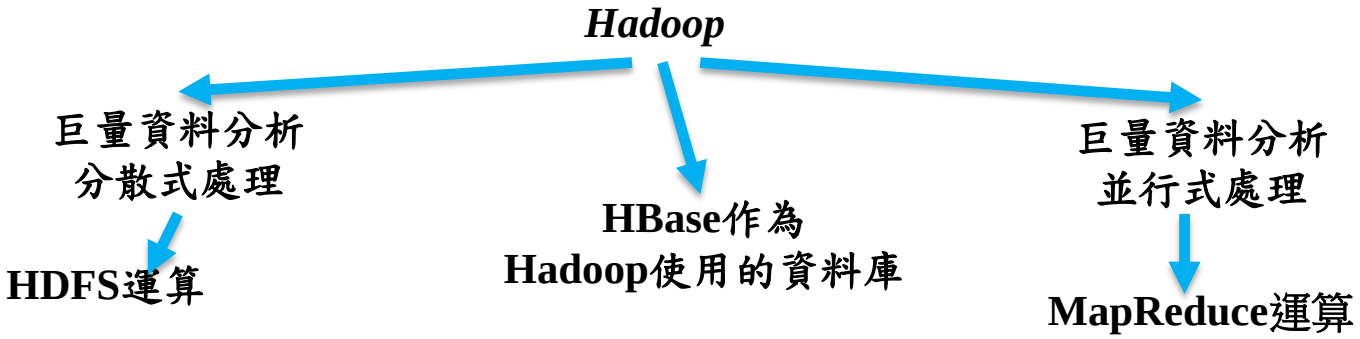
HADOOP MAPREDUCE

國立臺北科技大學資訊工程系
郭忠義

Hadoop

- Hadoop是Apache軟體基金會開放原始碼計劃
  - 以java寫成，提供巨量資料的分散式運算環境
- Hadoop架構由Google發表的BigTable及Google File System等概念實做，跟Google雲端運算架構相似。
  - Hadoop MapReduce如同Google MapReduce，提供分散式運算環境
  - Hadoop Distributed File System如同Google File System，提供大量儲存空間、HBase是一個類似 BigTable 的分散式資料庫 ，方便提供整合的雲端服務。



Hadoop

- Google File System
  - 可擴充的分散式檔案系統
  - 設計目的在於給大量用戶提供總體性能較高的服務
  - 適用於分散式、對大量資訊進行存取的應用
  - 可運作在一般的普通主機，提供錯誤容忍的能力
- The Google File System發表於SOSP' 03 October，並將設計概念公開

表一 Hadoop 與 Google 架構比較

Google	Hadoop
MapReduce	Hadoop MapReduce
GFS	HDFS
BigTable	HBase

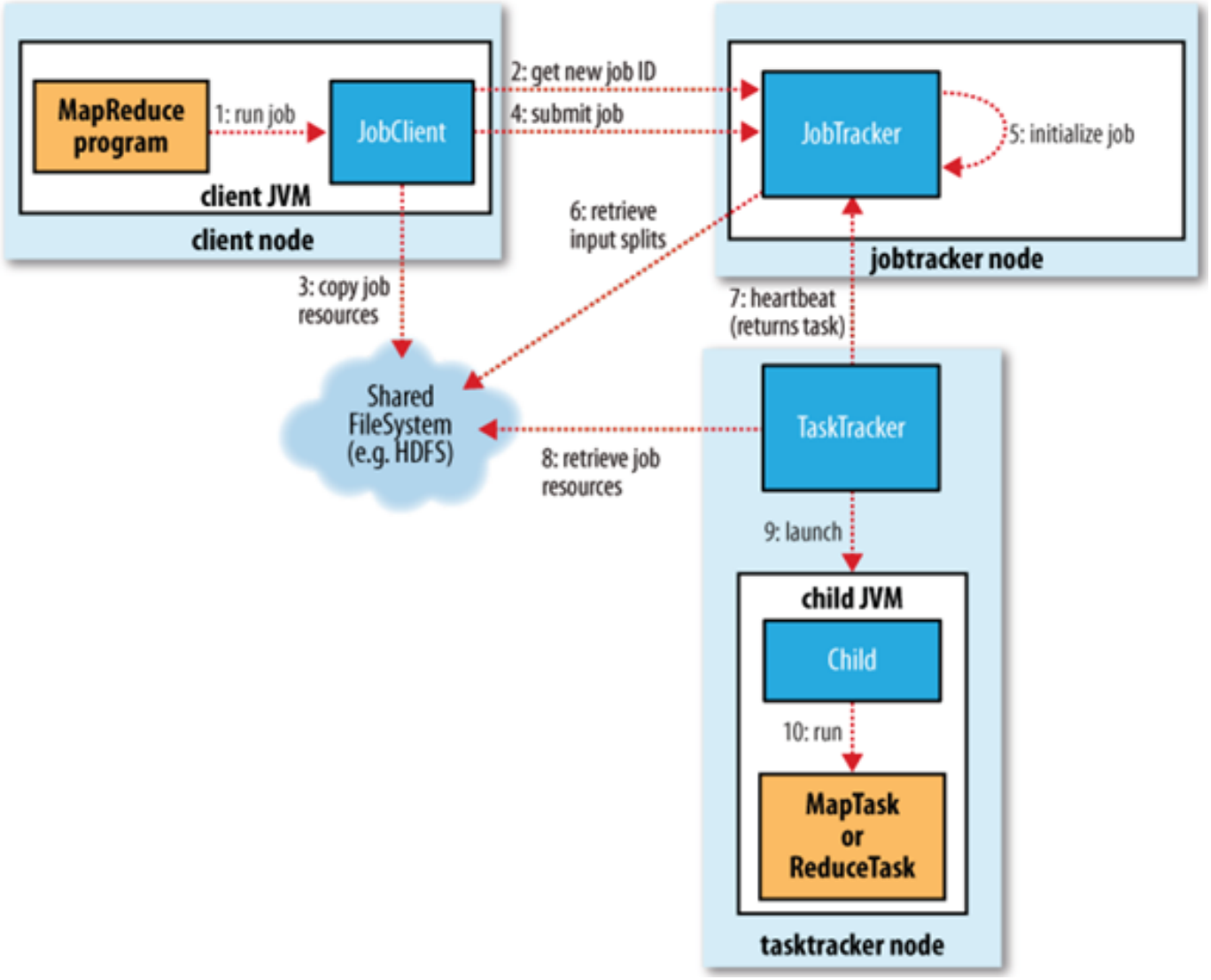
MapReduce Framework

- MapReduce是分散式程式框架，讓開發者簡單撰寫程式，利用大量運算資源，加速處理龐大的資料量。
- 一個MapReduce運算分成兩個部份—Map和Reduce
  - 大量資料在運算開始，被系統轉換成一組組 (key, value) 的序對並自動切割成許多部份
  - 分別傳給不同的Mapper處理，Mapper處理完後要將運算結果整理成一組組 (key, value)序對，再傳給Reducer整合所有Mapper結果，最後將整體結果輸出
- NameNode
  - HDFS file system 的中心區塊
- DataNode
  - HDFS儲存資料的地方

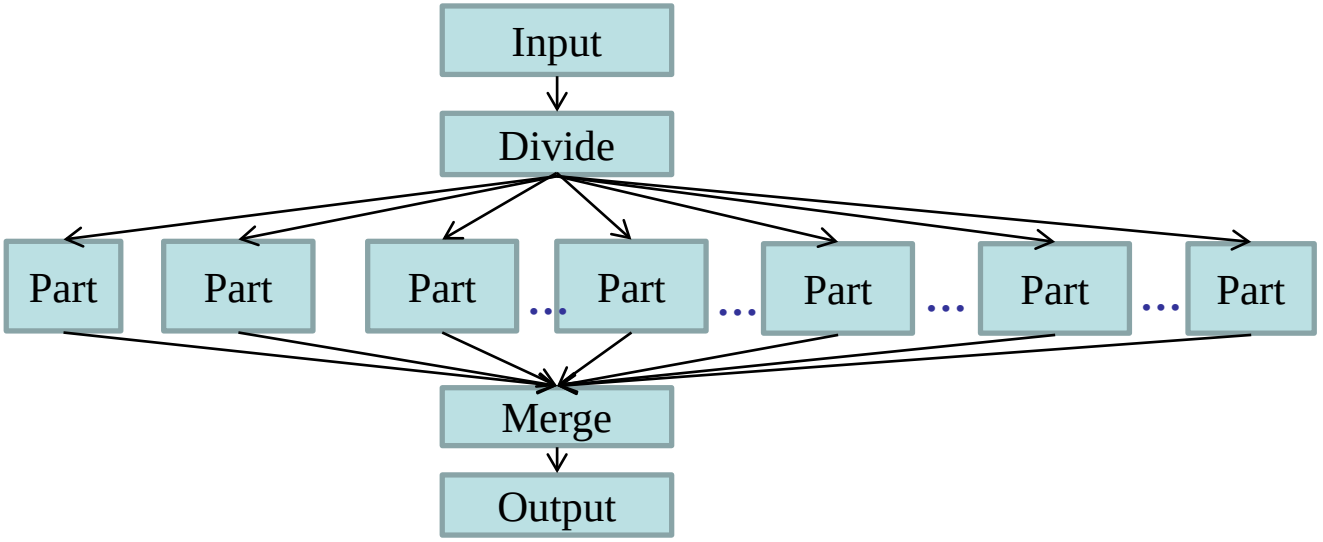
Hadoop術語

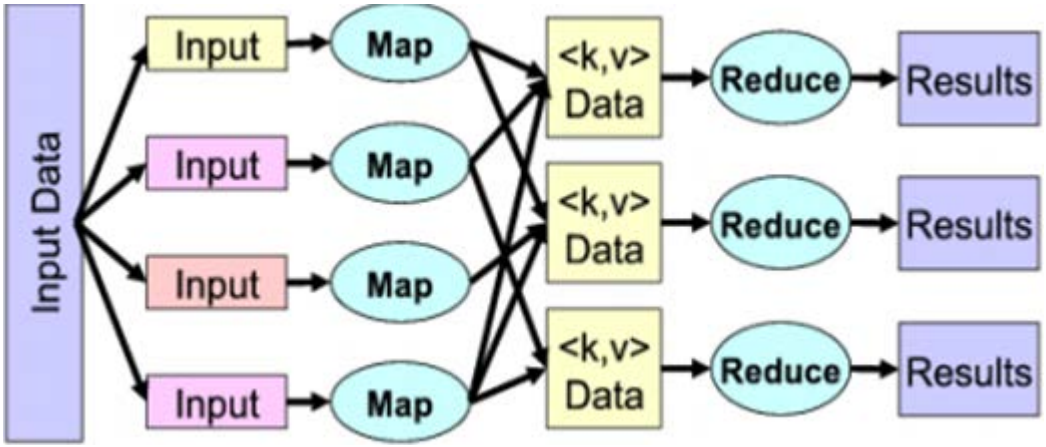
- Job 工作任務
- Task 由Job分解出的小工作
- JobTracker 工作任務分派者
- TaskTracker 小工作任務執行者
- Client 發起任務的客戶端
- Map 應對 Reduce 總和
- NameNode 名稱節點
- DataNode 資料節點
- Replication 資料檔案副本
- Block 檔案區塊 64M
- Metadata 屬性資料

MapReduce Framework

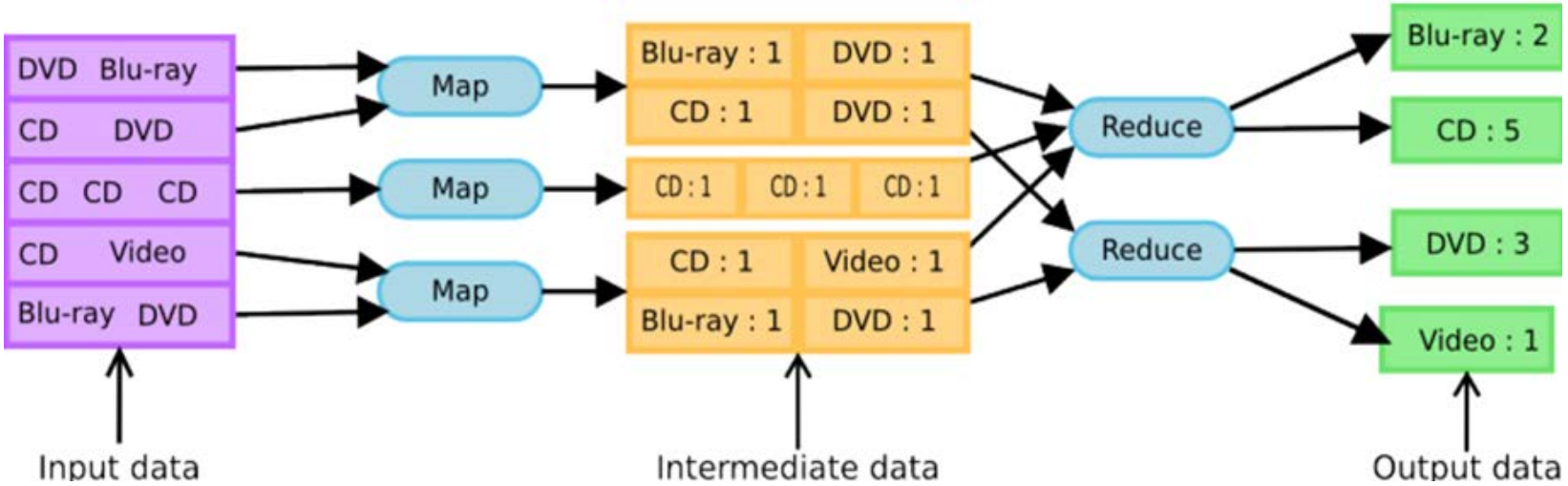


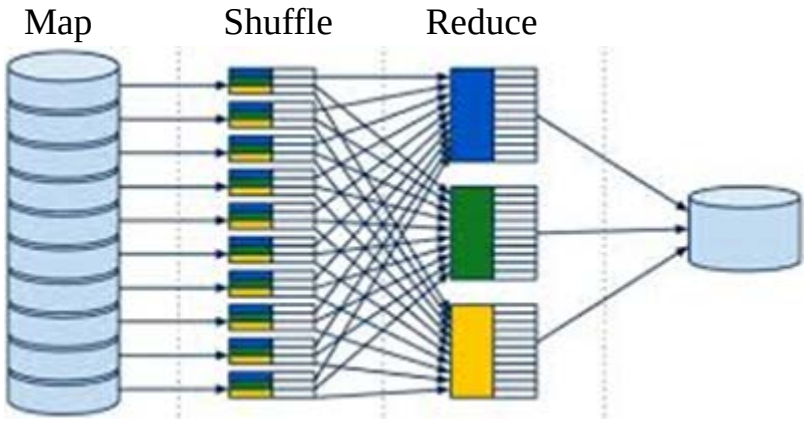
MapReduce Framework



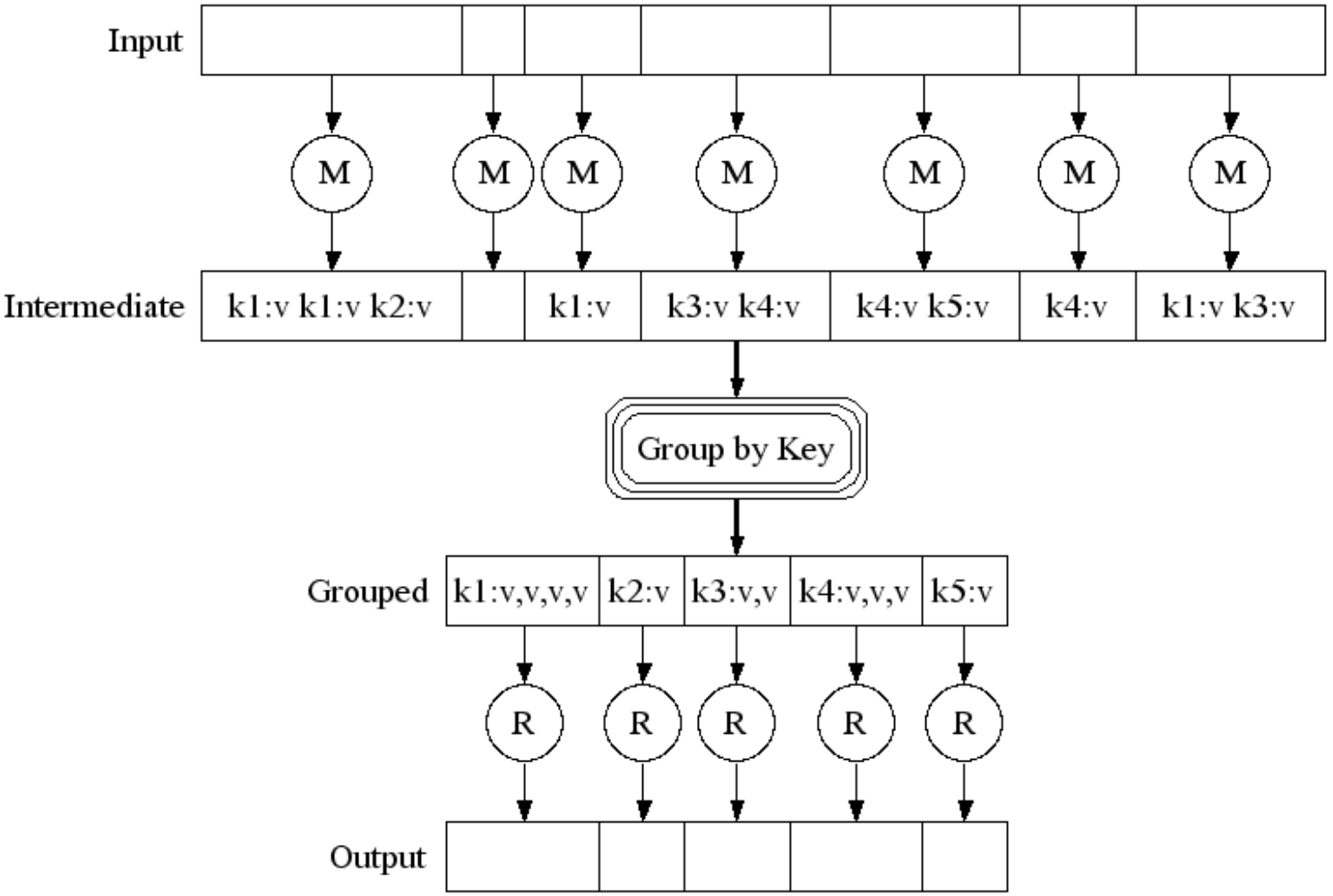


MapReduce Framework

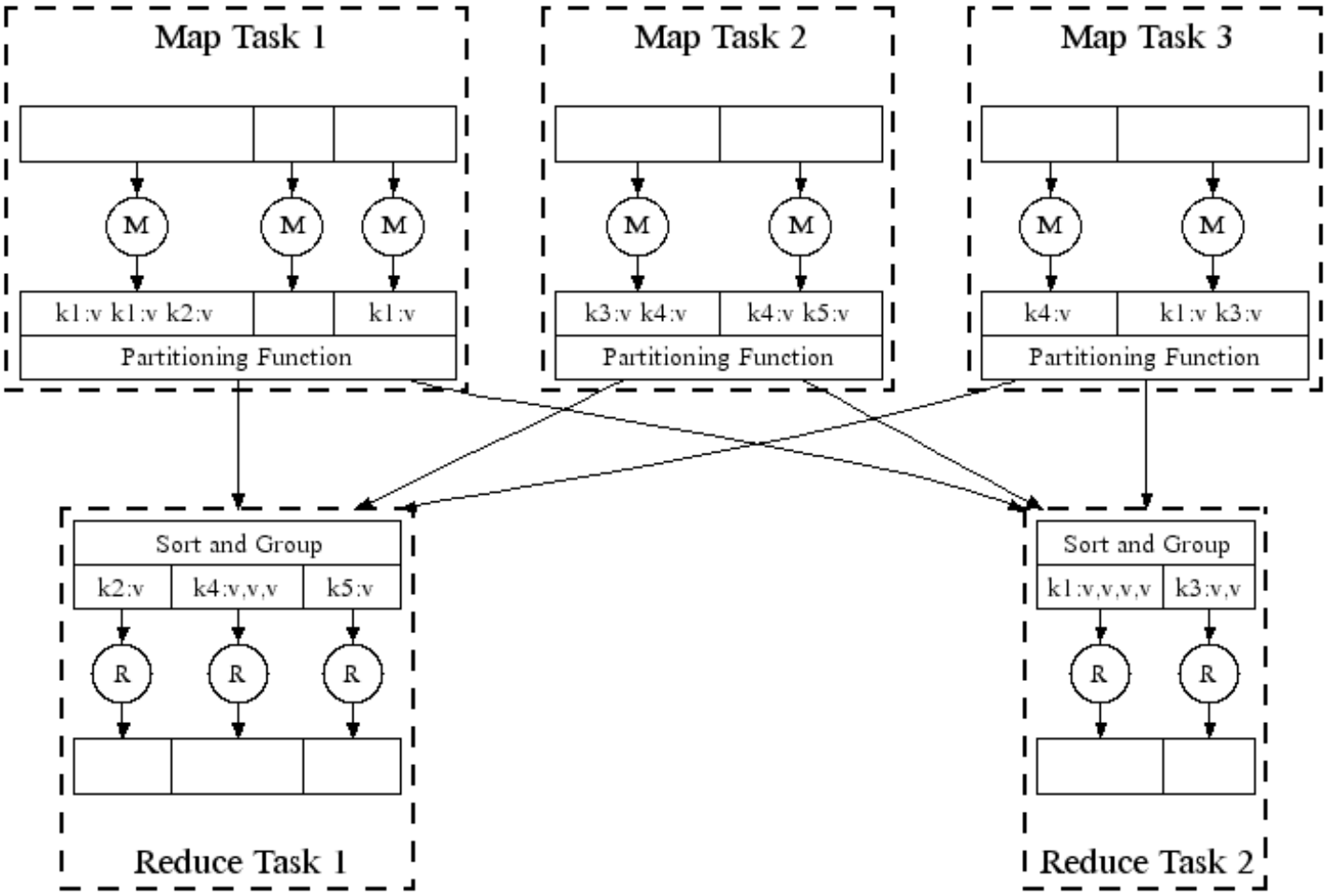




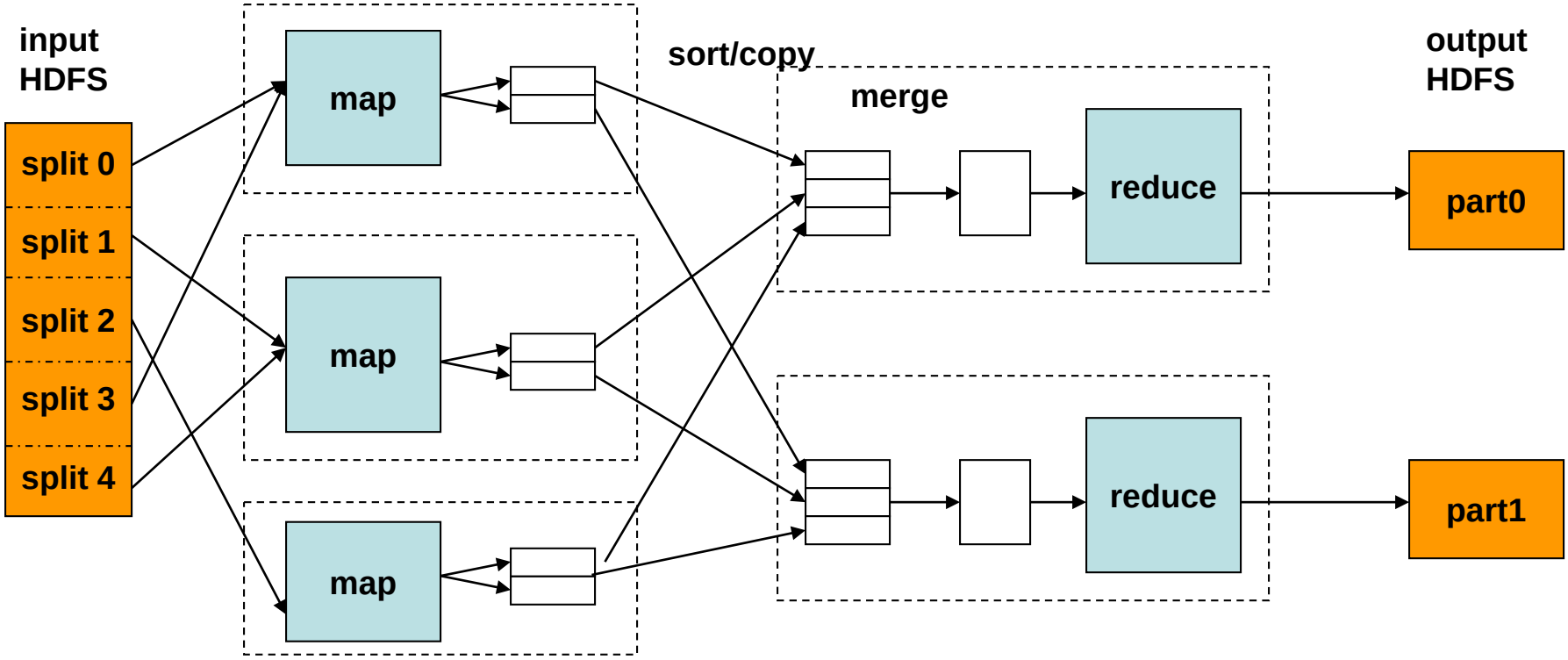
MapReduce Framework



MapReduce Framework



MapReduce Framework



JobTracker跟NameNode取得需要運算的blocks

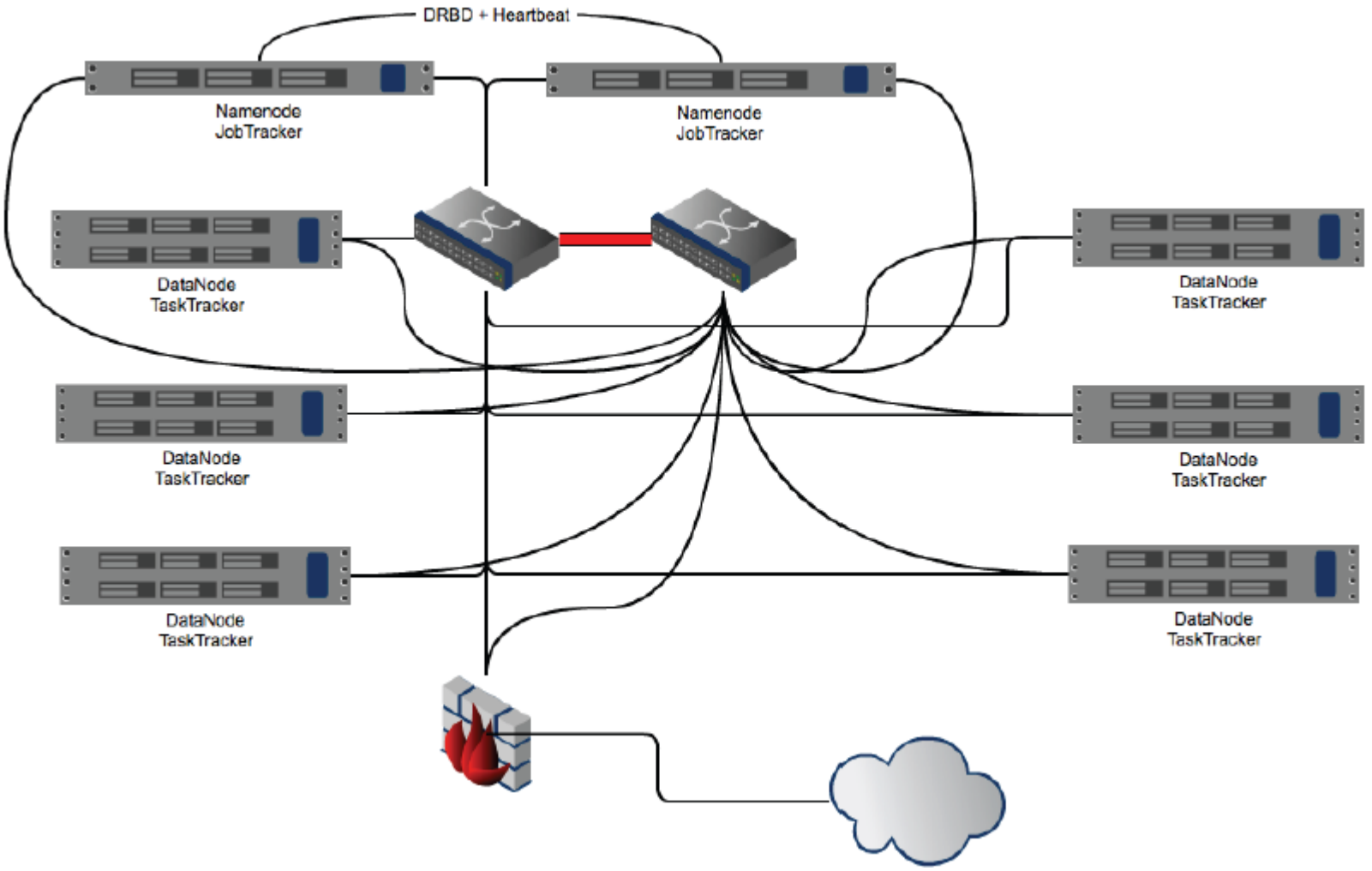
JobTracker選數個TaskTracker來作Map運算，產生中間檔案

JobTracker將中間檔案整合排序，複製到需要的TaskTracker

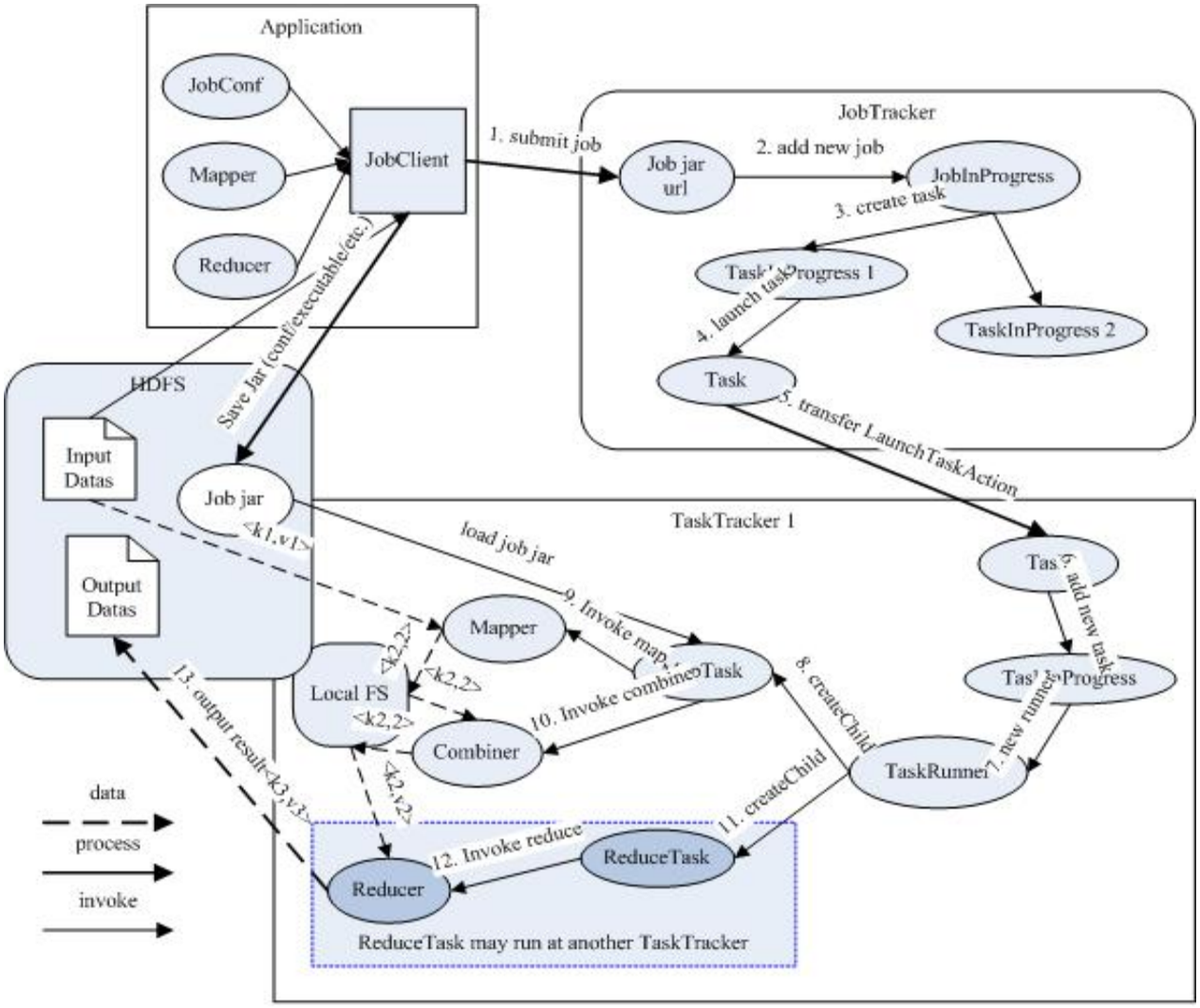
JobTracker派遣TaskTracker作reduce

reduce完後通知JobTracker與Namenode以產生output

MapReduce Framework

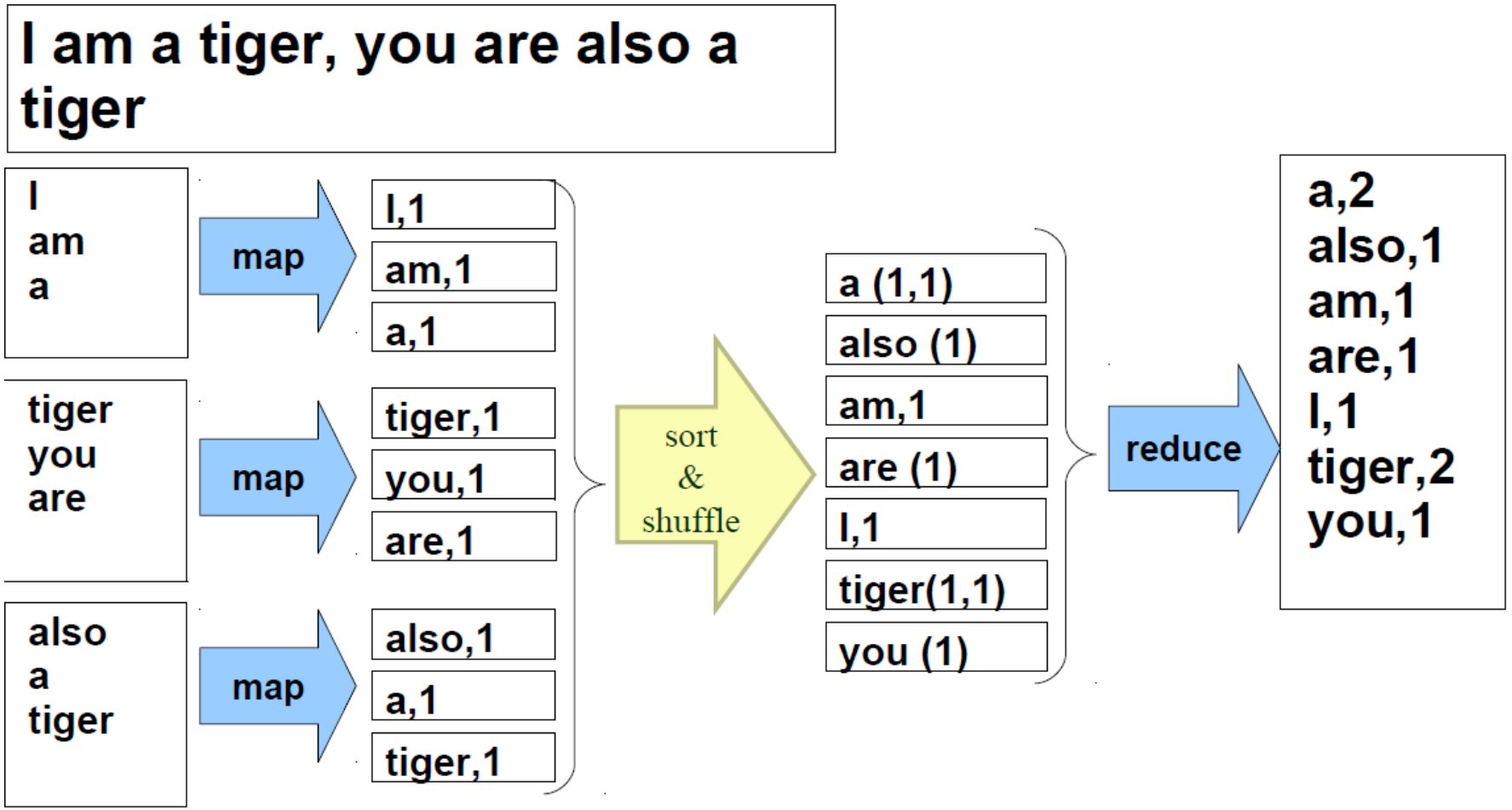


MapReduce Framework

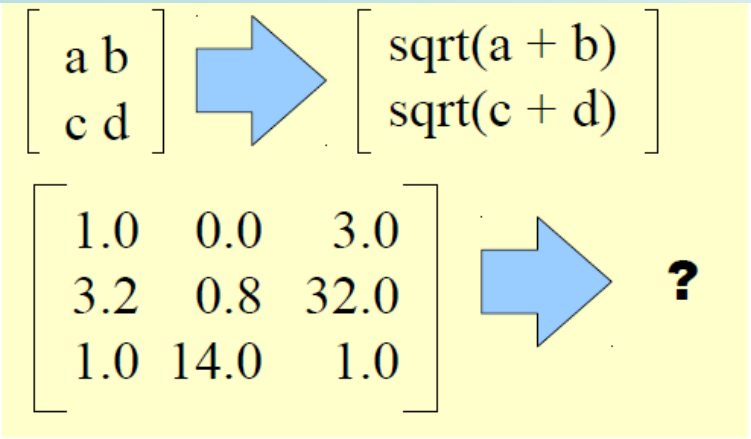


Hadoop MapReduce Framework Architecture

MapReduce Framework



MapReduce Framework




```
(0,sqrt(1.0 +  0.0 +  3.0))
(1,sqrt(3.2 +  0.8 + 32.0))
(2,sqrt(1.0 + 14.0 +  1.0))
```

Input File

```
0 0 1.0   // A[0][1] = 1.0
0 1 0.0   // A[0][1] = 0.0
0 2 3.0   // A[0][2] = 3.0
1 0 3.2   // A[1][0] = 3.2
1 1 0.8   // A[1][1] = 0.8
```

map

```
(0,1.0)
(0,0.0)
(0,3.0)
(1,3.2)
(1,0.8)
```

```
1 2 32.0 // A[1][2] = 32.0
2 0 1.0  // A[2][0] = 1.0
2 1 14.0 // A[2][1] = 14.0
2 2 1.0  // A[2][2] = 1.0
```

map

```
(1,32.0)
(2,1.0)
(2,14.0)
(2,1.0)
```

sort / merge

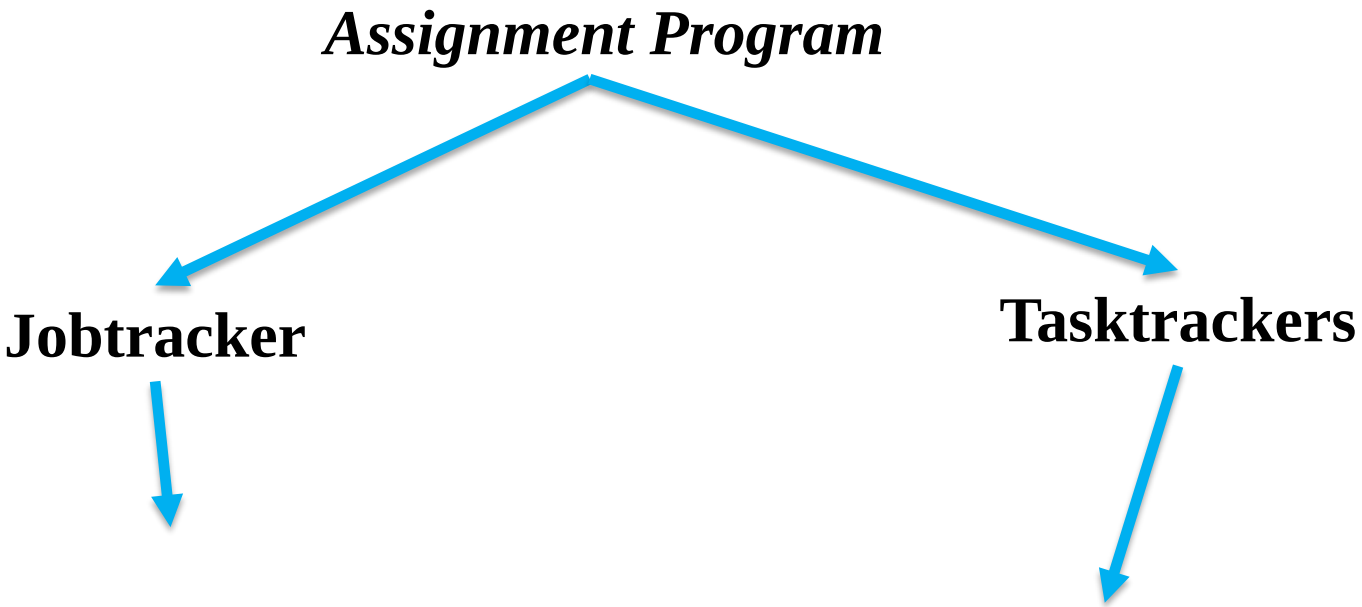
```
(0,{1.0,0.0,3.0})
(1,{3.2,0.8,32.0})
(2,{1.0,14.0,1.0})
```

reduce

MapReduce Framework

- 建立JobConf類別物件，設定運算工作的內容
  - setMapperClass/setReducerClass設定 Mapper及Reducer類別
  - setInputFormat/setOutputFormat 設定輸出輸入資料的格式
  - setOutputKeyClass / setOutputValueClass 設定輸出資料的類型
- 設定完成，依設定內容提交運算工作。
- 資料來源依InputFormat設定取得，分割轉換為一組組 (key, value) 序對，交由不同的Mapper同時進行運算，
- Mapper運算結果輸出為一組組(key, value) 序對，稱為中介資料，
- 系統將這些暫時結果排序 (sort) 並暫存，等到所有Mapper運算工作結束，依不同key值傳送給不同Reducer彙整，
- 所有同一key值的中介資料的value值，會放在一個容器 (container) 傳給同一個Reducer處理
- Reducer利用values.next()依序取得不同value值，快速完成結果整理，再依OutputFormat設定輸出為檔案。
- 進行運算的Mapper和Reducer系統會自動指派不同運算節點擔任，程式設計時不用做資料和運算切割，運算資源由JobTracker分配到各運算節點的TaskTracker，指派不同節點擔任Mapper和Reducer。

MapReduce Framework



Master
- ✓使用者發起工作
- ✓指派工作給 Tasktrackers
- ✓排程決策、工作分配

Workers
- ✓運作Map 與Reduce 工作
- ✓管理儲存 回覆運算結果

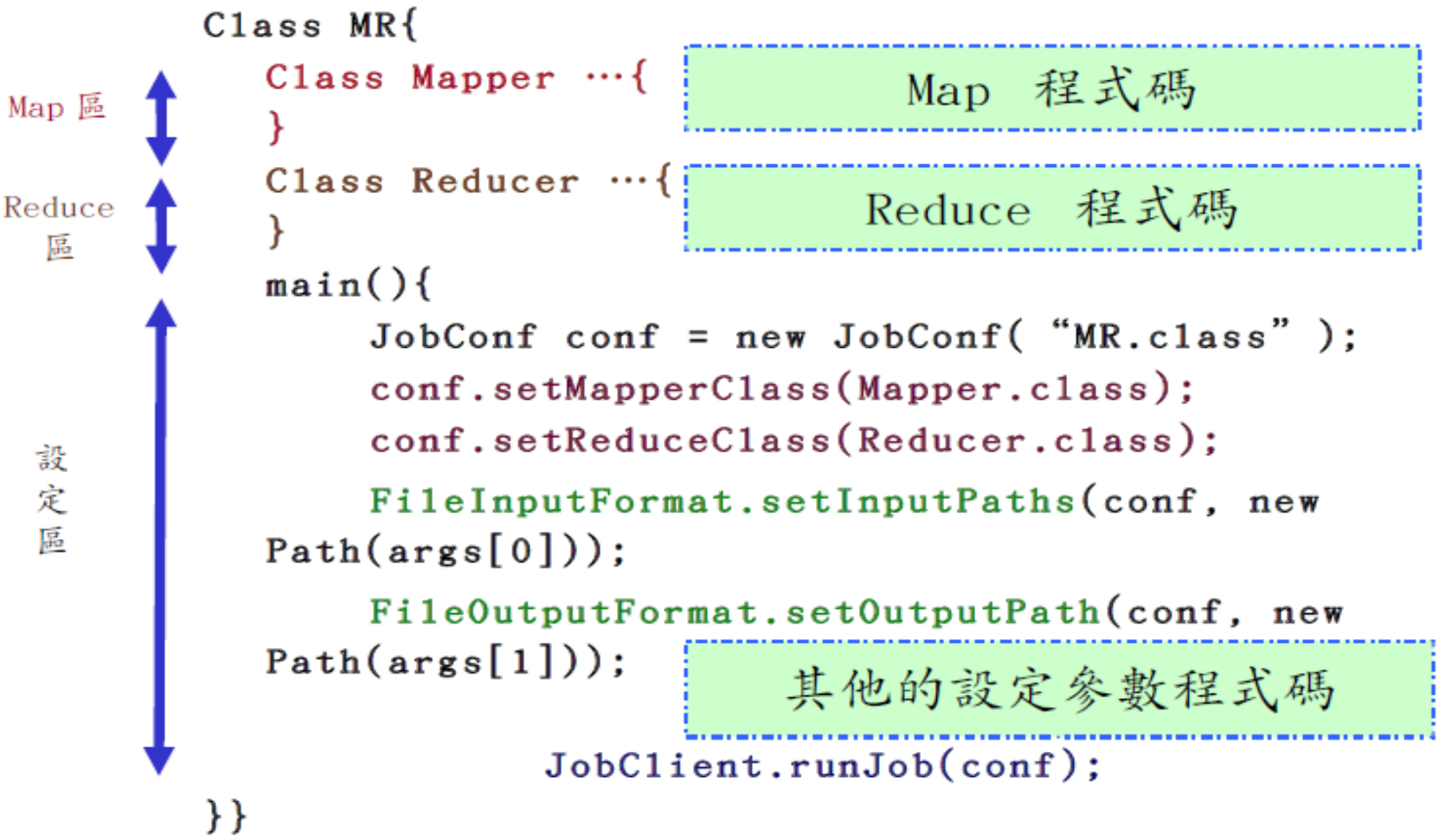
MapReduce Framework

- JobTracker: 分配工作
  - START Job
    - Main(), startTracker(), completedJobStatusStore(), interTrackerServer()
  - INIT Job
    - submitJob(), JobInitThread(), initTasks()
  - Scheduler
    - JobQueueTaskScheduler(), assignTasks()

MapReduce Framework

- TaskTracker: 接受工作
  - START
    - Main()
  - ADD Task To Child
    - run(), transmitHeartBeat(), addToTaskQueue(), startNewTask(), localizeJob(), launchTaskForJob(), createRunner()
  - Child Task Process
    - Map/Reduce, OutputCollector

Program Prototype



Class Mapper

```
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 ( key INPUT KEY , value INPUT 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  }
```

Class Reducer

```
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   }
```

Class Combiner

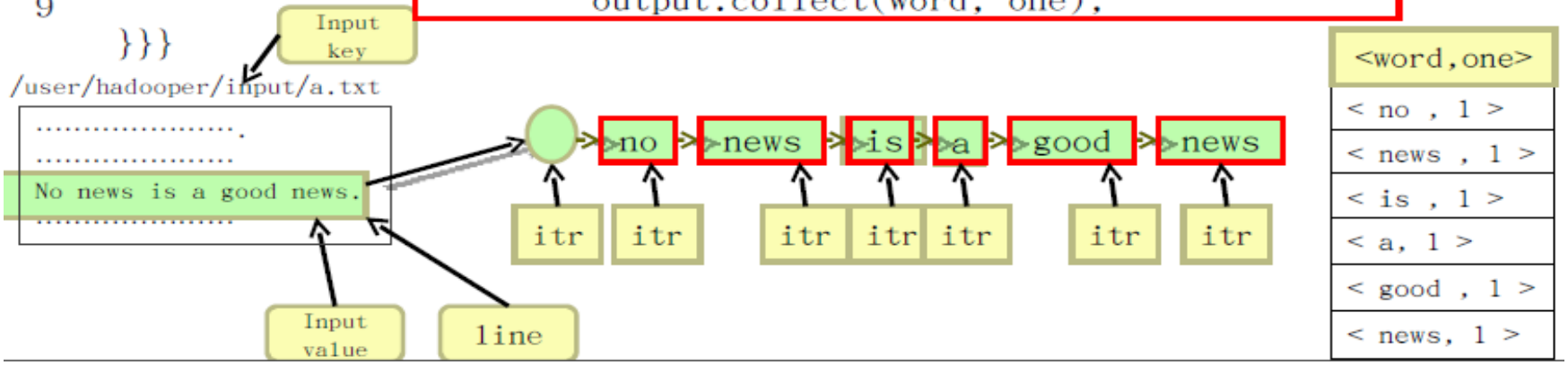
- 指定一個combiner，負責對中間過程的輸出進行聚集，會有助於降低從Mapper 到Reducer資料傳輸量。
- 可不用設定交由Hadoop預設也可不實做此程式，引用Reducer
- 設定
  - JobConf.setCombinerClass(Class)

Run Job

- RunJob(JobConf)
  - 提交作業，僅當作業完成時返回
- SubmitJob(JobConf)
  - 只提交作業，之後需要你輪詢它返回的
- RunningJob 句柄的狀態，並根據情況調度
- JobConf.setJobEndNotificationURI(String)
  - 設置一個作業完成通知，可避免輪詢

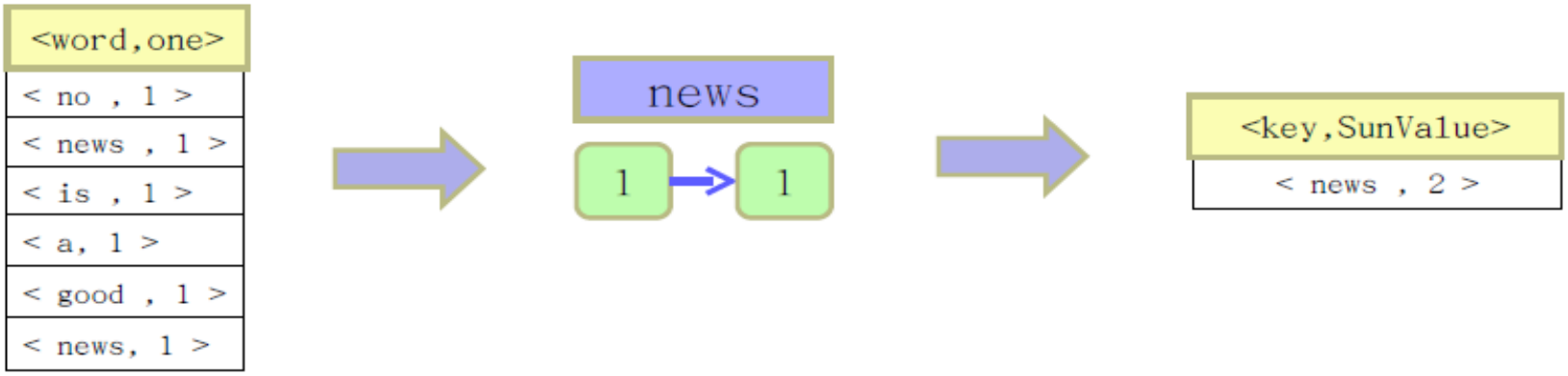
範例WordCount

```
1   class MapClass extends MapReduceBase implements
    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,
              OutputCollector<Text, IntWritable> output, Reporter
              reporter) throws IOException {
5                   String line = ((Text) value).toString();
6                   StringTokenizer itr = new StringTokenizer(line);
7                   while (itr.hasMoreTokens()) {
8                           word.set(itr.nextToken());
9                           output.collect(word, one);
    }}}
```



範例WordCount

```
1  class ReduceClass extends MapReduceBase implements Reducer< Text,
   IntWritable, Text, IntWritable> {
2          IntWritable SumValue = new IntWritable();
3          public void reduce( Text key, Iterator<IntWritable> values,
           OutputCollector<Text, IntWritable> output, Reporter reporter)
           throws IOException {
4                  int sum = 0;
5                  while (values.hasNext())
6                          sum += values.next().get();
7                  SumValue.set(sum);
8                  output.collect(key, SumValue);
   }}
```



範例WordCount

```
Class WordCount{
   static void main(){
      JobConf conf = new JobConf(WordCount.class);
      conf.setJobName("wordcount");
      // set path
      conf.setInputPath(new Path(“/user/waue/input”));
      conf.setOutputPath(new Path(“counts”));
      FileSystem.get(conf).delete(new Path(wc.outputPath));
      // set map reduce
      conf.setOutputKeyClass(Text.class);  // set every word as key
      conf.setOutputValueClass(IntWritable.class); // set 1 as value
      conf.setMapperClass(MapClass.class);
      conf.setReducerClass(ReduceClass.class);
      conf.setNumMapTasks(1);
      conf.setNumReduceTasks(1);// run
      JobClient.runJob(conf);
   }
}
```

作業

- 利用Hadoop分析1900年到2016年的天氣，從這些資料中找到某年的最高氣溫。

Mapper程式碼

Mapper：

- package com.charles.parseweather;
- import java.io.IOException;
- import org.apache.hadoop.io.IntWritable;
- import org.apache.hadoop.io.LongWritable;
- import org.apache.hadoop.io.Text;
- import org.apache.hadoop.mapreduce.Mapper;

——/**

Description: 是map類，定義map函數，base class有4個參數，分別是輸入key,輸入value，輸出key,輸出value的類型， 輸入key：LongWritable的位移量，代表某一行起始位置相對于檔案起始位置的位移。輸入value： 指定行的一行文字，包括氣温。輸出key:資料。輸出value：氣温 *0029029070999991901010106004+64333+023450FM12+000599999V0202701N015919999999N0000001N9 -00781+99999102001ADDGF108991999999999999999999 假如每一行都是106字，且這是第2行，則輸入key 106，輸入value ，輸出key:年份，是第15-19個字，所以是1901

* 輸出value:氣温是第87-92個字，可能是零下温度，所以是0（IntWritable類型）

```
public class MaxTemperatureMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
 //定義一個不可能的氣溫值，如果這一行的氣溫是此值，表示這一年沒有統計氣溫
   private static final int MISSING = 9999;
   //這個map方法將（檔案位移，本行）映射為（年份，氣溫），寫入Context中
   //網路序列化，轉為java.lang本類型
   public void map(LongWritable key,Text value ,Context context)
          throws IOException ,InterruptedException{
          //因為value代表這行文字，從中取資料
          //將網路傳輸的文字轉為String類型
         String line = value.toString();
        //從行中取得年份，為輸出key
        String year = line.substring(15,19);
        int airTemperature;
        //氣溫有正負，第87位是符號
      //正氣溫值，從下一個位置開始截取到92位置，然後轉為整數類型
      //負氣溫值，直接截取到92位置，然後轉為整數類型
```

```
        if(line.charAt(87) == '+'){
            airTemperature = Integer.parseInt(line.substring(88,92));
        }else{
            airTemperature =  Integer.parseInt(line.substring(87,92));
        }
        String quantity = line.substring(92,93);
        //quantity參數為正規表達式，quantity.matches("[01459]")是0,1,4,5,9才有效
        if(airTemperature != MISSING && quantity.matches("[01459]")){
            //只把正確氣溫值寫入Context，轉為Hadoop類型，透過網路傳到reduce中
            //key為年份，value為這一行中包含的氣溫值
            context.write(new Text(year),new IntWritable(airTemperature));
        }
    }
}
```

Reducer：

```
package com.charles.parseweather;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
// Description:是reduce類，定義reduce函數base class4個參數，輸入key,輸入value，輸出key,輸出value。
// reduce的輸入類型須和map輸出類型匹配，因在MaxTemperatureMapper中，輸出類型是
// (Text,IntWritable) reduce的輸入類型須(Text,IntWritable)。把 (年份，氣溫)傳給reduce
//  輸入key：從Mapper傳來年份,Text類型輸入value：從Mapper傳來當年氣溫，IntWritable類型
//  輸出key: 年份輸出value：這一年最高氣溫，比如(1949,111),(1949,78)
public class MaxTemperatureReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   //reduce可將（年份，<氣溫,氣溫,氣溫>）列表比較，找到最大值
   // map函數，所有氣溫依次存到context，年份對應的氣溫是List
   public void reduce(Text key,Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException{
         int maxValue=Integer.MIN_VALUE;
         //循環找到最大氣溫，如果是比當前最大的大，那麼取而代之
         for (IntWritable value : values){
             maxValue=Math.max(maxValue, value.get());
          }
          context.write(key,new IntWritable(maxValue));
     }
}
```

驅動類，負責給Map-Reduce，輸出結果：

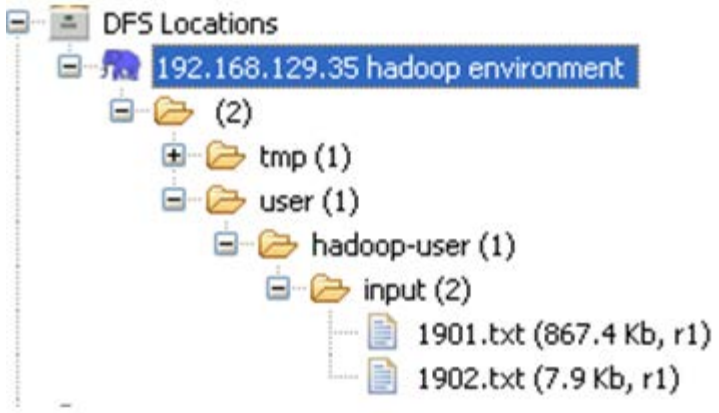
```
package com.charles.parseweather;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
// Description: 這個類定義且操作//
public class MaxTemperature {
public static void main(String[] args) throws Exception{
// TODO Auto-generated method stub
if (args.length !=2){
        System.err.println("Usage: MaxTemperature <input path> <output path>");
        System.exit(-1);
    }   //創建一個的Map-Reduce的作業
```

```
        Configuration conf = new Configuration();
        conf.set("hadoop.job.ugi", "hadoop-user,hadoop-user");
        Job job = new Job(conf,"Get Maximum Weather Information!");
        //設定作業的啟動類
        job.setJarByClass(MaxTemperature.class);
        //解析輸入和輸出參數，分別作為作業的輸入和輸出
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        //配置作業，設定Mapper類，Reducer類
        job.setMapperClass(MaxTemperatureMapper.class);
        job.setReducerClass(MaxTemperatureReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true)?0:1);
    }
}
```

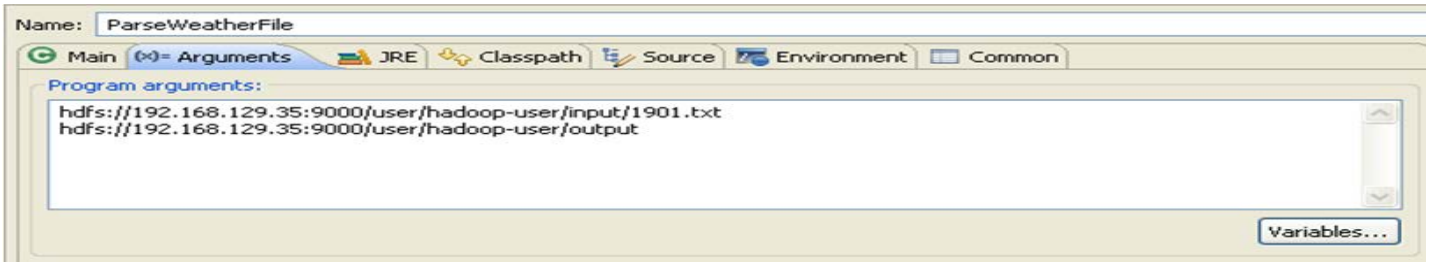
- 因為驅動要2個參數，第一個參數是HDFS檔案系統中包含樣本資料的檔案位置，第二個參數是HDFS檔案系統中處理完後的輸出檔案所在目錄。
- 先把樣本資料（1901.txt)放入指定位置且確認其存在：

```
[hadoop-user@localhost bin]$ hadoop fs -put /home/hadoop-user/copy/1902.txt /use
r/hadoop-user/input
[hadoop-user@localhost bin]$ hadoop fs -ls /user/hadoop-user/input
Found 2 items
-rw-r--r--   1 hadoop-user supergroup     888190 2012-05-25 17:41 /user/hadoop-u
ser/input/1901.txt
-rw-r--r--   1 hadoop-user supergroup       8117 2012-05-25 18:02 /user/hadoop-u
ser/input/1902.txt
```

- 可在IDE中用Hadoop 視圖查看檔案系統：



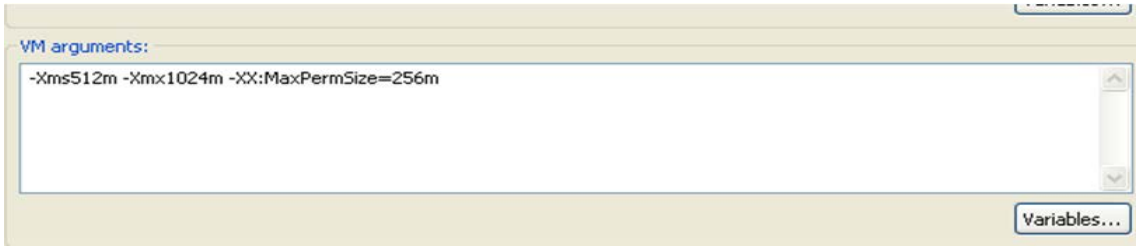
- 然後傳入2個參數（HDFS中輸入檔案位置和輸出目錄）：



- stack memory overflow

```
12/05/25 18:08:34 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
12/05/25 18:08:34 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should
12/05/25 18:08:34 WARN mapred.JobClient: No job jar file set.  User classes may not be found. See JobConf(Class)
12/05/25 18:08:39 INFO input.FileInputFormat: Total input paths to process : 1
12/05/25 18:08:39 INFO mapred.JobClient: Running job: job_local_0001
12/05/25 18:08:39 INFO input.FileInputFormat: Total input paths to process : 1
12/05/25 18:08:39 INFO mapred.MapTask: io.sort.mb = 100
12/05/25 18:08:39 WARN mapred.LocalJobRunner: job_local_0001
java.lang.OutOfMemoryError: Java heap space
        at org.apache.hadoop.mapred.MapTask$MapOutputBuffer.<init>(MapTask.java:781)
        at org.apache.hadoop.mapred.MapTask$NewOutputCollector.<init>(MapTask.java:524)
        at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:613)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:305)
        at org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:177)
12/05/25 18:08:40 INFO mapred.JobClient:  map 0% reduce 0%
12/05/25 18:08:40 INFO mapred.JobClient: Job complete: job_local_0001
12/05/25 18:08:40 INFO mapred.JobClient: Counters: 0
```

- 因為預設JDK的stack記憶體是64M，所以把它調大：



執行成功，正確顯示過程在控制台上：

```
14/01/11 17:24:25 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
14/01/11 17:24:25 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
14/01/11 17:24:25 WARN mapred.JobClient: No job jar file set.  User classes may not be found. See JobConf(Class) or JobConf#setJar(String).
14/01/11 17:24:25 INFO input.FileInputFormat: Total input paths to process : 1
14/01/11 17:24:25 INFO mapred.JobClient: Running job: job_local_0001
14/01/11 17:24:25 INFO input.FileInputFormat: Total input paths to process : 1
14/01/11 17:24:25 INFO mapred.MapTask: io.sort.mb = 100
14/01/11 17:24:25 INFO mapred.MapTask: data buffer = 79691776/99614720
14/01/11 17:24:25 INFO mapred.MapTask: record buffer = 262144/327680
14/01/11 17:24:25 INFO mapred.MapTask: Starting flush of map output
14/01/11 17:24:25 INFO mapred.MapTask: Finished spill 0
14/01/11 17:24:25 INFO mapred.TaskRunner: Task:attempt_local_0001_m_000000_0 is done. And is in the process of commiting
14/01/11 17:24:25 INFO mapred.LocalJobRunner:
14/01/11 17:24:25 INFO mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' done.
14/01/11 17:24:25 INFO mapred.LocalJobRunner:
14/01/11 17:24:25 INFO mapred.Merger: Merging 1 sorted segments
14/01/11 17:24:25 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 72206 bytes
14/01/11 17:24:25 INFO mapred.LocalJobRunner:
14/01/11 17:24:25 INFO mapred.TaskRunner: Task:attempt_local_0001_r_000000_0 is done. And is in the process of commiting
```

```
14/01/11 17:24:25 INFO mapred.LocalJobRunner:
14/01/11 17:24:25 INFO mapred.TaskRunner: Task attempt_local_0001_r_000000_0 is allowed to commit now
14/01/11 17:24:25 INFO output.FileOutputCommitter: Saved output of task 'attempt_local_0001_r_000000_0' to
hdfs://192.168.129.35:9000/user/hadoop-user/output
14/01/11 17:24:25 INFO mapred.LocalJobRunner: reduce > reduce
14/01/11 17:24:25 INFO mapred.TaskRunner: Task 'attempt_local_0001_r_000000_0' done.
14/01/11 17:24:25 INFO mapred.JobClient:  map 100% reduce 100%
14/01/11 17:24:25 INFO mapred.JobClient: Job complete: job_local_0001
14/01/11 17:24:25 INFO mapred.JobClient: Counters: 14
14/01/11 17:24:25 INFO mapred.JobClient:   FileSystemCounters
14/01/11 17:24:25 INFO mapred.JobClient:     FILE_BYTES_READ=105868
14/01/11 17:24:25 INFO mapred.JobClient:     HDFS_BYTES_READ=1776380
14/01/11 17:24:25 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=212428
14/01/11 17:24:25 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=9
14/01/11 17:24:25 INFO mapred.JobClient:   Map-Reduce Framework
14/01/11 17:24:25 INFO mapred.JobClient:     Reduce input groups=1
14/01/11 17:24:25 INFO mapred.JobClient:     Combine output records=0
14/01/11 17:24:25 INFO mapred.JobClient:     Map input records=6565
14/01/11 17:24:25 INFO mapred.JobClient:     Reduce shuffle bytes=0
14/01/11 17:24:25 INFO mapred.JobClient:     Reduce output records=1
14/01/11 17:24:25 INFO mapred.JobClient:     Spilled Records=13128
14/01/11 17:24:25 INFO mapred.JobClient:     Map output bytes=59076
14/01/11 17:24:25 INFO mapred.JobClient:     Combine input records=0
14/01/11 17:24:25 INFO mapred.JobClient:     Map output records=6564
14/01/11 17:24:25 INFO mapred.JobClient:     Reduce input records=6564
```

從控制台，可以看到

第2行：使用GenericOptionsParser可以解析傳入參數（輸入檔案，輸出目錄），分析結果只有一個輸入檔案。

第5行：為作業(也就是main應用）分配一個作業id叫job_local_0001

第7到11行：是MapTask的任務，它對Map進行設定。

第7行：io.sort.mb表明map輸出結果在記憶體中佔用的buffer的大小，設為100MB,因map的輸出不直接寫硬碟，而是寫入暫存，直到達一定數量，則後台去寫硬碟。

第11行：每次記憶體向硬碟flush會產生一個spill檔案，這一行是這個spill檔案。

第12-14行：map任務完成，map任務的id為attempt_local_0001_m_000000_0

第16-17行：進行合併(Merge)Map 過程的結果。

第18-19行：reduce任務完成，這個reduce任務的id為attempt_local_0001_r_000000_0

第20-24行：reduce任務用於輸出結果到HDFS系統，結果檔案存在命令行參數指定的目錄中

第25-43行：是map-reduce過程的一個總結性報告。

校驗檔案系統，發現最終結果（1901年最高氣溫為37度）：

```
[hadoop-user@localhost ~]$ hadoop fs -cat /user/hadoop-user/output/part-r-00000
1901    317
```