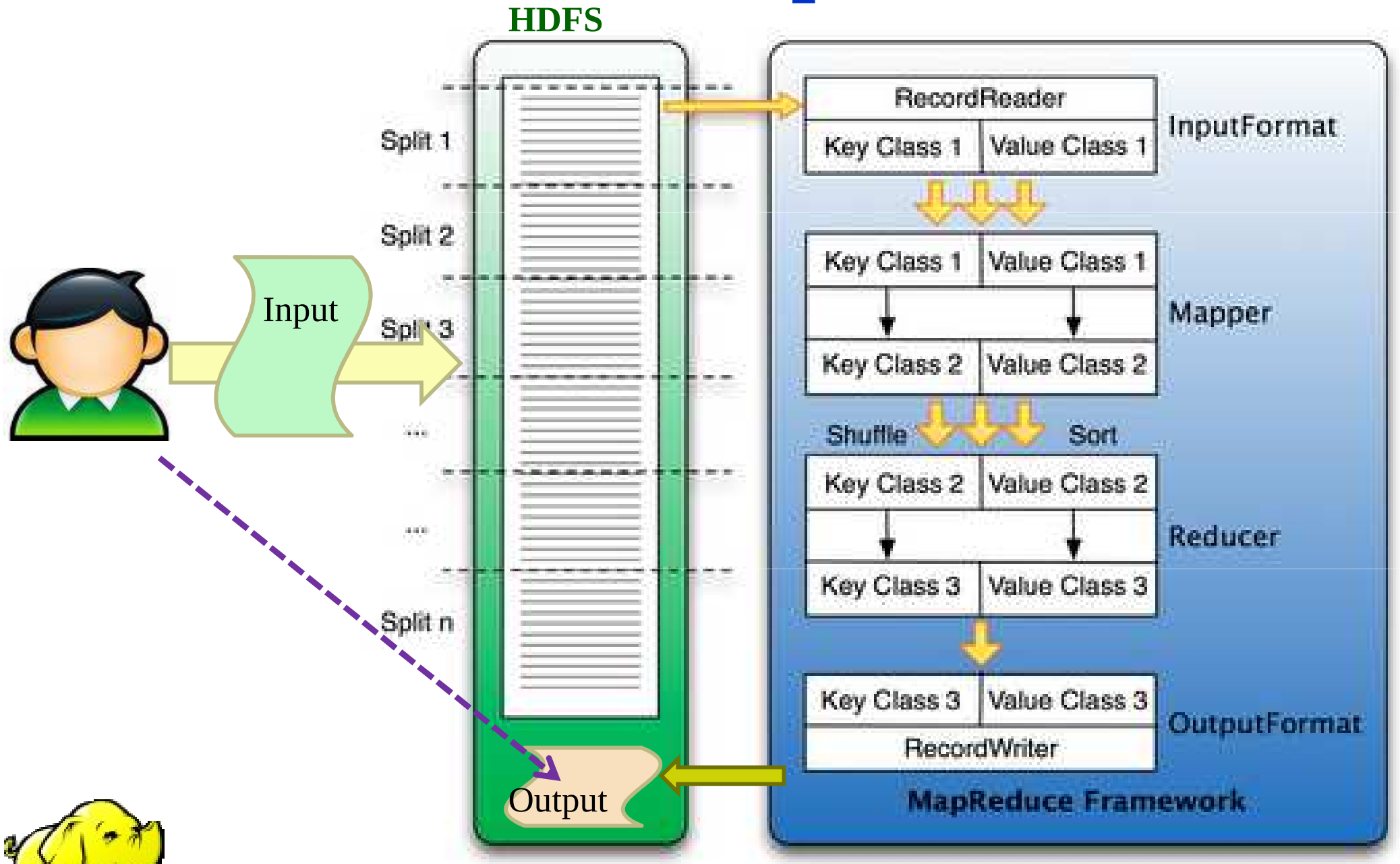


Hadoop 的MapReduce API提供

- 自動的平行化與工作分配
- 容錯特性
- 狀態監控工具
- 一個乾淨的抽象化(abstration)供程式設計師使用

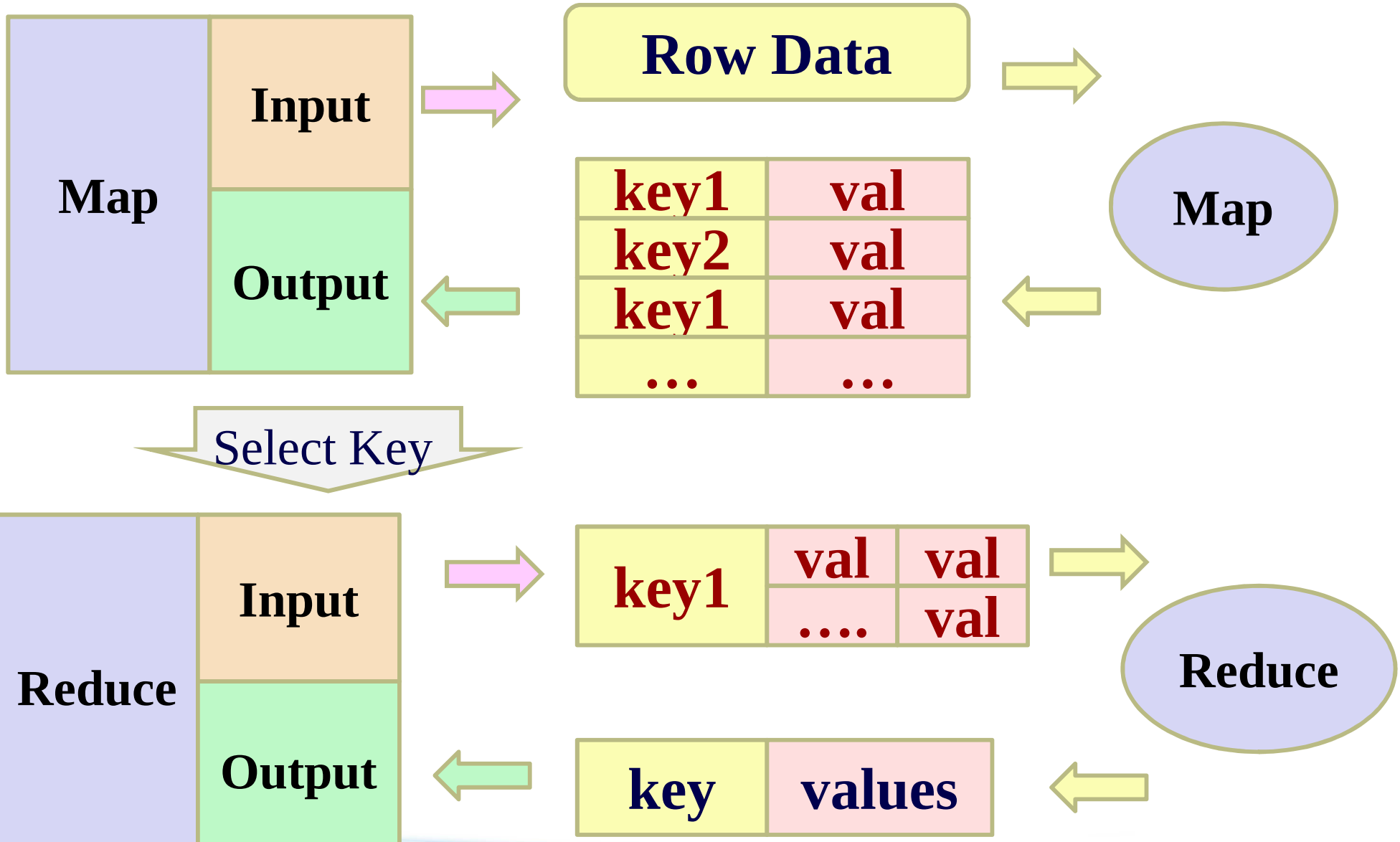


HDFS & MapReduce



部份圖片來源：http://www.larsgeorge.com/2009/05/hbase-mapreduce-101-part-i.html

<Key, Value> Pair



Program Prototype (v 0.20)

Map 區

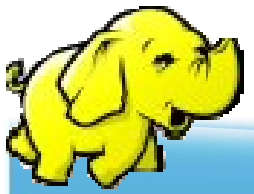
Reduce 區

設定區

```
Class MR{
    static public Class Mapper …{
    }
     static public Class Reducer …{
    }
    main(){
        Configuration conf = new Configuration();
        Job job = new Job(conf, “job name");
        job.setJarByClass(thisMainClass.class);
        job.setMapperClass(Mapper.class);
        job.setReduceClass(Reducer.class);
        FileInputFormat.addInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        其他的設定參數程式碼
        job.waitForCompletion(true);
}}
```

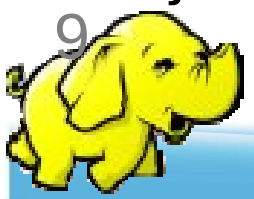
Map 程式碼

Reduce 程式碼



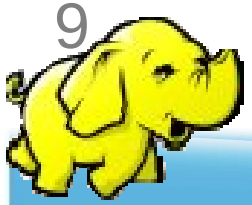
Class Mapper (v 0.20)

```
import org.apache.hadoop.mapreduce.Mapper;
1  class MyMap extends
                    Mapper <INPUT KEY Class , INPUT VALUE Class , OUTPUT KEY Class , OUTPUT VALUE Class >
2  {
3  // 全域變數區
4  public void map ( INPUT KEY Class key, INPUT VALUE Class value,
                Context context )throws IOException,InterruptedException
               {
5              // 區域變數與程式邏輯區
6              context.write( NewKey, NewValue);
7              }
8  }
9
```



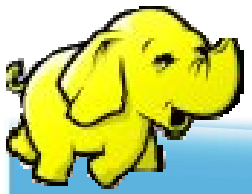
Class Reducer (v 0.20)

```
import org.apache.hadoop.mapreduce.Reducer;
1  class MyRed extends
                Reducer < INPUT KEY Class , INPUT VALUE Class , OUTPUT KEY Class , OUTPUT VALUE Class >
2  {
3  // 全域變數區
4  public void reduce ( INPUT KEY Class key, Iterable< INPUT VALUE Class > values,
          Context context) throws IOException,InterruptedException
          {
5         // 區域變數與程式邏輯區
6         context.write( NewKey, NewValue);
7         }
8  }
9
```



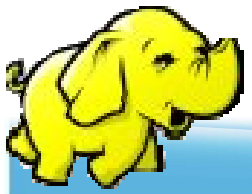
其他常用的設定參數

- 設定 Combiner
  - Job.setCombinerClass (…);
- 設定 output class
  - Job.setMapOutputKeyClass(…);
  - Job.setMapOutputValueClass(…);
  - Job.setOutputKeyClass(…);
  - Job.setOutputValueClass(…);



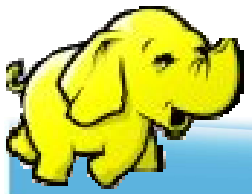
Class Combiner

- 指定一個combiner，它負責對中間過程的輸出進行聚集，這會有助於降低從Mapper 到 Reducer數據傳輸量。
- 可不用設定交由Hadoop預設
- 也可不實做此程式，引用Reducer
- 設定
  - JobConf.setCombinerClass(Class)



範例一 (1) - mapper

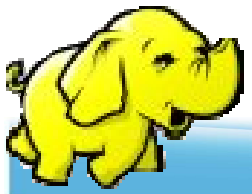
```
public class HelloHadoop {

    static public class HelloMapper extends
                    Mapper<LongWritable, Text, LongWritable, Text> {
            public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
                    context.write((LongWritable) key, (Text) value);
            }
    }  // HelloReducer end

..(待續) …
```



範例一 (2) - reducer

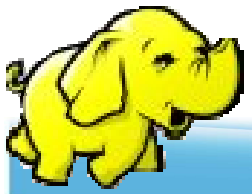
```
static public class HelloReducer extends
        Reducer<LongWritable, Text, LongWritable, Text> {
    public void reduce(LongWritable key, Iterable<Text> values,
    Context context) throws IOException, InterruptedException {
        Text val = new Text();
        for (Text str : values) {
            val.set(str.toString());
        }
        context.write(key, val);
    }
} // HelloReducer end
```

..(待續) …



範例一 (3) - main

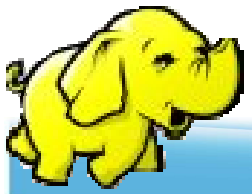
```
    public static void main(String[] args) throws IOException,
                        InterruptedException, ClassNotFoundException {
            Configuration conf = new Configuration();
            Job job = new Job(conf, "Hadoop Hello World");
            job.setJarByClass(HelloHadoop.class);
            FileInputFormat.setInputPaths(job, "input");
            FileOutputFormat.setOutputPath(job, new Path("output-hh1"));
            job.setMapperClass(HelloMapper.class);
            job.setReducerClass(HelloReducer.class);
            job.waitForCompletion(true);
    } // main end
} // wordcount class end
// 完
```



七、Hadoop 程式範例

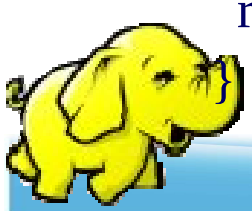
7.1：HDFS 操作篇

7.2：MapReduce運算篇



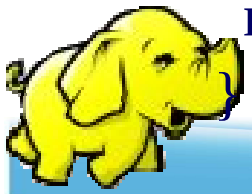
傳送檔案至HDFS

```
// 將檔案從local上傳到 hdfs ， src 為 local 的來源 ， dst 為 hdfs 的目的端

public class PutToHdfs {
  static boolean putToHdfs(String src, String dst, Configuration conf) {
    Path dstPath = new Path(dst);
    try {
      // 產生操作hdfs的物件
      FileSystem hdfs = dstPath.getFileSystem(conf);
      // 上傳
      hdfs.copyFromLocalFile(false, new Path(src),new Path(dst));
    } catch (IOException e) {
      e.printStackTrace();
      return false;
    }
    return true;
  }
```



從HDFS取回檔案

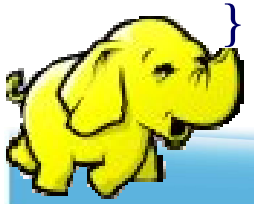
```
// 將檔案從hdfs下載回local, src 為 hdfs的來源,
dst 為 local 的目的端
public class GetFromHdfs {
static boolean getFromHdfs(String src,String dst, Configuration conf) {
    Path dstPath = new Path(src);
    try {
     // 產生操作hdfs的物件
     FileSystem hdfs = dstPath.getFileSystem(conf);
     // 下載
     hdfs.copyToLocalFile(false, new Path(src),new Path(dst));
    } catch (IOException e) {
     e.printStackTrace();
     return false;
    }
    return true;
}
```



檢查與刪除檔案

// checkAndDelete函式，檢查是否存在該資料夾，若有則刪除之

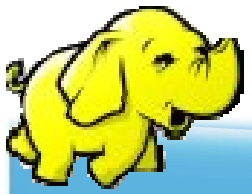
```
public class CheckAndDelete {
  static boolean checkAndDelete(final String path, Configuration conf) {
    Path dst_path = new Path(path);
    try {
      // 產生操作hdfs的物件
      FileSystem hdfs = dst_path.getFileSystem(conf);
      // 檢查是否存在
      if (hdfs.exists(dst_path)) {
        // 有則刪除
        hdfs.delete(dst_path, true);
      } } catch (IOException e) {
      e.printStackTrace();
      return false;
    }     return true;    }
```



七、Hadoop 程式範例

7.1：HDFS 操作篇

7.2：MapReduce運算篇



範例二 (1) HelloHadoopV2

說明：

此程式碼比HelloHadoop 增加了

* 檢查輸出資料夾是否存在並刪除
* input 資料夾內的資料若大於兩個，則資料不會被覆蓋
* map 與 reduce 拆開以利程式再利用

測試方法：

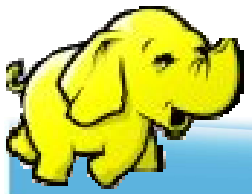
將此程式運作在hadoop 0.20 平台上，執行：

---

hadoop jar V2.jar HelloHadoopV2

注意：

1. 在hdfs 上來源檔案的路徑為 "/user/$YOUR_NAME/input" , 請注意必須先放資料到此hdfs上的資料夾內，且此資料夾內只能放檔案，不可再放資料夾
2. 運算完後，程式將執行結果放在hdfs 的輸出路徑為 "/user/$YOUR_NAME/output-hh2"



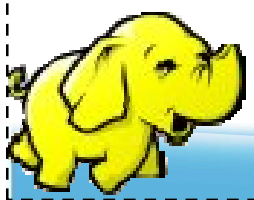
範例二 (2)

```
public class HelloHadoopV2 {
public static void main(String[] args) throws
      IOException,
     InterruptedException, ClassNotFoundException
      {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "Hadoop Hello World
      2");
    job.setJarByClass(HelloHadoopV2.class);
    // 設定 map and reduce 以及 Combiner class
    job.setMapperClass(HelloMapperV2.class);
    job.setCombinerClass(HelloReducerV2.class);
    job.setReducerClass(HelloReducerV2.class);
    // 設定map的輸出型態
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    // 設定reduce的輸出型態
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath
              (job, new Path("input"));
    FileOutputFormat.setOutputPath
              (job, new Path("output-hh2"));
    // 呼叫checkAndDelete函式，
    // 檢查是否存在該資料夾，若有則刪除之
    CheckAndDelete.checkAndDelete("output-hh2",
      conf);
    boolean status = job.waitForCompletion(true);
    if (status) {
      System.err.println("Integrate Alert Job Finished
       !");
    } else {
      System.err.println("Integrate Alert Job Failed
       !");
      System.exit(1);
} } }
```

範例二 (3)

```
public class HelloMapperV2 extends
    Mapper <LongWritable, Text, Text,
    Text> {
  public void map(LongWritable key, Text
    value, Context context)
    throws IOException,
    InterruptedException {
   context.write(new Text(key.toString()),
    value);
} }
```

```
public class HelloReducerV2 extends
    Reducer<Text, Text, Text, Text> {
  public void reduce(Text key, Iterable<Text>
    values, Context context)
    throws IOException, InterruptedException {
   String str = new String("");
   Text final_key = new Text();
   Text final_value = new Text();
   // 將key值相同的values，透過 && 符號分隔
    之
   for (Text tmp : values) {
    str += tmp.toString() + " &&";
   }
   final_key.set(key);
   final_value.set(str);
   context.write(final_key, final_value);
} }
```



範例三 (1) HelloHadoopV3

說明：

此程式碼再利用了 HelloHadoopV2 的 map , reduce 檔，並且自動將檔案上傳到hdfs上運算並自動取回結果，還有提示訊息 、參數輸入 與 印出運算時間 的功能

測試方法：

將此程式運作在hadoop 0.20 平台上，執行：

```
-------------------------
hadoop jar V3.jar HelloHadoopV3 <local_input> <local_output>
-------------------------
```

注意：

1. 第一個輸入的參數是在local 的 輸入資料夾，請確認此資料夾內有資料並無子目錄
2. 第二個輸入的參數是在local 的 運算結果資料夾，由程式產生不用事先建立，若有請刪除之

範例三 (2)

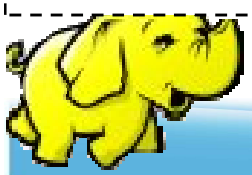
```
public class HelloHadoopV3 {
  public static void main(String[] args) throws
      IOException,
      InterruptedException, ClassNotFoundException
      {
String hdfs_input = "HH3_input";
    String hdfs_output = "HH3_output";
    Configuration conf = new Configuration();
    // 宣告取得參數
    String[] otherArgs = new
      GenericOptionsParser(conf, args)
       .getRemainingArgs();
    // 如果參數數量不為2 則印出提示訊息
    if (otherArgs.length != 2) {
      System.err
          .println("Usage: hadoop jar
       HelloHadoopV3.jar <local_input>
       <local_output>");
      System.exit(2);
    }
```

```
    Job job = new Job(conf, "Hadoop Hello World");
    job.setJarByClass(HelloHadoopV3.class);
    // 再利用上個範例的map 與 reduce
    job.setMapperClass(HelloMapperV2.class);
    job.setCombinerClass(HelloReducerV2.class);
    job.setReducerClass(HelloReducerV2.class);
   // 設定map reduce  的key value輸出型態
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
```

範例三 (2)

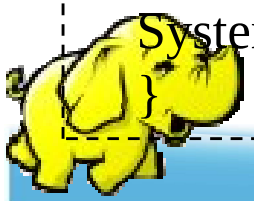
```
// 用 checkAndDelete 函式防止overhead的錯誤
  CheckAndDelete.checkAndDelete(hdfs_input,
    conf);
  CheckAndDelete.checkAndDelete(hdfs_output,
    conf);
// 放檔案到hdfs
  PutToHdfs.putToHdfs(args[0], hdfs_input, conf);
// 設定hdfs 的輸入輸出來源路定
  FileInputFormat.addInputPath(job, new
    Path(hdfs_input));
  FileOutputFormat.setOutputPath(job, new
    Path(hdfs_output));
  long start = System.nanoTime();
  job.waitForCompletion(true);
```

```
// 把hdfs的結果取下
  GetFromHdfs.getFromHdfs(hdfs_output, args[1],
    conf);
  boolean status = job.waitForCompletion(true);
  // 計算時間
  if (status) {
    System.err.println("Integrate Alert Job Finished
    !");
    long time = System.nanoTime() - start;
    System.err.println(time * (1E-9) + " secs.");
  } else {
    System.err.println("Integrate Alert Job Failed !");
    System.exit(1);
} } }
```



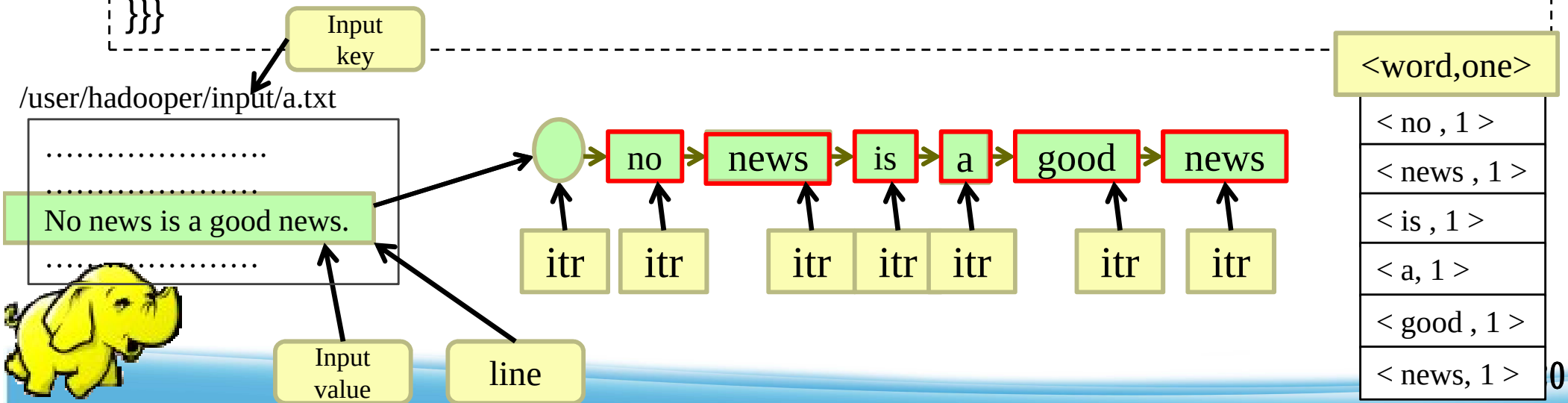
範例四 (1)

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: hadoop jar WordCount.jar <input> <output>");
      System.exit(2);
    }
    Job job = new Job(conf, "Word Count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    CheckAndDelete.checkAndDelete(args[1], conf);
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```



範例四 (2)

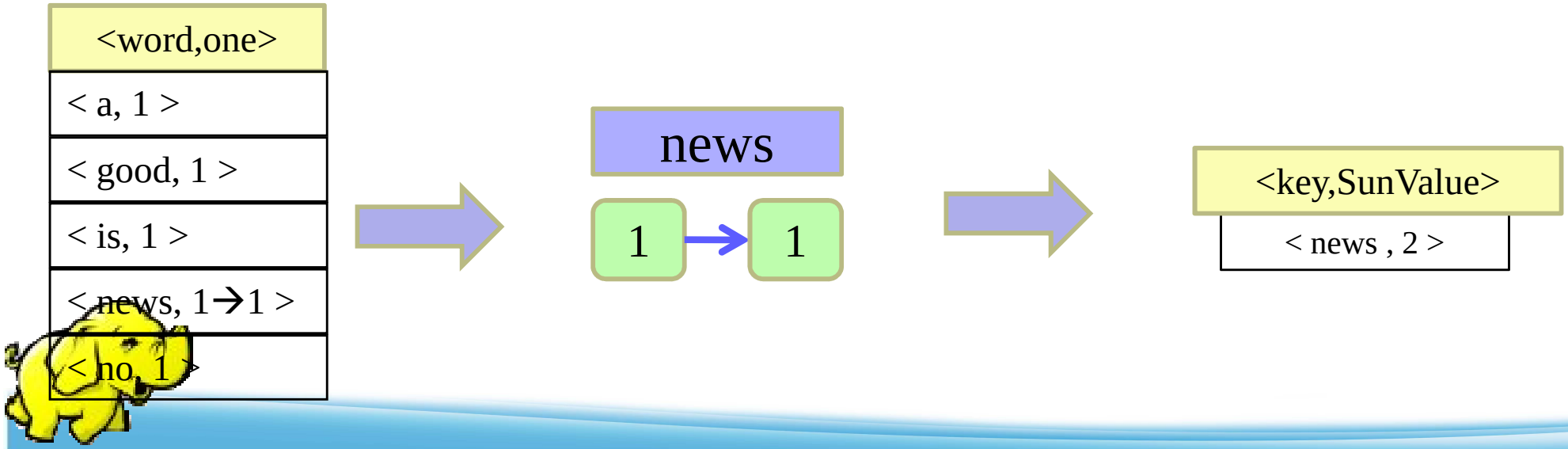
```
1  class TokenizerMapper extends Mapper<LongWritable, Text, Text,
   IntWritable> {
2          private final static IntWritable one = new IntWritable(1);
3          private Text word = new Text();
4          public void map( LongWritable key, Text value, Context context)
           throws IOException , InterruptedException {
5                  String line = ((Text) value).toString();
6                  StringTokenizer itr = new StringTokenizer(line);
7                  while (itr.hasMoreTokens()) {
8                          word.set(itr.nextToken());
9                          context.write(word, one);
   }}}
```



範例四 (3)

```
1  class IntSumReducer extends Reducer< Text, IntWritable, Text, IntWritable> {
2          IntWritable result = new IntWritable();
3          public void reduce( Text key, Iterable <IntWritable> values, Context context)
           throws IOException, InterruptedException {
4              int sum = 0;
5              for ( IntWritable val : values )
6                      sum += val.get();
7              result.set(sum);
8              context.write ( key, result);
   }}
```

```
for ( int i ; i < values.length ; i ++ ){
  sum += values[i].get()
}
```



範例五 (1) WordCountV2

說明：

用於字數統計，並且增加略過大小寫辨識、符號篩除等功能

測試方法：

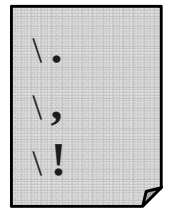
將此程式運作在hadoop 0.20 平台上，執行：

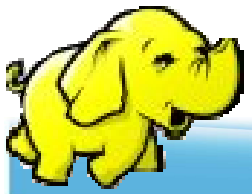
```
-------------------------
hadoop jar WCV2.jar WordCountV2 -Dwordcount.case.sensitive=false \
  <input> <output> -skip patterns/patterns.txt
-------------------------
```

注意：

1. 在hdfs 上來源檔案的路徑為 你所指定的 <input>
   請注意必須先放資料到此hdfs上的資料夾內，且此資料夾內只能放檔案，不可再放資料夾
2. 運算完後，程式將執行結果放在hdfs 的輸出路徑為 你所指定的 <output>
3. 請建立一個資料夾 pattern 並在裡面放置pattern.txt，內容如

（一行一個，前置提示符號\）→





範例五 (2)

```
public class WordCountV2 extends Configured implements Tool {
  public static class Map extends MapReduceBase implements
     Mapper<LongWritable, Text, Text, IntWritable> {
   static enum Counters {       INPUT_WORDS     }
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();
   private boolean caseSensitive = true;
   private Set<String> patternsToSkip = new HashSet<String>();
   private long numRecords = 0;
   private String inputFile;
   public void configure(JobConf job) {
   caseSensitive = job.getBoolean("wordcount.case.sensitive", true);
   inputFile = job.get("map.input.file");
   if (job.getBoolean("wordcount.skip.patterns", false)) {
      Path[] patternsFiles = new Path[0];
      try {
        patternsFiles = DistributedCache.getLocalCacheFiles(job);
      } catch (IOException ioe) {
        System.err
           .println("Caught exception while getting cached files: "
              + StringUtils.stringifyException(ioe));
      }
      for (Path patternsFile : patternsFiles) {
        parseSkipFile(patternsFile);
   }}}
```

```
private void parseSkipFile(Path patternsFile) {
    try {
      BufferedReader fis = new BufferedReader(new FileReader(
         patternsFile.toString()));
      String pattern = null;
      while ((pattern = fis.readLine()) != null) {
        patternsToSkip.add(pattern);       }
    } catch (IOException ioe) {
      System.err.println("Caught exception while parsing the cached
       file '"+ patternsFile + "' : " + tringUtils.stringifyException(ioe));
       }}
   public void map(LongWritable key, Text value,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {
   String line = (caseSensitive) ? value.toString() : value.toString()
       .toLowerCase();
   for (String pattern : patternsToSkip)
      line = line.replaceAll(pattern, "");
   StringTokenizer tokenizer = new StringTokenizer(line);
   while (tokenizer.hasMoreTokens()) {
     word.set(tokenizer.nextToken());
     output.collect(word, one);
     reporter.incrCounter(Counters.INPUT_WORDS, 1); }
```

範例五 (3)

```
if ((++numRecords % 100) == 0) {
      reporter.setStatus("Finished processing " + numRecords
          + " records " + "from the input file: " + inputFile);
} } }
 public static class Reduce extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get(); }
    output.collect(key, new IntWritable(sum));     }  }
  public int run(String[] args) throws Exception {
   JobConf conf = new JobConf(getConf(), WordCount.class);
   conf.setJobName("wordcount");
   String[] otherArgs = new GenericOptionsParser(conf, args)
      .getRemainingArgs();
   if (otherArgs.length < 2) {
     System.out.println("WordCountV2 [-
       Dwordcount.case.sensitive=<false|true>] \\ ");
     System.out.println("           <inDir> <outDir> [-skip
       Pattern_file]");
     return 0;     }
   conf.setOutputKeyClass(Text.class);
   conf.setOutputValueClass(IntWritable.class);
```

```
   conf.setMapperClass(Map.class);
   conf.setCombinerClass(Reduce.class);
   conf.setReducerClass(Reduce.class);
   conf.setInputFormat(TextInputFormat.class);
   conf.setOutputFormat(TextOutputFormat.class);
   List<String> other_args = new ArrayList<String>();
   for (int i = 0; i < args.length; ++i) {
    if ("-skip".equals(args[i])) {
      DistributedCache
          .addCacheFile(new Path(args[++i]).toUri(), conf);
      conf.setBoolean("wordcount.skip.patterns", true);
    } else {other_args.add(args[i]); }    }
   FileInputFormat.setInputPaths(conf, new Path(other_args.get(0)));
   FileOutputFormat.setOutputPath(conf, new
       Path(other_args.get(1)));
   CheckAndDelete.checkAndDelete(other_args.get(1), conf);
   JobClient.runJob(conf);
   return 0;  }
public static void main(String[] args) throws Exception {
int res = ToolRunner.run(new Configuration(), new WordCountV2(),
       args);
  System.exit(res);
}}
```

範例六 (1) WordIndex

說明：

將每個字出於哪個檔案，那一行印出來

測試方法：

將此程式運作在hadoop 0.20 平台上，執行：

---

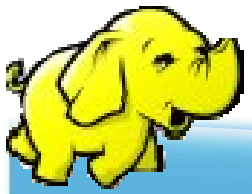
hadoop jar WI.jar WordIndex <input> <output>

注意：

1. 在hdfs 上來源檔案的路徑為 你所指定的 <input>

請注意必須先放資料到此hdfs上的資料夾內，且此資料夾內只能放檔案，不可再放資料夾

2. 運算完後，程式將執行結果放在hdfs 的輸出路徑為 你所指定的 <output>



範例六 (2)

```
public class WordIndex {
  public static class wordindexM extends
    Mapper<LongWritable, Text, Text, Text> {
   public void map(LongWritable key, Text value,
     Context context)
      throws IOException, InterruptedException {

    FileSplit fileSplit = (FileSplit)
     context.getInputSplit();

    Text map_key = new Text();
    Text map_value = new Text();
    String line = value.toString();
    StringTokenizer st = new
     StringTokenizer(line.toLowerCase());
    while (st.hasMoreTokens()) {
      String word = st.nextToken();
      map_key.set(word);
      map_value.set(fileSplit.getPath().getName() +
    ":" + line);
      context.write(map_key, map_value);
    }   }   }
```

```
static public class wordindexR extends
    Reducer<Text, Text, Text, Text> {
 public void reduce(Text key, Iterable<Text>
   values,
    OutputCollector<Text, Text> output, Reporter
   reporter)
    throws IOException {
  String v = "";
  StringBuilder ret = new StringBuilder("\n");
  for (Text val : values) {
    v += val.toString().trim();
    if (v.length() > 0)
      ret.append(v + "\n");
  }
  output.collect((Text) key, new
   Text(ret.toString()));
 }  }
```

範例六 (2)

```
public static void main(String[] args) throws
    IOException,
    InterruptedException,
    ClassNotFoundException {
  Configuration conf = new Configuration();
  String[] otherArgs = new
    GenericOptionsParser(conf, args)
      .getRemainingArgs();
  if (otherArgs.length < 2) {
    System.out.println("hadoop jar
    WordIndex.jar <inDir> <outDir>");
    return;
  }
  Job job = new Job(conf, "word index");
  job.setJobName("word inverted index");
  job.setJarByClass(WordIndex.class);
  job.setMapOutputKeyClass(Text.class);
  job.setMapOutputValueClass(Text.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(Text.class);
  job.setMapperClass(wordindexM.class);
  job.setReducerClass(wordindexR.class);
  job.setCombinerClass(wordindexR.class);
  FileInputFormat.setInputPaths(job, args[0]);
  CheckAndDelete.checkAndDelete(args[1],
    conf);
  FileOutputFormat.setOutputPath(job, new
    Path(args[1]));
  long start = System.nanoTime();
  job.waitForCompletion(true);
  long time = System.nanoTime() - start;
  System.err.println(time * (1E-9) + " secs.");
}}
```

範例七 (1) YourMenu

說明：

將之前的功能整合起來

測試方法：

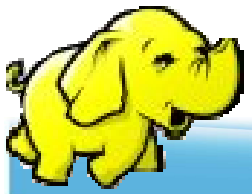
將此程式運作在hadoop 0.20 平台上，執行：

---

hadoop jar YourMenu.jar <功能>

注意：

1. 此程式需與之前的所有範例一起打包成一個jar檔



範例七 (2)

```
public class YourMenu {
  public static void main(String argv[]) {
    int exitCode = -1;
    ProgramDriver pgd = new ProgramDriver();
    if (argv.length < 1) {

      System.out.print("**********************************
      ********\n"
        + "歡迎使用 NCHC 的運算功能 \n" + "指令： \n"
        + "  Hadoop jar NCHC-example-*.jar <功能> \n" + "功能：
      \n"
        + "  HelloHadoop:  秀出Hadoop的<Key,Value>為何 \n"
        + "  HelloHadoopV2:  秀出Hadoop的<Key,Value> 進階版
      \n"
        + "  HelloHadoopV3:  秀出Hadoop的<Key,Value> 進化版
      \n"
        + "  WordCount:  計算輸入資料夾內分別在每個檔案的
      字數統計 \n"
        + "  WordCountV2:  WordCount 進階版 \n"
        + "  WordIndex:  索引每個字與其所有出現的所在列 \n"
        + "**********************************************\n");
    } else {
```

```
try {
      pgd.addClass("HelloHadoop", HelloHadoop.class, "
      Hadoop hello world");
      pgd.addClass("HelloHadoopV2",
      HelloHadoopV2.class, " Hadoop hello world V2");
      pgd.addClass("HelloHadoopV3",
      HelloHadoopV3.class, " Hadoop hello world V3");
      pgd.addClass("WordCount", WordCount.class, "
      word count.");
      pgd.addClass("WordCountV2",
      WordCountV2.class, " word count V2.");
      pgd.addClass("WordIndex", WordIndex.class,
      "invert each word in line");
      pgd.driver(argv);
      // Success
      exitCode = 0;
      System.exit(exitCode);
    } catch (Throwable e) {
      e.printStackTrace();
}}}}
```

補充

ogram Prototype (v 0.18)

```
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 區

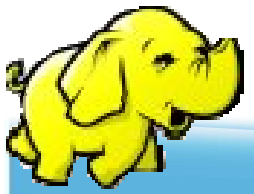
Reduce 區

設定區

Map 程式碼

Reduce 程式碼

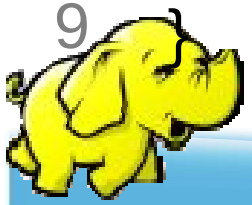
其他的設定參數程式碼



補充

Class Mapper (v0.18)

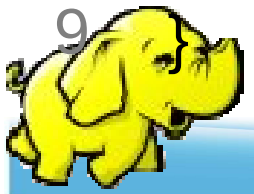
```
import org.apache.hadoop.mapred.*;
1  class MyMap extends MapReduceBase
   implements Mapper < INPUT KEY , INPUT VALUE , OUTPUT KEY , OUTPUT VALUE >
2  {
3  // 全域變數區
4  public void map ( INPUT KEY key, INPUT VALUE value,
           OutputCollector< OUTPUT KEY , OUTPUT VALUE > output,
           Reporter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



補充

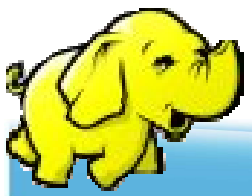
Class Reducer (v0.18)

```
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  }
```



Conclusions

- 以上範例程式碼包含
  - Hadoop 的key,value 架構
  - 操作Hdfs 檔案系統
  - Map Reduce運算方式
- 執行hadoop 運算時，程式檔不用上傳至hadoop上，但資料需要再HDFS內
- 可運用範例七的程式達成連續運算
- Hadoop 0.20 與 Hadoop 0.18 有些API有些許差異，因此在網路上找到Hadoop的程式如果compiler有錯，可以換換對應的Function試試

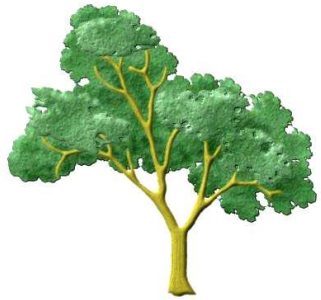


進階課程

Hadoop專案分享

透過 一些使用Hadoop的案例帶給
大家一些新的啟發

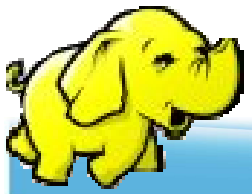
財團法人國家實驗研究院
國家高速網路與計算中心
NATIONAL CENTER FOR HIGH-PERFORMANCE COMPUTING



A：用Hadoop 打造 Location Plus

2009/11/17

- 「Location Plus！」服務，擊敗了其他38組作品獲得Yahoo開發競賽的優勝
- 從大量的批踢踢BBS文章中，找出臺灣17個城市的熱門話題
- 每個城市提供30個熱門話題和相關的參考詞
- 可以讓使用者用這些熱門話題來搜尋 Yahoo知識+、生活+、無名小站等內容
- 提供了手機版介面，讓使用者到任何地方就知道當地有哪些熱門話題

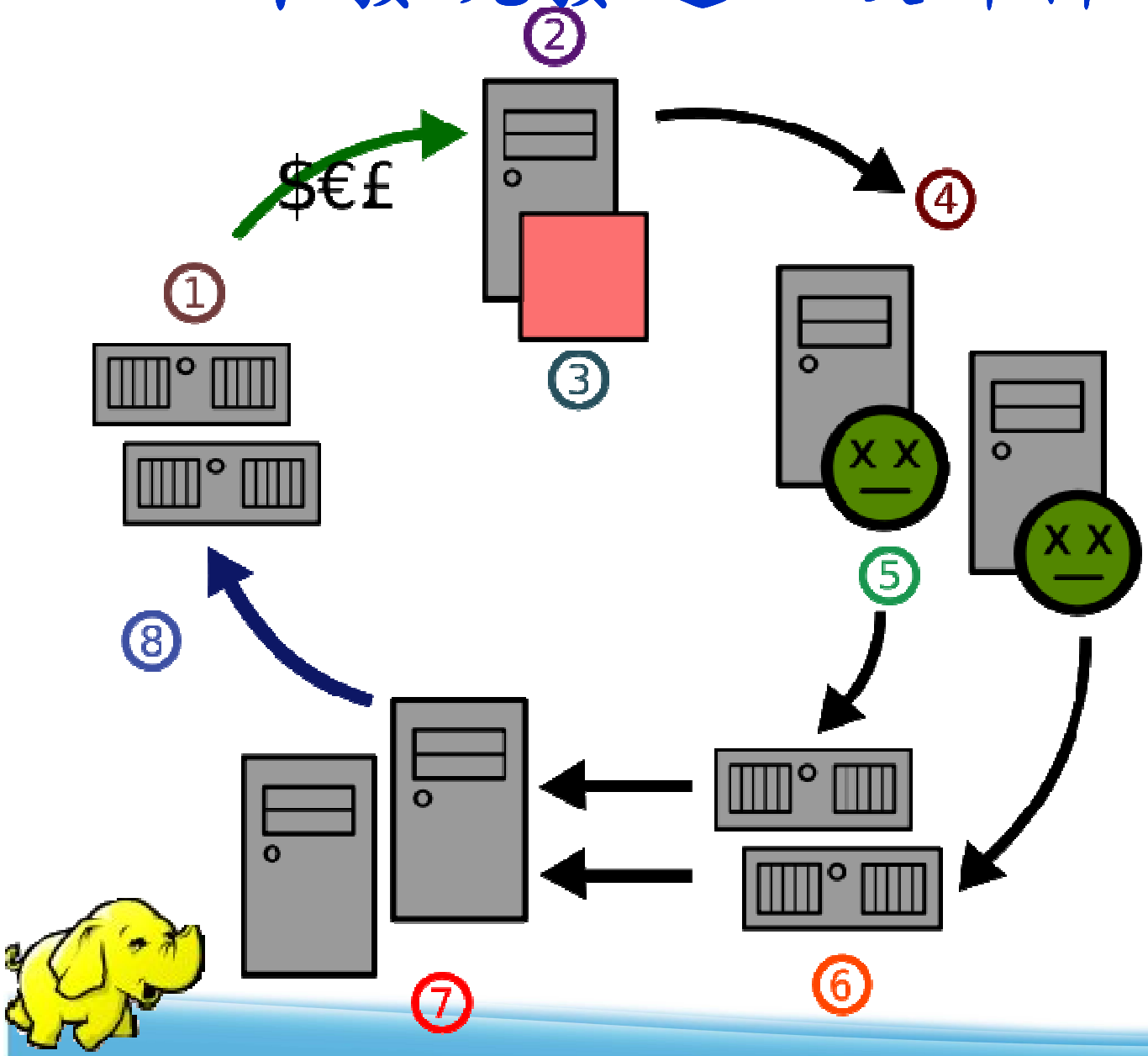


A：Location Plus

- 用RSS蒐集批踢踢BBS站的十大熱門看板的文章，約20萬筆文章記錄
- 維基百科提供的30萬筆詞條作為關鍵詞
- 將所有關鍵詞套用到所有文章紀錄，共需 600 億次比對（還不包含排序..）
- 這些都交給Hadoop吧！

http://www.ithome.com.tw/itadm/article.php?c=58008

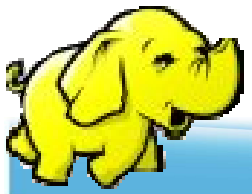
B：Yahoo 使用Hadoop平台
來發現發送垃圾郵件的殭屍網絡



（1） 垃圾郵件網站
（2） 垃圾郵件
（3） 垃圾郵件軟體
（4） 被感染的電腦
（5） 病毒或木馬
（6） 信件伺服器
（7） 用戶
（8） 網路流量

C：警訊整合系統

- 目的：
  - 將原本複雜難懂的警訊日誌整合成易於明瞭的報告
  - 透過“雲端”來運算大量資料
- 環境：
  - hadoop 0.20
  - Java 1.6
  - Apache 2



輸入資料

[**] [1:538:15] NETBIOS SMB IPC$ unicode share access [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]
09/04-17:53:56.363811 168.150.177.165:1051 ->
168.150.177.166:139
TCP TTL:128 TOS:0x0 ID:4000 IpLen:20 DgmLen:138 DF
AP Seq: 0x2E589B8 Ack: 0x642D47F9 Win: 0x4241
TcpLen: 20

[**] [1:1917:6] SCAN UPnP service discover attempt [**]
[Classification: Detection of a Network Scan] [Priority: 3]
09/04-17:53:56.385573 168.150.177.164:1032 ->
239.255.255.250:1900
UDP TTL:1 TOS:0x0 ID:80 IpLen:20 DgmLen:161
Len: 133

[**] [1:1917:6] SCAN UPnP service discover attempt [**]
[Classification: Detection of a Network Scan] [Priority: 3]
09/04-17:53:56.386910 168.150.177.164:1032 ->
239.255.255.250:1900
UDP TTL:1 TOS:0x0 ID:82 IpLen:20 DgmLen:161
Len: 133

[**] [1:1917:6] SCAN UPnP service discover attempt [**]
[Classification: Detection of a Network Scan] [Priority: 3]
09/04-17:53:56.388244 168.150.177.164:1032 ->
239.255.255.250:1900
UDP TTL:1 TOS:0x0 ID:84 IpLen:20 DgmLen:161
Len: 133

[**] [1:1917:6] SCAN UPnP service discover attempt [**]
[Classification: Detection of a Network Scan] [Priority: 3]
09/04-17:53:56.417045 168.150.177.164:45461 -> 168.150.177.1:1900
UDP TTL:1 TOS:0x0 ID:105 IpLen:20 DgmLen:161
Len: 133

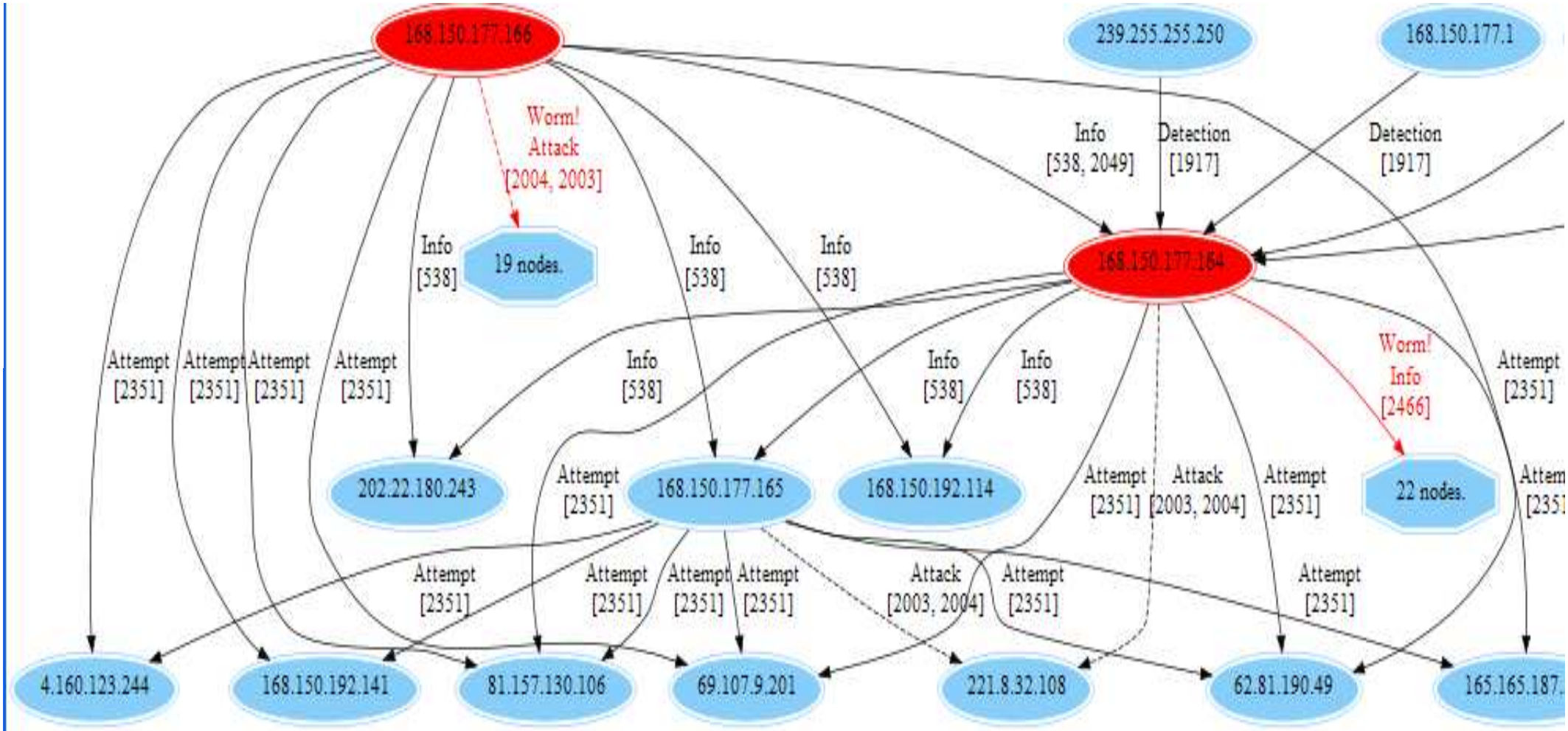
[**] [1:1917:6] SCAN UPnP service discover attempt [**]
[Classification: Detection of a Network Scan] [Priority: 3]
09/04-17:53:56.420759 168.150.177.164:45461 -> 168.150.177.1:1900
UDP TTL:1 TOS:0x0 ID:117 IpLen:20 DgmLen:160
Len: 132

[**] [1:1917:6] SCAN UPnP service discover attempt [**]
[Classification: Detection of a Network Scan] [Priority: 3]
09/04-17:53:56.422095 168.150.177.164:45461 -> 168.150.177.1:1900
UDP TTL:1 TOS:0x0 ID:118 IpLen:20 DgmLen:161
Len: 133

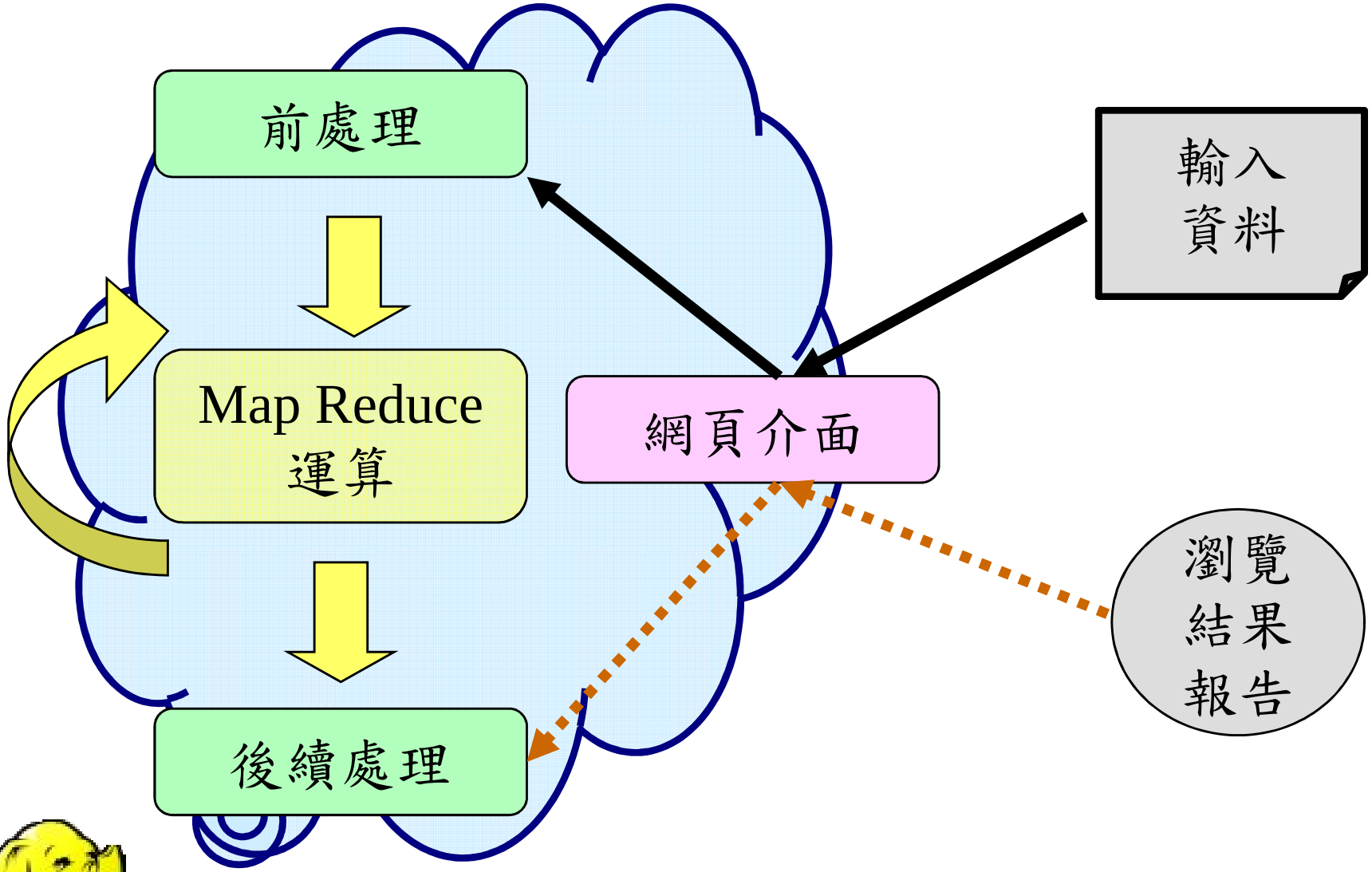
[**] [1:2351:10] NETBIOS DCERPC ISystemActivator path overflow
attempt little endian unicode [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
09/04-17:53:56.442445 198.8.16.1:10179 -> 168.150.177.164:135
TCP TTL:105 TOS:0x0 ID:49809 IpLen:20 DgmLen:1420 DF
A* Seq: 0xF9589BBF Ack: 0x82CCF5B7 Win: 0xFFFF
TcpLen: 20
[Xref => http://www.microsoft.com/technet/security/bulletin/MS03-
026.mspx][Xref =>
http://cgi.nessus.org/plugins/dump.php3?id=11808][Xref =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2003-0352][Xref =>
http://www.securityfocus.com/bid/8205]

輸出資料

Generate dot graph format

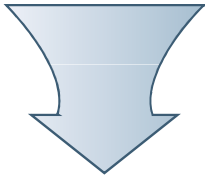


系統分析



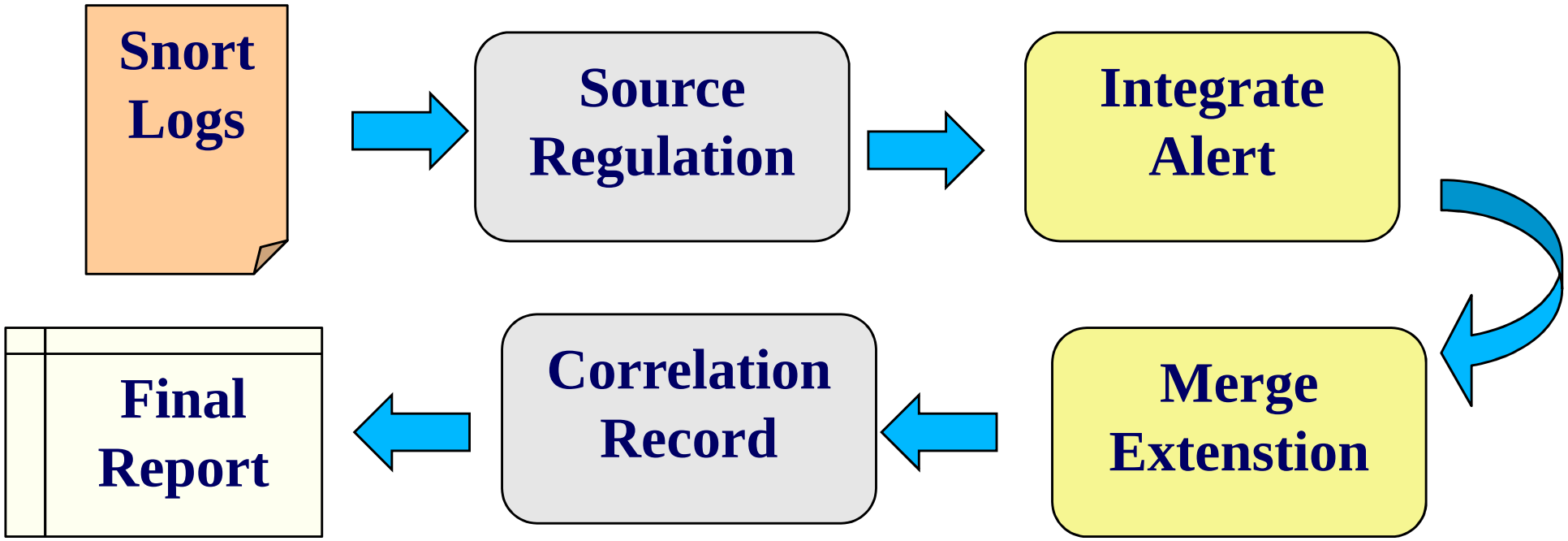
Alert Merge Example

Destination IP	Attack Signature	Source IP	Destination Port	Source Port	Packet Protocol	Timestamp
Host_1	Trojan	Sip1	80	4077	tcp	T1
Host_1	Trojan	Sip2	80	4077	tcp	T2
Host_1	Trojan	Sip1	443	5002	tcp	T3
Host_2	Trojan	Sip1	443	5002	tcp	T4
Host_3	D.D.O.S	Sip3	53	6007	udp	T5
Host_3	D.D.O.S	Sip4	53	6008	tcp	T5
Host_3	D.D.O.S	Sip5	53	6007	udp	T5
Host_3	D.D.O.S	Sip6	53	6008	tcp	T5



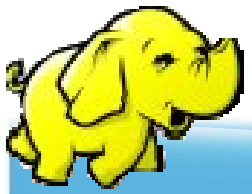
Key		Values				
Host_1	Trojan	Sip1,Sip2	80,443	4077,5002	tcp	T1,T2,T3
Host_2	Trojan	Sip1	443	5002	tcp	T4
Host_3	D.D.O.S.	Sip3,Sip4,Sip5 ,Sip6	53	6007,6008	tcp, udp	T5

程式流程圖


Snort Logs
Source Regulation
Integrate Alert
Merge Extenstion
Correlation Record
Final Report

Conclusions

- 評估
- 系統分析
  - 輸入輸出
  - 系統元件
  - 各元件參數與串流
- 實做
  - 多次運算
  - 前處理與後處理



進階課程

QUESTIONS
&
THANKS

財團法人國家實驗研究院
國家高速網路與計算中心
NATIONAL CENTER FOR HIGH-PERFORMANCE COMPUTING