Spark建置與開發實務

軟體發展組
莊家雋

Outline

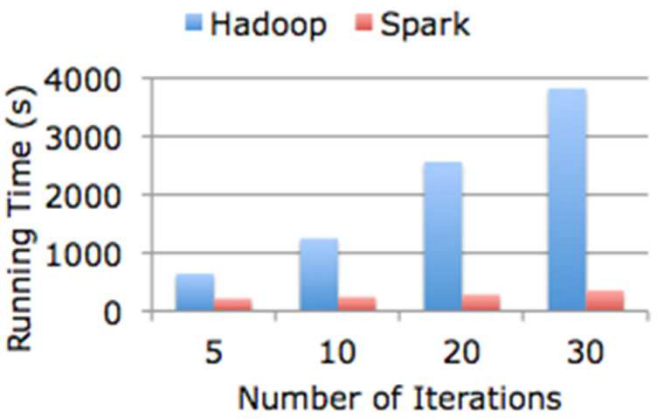
- Part I : Quick tour of Spark
  - Why Spark
  - Prepare Lab VMs
  - Run first Spark example
    - {Java / Python / Scala } code via {Shell / Standalone application }
    - Introduction to FP, Java 8, and Scala
- Part II : Programming with RDD
  - RDD Concepts
  - RDD, RDD API, convert RDD
  - Load and save data
  - Spark application

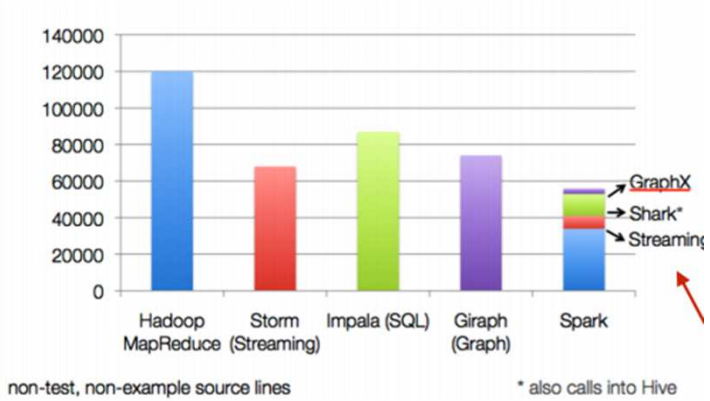
Outline

- Part III: Run on Cluster
  - Cluster Manager
    - Standalone, YARN, Mesos
  - Components of Execution
    - Application, Job, stage, task, RDD, partition, DAG

Why Spark

- Compare with Hadoop ecosystem
  - More efficient execution
  - More unified program abstraction
  - More flexible program operation





transformation
map(*func*)
filter(*func*)
flatMap(*func*)
sample(*withReplacement*, *fraction*, *seed*)
union(*otherDataset*)
distinct([*numTasks*]))
groupByKey([*numTasks*])
reduceByKey(*func*, [*numTasks*])
sortByKey([*ascending*], [*numTasks*])
join(*otherDataset*, [*numTasks*])
cogroup(*otherDataset*, [*numTasks*])
cartesian(*otherDataset*)

Exercise i-1:
Bring up Lab VMs

- Lab VM has
  - Ubuntu OS / Java8 / maven3 / spark 1.5 tarball / R
- Download and Install Vagrant
  - https://www.vagrantup.com/downloads.html
- Download Vagrant file
  - https://raw.githubusercontent.com/ogre0403/NCHC-Spark-Tutorial/master/vagrant/Vagrantfile
- Bring up Lab VMs
  - vagrant up [spark-master|spark-slave]
- Shutdown Lab VMs
  - vagrant halt [spark-master|spark-slave]

Exercise i-2:
Try interactive Spark shell

- $cp /opt/spark-1.5.1-bin-hadoop2.6.tgz ~
- $tar zxvf spark-1.5.1-bin-hadoop2.6.tgz
- $mv spark-1.5.1-bin-hadoop2.6 spark
- Launch interactive shell
  - $ ~/spark/bin/spark-shell
  - $ ~/spark/bin/pyspark

- Hadoop-free version v.s. pre-built for Hadoop
  - http://spark.apache.org/downloads.html
  - http://spark.apache.org/docs/latest/hadoop-provided.html

Shell example

- spark-shell
  - **scala>** val lines = sc.textFile("README.md")
  - **scala>** val pythonLines = lines.filter(s => s.contains("Python"))
  - **scala>** pythonLines.collect()

- pyspark
  - **>>>** lines = sc.textFile("README.md")
  - **>>>** pythonLines = lines.filter(lambda line: "Python" in line)
  - **>>>** pythonLines.collect()

Functional Programming

- Pass FUNCTION (what to do) as method parameter, rather than OBJECT (what)

- Scala: FP language based on JVM
- Java: support FP since Java 8
- Python: support FP

non-Java Lambda v.s. Java 8 Lambda

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        System.out.println("button clicked");
    }
});
```

```
button.addActionListener(event ->
    System.out.println("button clicked"));
```

```
Thread thread1 = new Thread(new Runnable() {
    @Override
    public void run(){
        System.out.println("Task #1 is running");
    }
});
```

```
Runnable task2 = () -> {
    System.out.println("Task #2 is running"); };
```

```
//Sorting using Anonymous Inner class.
Collections.sort(personList, new Comparator<Person>(){
  public int compare(Person p1, Person p2){
    return p1.firstName.compareTo(p2.firstName);
  }
});
```

```
//Anonymous Inner class replaced with Lambda
expression.
Collections.sort(personList, (Person p1, Person p2) ->
p1.firstName.compareTo(p2.firstName));
```

MapReduce Recap

- The idea of MR originates from FP
- map() and reduce() both are FP operations
- Pass what to do as map() and reduce() parameter

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

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

            }
}
```

Passing function **expression** to Spark

- Using function expression

```
JavaRDD<String> lines = sc.textFile("hdfs://log.txt")
                          .filter(s -> s.contains("error"));
long numErrors = lines.count();
```

Passing function **class** to Spark

- We will use function class in our tutorial
  - Override different call() in different Function class

```
Thread thread1 = new Thread(new Runnable() {
    @Override
    public void run(){
        System.out.println("Task #1 is running");
    }
});
```

Function name	Method to implement	Usage
`Function<T, R>`	`R call(T)`	Take in one input and return one output, for use with operations like `map()` and `filter()`.
`Function2<T1, T2, R>`	`R call(T1, T2)`	Take in two inputs and return one output, for use with operations like `aggregate()` or `fold()`.

Passing function **class** to Spark

- Using anonymous function class

```
JavaRDD<String> lines = sc.textFile("hdfs://log.txt").filter(
  new Function<String, Boolean>() {
    public Boolean call(String s) {
      return s.contains("error");
    }
});
long numErrors = lines.count();
```

- Using named class

```
class Contains implements Function<String, Boolean>() {
  private String query;
  public Contains(String query) { this.query = query; }
  public Boolean call(String x) { return x.contains(query); }
}

RDD<String> errors = lines.filter(new Contains("error"));
```

Exercise i-3:
Pass function to Spark in Java

- Download example code
  - git clone https://github.com/ogre0403/NCHC-Spark-Tutorial.git

- 篩選出含 “Python”的內容
  - JavaRDD<String> pythonLines = lines.filter(…)
  - Function class and expression

- 在IDE裡以local 模式執行spark application

Exercise i-4:
Build and launch standalone project

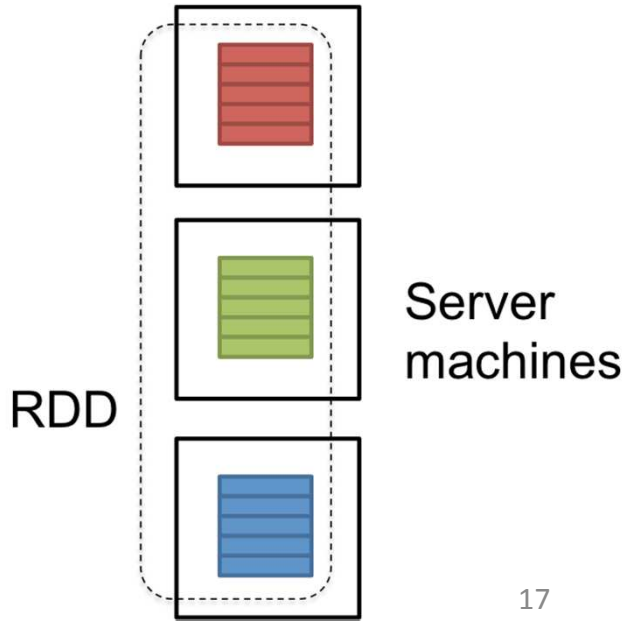
- Scala project
  - Built by sbt, launch by spark-submit
- Python project
  - Launch by spark-submit
- Java project
  - With-Lambda & non-Lambda
  - Built by maven, launch by spark-submit

Outline

- Part I : Quick tour of Spark
  - Why Spark
  - Prepare Lab Environment
  - Run first Spark example
    - {Java / Python / Scala } code via {Shell / Standalone application }
    - Introduction to FP, Java 8, and Scala
- Part II : Programming with RDD
  - RDD Concepts
  - RDD, RDD API, convert RDD
  - Load and save data
  - Spark Application

RDD Essentials

- Resilient distributed dataset
- Each RDD is split into multiple **partitions**
  - Partitions may exist on different machines
- immutable distributed collection of objects
  - Transform creates new RDD
- Spark keeps track lineage graph
  - Fast recovery from failure
- Lazy Evaluation
  - Until action is called



RDD operations

Transformations

- Create a new dataset from and existing one.
- Lazy in nature. They are executed only when some action is performed.
- Example :
  - Map(func)
  - Filter(func)
  - Distinct()

Actions

- Returns to the driver program a value or exports data to a storage system after performing a computation.
- Example:
  - Count()
  - Reduce(funct)
  - Collect
  - Take()

Persistence

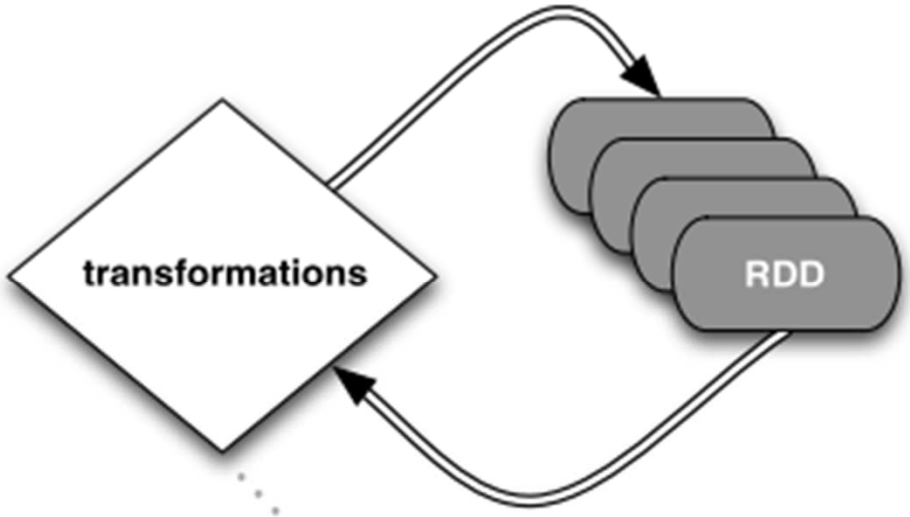
- For caching datasets in-memory for future operations.
- Option to store on disk or RAM or mixed (Storage Level).
- Example:
  - Persist()
  - Cache()

Create RDD


```
// base RDD
val lines = sc.textFile("hdfs://...")
```

```
JavaRDD<String> lines = sc.textFile("/path/to/README.md");
```

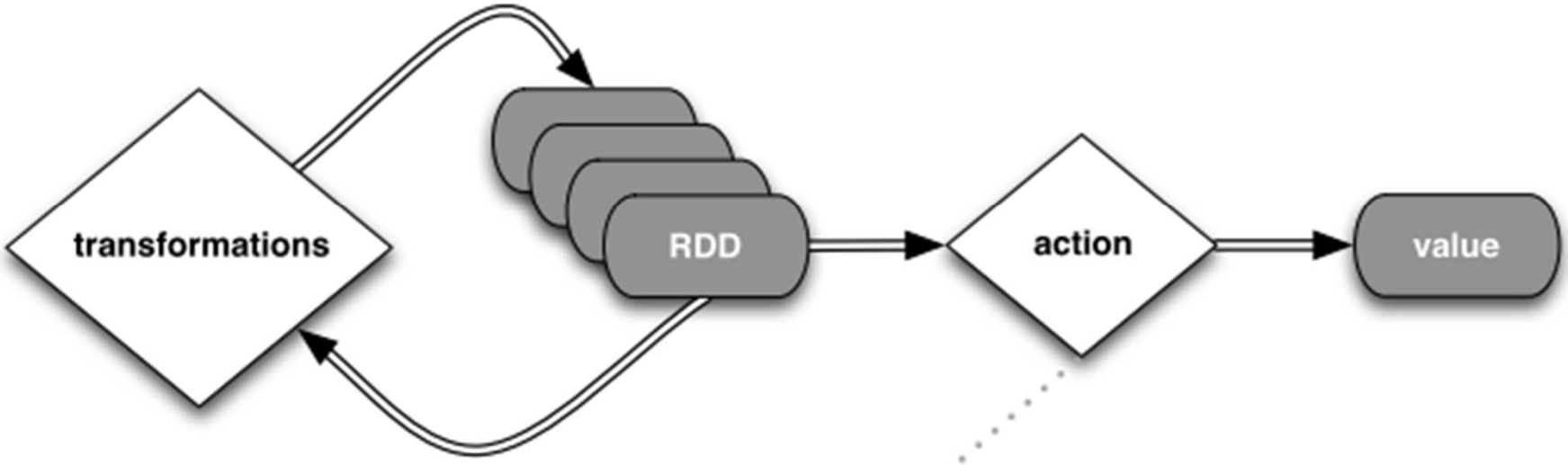
RDD Transformation




```
// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
messages.cache()
```

```
JavaRDD<String> errorsRDD = inputRDD.filter(
  new Function<String, Boolean>() {
    public Boolean call(String x) { return x.contains("error"); }
  }
});
```

RDD Action



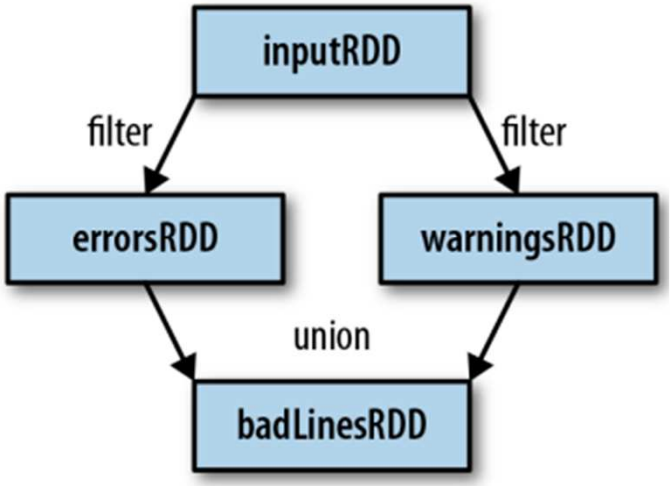

```
// action 1
messages.filter(_.contains("mysql")).count()
```

```
System.out.println("Input had " + badLinesRDD.count() + " concerning lines")
System.out.println("Here are 10 examples:")
for (String line: badLinesRDD.take(10)) {
  System.out.println(line);
}
```

Lineage Graph

- Think of each RDD as consisting of instructions on how to compute the data through transformations.

```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```



Cache() or Persist()

- Each time call a action, the entire RDD must be computed “from scratch”
  - Cache/persist the computed lineage result

Example 3-39. Double execution in Scala

```
val result = input.map(x => x*x)
println(result.count())
println(result.collect().mkString(","))
```

Example 3-40. persist() in Scala

```
val result = input.map(x => x * x)
result.persist(StorageLevel.DISK_ONLY)
println(result.count())
println(result.collect().mkString(","))
```

- If there are no enough memory for caching RDD partitions
  - MEMORY_ONLY:
    - Oldest will be deleted, then recomputed if necessary
  - MEMORY_AND_DISK:
    - Swap to disk and read back when necessary

Level	Space used	CPU time	In memory	On disk	Comments
MEMORY_ONLY	High	Low	Y	N	
MEMORY_ONLY_SER	Low	High	Y	N	
MEMORY_AND_DISK	High	Medium	Some	Some	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_SER	Low	High	Some	Some	Spills to disk if there is too much data to fit in memory. Stores serialized representation in memory.
DISK_ONLY	Low	High	N	Y	

RDD Type

- Basic RDD[T]
  - considers each data item as a single value
  - Convert to other RDD Type
- PairRDDs
  - each data item containing key/value pairs.
- DoubleRDD
  - Data items are convertible to the Scaladata-type double

http://blog.csdn.net/pelick/article/details/44922619
http://lxw1234.com/archives/2015/07/363.htm
http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html

Create RDD

- 從集合建立RDD
  - sc.parallelize(1 to 10)
  - sc.parallelize(Array("1","2","3"))

- 從檔案建立RDD
  - sc.textFile("file://path/to/file")
  - sc.textFile(”hdfs://path/to/file")

Basic RDD Transformation (part)

- RDD.map(func)

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
map()	Apply a function to each element in the RDD and return an RDD of the result.	rdd.map(x => x + 1)	{2, 3, 4, 4}

```
JavaRDD<Integer> rdd = sc.parallelize(Arrays.asList(1, 2, 3, 4));
JavaRDD<Integer> result = rdd.map(new Function<Integer, Integer>() {
  public Integer call(Integer x) { return x*x; }
});
System.out.println(StringUtils.join(result.collect(), ","));
```

Basic RDD Transformation (part)

- RDD.filter(func)

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`

```
RDD<String> errors = lines.filter(new Function<String, Boolean>() {
  public Boolean call(String x) { return x.contains("error"); }
});
```

Basic RDD Transformation (part)

- RDD.flatMap(func)

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`

```
JavaRDD<String> lines = sc.parallelize(Arrays.asList("hello world", "hi"));
JavaRDD<String> words = lines.flatMap(new FlatMapFunction<String, String>() {
  public Iterable<String> call(String line) {
    return Arrays.asList(line.split(" "));
  }
});
words.first();  // returns "hello"
```

Exercise ii-1:
RDD transformation operation

- Create RDD
  - scala> val rdd = sc.parallelize(1 to 10)
- map
  - scala> rdd.map(x => x +1).collect()
  - scala> rdd.map(x=>x.toFloat).collect()
- flatMap
  - scala > rdd.flatMap(x => (1 to x)).collect()
- filter
  - scala > rdd.filter(x => x > 5)

Basic RDD Action (part)

- RDD.collect()

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`

  - collect() will attempt to copy every single element in the RDD onto the single driver program, and then run out of memory and crash.

Basic RDD Action (part)

- RDD.reduce(func)

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`reduce(func)`	Combine the elements of the RDD together in parallel (e.g., `sum`).	`rdd.reduce((x, y) => x + y)`	9

```
Integer sum = rdd.reduce(new Function2<Integer, Integer, Integer>() {
  public Integer call(Integer x, Integer y) { return x + y; }
});
```

Basic RDD Action (part)

- RDD.fold(zero)(func)
  - Reduce()的一般式
  - Zero value會運用在每個partition中
  - Zero value也會用在合併partition的結果中

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`fold(zero)(func)`	Same as `reduce()` but with the provided zero value.	`rdd.fold(0)((x, y) => x + y)`	9

Basic RDD Action (part)

- RDD.foreach(func)
  - Foreach() iterates over a list and applies some operation with side effects to each list member
  - Map() iterates over a list, transforms each member of that list, and returns another list of the same size with the transformed members

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`foreach(func)`	Apply the provided function to each element of the RDD.	`rdd.foreach(func)`	Nothing

Basic RDD Action (part)

- RDD.saveAsTextFile()
  - 若路徑為本地路徑，則只會存在executor所在機器上

Exercise ii-2:
RDD action operation

- Reduce
  - scala> var rdd1 = sc.makeRDD(1 to 10,3)
  - scala> rdd1.reduce((a,b) => a+b)

  - scala> val range = ('a' to 'z').map(_.toString)
  - scala> val rdd = sc.parallelize(range,3)
  - scala> rdd.reduce((a,b) => a+b)
  - scala > rdd.partitions.size
- Fold
  - scala> var rdd1 = sc.makeRDD(1 to 10,3)
  - scala> rdd1.fold(0)((a,b) => a+b)

  - scala> val range = ('a' to 'z').map(_.toString)
  - scala> val rdd = sc.parallelize(range,3)
  - scala> rdd.fold(“1”)((a,b) => a+b)
  - scala > rdd.partitions.size

- foreach
  - scala> var rdd1 = sc.makeRDD(1 to 10,3)
  - scala> rdd1.foreach(x=>println(x))

- saveAsTextFile
  - scala> var rdd1 = sc.makeRDD(1 to 10,3)
  - scala> rdd1.saveAsTextFile(“/tmp/test”)

RDD Type

- Basic RDD[T]
  - considers each data item as a single value
  - Convert to other RDD Type
- PairRDD[K,V]
  - each data item containing key/value pairs.
- DoubleRDD
  - Data items are convertible to the Scaladata-type double

http://blog.csdn.net/pelick/article/details/44922619
http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html

Convert between RDD Type

- Scala: Basic RDD convert to
  - PairRDD: Implicit conversion on RDDs of tuple exists.
  - DoubleRDD: Sometimes, implicit convert RDD to RDD[Double]

- Java Basic RDD convert to
  - PairRDD : **explicitly** implement PairFunction()
  - DoubleRDD: **explicitly** implement DoubleFunction()

Function name	Equivalent function*<A, B,...>	Usage
PairFunction<T, K, V>	Function<T, Tuple2<K, V>>	PairRDD<K, V> from a mapToPair
DoubleFunction<T>	Function<T, double>	DoubleRDD from map ToDouble

```
PairFunction<String, String, String> keyData =
  new PairFunction<String, String, String>() {
  public Tuple2<String, String> call(String x) {
    return new Tuple2(x.split(" ")[0], x);
  }
};
JavaPairRDD<String, String> pairs = lines.mapToPair(keyData);
```

```
JavaDoubleRDD result = rdd.mapToDouble(
  new DoubleFunction<Integer>() {
    public double call(Integer x) {
      return (double) x * x;
    }
});
System.out.println(result.mean());
```

Exercise ii-3:
RDD conversion

- scala> val rdd = sc.parallelize(1 to 10)
- scala> val pairs = rdd.map(x=> (x,x+1))
- scala> pairs.collect()

- scala> val rdd1 = sc.parallelize(Array("1","2","3"))
- scala> rdd1.mean()
- scala> val rdd2 = rdd1.map(x=>x.toInt)
- scala> rdd2.mean()

RDD Type

- Basic RDD[T]
  - considers each data item as a single value
  - Convert to other RDD Type
- PairRDD[K,V]
  - each data item containing key/value pairs.
- DoubleRDD
  - Data items are convertible to the Scaladata-type double

http://blog.csdn.net/pelick/article/details/44922619
http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html

DoubleRDD

- 針對數值資料(Double)的RDD，提供敘述統計運算
- stats()
  - count()、mean()、sum()、variance()、stdev()、max()、min()

Exercise ii-4:
Remove Outliers

- scala> val rdd1 = sc.parallelize(Array(1,1,1,1,1,1,1,1,1,10))
- scala> val distanceDouble = distance.map(string => string.toDouble)
- scala> val stats = distanceDoubles.stats()
- scala> val stddev = stats.stdev
- scala> val mean = stats.mean
- scala> val reasonableDistances = distanceDoubles.filter(x =>

  math.abs(x-mean) < 3 * stddev)

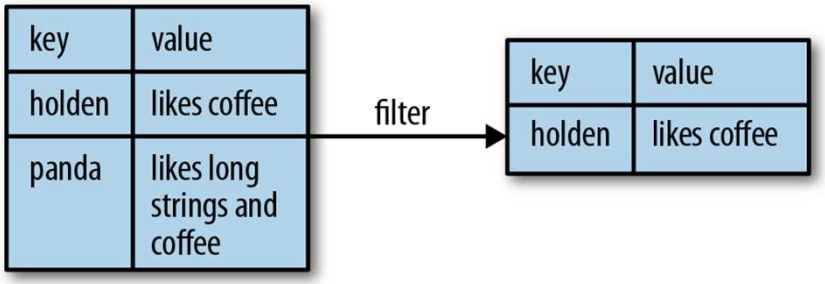
RDD Type

- Basic RDD[T]
  - considers each data item as a single value
- Convert to other RDD Type
- PairRDD[K,V]
  - each data item containing key/value pairs.
- DoubleRDD
  - Data items are convertible to the Scaladata-type double

http://blog.csdn.net/pelick/article/details/44922619
http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html

PairRDD Transformation

- PairRDD is also a RDD
  - RDD.filter(func)
  - RDD.map(func)
  - RDD.flatMap(func)
  - …




```
Function<Tuple2<String, String>, Boolean> longWordFilter =
  new Function<Tuple2<String, String>, Boolean>() {
    public Boolean call(Tuple2<String, String> keyValue) {
      return (keyValue._2().length() < 20);
    }
  };
JavaPairRDD<String, String> result = pairs.filter(longWordFilter);
```

PairRDD Transformation

- RDD.mapValues(func)

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`mapValues(func)`	Apply a function to each value of a pair RDD without changing the key.	`rdd.mapValues(x => x+1)`	`{(1, 3), (3, 5), (3, 7)}`

```
JavaPairRDD<Integer,Integer> result =
        prdd.mapValues(new Function<Integer, Integer>() {
            @Override
            public Integer call(Integer v1) throws Exception {
            return v1 +1;
        }
});
```

PairRDD Transformation

- RDD.reduceByKey(func)

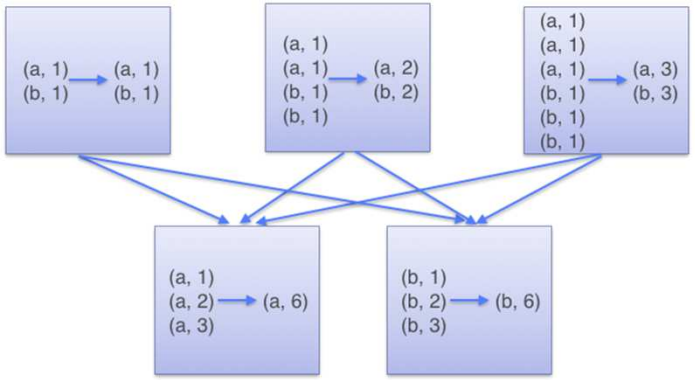


Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`reduceByKey(func)`	Combine values with the same key.	`rdd.reduceByKey( (x, y) => x + y)`	`{(1, 2), (3, 10)}`

```
JavaPairRDD<Integer,Integer> result =
        prdd.reduceByKey(new Function2<Integer, Integer, Integer>() {
            @Override
            public Integer call(Integer v1, Integer v2) throws Exception {
                return v1 + v2;
            }
        });
```

PairRDD Transformation

- RDD.groupByKey()

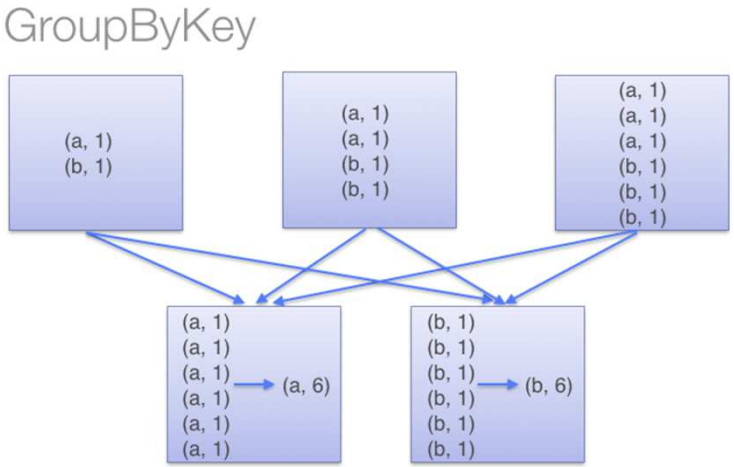
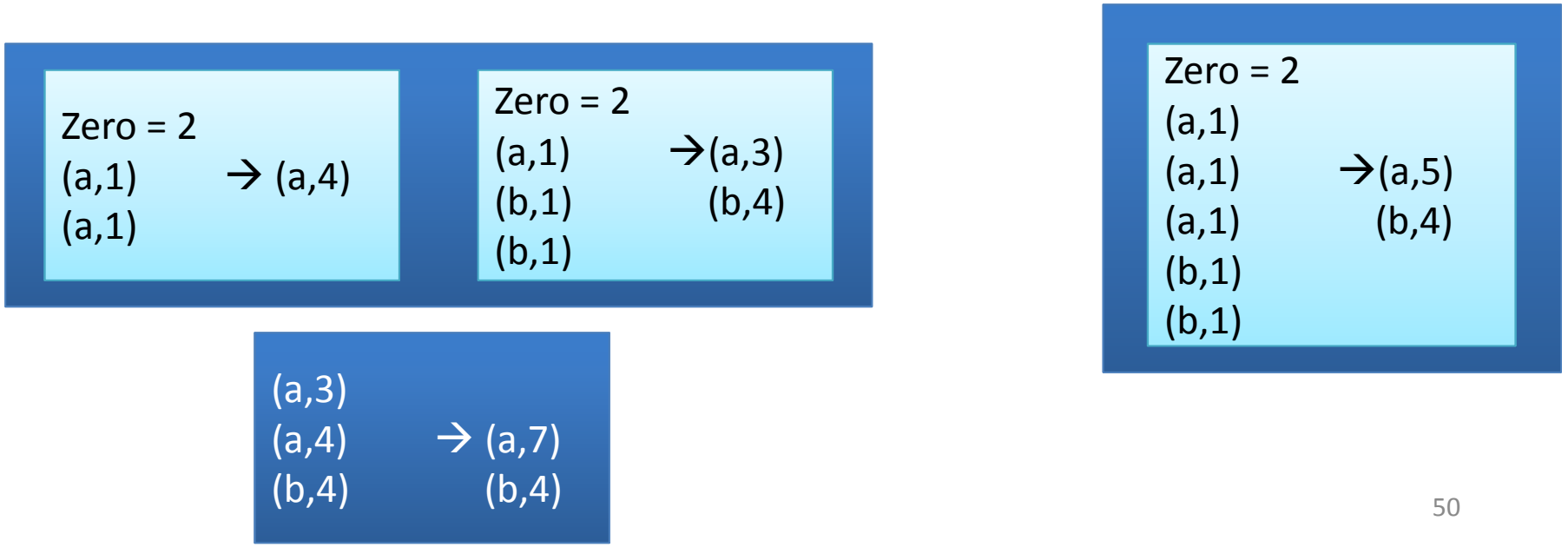


Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`groupByKey()`	Group values with the same key.	`rdd.groupByKey()`	`{(1, [2]), (3, [4, 6])}`

PairRDD Transformation

- RDD.foldByKey(zero)(func)
  - reduceByKey()的一般式
  - Zero value會運用在每個partition同key的值



PairRDD Action

- RDD.countByKey()

Table 4-3. Actions on pair RDDs (example ({(1, 2), (3, 4), (3, 6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key.	`rdd.countByKey()`	`{(1, 1), (3, 2)}`

Exercise ii-5:
Pair RDD

- scala> var rdd1 = sc.makeRDD(Array(("A",0),("A",2),("B",1),("B",2),("C",1)))
- scala > rdd1.reduceByKey(_ + _).collect
- scala > rdd1.groupByKey()
- scala > rdd1.countByKey()
- scala> rdd1.foldByKey(0)(_+_).collect
- scala> rdd1.foldByKey(2)(_+_).collect

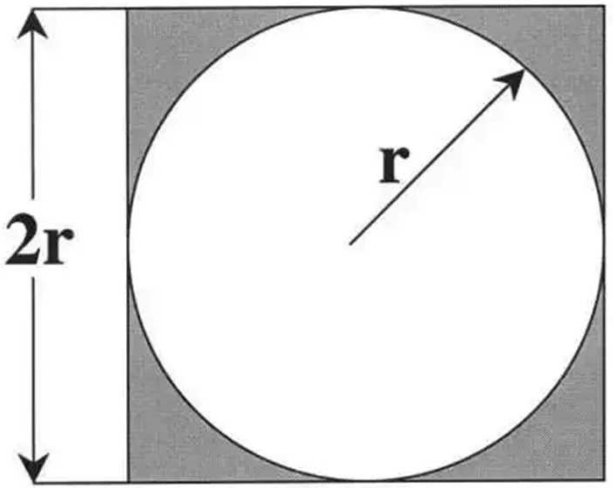
- scala> var rdd1 = sc.makeRDD(Array(("A","a"),("A","b"),("A","c"),("B","a"),("B","b")),3)
- scala> rdd1.foldByKey("0")(_+_).collect
- scala> rdd1.mapPartitionsWithIndex((index: Int, it: Iterator[(String,String)]) =>it.toList.map(x => if (index ==**0**) {println(x)}).iterator).collect

Spark application

- Pi Estimation
- Word Count
- K-means
- Apache Log analysis

Pi Estimation

- 在單位方型內隨機生成資料點。
  - mapToPair(…)
- 判斷那些點在單位圓內，那些點在單位圓外面。
  - filter(…)
- 統計在單位圓內點的個數
  - count()
- Pi = 4*圓內點總數/全部點數



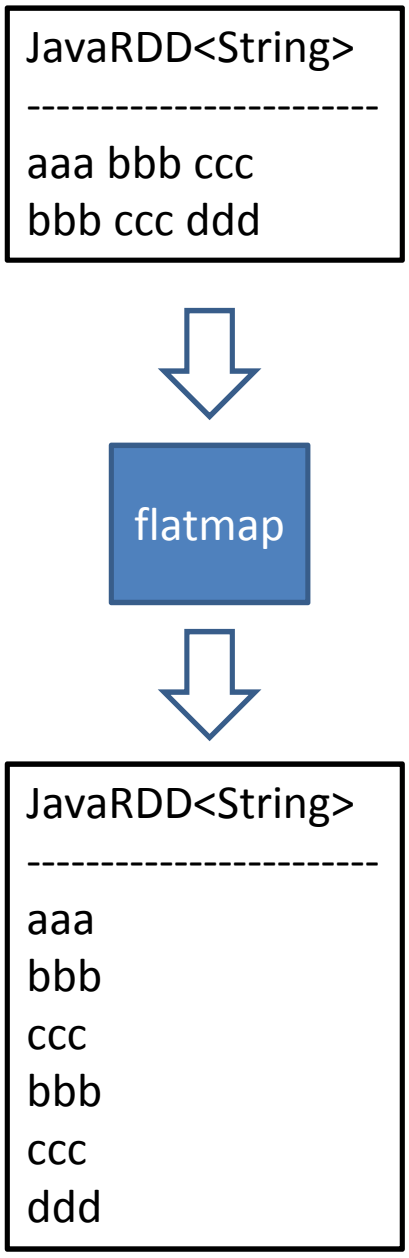
$$\frac{Area\ of\ Circle}{Area\ of\ Square} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

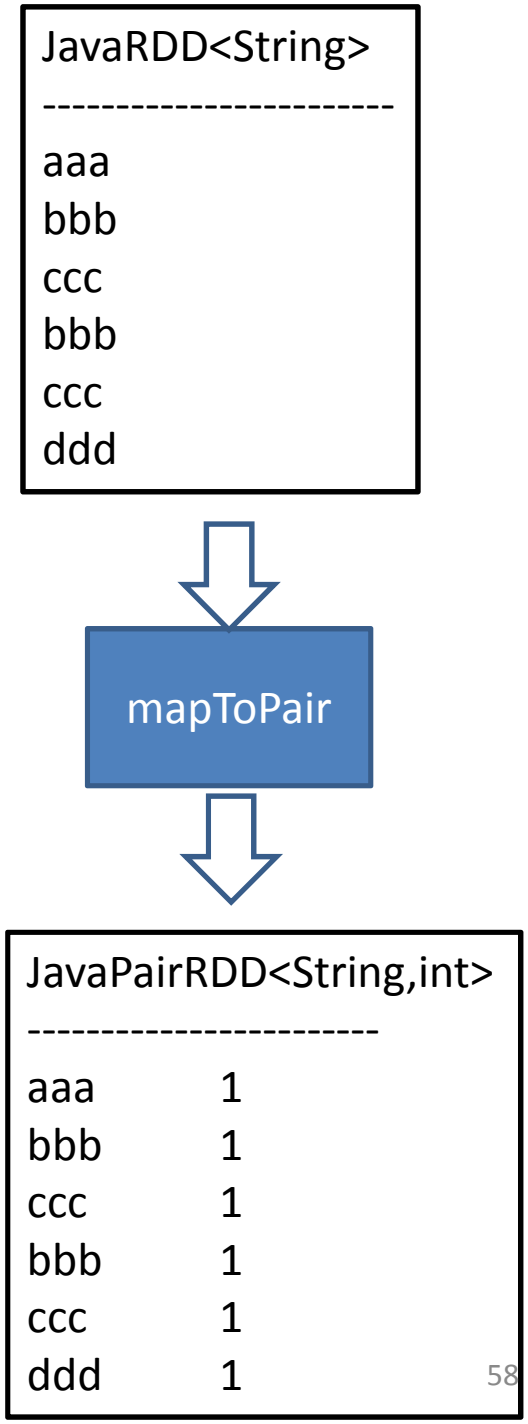
Exercise ii-6:
Pi Estimation

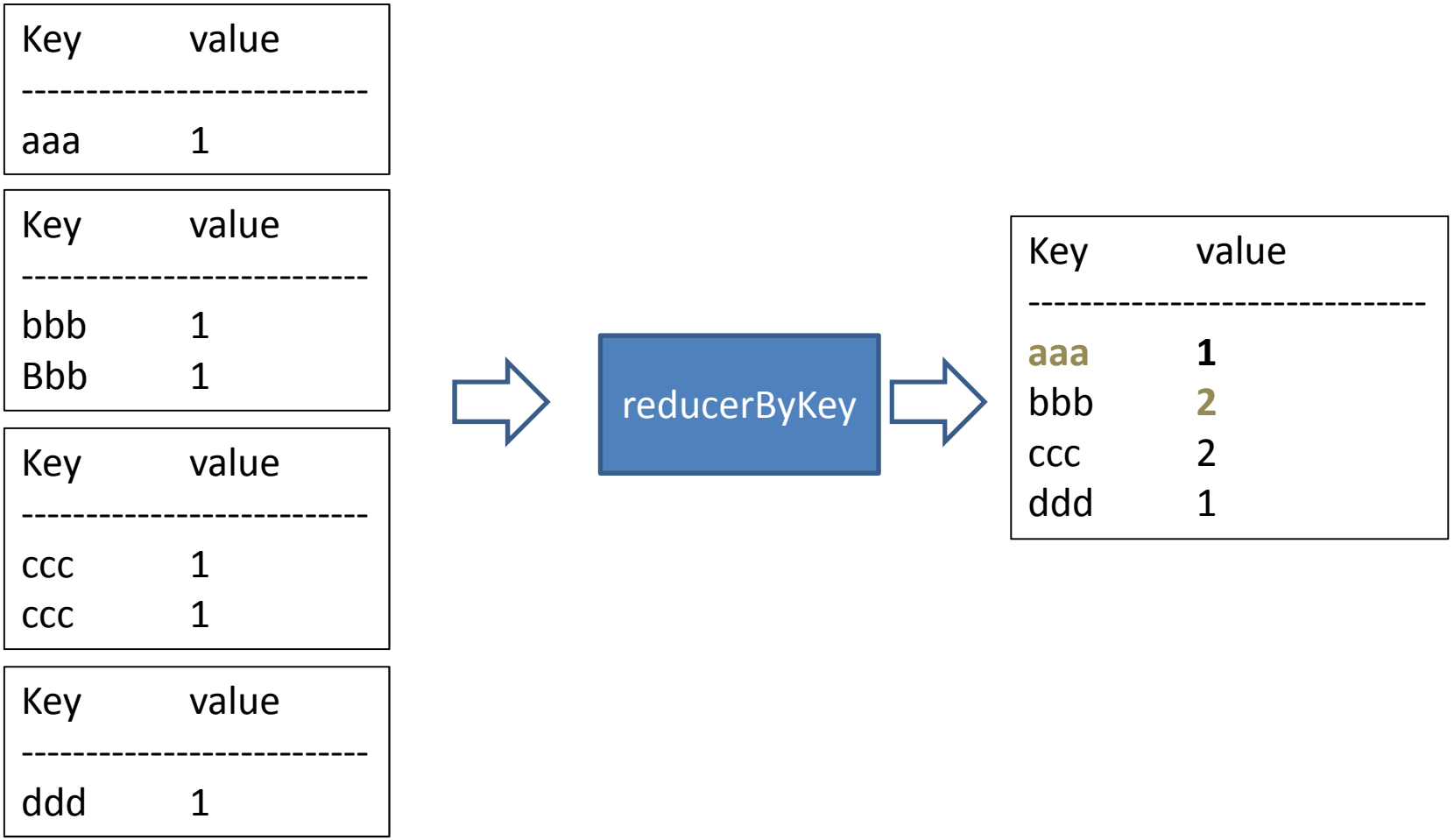
- 分別用Function class和expression完成
  - mapToPair(…)
  - filter(…)

MR convert to Spark

- Map() in MR
  - flatmap() + map()/mapToPair()
- Reduce() in MR
  - reduceByKey()
  - foldByKey() (reduceByKey() 的一般化)
  - groupByKey() + mapValue() (效能差，不建議使用)


JavaRDD<String>
aaa bbb ccc
bbb ccc ddd
flatmap
JavaRDD<String>
aaa
bbb
ccc
bbb
ccc
ddd


JavaRDD<String>
aaa
bbb
ccc
bbb
ccc
ddd
mapToPair
JavaPairRDD<String,int>
aaa 1
bbb 1
ccc 1
bbb 1
ccc 1
ddd 1


Key value
aaa 1
Key value
bbb 1
Bbb 1
Key value
ccc 1
ccc 1
Key value
ddd 1
reducerByKey
Key value
aaa 1
bbb 2
ccc 2
ddd 1

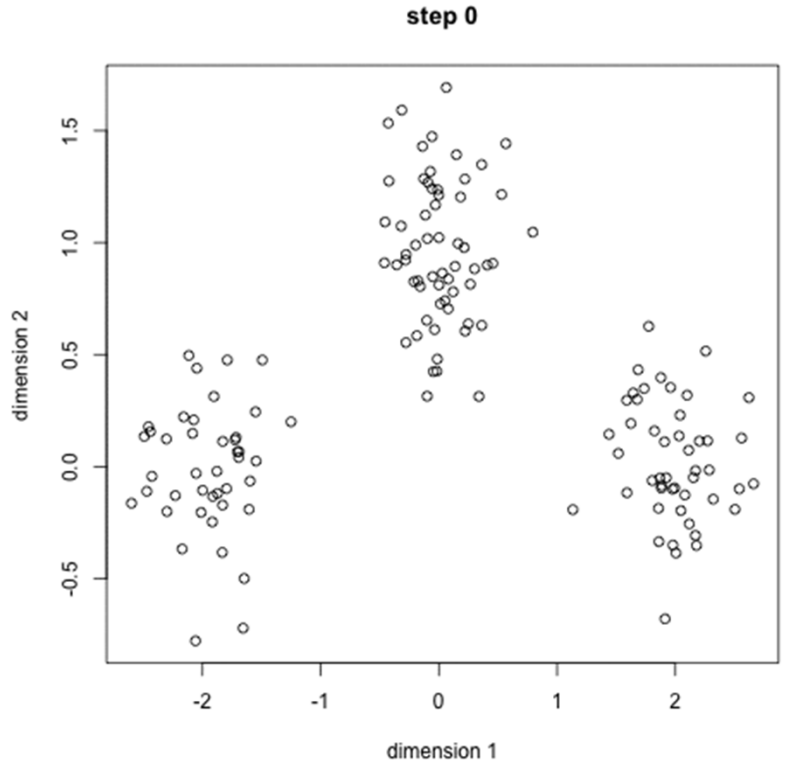
Exercise ii-7:
Word Count

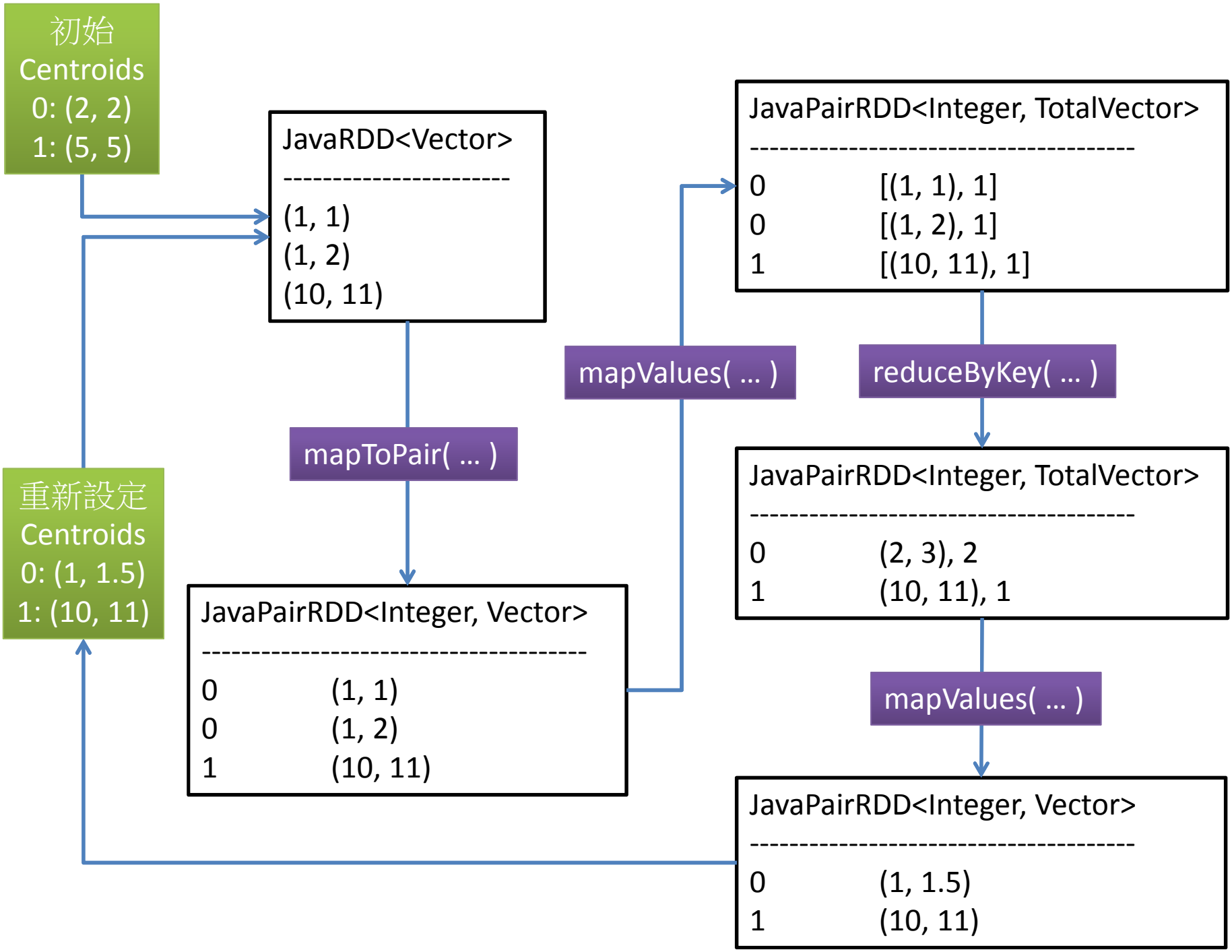
- 用 foldByKey(…)改寫 word count

- 用 groupByKey() + mapValue(…) 改寫 word count

K-means

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




初始
Centroids
0: (2, 2)
1: (5, 5)
JavaRDD<Vector>
(1, 1)
(1, 2)
(10, 11)
mapToPair(…)
JavaPairRDD<Integer, Vector>
0 (1, 1)
0 (1, 2)
1 (10, 11)
mapValues(…)
JavaPairRDD<Integer, TotalVector>
0 [(1, 1), 1]
0 [(1, 2), 1]
1 [(10, 11), 1]
reduceByKey(…)
JavaPairRDD<Integer, TotalVector>
0 (2, 3), 2
1 (10, 11), 1
mapValues(…)
JavaPairRDD<Integer, Vector>
0 (1, 1.5)
1 (10, 11)
重新設定
Centroids
0: (1, 1.5)
1: (10, 11)

Exercise ii-8:
K-means

- 分群, 計算每個Vector離那一組中心最近
  - JavaPairRDD<Integer, Vector> result1 = data1.mapToPair(…);
- 將PairRDD的Vector value轉成TotalVector方便後續做reduceByKey
  - JavaPairRDD<Integer, TotalVector> result2 = result1.mapValues(…);
- 將所有點的座標加總
  - JavaPairRDD<Integer, TotalVector> result3 = result2.reduceByKey(…);
- 計算所有點的幾何中心, 找出新的centroid 座標
  - JavaPairRDD<Integer, Vector> result4 = result3.mapValues(…);

Exercise ii-9: Apache Log analysis

- Format
  - 64.242.88.10 - - [07/Mar/2004:16:47:12 -0800] "GET /robots.txt HTTP/1.1" 200 68
  - 1:IP 2:client 3:user 4:date time 5:method 6:req 7:proto 8:respcode 9:size

- 用parseFromLogLine()將日志字串轉成ApacheAccessLog物件
  - JavaRDD<ApacheAccessLog> result1 = logLines.map(…);

- 將不符合格式的資料篩除
  - JavaRDD<ApacheAccessLog> result2 = result1.filter(…);

- 利用ApacheAccessLog的getContentSizeDouble(), 產生DoubleRDD做敘述統計
  - JavaDoubleRDD contentSizes = accessLogs.mapToDouble(…).cache()

- countByKey()找出各種回傳值總數
  - Map<Integer, Object> responseCodeToCount = accessLogs.
    mapToPair(...).countByKey()

- 篩選出大於threshold的ip
  - JavaPairRDD<String, Long> result6 = result5.filter(…);

- 利用map()將JavaPairRDD<String, Long>轉成JavaRDD<String>
  - JavaRDD<String> result7 = result6.map(…);

- 印出RDD中的每個元素
  - result7.foreach(…)

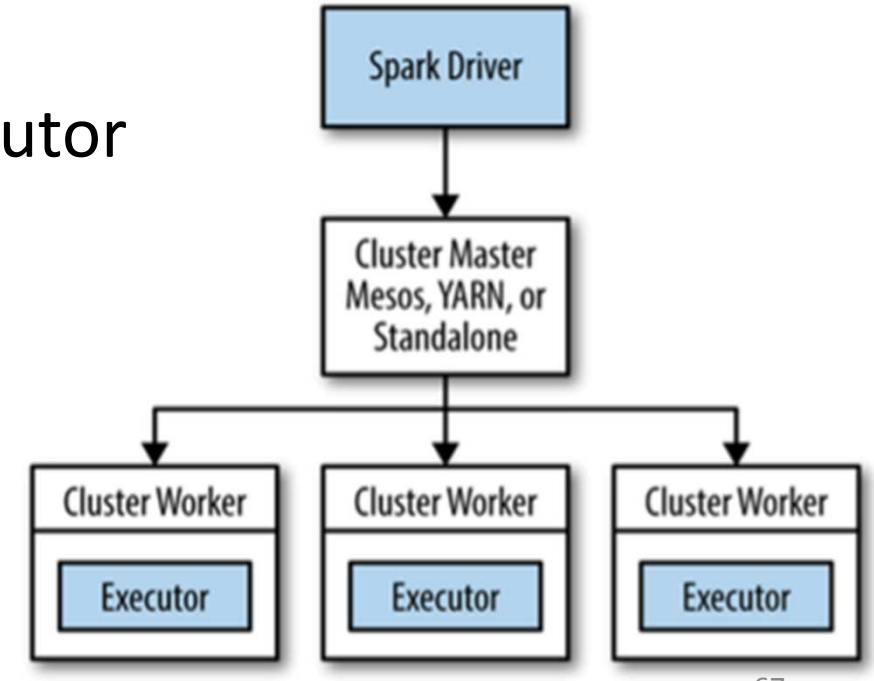
- 利用mapValues()計算groupByKey()結果的總合
  - JavaPairRDD<String, Long> result9 =
    result8.groupByKey().mapValues(…);

Outline

- Part III: Run on Cluster
  - Cluster Manager
    - Standalone, YARN, Mesos
  - Components of Execution
    - Application, Job, stage, task, RDD, partition, DAG
- Part IV: Play with Spark Ecosystem
  - Spark SQL
  - Spark Streaming
  - SparkR

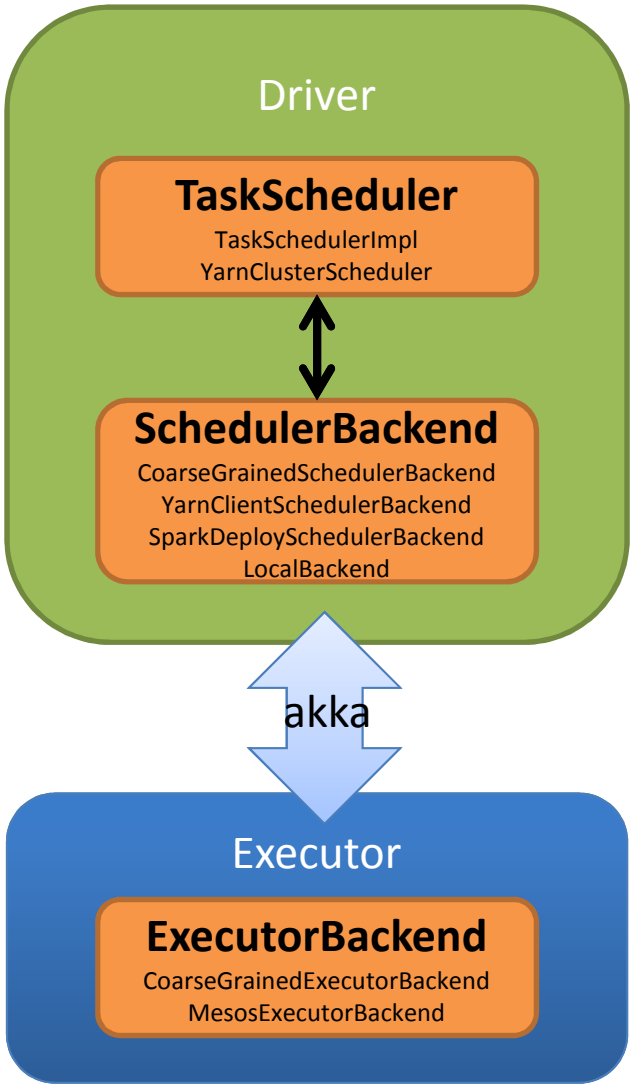
Spark Runtime Architecture

- Driver
  - Process which has the main() method
  - Convert application to tasks
    - DAGScheduler
  - Scheduling tasks on executor
    - TaskScheduler
    - SchedulerBackend
- Executor
  - Running individual task



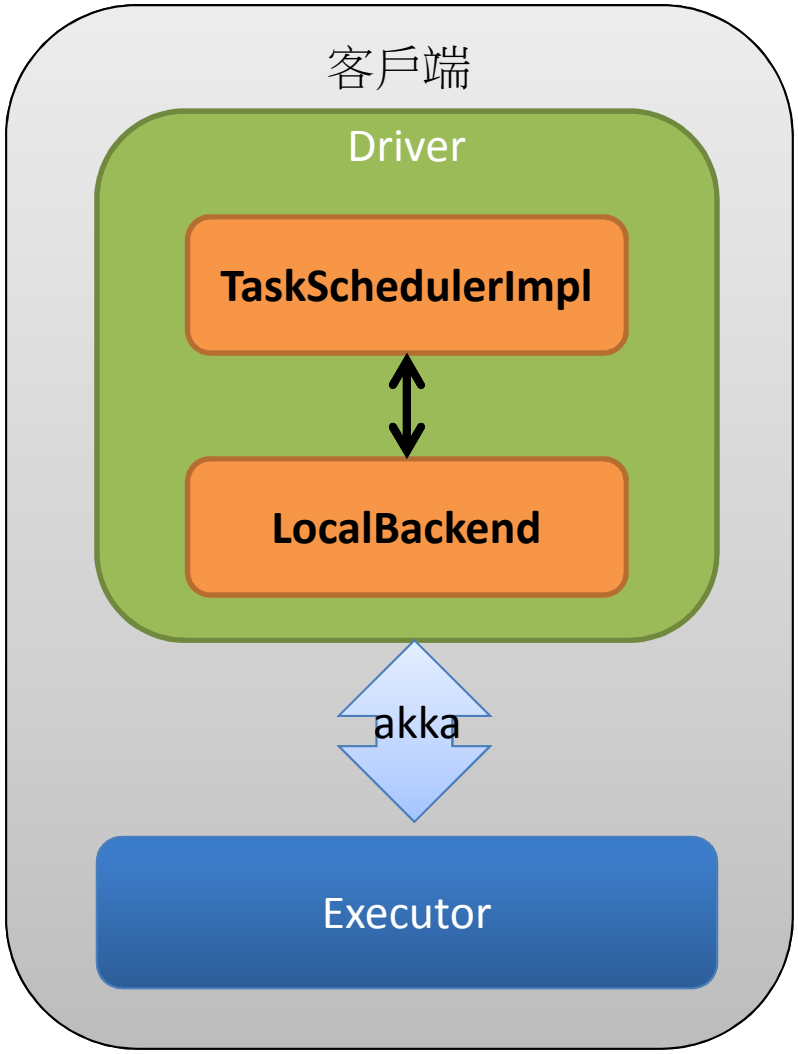
Spark Deployment

- Spark on Local
- Spark on Cluster
  - Spark Standalone
    - Cluster / Client
  - Spark on YARN
    - YARN-Cluster / YARN-Client
  - Spark on Mesos
    - Coarse grained
    - Find grained



Local mode

- local
  - local mode with 1 core
- local[N]
  - local mode with N core
- local[*]
  - Local mode with as many cores the node has

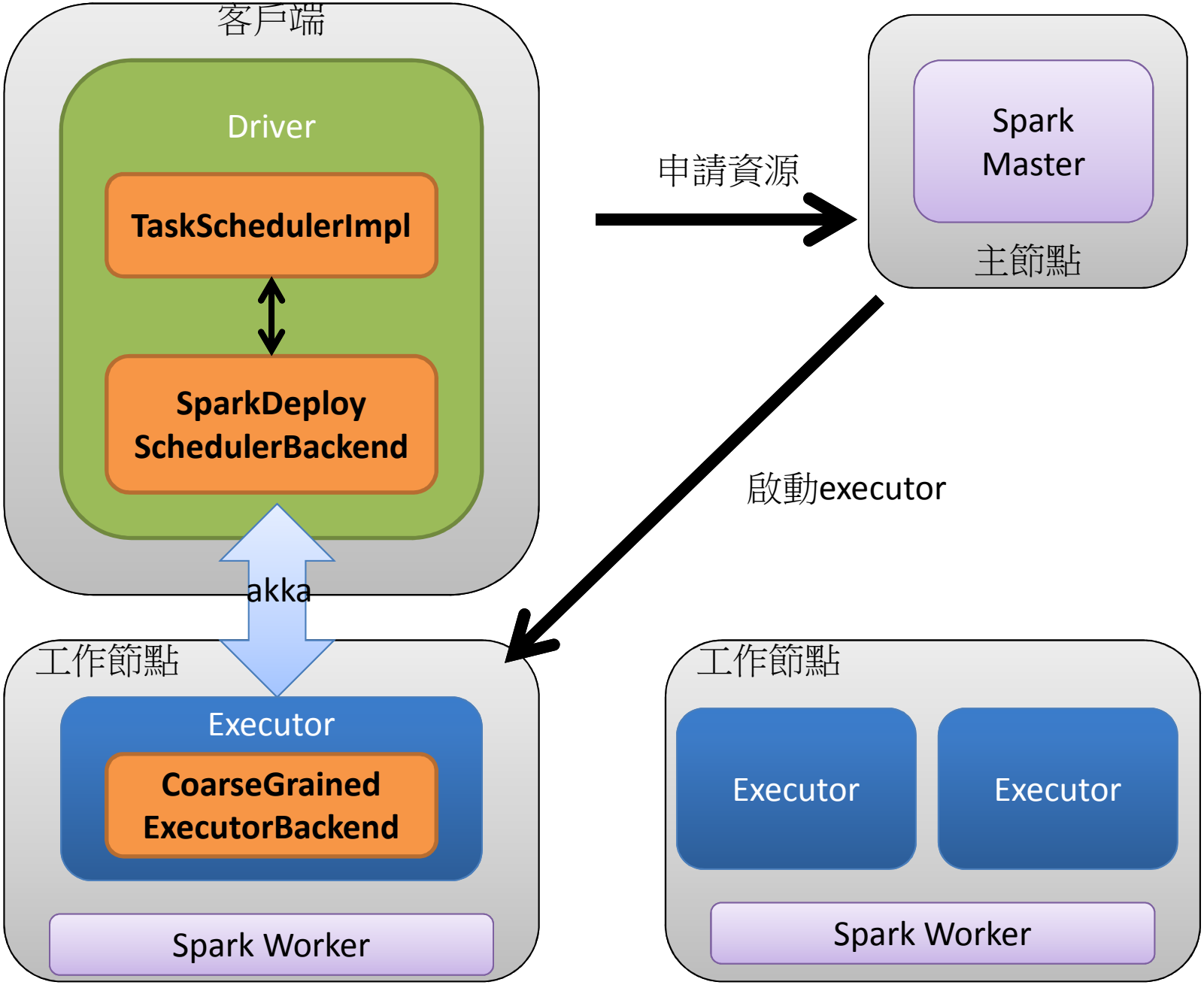


Exercise iii-1:
use different mode

- $ ./spark-shell --master local
  - **scala>** sc.master
  - **scala>** sc.isLocal

- $./spark-submit --master local[2] --class xxx.yyy.zzz xyz.jar
  - **DON'T** hard code setMaster() in source

Standalone mode



Exercise iii-2:
Setup Standalone cluster

- Step 1: edit /etc/hosts
- Step 2: Password less login
  - spark@master $ ssh-keygen –t rsa
  - spark@master $ ssh-copy-id spark@spark-master
  - spark@master $ ssh-copy-id spark@spark-slave
- Step 3:
  - Edit $SPARK_HOME/conf/slaves
  - Edit $SPARK_HOME/conf/spark-env.sh (OPTION)
    - SPARK_WORKER_MEMORY=512m
- Step 4: distribute $SPARK_HOME to all nodes
  - $scp –r sprak spark@spark-slave:/home/spark

- Step 5: Start / Stop Spark Cluster
  - $SPARK_HOME/sbin/start-all.sh
  - $SPARK_HOME/sbin/stop-all.sh
- Step 6: connect to Spark master using **hostname**
  - $ spark-shell --master spark://**spark-master**:7077
  - $ spark-submit --master spark://**spark-master**:7077 --class xx.yy.zz xyz.jar
  - export MASTER=spark://spark-master:7077
- Step 7: submit using cluster mode
  - $ spark-submit --deploy-mode cluster \
    --master XXX --class xx.yy.zz \
    --executor-memory YYYm --driver-memory ZZZm \
    xyz.jar

- Step 8: Setup HDFS (OPTION)
  - Edit $HADOOP_HOME/libexec/hadoop-config.sh
    - export JAVA_HOME=/usr/lib/jvm/java-8-oracle
  - Edit $HADOOP_HOME/etc/hadoop/hadoop-env.sh
    - export JAVA_HOME=/usr/lib/jvm/java-8-oracle
  - Edit $HADOOP_HOME/etc/hadoop/slaves
  - Edit $HADOOP_HOME/etc/hadoop/core-site.xml
  - Edit $HADOOP_HOME/etc/hadoop/hdfs-site.xml
  - Edit ~/.bashrc
  - Start / stop HDFS:
    - start-dfs.sh / stop-dfs.sh
  - Read file from HDFS:
    - sc.textFile(“hdfs://spark-master:9000/path/to/file”)

core-site.xml

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

hdfs-site.xml

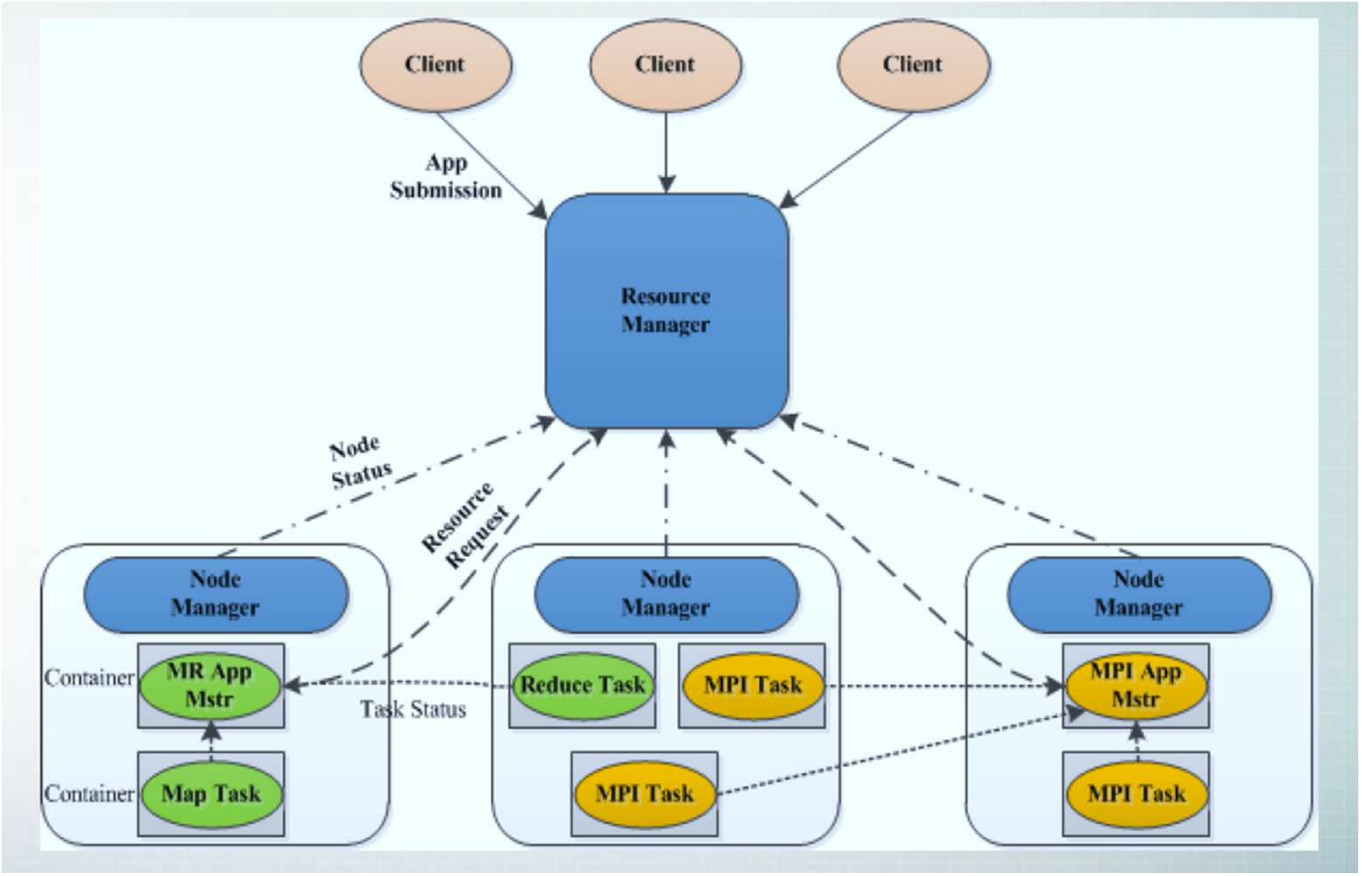
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
 </property>
 <property>
  <name>dfs.permissions</name>
  <value>false</value>
 </property>
</configuration>
```

.bashrc

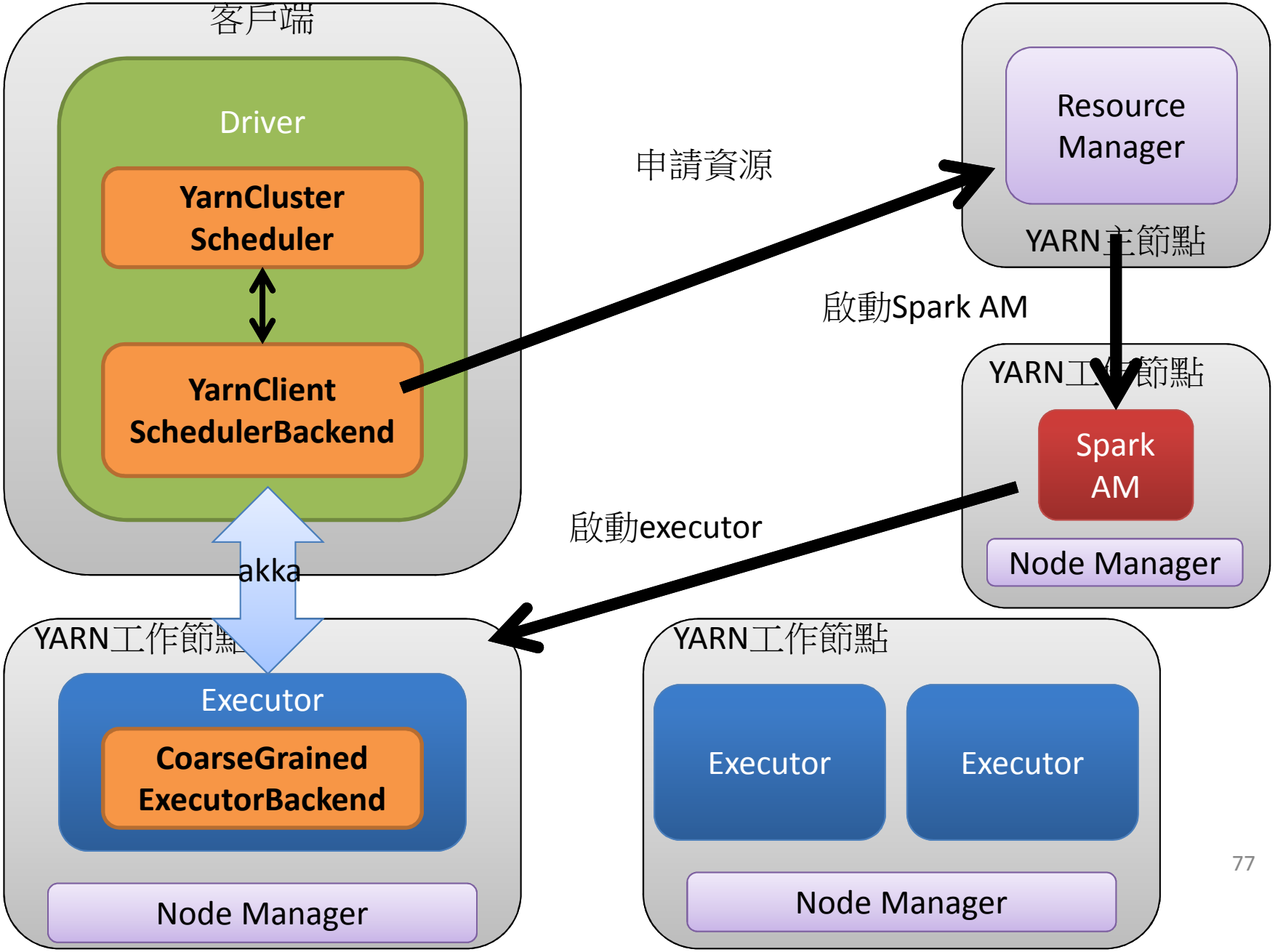
```
export HADOOP_HOME=/home/spark/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop

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

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

Spark on YARN

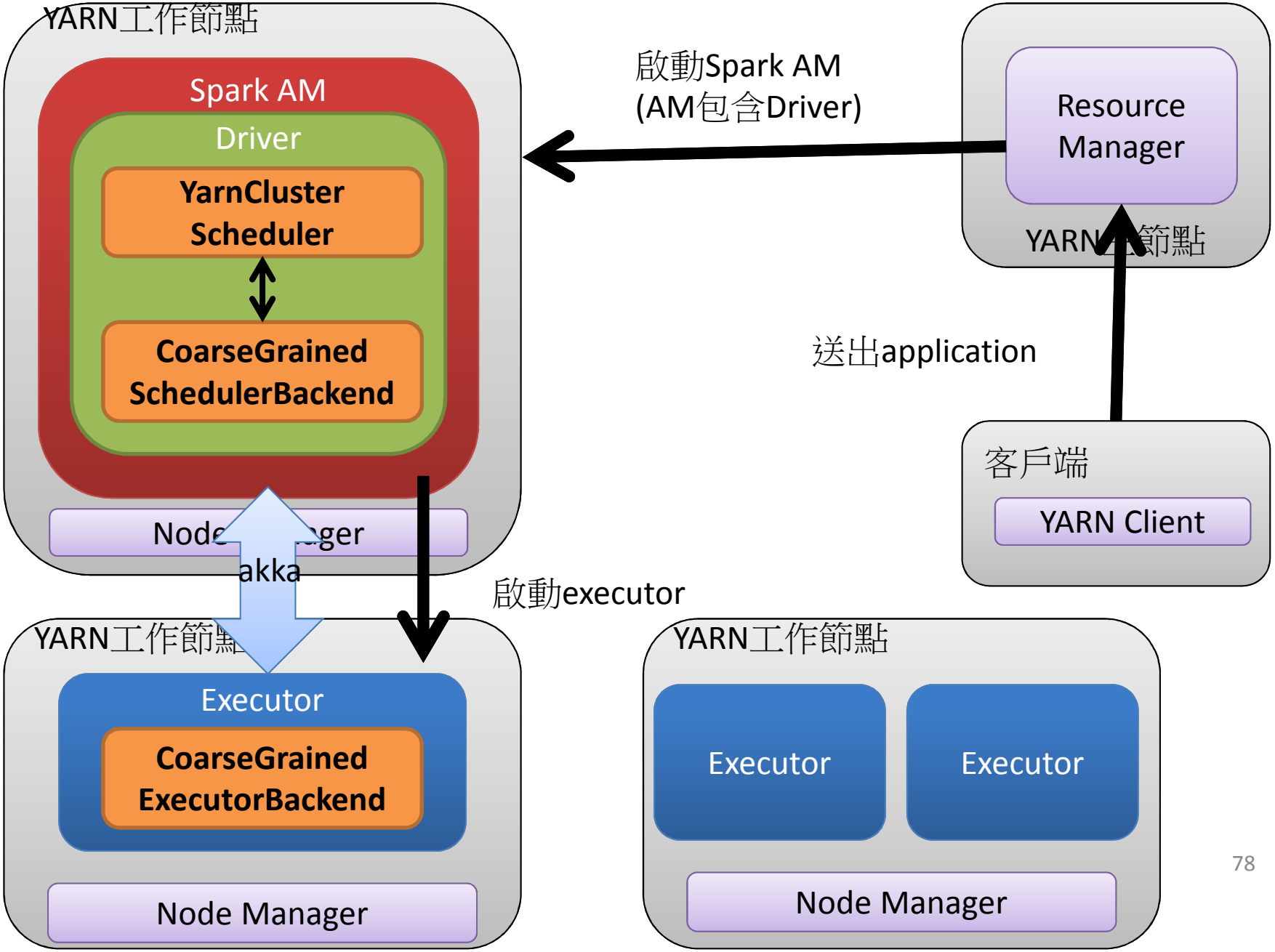
1. 由RM做全局的資源分配
2. NM定時回報目前的資源使用量
3. 每個JOB會有一個負責的AppMaster控制Job
4. 將資源管理與工作控制分開
5. YARN為一通用的資源管理系統
   可達成在YARN上運行多種框架



YARN Client mode



YARN Cluster mode



Exercise iii-3:
Setup Spark on YARN cluster

- Step 1: Start HDFS
- Step 2: Configure YARN
  - Edit $HADOOP_HOME/etc/hadoop/mapred-site.xml
  - Edit $HADOOP_HOME/etc/hadoop/yarn-site.xml
  - Distribute $HADOOP_HOME to all nodes
  - $SPARK_HOME **NO NEEDED** to distribute
- Step 3: Start / Stop YARN Cluster
  - start-yarn.sh / stop-yarn.sh

- Step 4: connect to YARN cluster using **yarn**
  - $ spark-shell --master yarn
  - $ spark-submit **--master yarn** \
    --class xx.yy.zz xyz.jar

- Step 5: using yarn cluster mode
  - $ spark-submit --master yarn \
    --deploy-mode cluster \
    --class xx.yy.zz xyz.jar

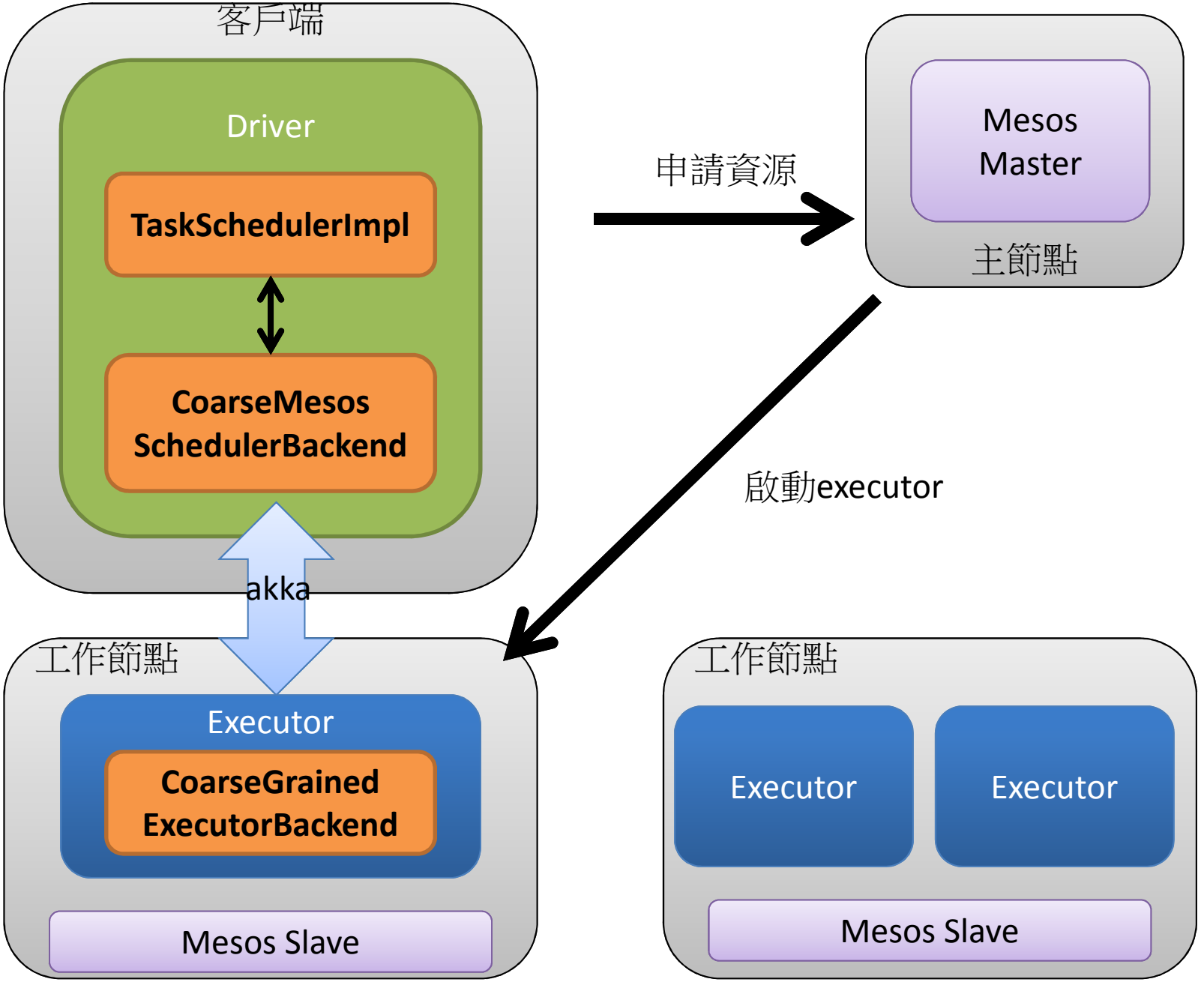
yarn-site.xml

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

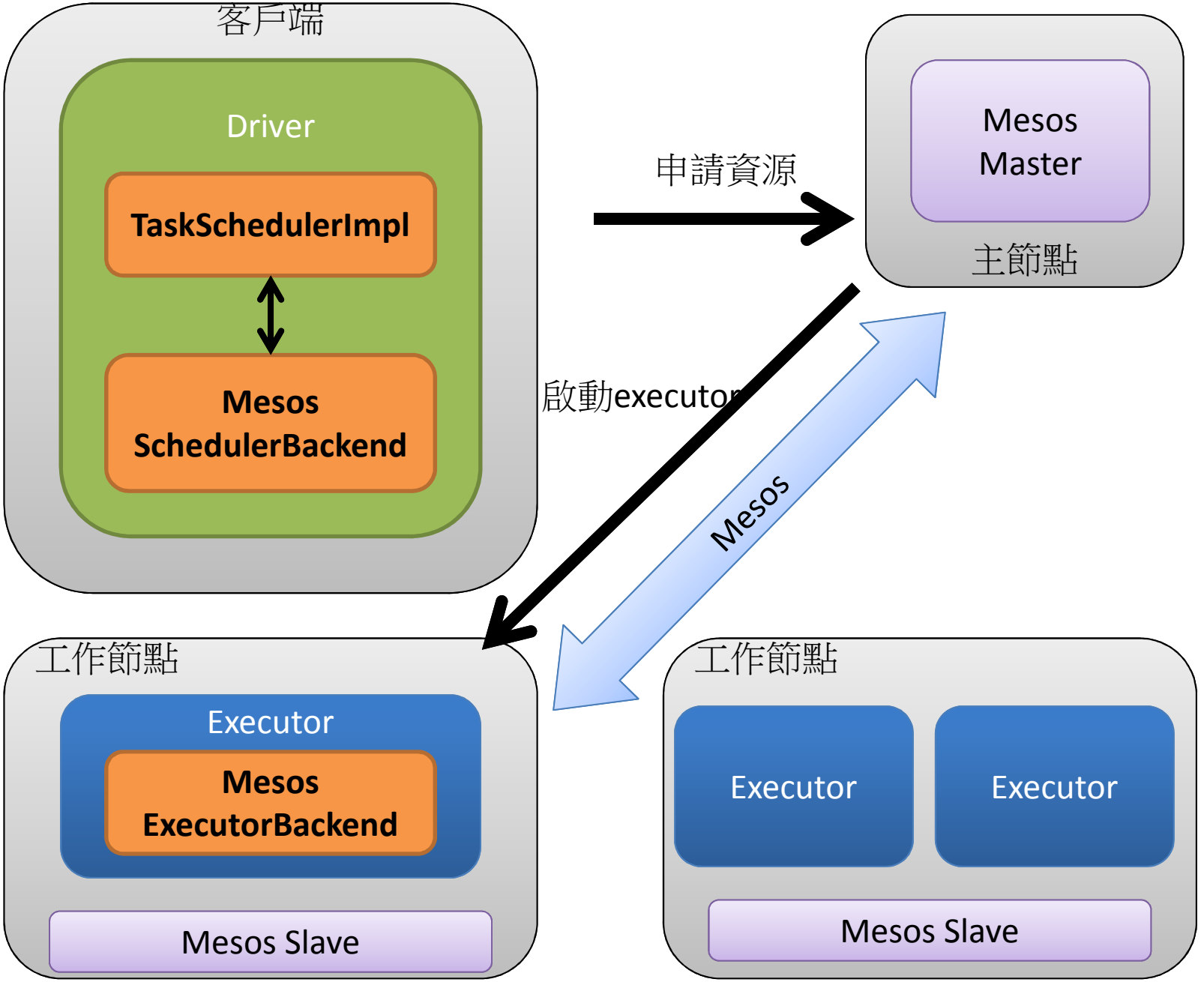
mapred-site.xml

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

Mesos Coarse Grained



Mesos Fine Grained



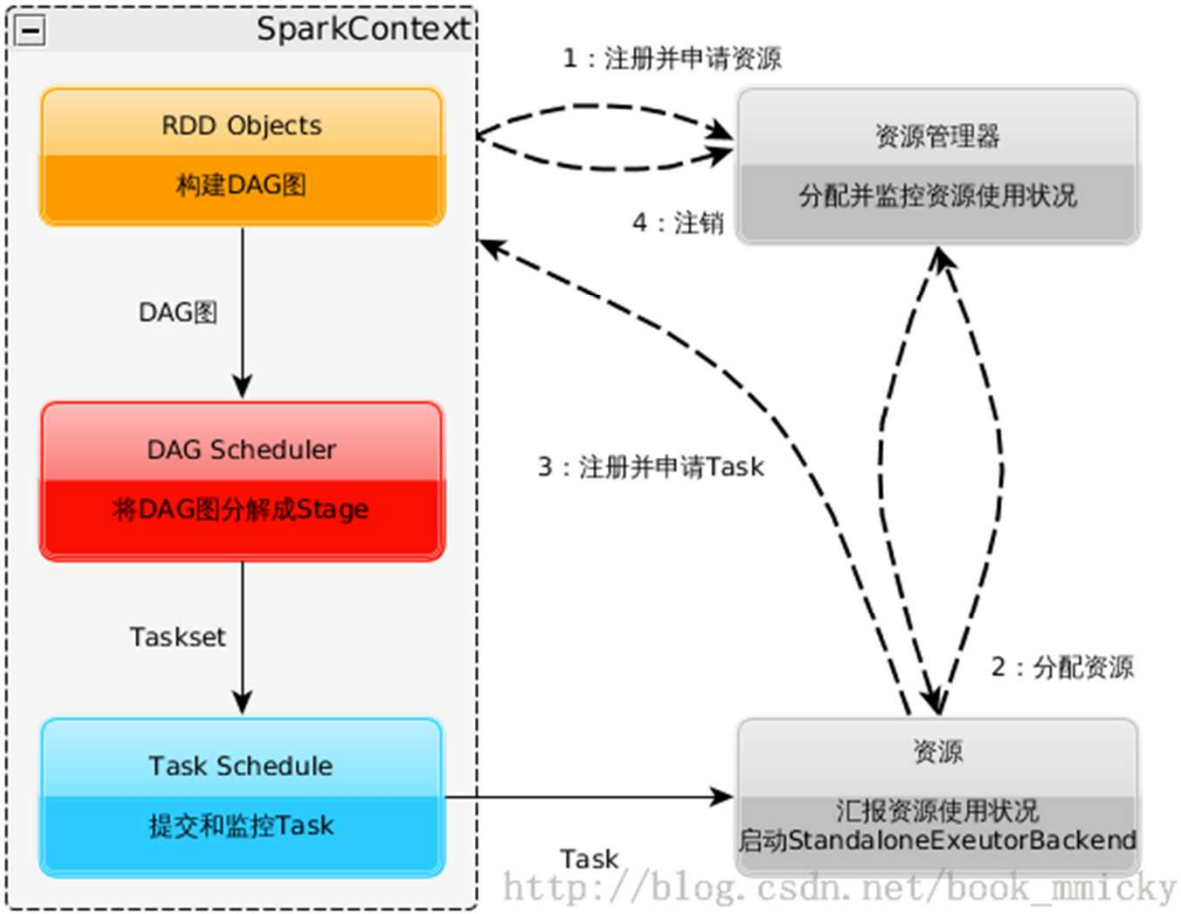
選擇那一種Cluster mode

- 初學者、沒有特殊需求與包袱
  - Standalone mode
- 同時要跑Spark與MapReduce
  - Spark on YARN
- 需要fine grained scheduling
  - Spark on Mesos

Outline

- Part III: Run on Cluster
  - Cluster Manager
    - Standalone, YARN, Mesos
  - Components of Execution
    - Application, Job, stage, task, RDD, partition, DAG
- Part IV: Play with Spark Ecosystem
  - Spark SQL
  - Spark Streaming
  - SparkR

Spark Runtime



http://blog.csdn.net/book_mmicky/article/details/25714419

- 建構Spark Application執行環境(啟動sc)

- SC透過cluster manager (Standalone、Mesos、Yarn)申請Executor資源後啟動ExecutorBackend，executor會向sc註冊。

- **SC利用DAGScheduler生成DAG圖(logic plan)，將DAG圖分解成Stage(physical plan)**，將Taskset發送給Task Scheduler，最後由最Task Scheduler將Task分發給Executor執行。

- Task在Executor上執行完成後，釋放資源。

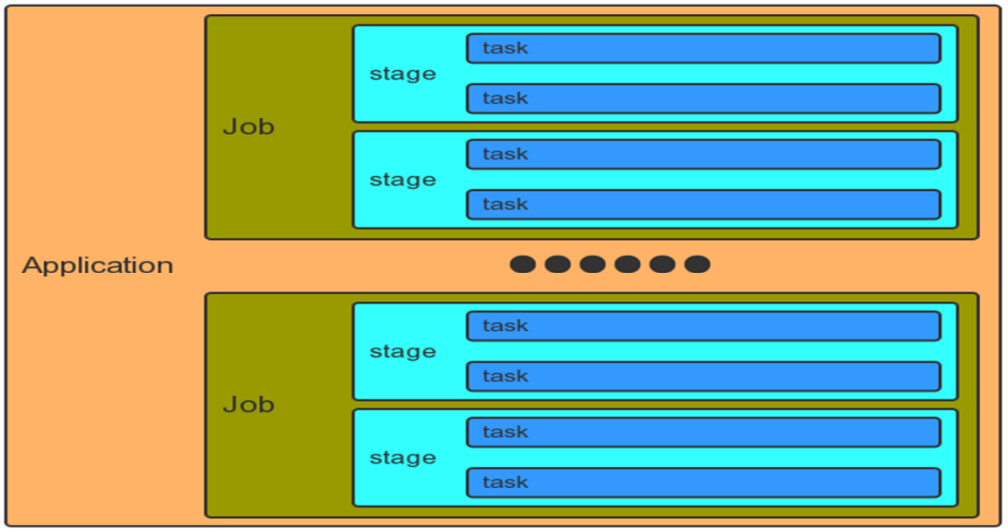
Hadoop application

- 一個Mapreduce application就是一個Job
- 一個Job裡會有多個task
- Task 可分成map task與reduce task

Spark技术博客：http://www.iteblog.com

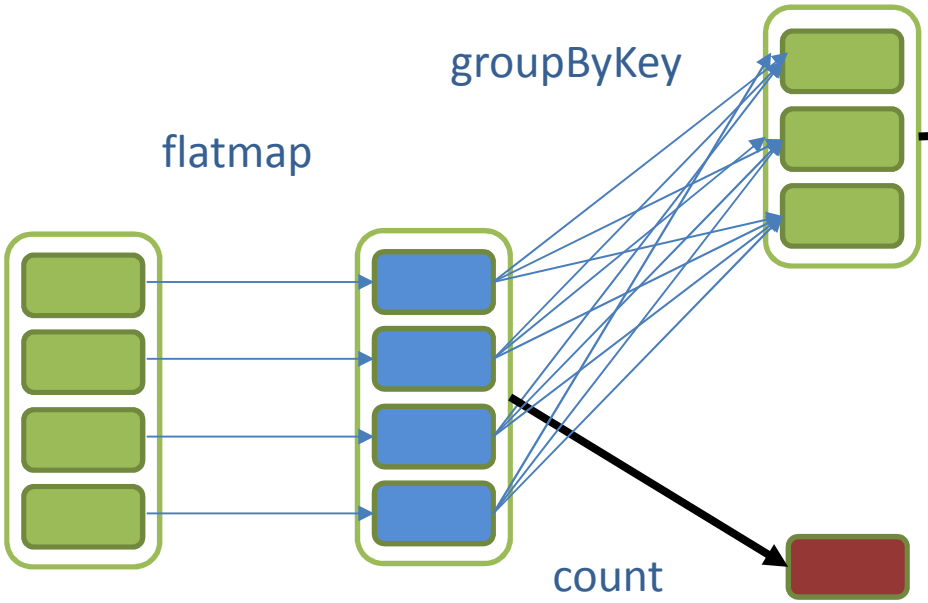
Spark Application

- 一個sparkcontext對應一個Application
- 每個applicatio可有多個job，一個action產生一個job，job可平行或依序執行
- Job由多個stage組成，job裡的stage是由shuffle畫分
- Stage裡有多個task，task數由最後一個RDD的partition決定



How to determine Job

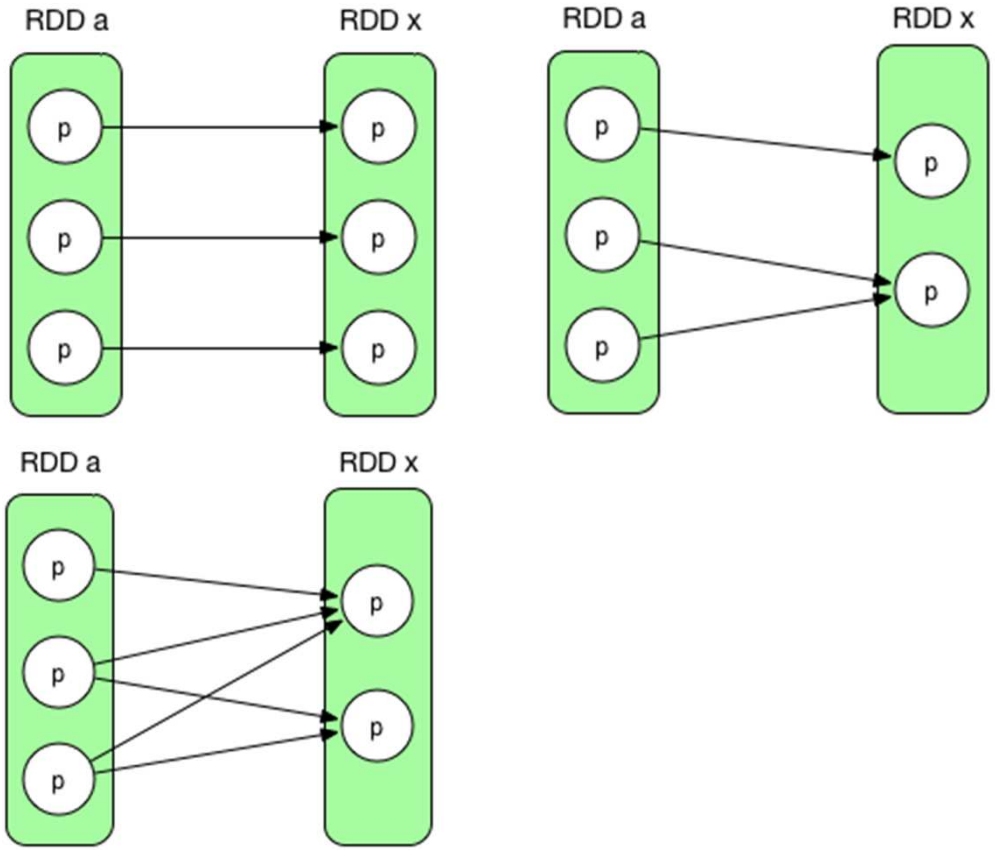
- application 可以包含多個job
- Driver 中每執行一個action，就會產生一個job




```
val pairs1 = sc.parallelize().flatMap {
                p =>  …..
}.cache

pairs1.count

pairs1.groupByKey(numReducers).count
```

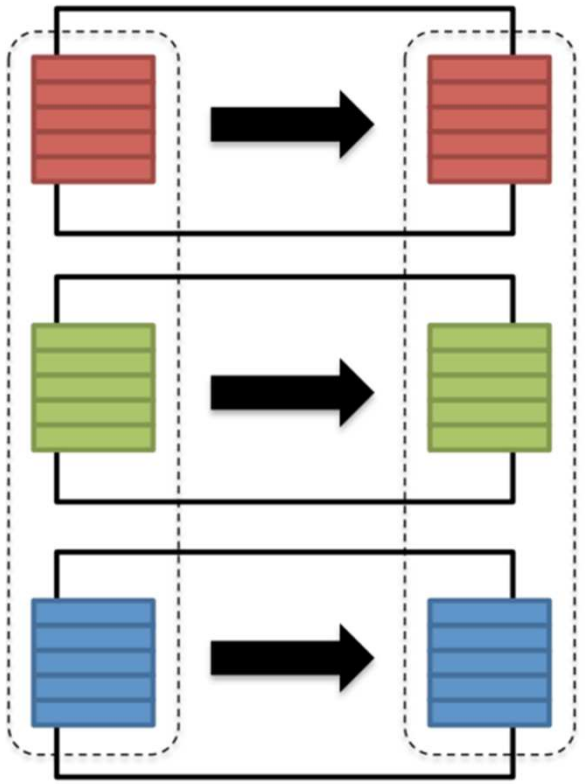
Narrow Dependency

- depends only on data that is already residing in the partition and data shuffling is unnecessary
- Filter()、map()、…



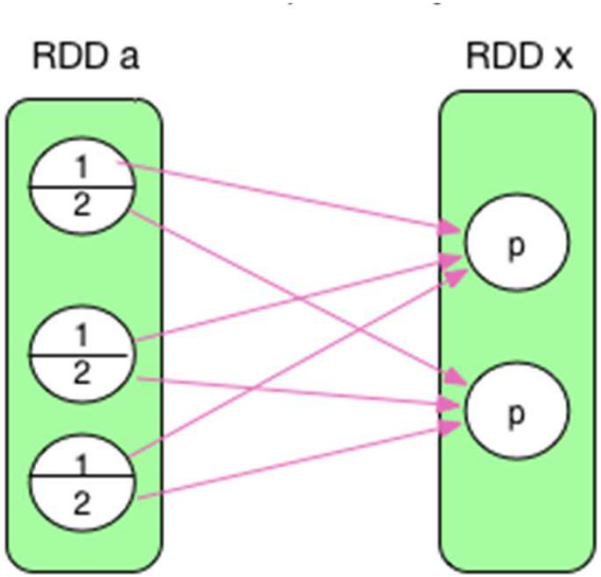
Narrow transformation

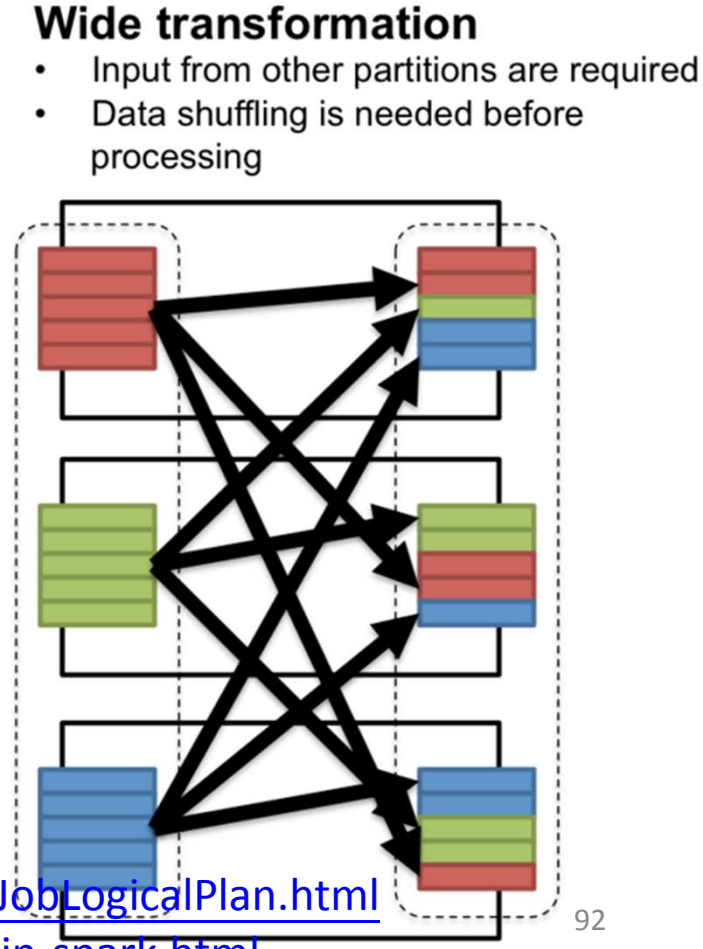
- Input and output stays in same partition
- No data movement is needed



Wide Dependency

- depends on data residing in multiple partitions and therefore data shuffling is needed to bring them together in one place
- groupByKey()、reduceByKey()

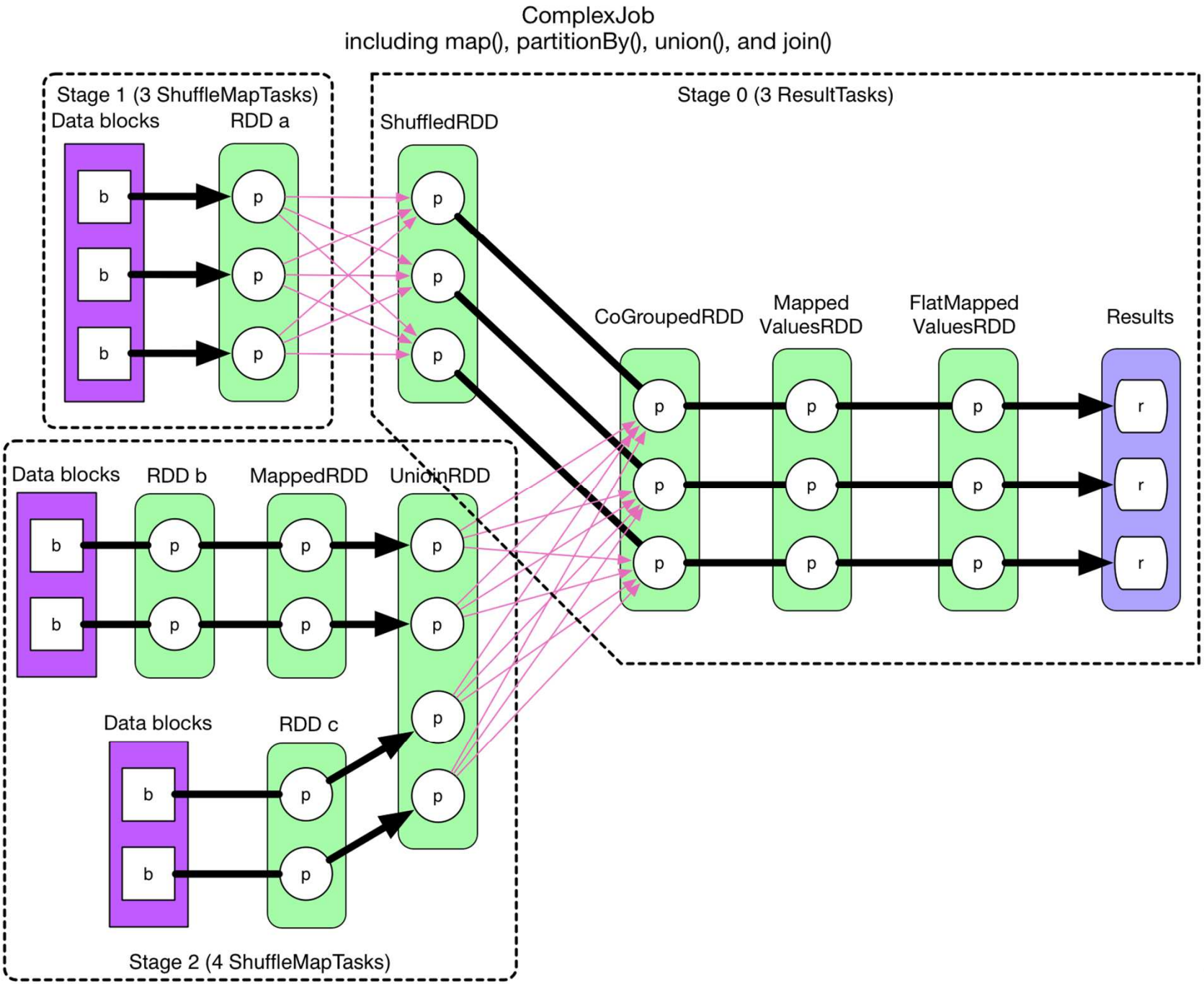




http://spark-internals.books.yourtion.com/markdown/2-JobLogicalPlan.html
http://horicky.blogspot.tw/2015/02/big-data-processing-in-spark.html

How to determine stage and task

- Stage
  - Stages are sequences of RDDs, that don't have a **Shuffle** in between
  - 由後往前推，遇到wideDependency (or Shuffle) 就斷開，遇到NarrowDependency 就加入stage。
- Task
  - 每個Stage裡的 task數目由該 stage最後一個 RDD 中的 partition個數決定。


ComplexJob
including map(), partitionBy(), union(), and join()
Stage 1 (3 ShuffleMapTasks)
Data blocks
RDD a
ShuffledRDD
Stage 0 (3 ResultTasks)
CoGroupedRDD
Mapped ValuesRDD
FlatMapped ValuesRDD
Results
Data blocks
RDD b
MappedRDD
UnionRDD
Data blocks
RDD c
Stage 2 (4 ShuffleMapTasks)

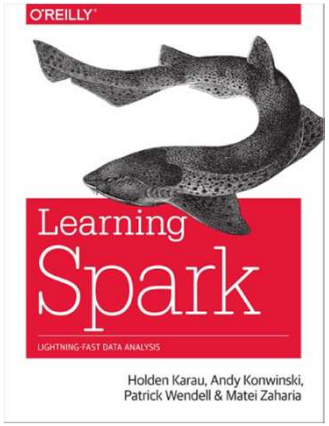
Exercise iii-4 : Setup history server

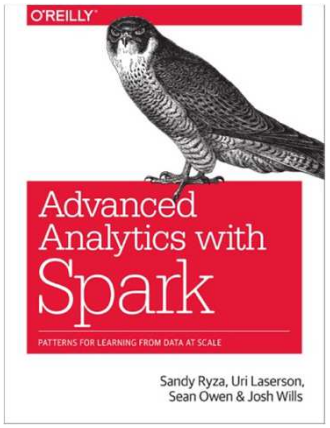
- Start HDFS and create log directory
- Edit $SPARK_HOME/conf/spark-defaults.conf
  - 讓Executor 輸出event log
    - spark.eventLog.enabled true
    - spark.eventLog.dir hdfs://spark-master:9000/sparkLogs
  - 設定history server讀取目錄
    - spark.history.fs.logDirectory hdfs://spark-master:9000/sparkLogs
- Distribute spark-defaults.conf to all nodes
- Start history server
  - $SPARK_HOME/sbin/start-history-server.sh

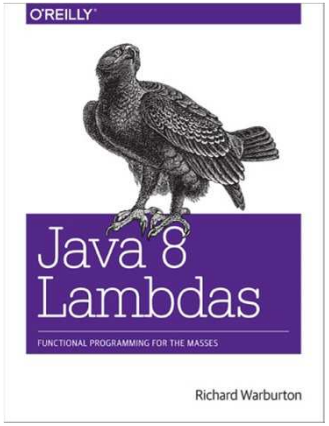
Exercise iii-5 :
觀察Job、Stage、Task in UI

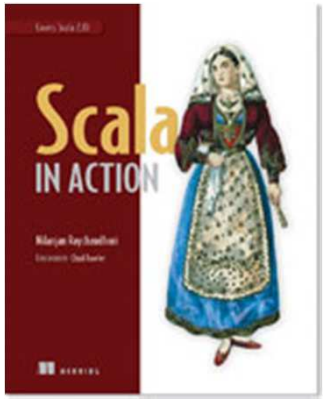
- 分別執行simpleJob與complexJob
- 在job-history-server上驗證job數、stage數、task數

Reference









- Spark
  - Learning Spark
  - Advanced Analytics with Spark
- Java 8
  - Java 8 Lambdas
- Scala
  - Scala in action
  - 为Java程序员编写的Scala的入门教程
    - http://www.iteblog.com/archives/1325