Spark建置與開發實務

國網中心核心技術組
莊家雋

Outline

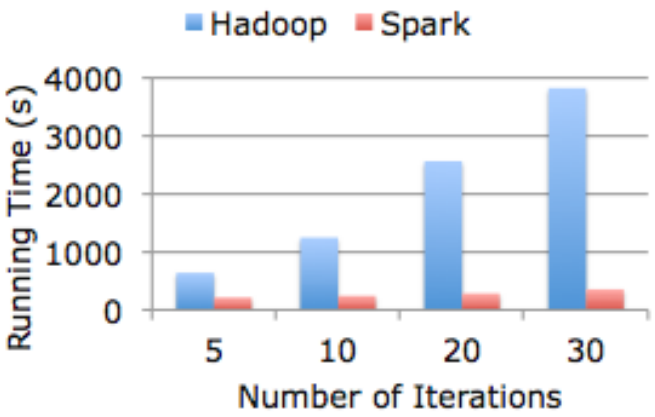
- Part I : Quick tour of Spark
  - Why Spark
  - Prepare Lab VMs
  - Run first Spark example
    - {Java / Python / Scala } code via {Shell / Standalone application }
    - Introduction to FP, Java 8, and Scala
- Part II : Programming with RDD
  - RDD Concepts
  - RDD, RDD API, convert RDD
  - Load and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

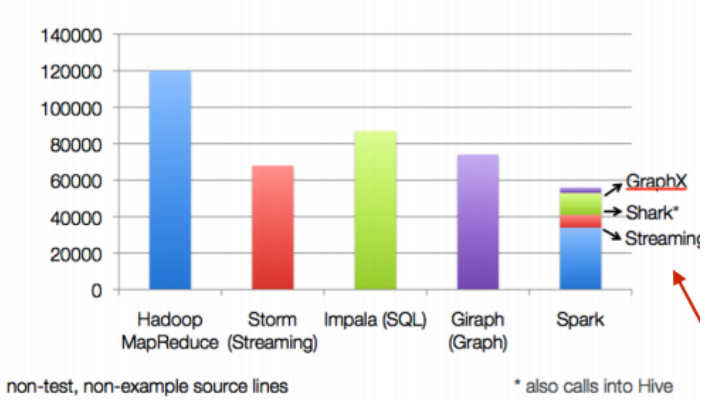
Outline

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

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*)

Spark 版本演進

Version	Original release date	Latest version	Release date
0.5	2012-06-12	0.5.1	2012-10-07
0.6	2012-10-14	0.6.2	2013-02-07[28]
0.7	2013-02-27	0.7.3	2013-07-16
0.8	2013-09-25	0.8.1	2013-12-19
0.9	2014-02-02	0.9.2	2014-07-23
1.0	2014-05-30	1.0.2	2014-08-05
1.1	2014-09-11	1.1.1	2014-11-26
1.2	2014-12-18	1.2.2	2015-04-17
1.3	2015-03-13	1.3.1	2015-04-17
1.4	2015-06-11	1.4.1	2015-07-15
1.5	2015-09-09	1.5.2	2015-11-09
1.6	2016-01-04	1.6.3	2016-11-07
2.0	2016-07-26	2.0.2	2016-11-14
2.1	2016-12-28	2.1.1	2017-05-02
Legend: Old version	Older version, still supported	Latest version	Latest preview version

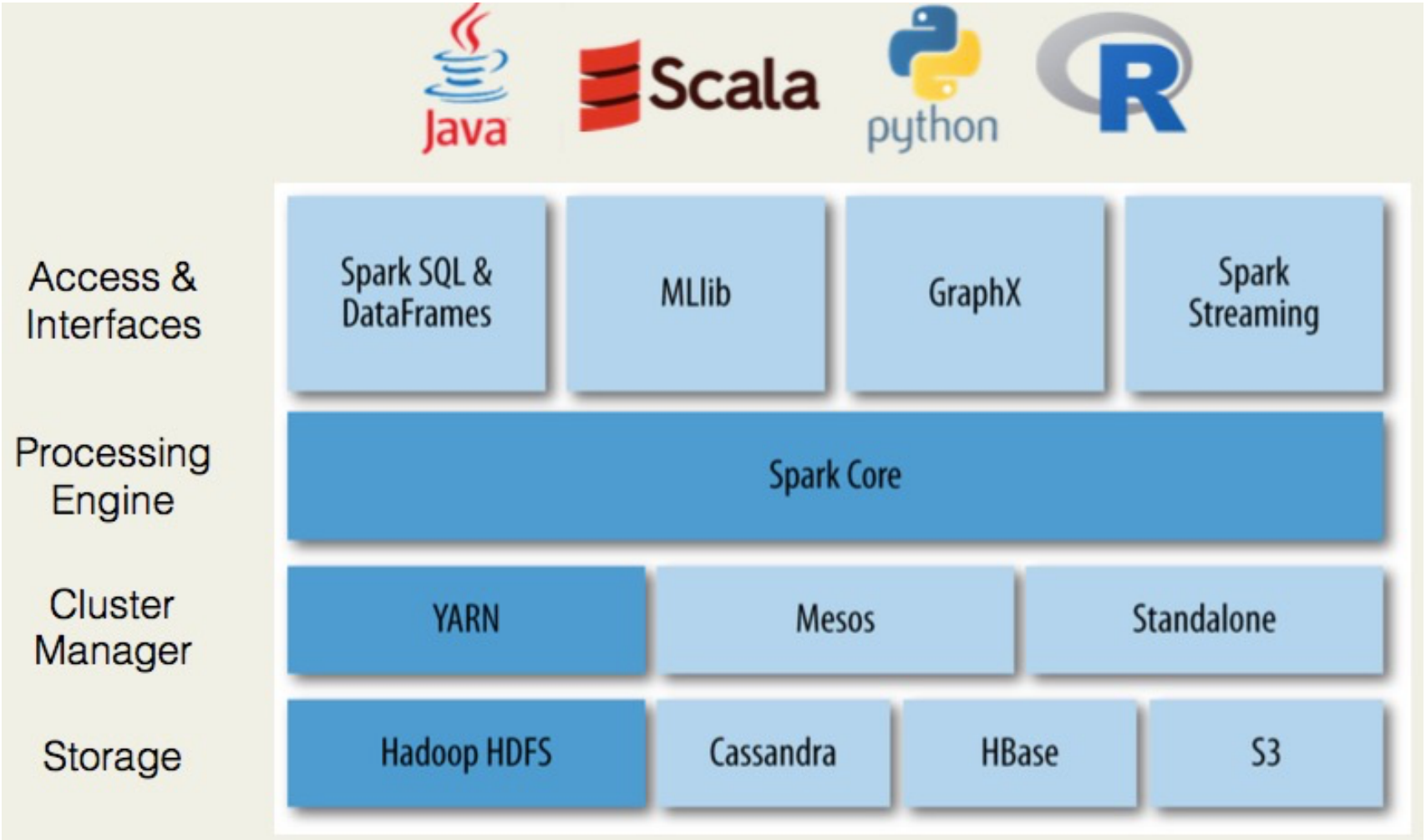
RDD

PySpark

DataFrames

SparkR

Spark Stack


Java
Scala
python
R
Access & Interfaces
Spark SQL & DataFrames
MLlib
GraphX
Spark Streaming
Processing Engine
Spark Core
Cluster Manager
YARN
Mesos
Standalone
Storage
Hadoop HDFS
Cassandra
HBase
S3

Exercise i-1:
Setup Lab Environment

- Lab VM has
  - Ubuntu / Java8 / maven3/ python / spark 2.1.0 / R

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

- 安裝
  - $cp /opt/spark-2.1.0-bin-hadoop2.6.tgz ~
  - $tar zxvf spark-2.1.0-bin-hadoop2.6.tgz
  - $mv spark-2.1.0-bin-hadoop2.6 spark

Exercise i-2:
Try interactive Spark shell

- Launch interactive shell
  - $ ~/spark/bin/spark-shell
  - $ ~/spark/bin/pyspark

- via Juypter (primary interface in this workshop)
  - source ~/py_juypter.profile

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
- Function is passed into RDD operation in Spark

FP in python

```
Help on built-in function filter in module __builtin__:

filter(...)
    filter(function or None, sequence) -> list, tuple, or string

    Return those items of sequence for which function(item) is true.  If
    function is None, return the items that are true.  If sequence is a tuple
    or string, return the same type, else return a list.
```

- Named function
  - i.e. normal function

```
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>> def g(x): return x% 3 ==0
...
>>> print filter(g, foo)
[18, 9, 24, 12, 27]
```

- anonymous function
  - functions that are not bound to a name

```
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>> print filter(lambda x: x%3==0, foo)
[18, 9, 24, 12, 27]
```

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

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

```
$<SPARK_HOME>/bin/spark-submit  \
  ./python/HwlloWorld.py  \
  ./README.md
```

```
$mvn clean package
$<SPARK_HOME>/bin/spark-submit  \
  --class org.nchc.spark.java.sample.i.HelloWorld \
  ./target/spark-sample-0.0.1.jar \
  ./README.md
```

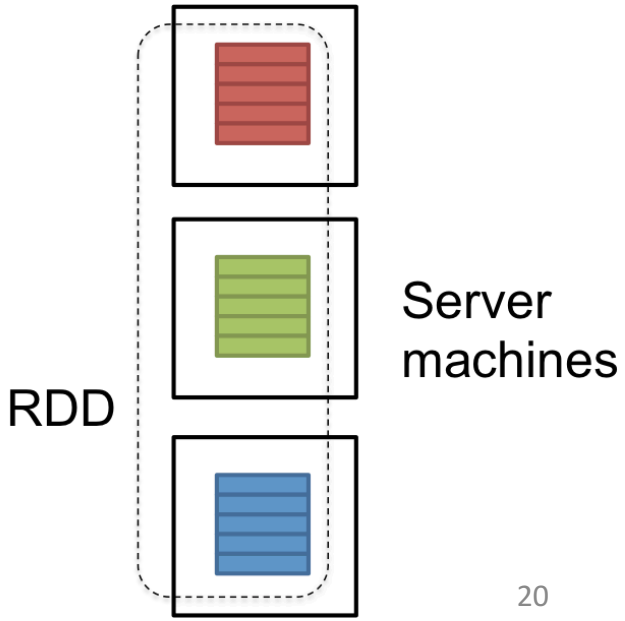
```
$ sbt clean package
$ <SPARK_HOME>/bin/spark-submit  \
   --class org.nchc.spark.scala.HelloWorld \
   ./target/scala-2.10/spark-sample_2.10-0.0.1.jar \
   ./README.md
```

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
  - Coarse-grained transformation
- 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()

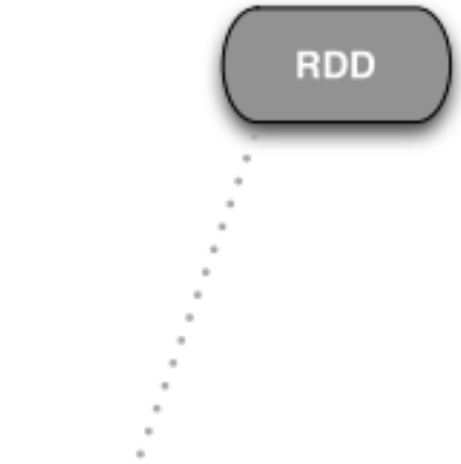
Actions

- 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()

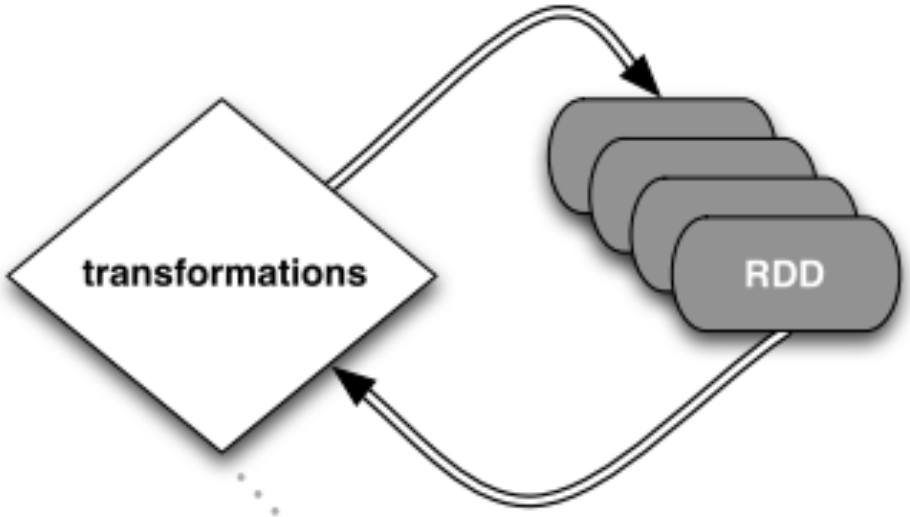
Create RDD




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

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

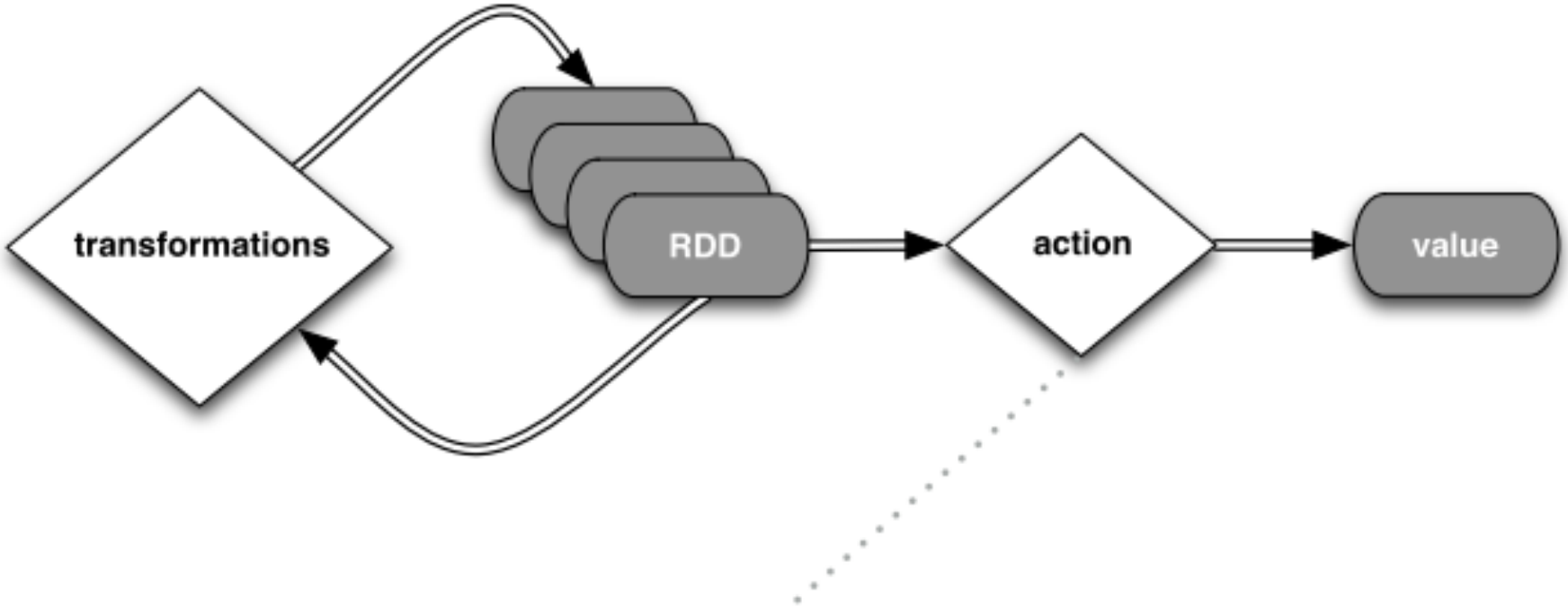
RDD Transformation




```
// 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



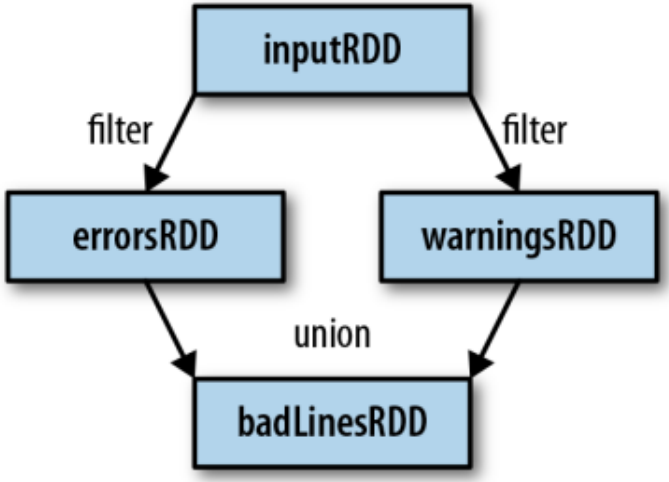

```
// action 1
messages.filter(_.contains("mysql")).count()
```

```
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
  - For iterative algorithm, keep temporary in memory can improve performance
    - The reason for in-memory computing

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)

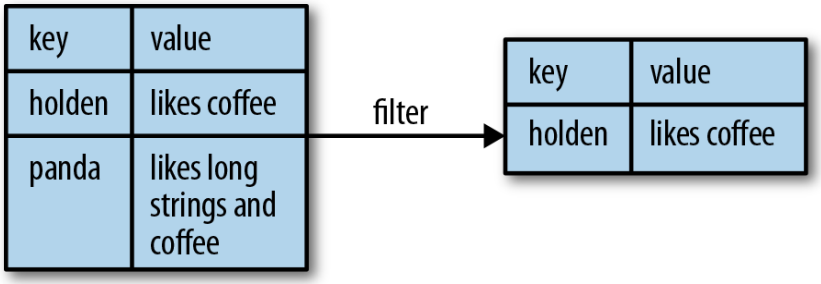
RDD Type

- 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)

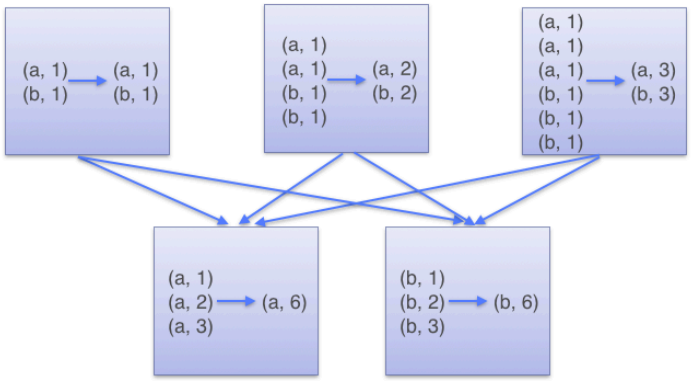


Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

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()

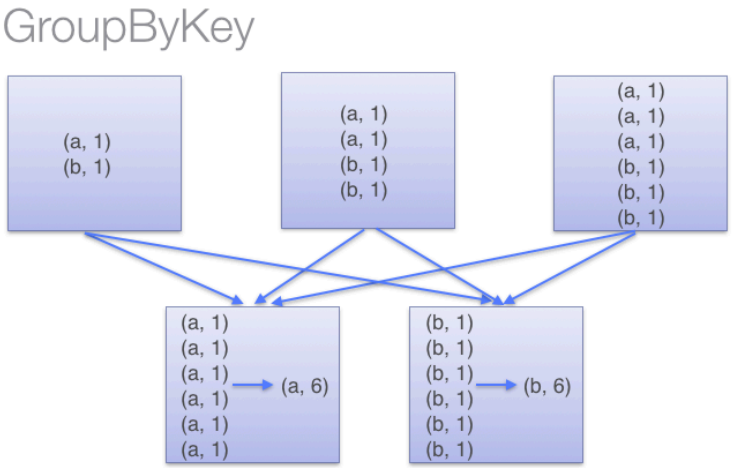
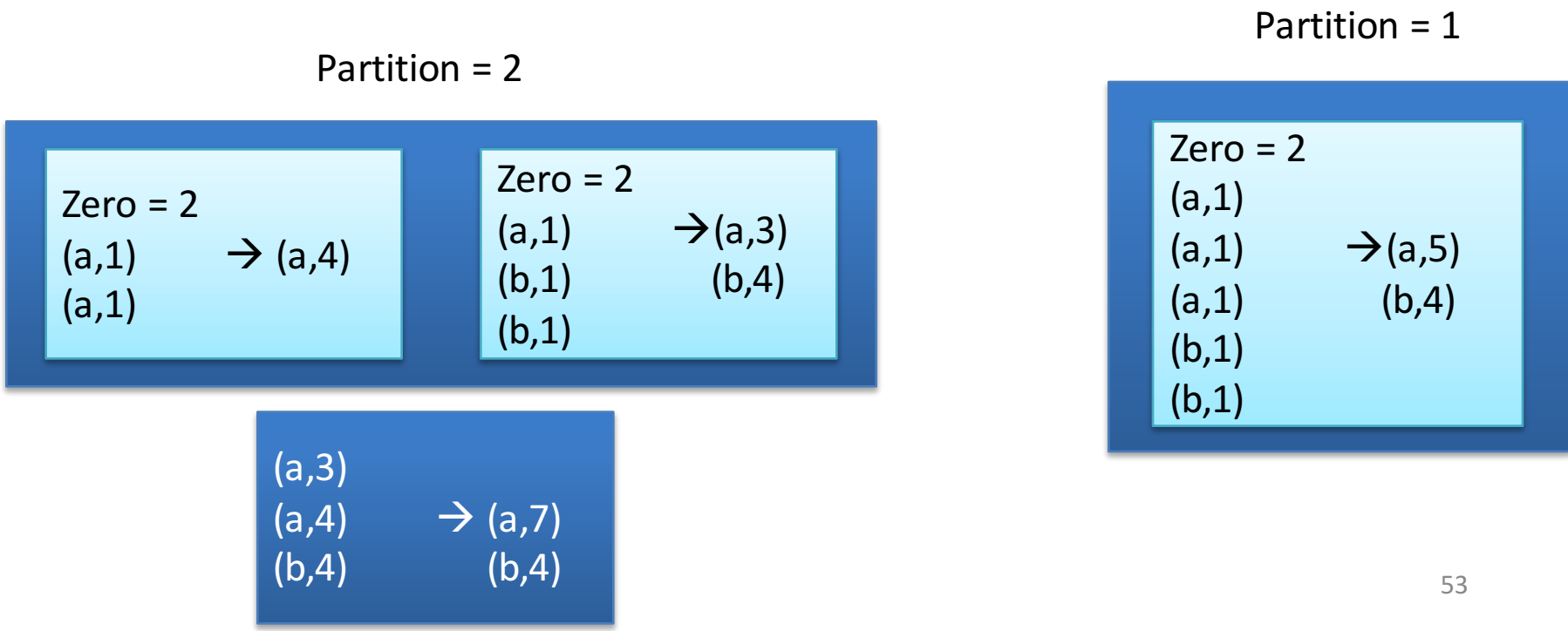


Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

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

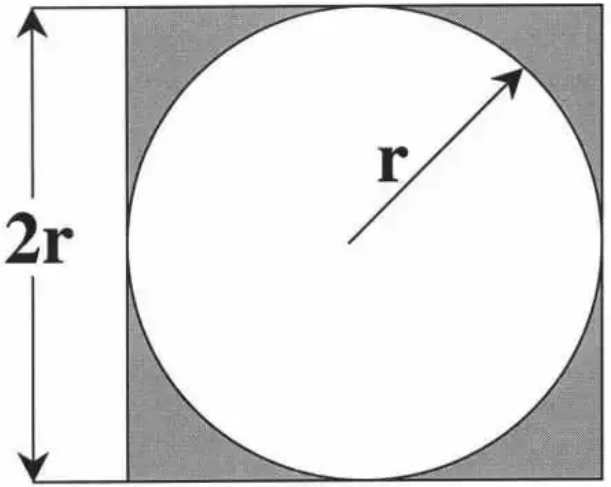
- scala> var rdd1 = sc.makeRDD(Array(("A","a"),("A","b"),("A","c"),("B","a"),("B","b")),3)
- scala> rdd1.foldByKey("0")(_+_).collect
- scala> rdd1.mapPartitionsWithIndex((index: Int, it: Iterator[(String,String)]) =>it.toList.map(x => if (index ==**0**) {println(x)}).iterator).collect

Spark application

- Pi Estimation
- Word Count
- K-means
- PageRank
- 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



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

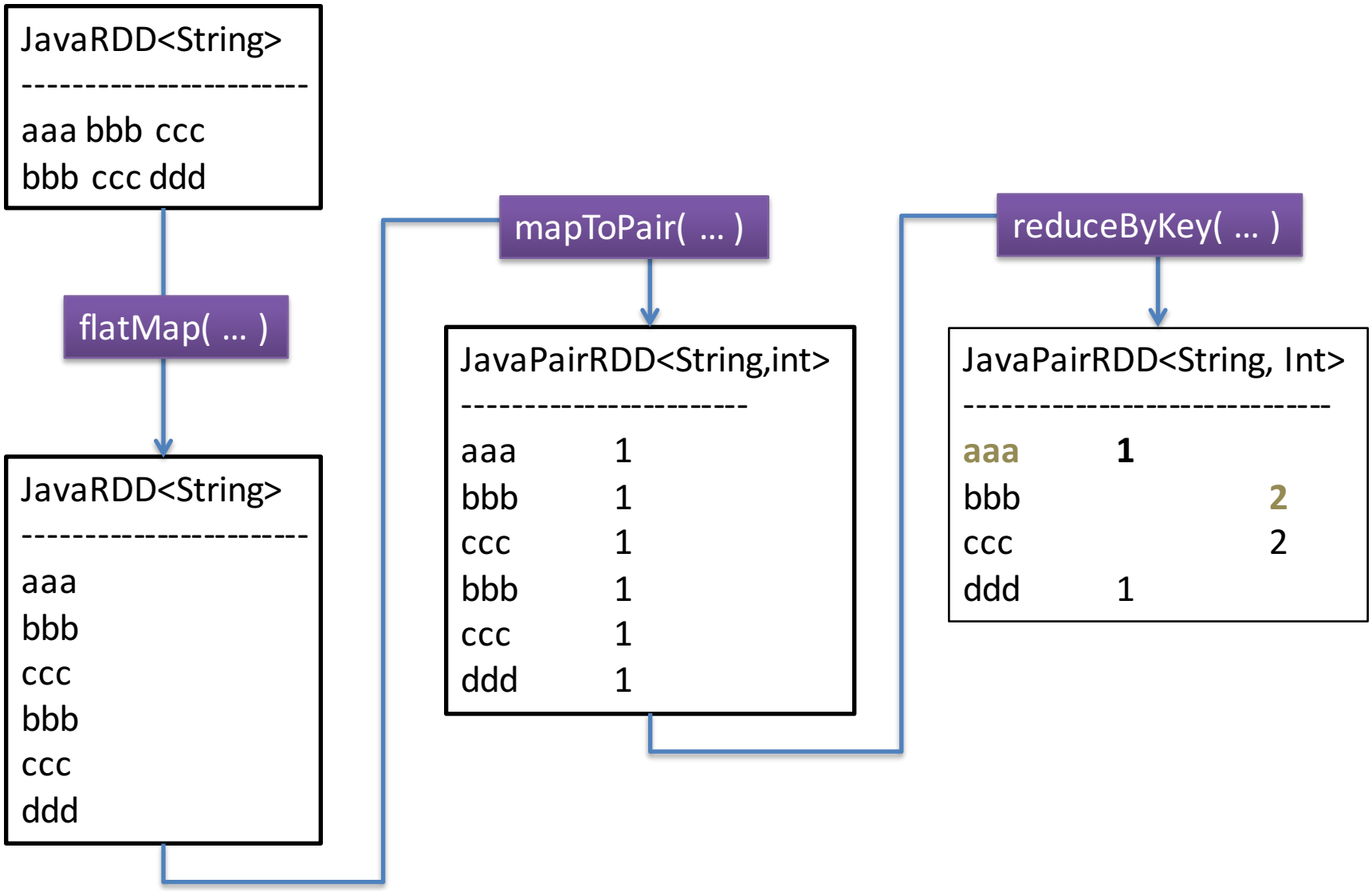
Exercise ii-6:
Pi Estimation

- 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
mapToPair(…)
JavaPairRDD<String,int>
aaa 1
bbb 1
ccc 1
bbb 1
ccc 1
ddd 1
reduceByKey(…)
JavaPairRDD<String, Int>
aaa 1
bbb 2
ccc 2
ddd 1

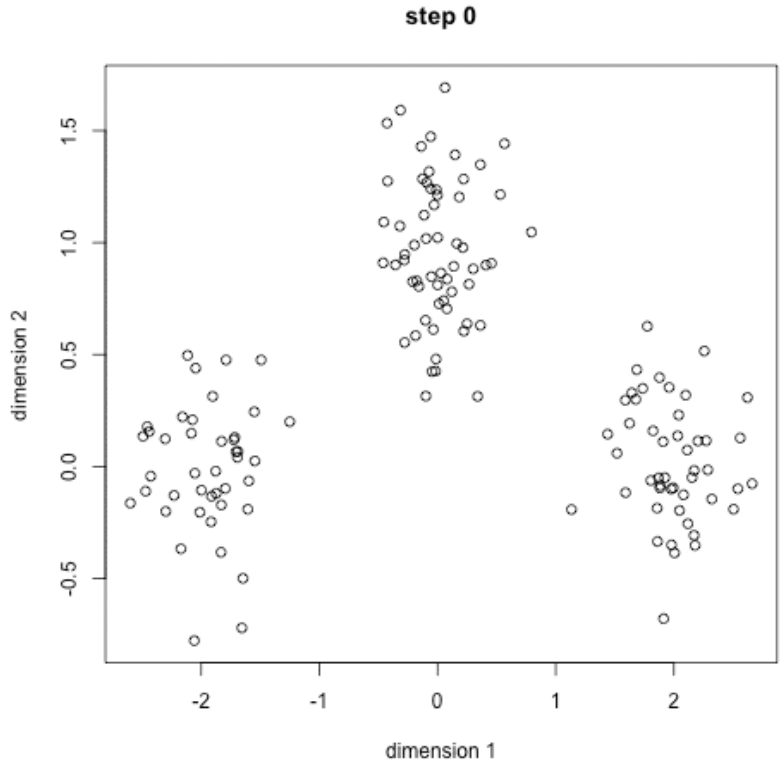
Exercise ii-7: Word Count

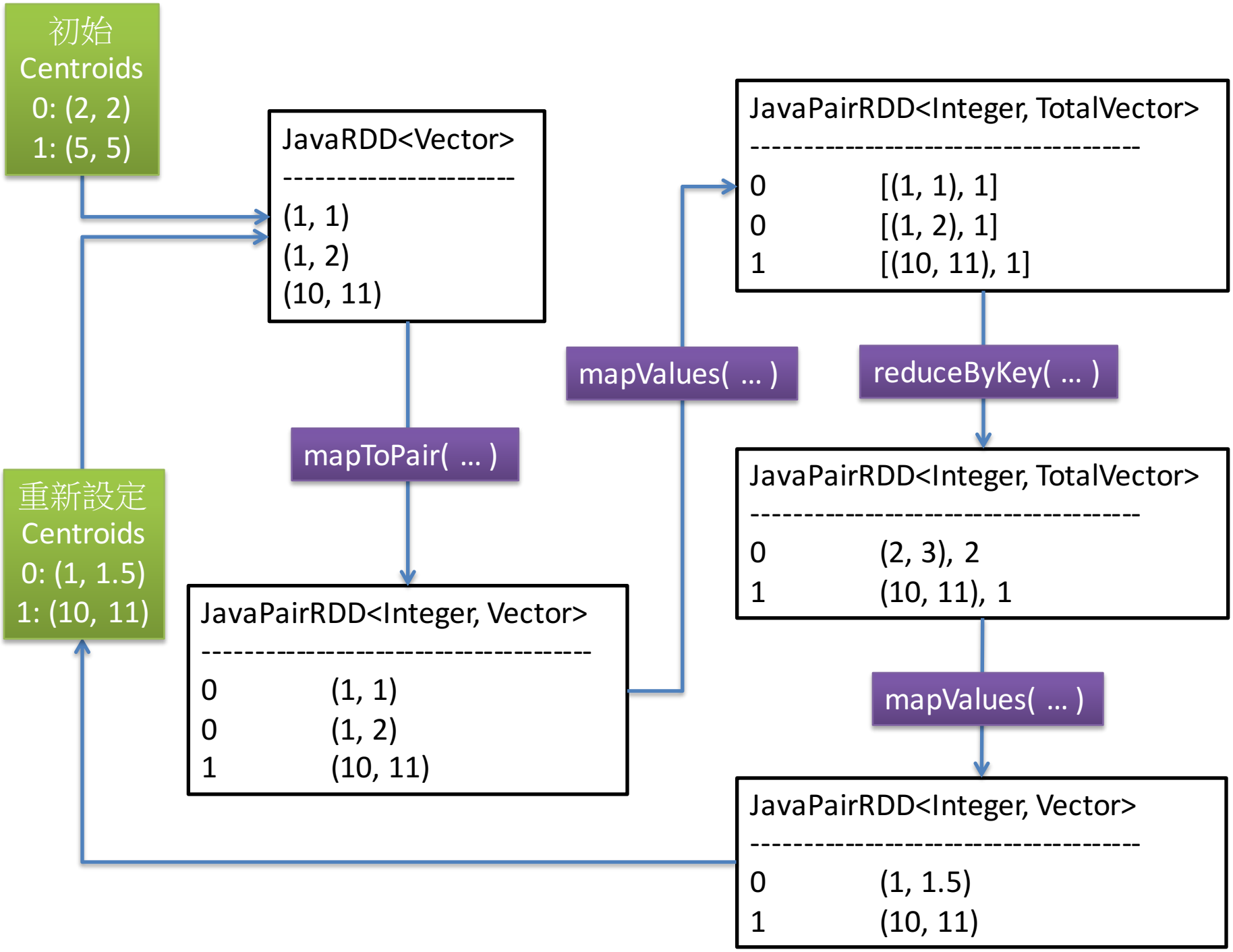
- 用foldByKey(…)改寫word count

- 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)



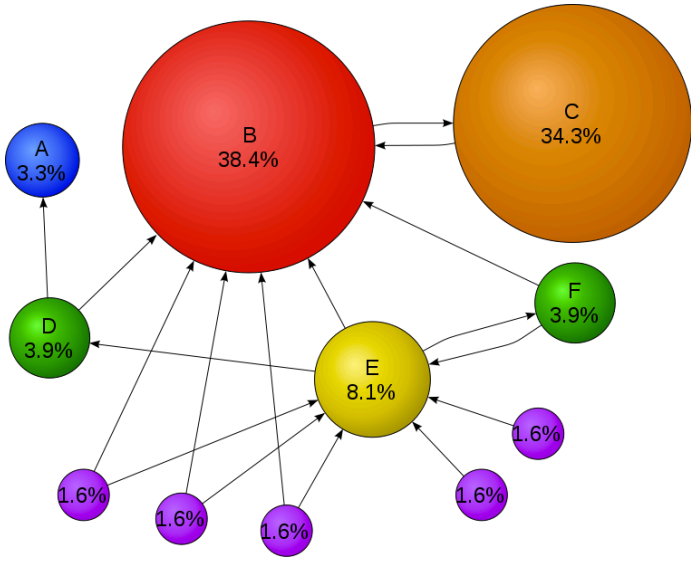

初始
Centroids
0: (2, 2)
1: (5, 5)
JavaRDD<Vector>
(1, 1)
(1, 2)
(10, 11)
mapToPair(…)
JavaPairRDD<Integer, Vector>
0 (1, 1)
0 (1, 2)
1 (10, 11)
mapValues(…)
JavaPairRDD<Integer, TotalVector>
0 [(1, 1), 1]
0 [(1, 2), 1]
1 [(10, 11), 1]
reduceByKey(…)
JavaPairRDD<Integer, TotalVector>
0 (2, 3), 2
1 (10,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

PageRank

- 評估網頁重要程度的指標



$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}$$

$$= \sum_{p_j} \frac{\text{PageRank}(p_j)}{L(p_j)}$$

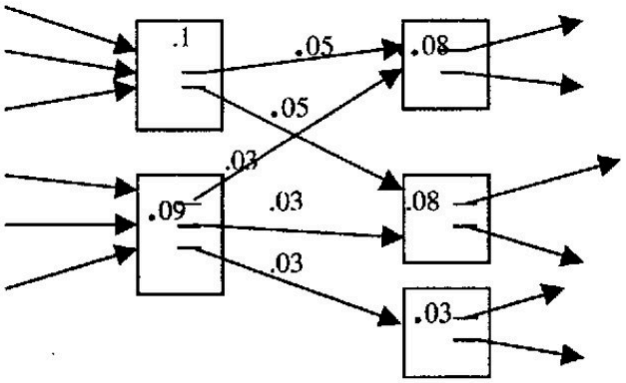


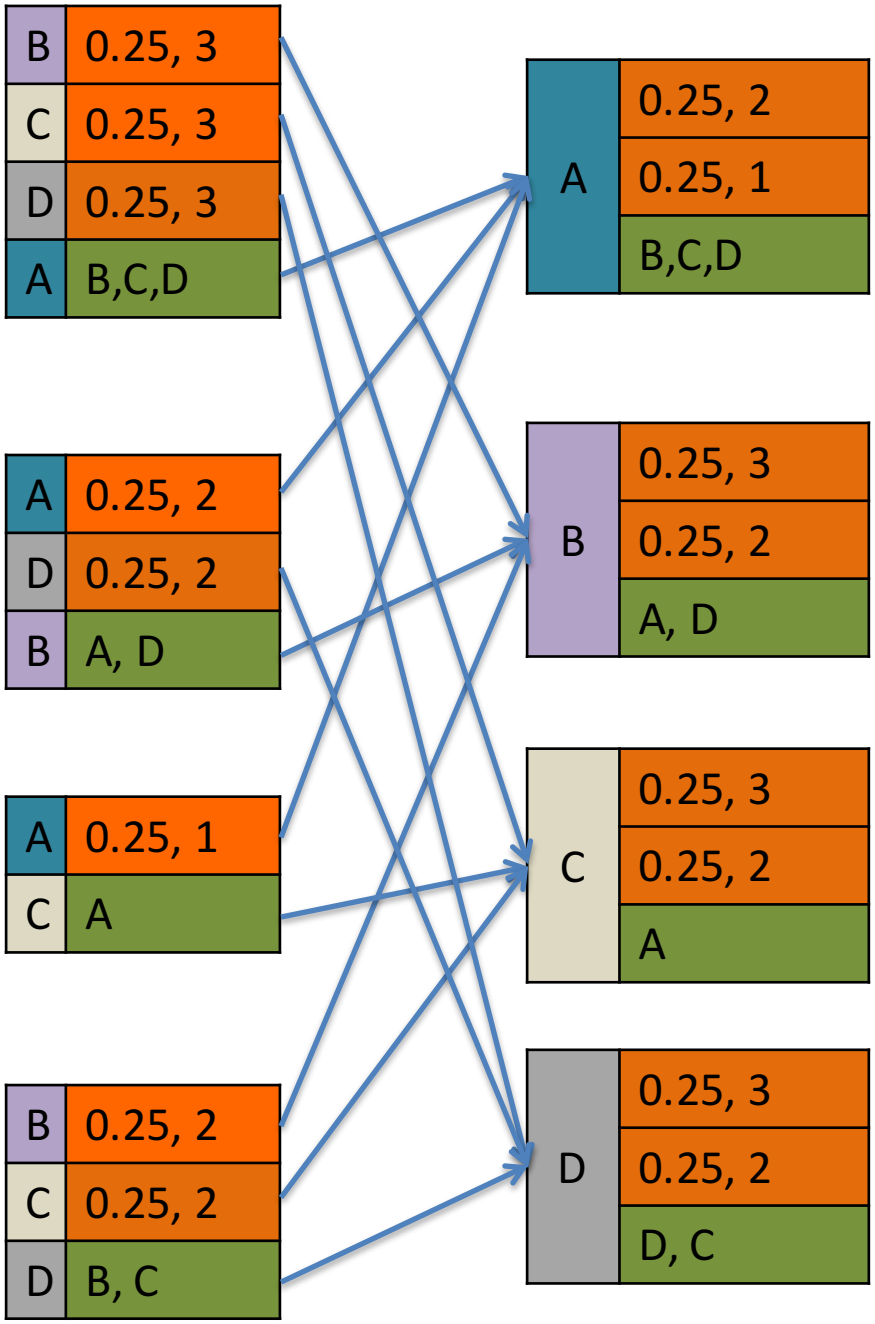
图 1　链接结构中的部分网页及其 PageRank 值

A, 0.25	B,C,D

B, 0.25	A, D

C, 0.25	A

D, 0.25	B,C

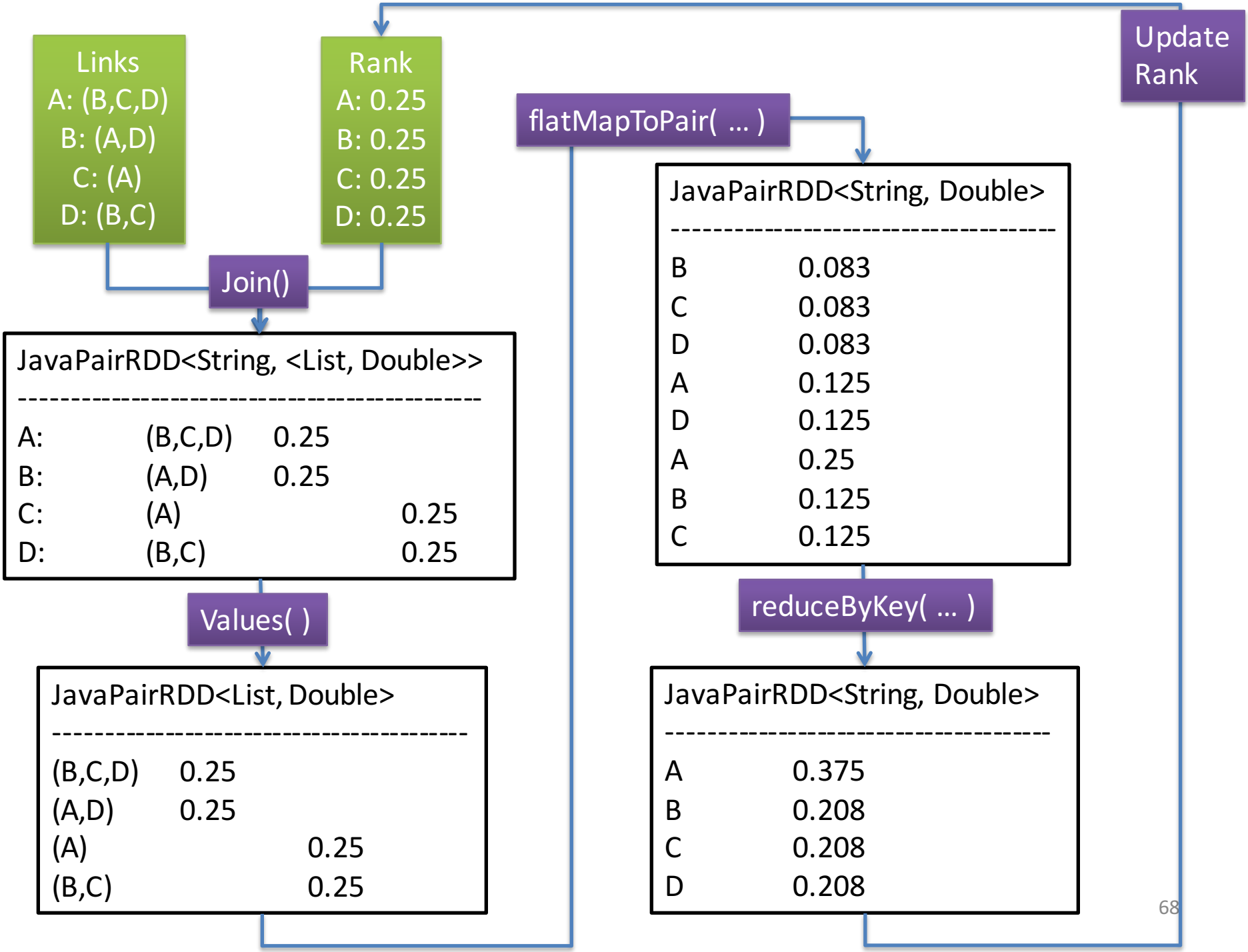


A, 0.375	B,C,D

B, 0.208	A, D

C, 0.208	A

D, 0.208	B,C


Links
A: (B,C,D)
B: (A,D)
C: (A)
D: (B,C)
Rank
A: 0.25
B: 0.25
C: 0.25
D: 0.25
Update Rank
Join()
flatMapToPair(…)
JavaPairRDD<String, <List, Double>>
A: (B,C,D) 0.25
B: (A,D) 0.25
C: (A) 0.25
D: (B,C) 0.25
JavaPairRDD<String, Double>
B 0.083
C 0.083
D 0.083
A 0.125
D 0.125
A 0.25
B 0.125
C 0.125
Values( )
reduceByKey(…)
JavaPairRDD<List, Double>
(B,C,D) 0.25
(A,D) 0.25
(A) 0.25
(B,C) 0.25
JavaPairRDD<String, Double>
A 0.375
B 0.208
C 0.208
D 0.208

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()

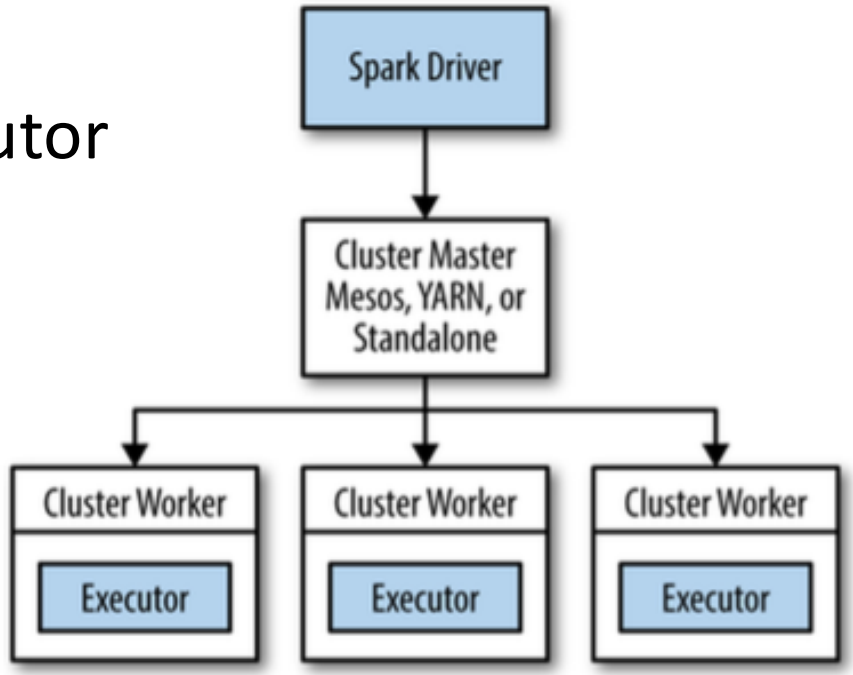
- countByKey()找出各種回傳值總數
  - Map<Integer, Object> responseCodeToCount = accessLogs.mapToPair(...).countByKey()
- 篩選出大於threshold的ip
  - JavaPairRDD<String, Long> result6 = result5.filter(…);
- 利用map()將JavaPairRDD<String, Long>轉成JavaRDD<String>
  - JavaRDD<String> result7 = result6.map(…);
- 印出RDD中的每個元素
  - 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

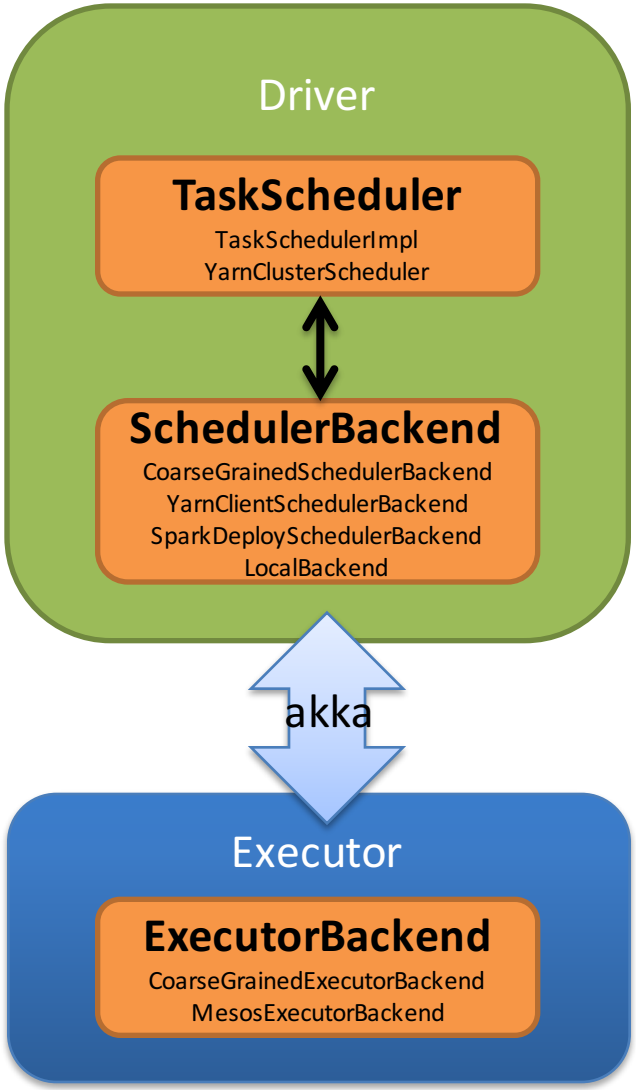
Spark Runtime Architecture

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



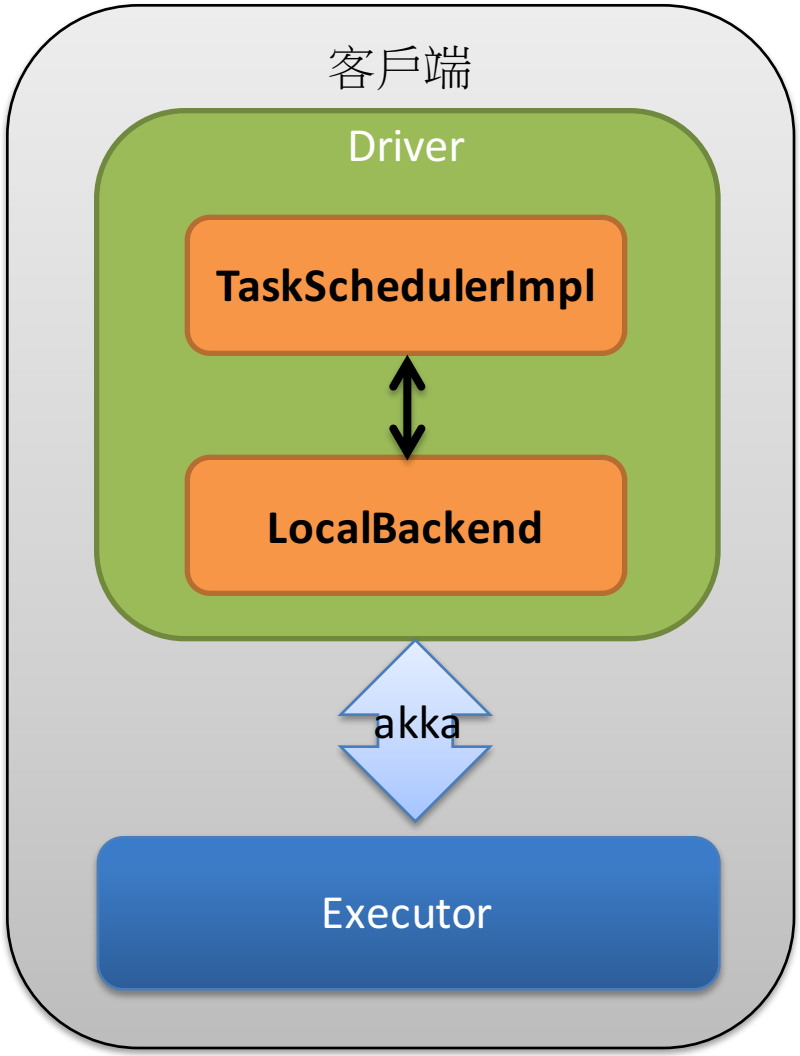
Spark Deployment

- Spark on Local
- Spark on Cluster
  - Spark Standalone
    - Cluster / Client
  - Spark on YARN
    - YARN-Cluster / YARN-Client
  - Spark on Mesos
    - Coarse / Fine grained
    - Cluster/client mode



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

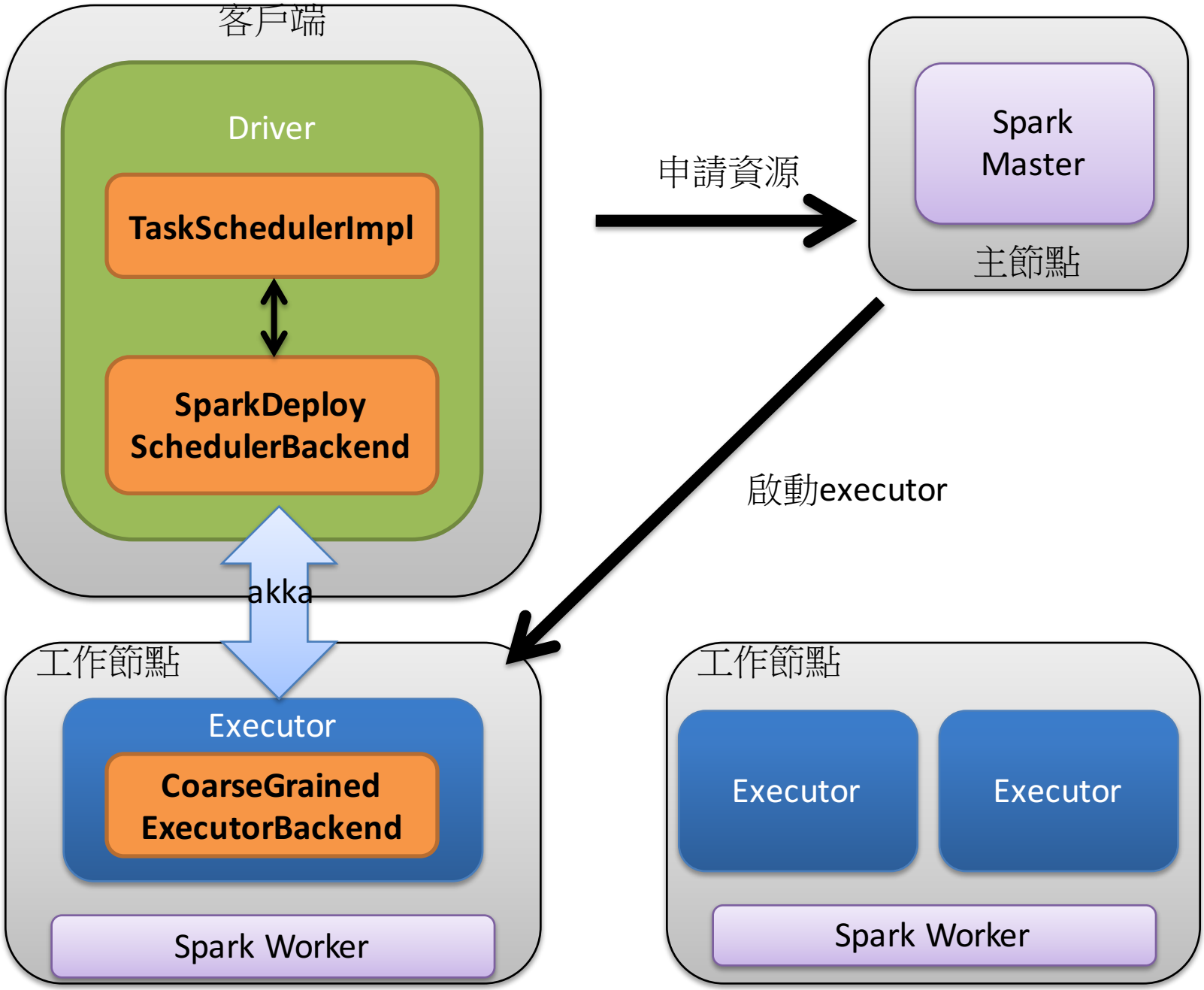


Exercise iii-1: use different mode

- $ ./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
  - Format HDFS
    - $hadoop namenode -format
  - 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

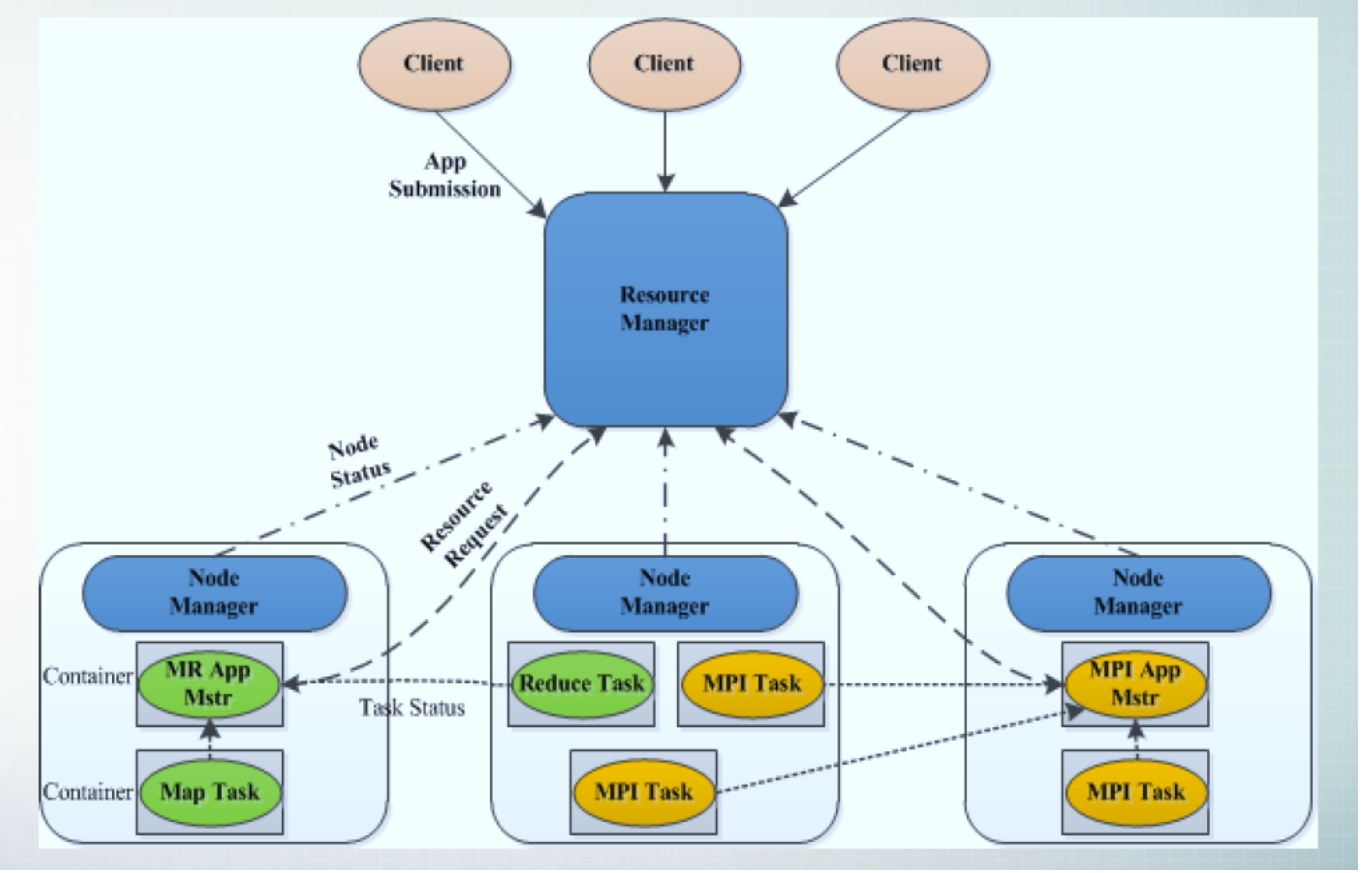
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
 </property>
 <property>
  <name>dfs.permissions</name>
  <value>false</value>
 </property>
</configuration>
```

.bashrc

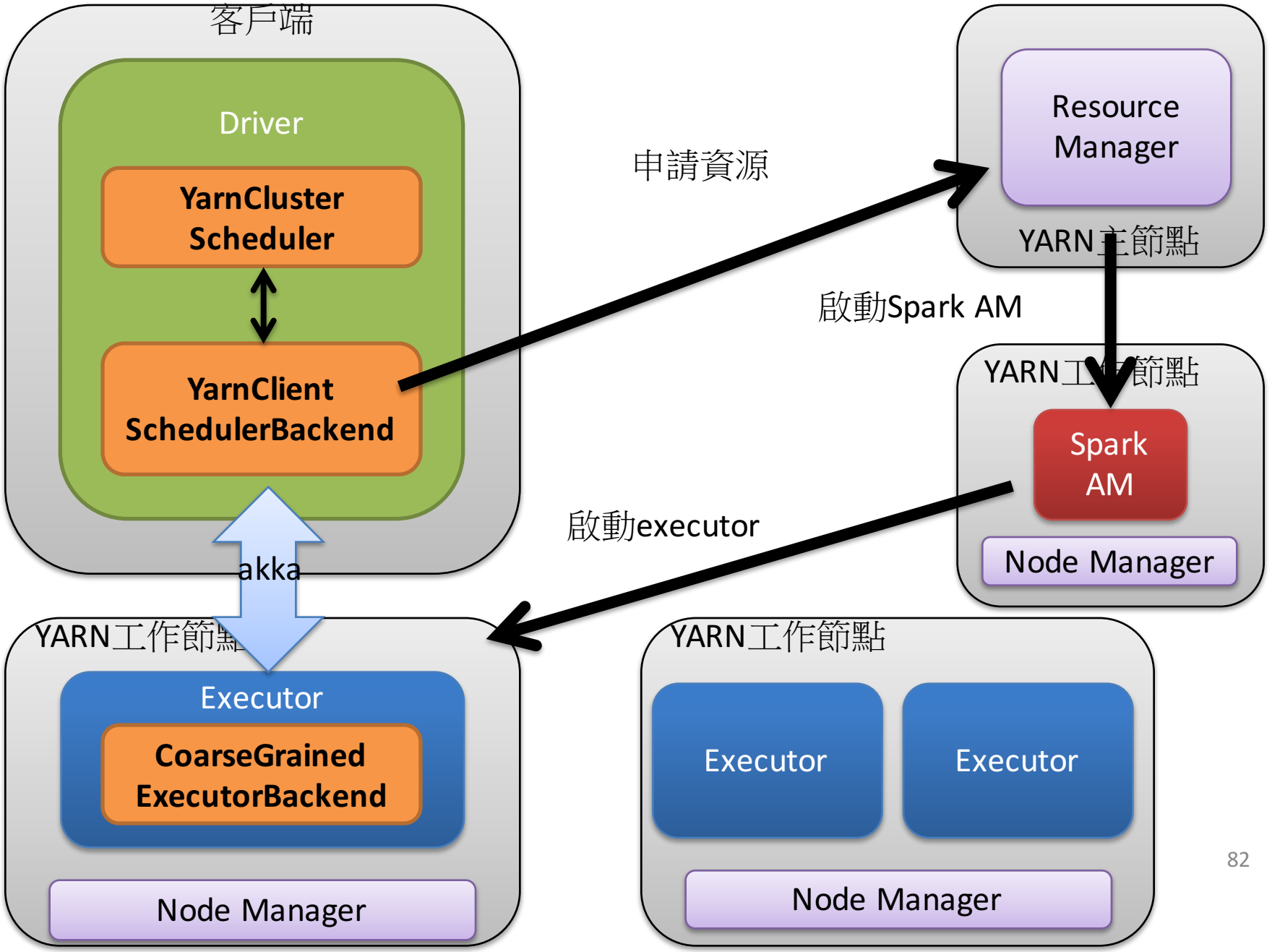
```
export HADOOP_HOME=/home/spark/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop

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

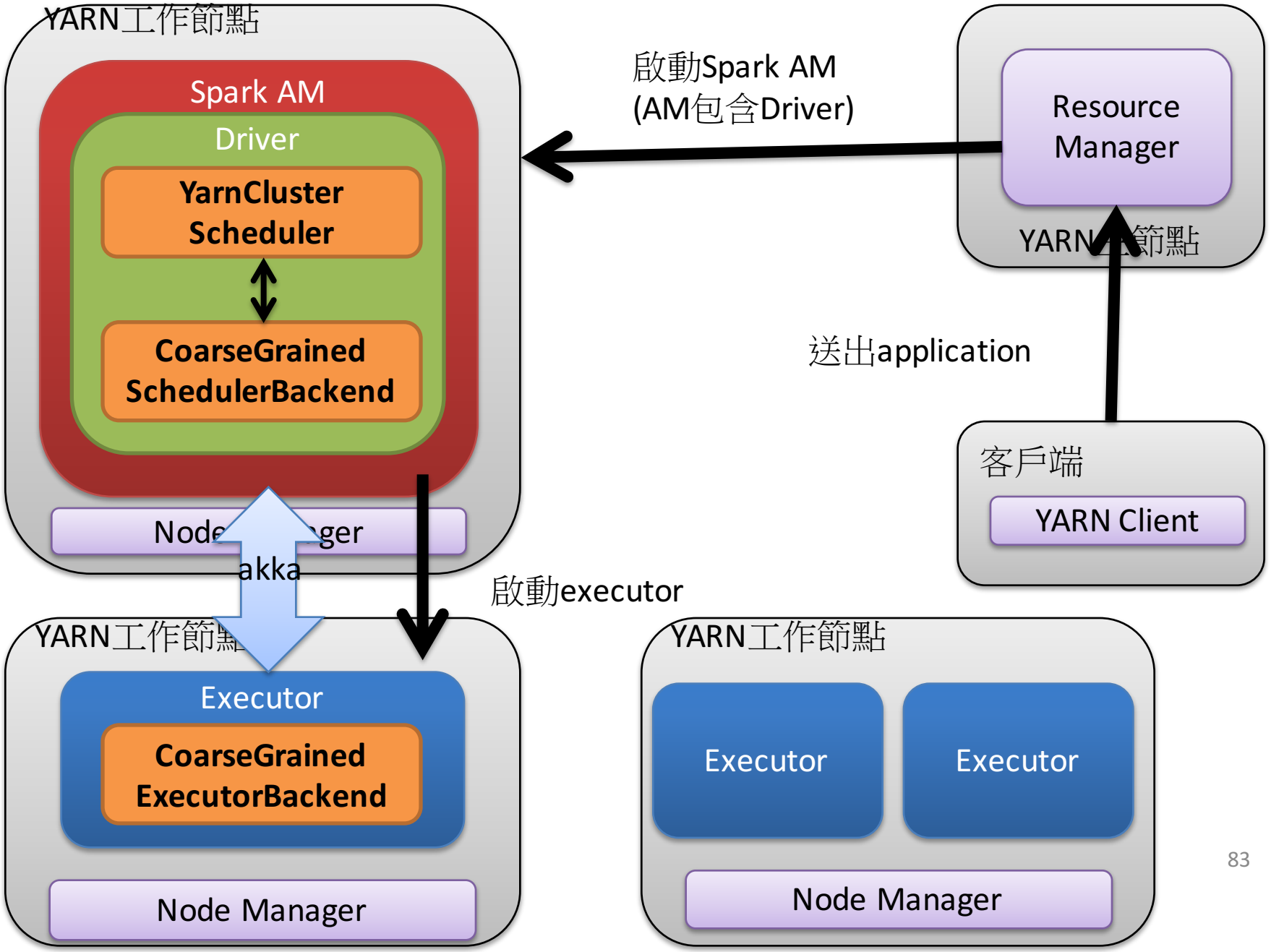
1. 由RM做全局的資源分配
2. NM定時回報目前的資源使用量
3. 每個JOB會有一個負責的AppMaster控制Job
4. 將資源管理與工作控制分開
5. YARN為一通用的資源管理系統
   可達成在YARN上運行多種框架



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

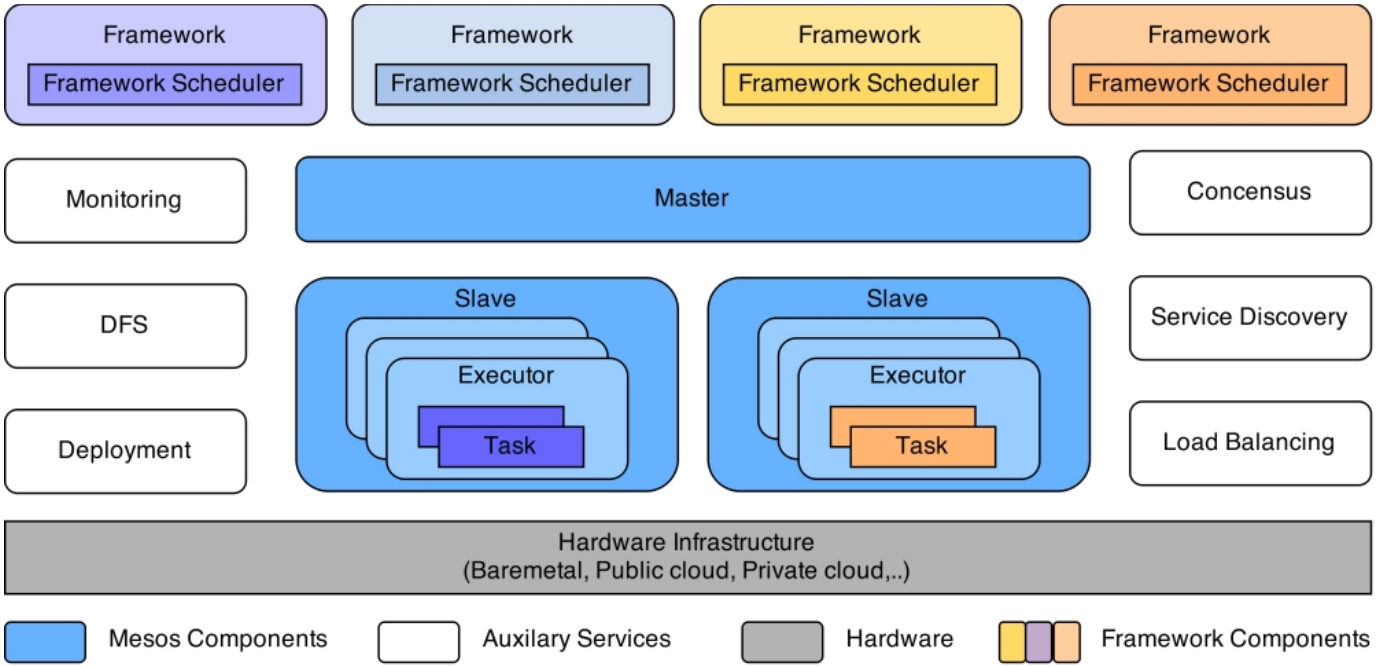
yarn-site.xml

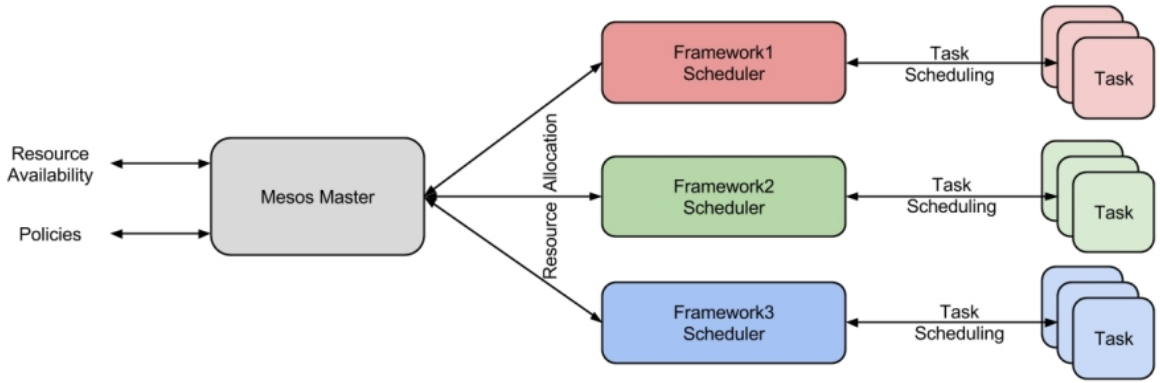
```
<?xml version="1.0"?>
<configuration>
<property>
   <name>yarn.resourcemanager.scheduler.address</name>
   <value>spark-master:8030</value>
 </property>
<property>
   <name>yarn.resourcemanager.resource-tracker.address</name>
   <value>spark-master:8031</value>
 </property>
<property>
   <name>yarn.resourcemanager.address</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

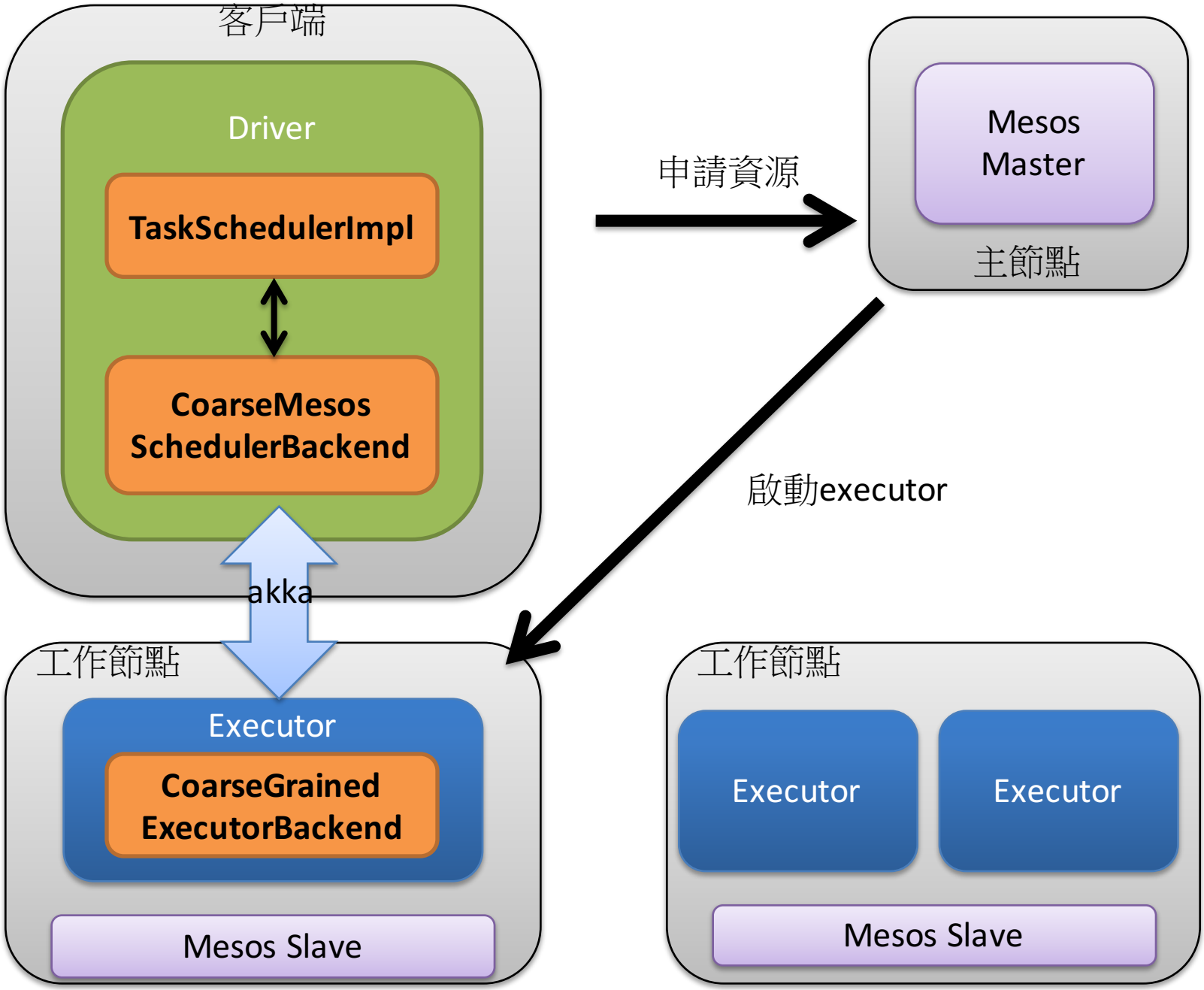
```
<?xml version="1.0"?>
<configuration>
 <property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
 </property>
</configuration>
```

Mesos Architecture and two level scheduler

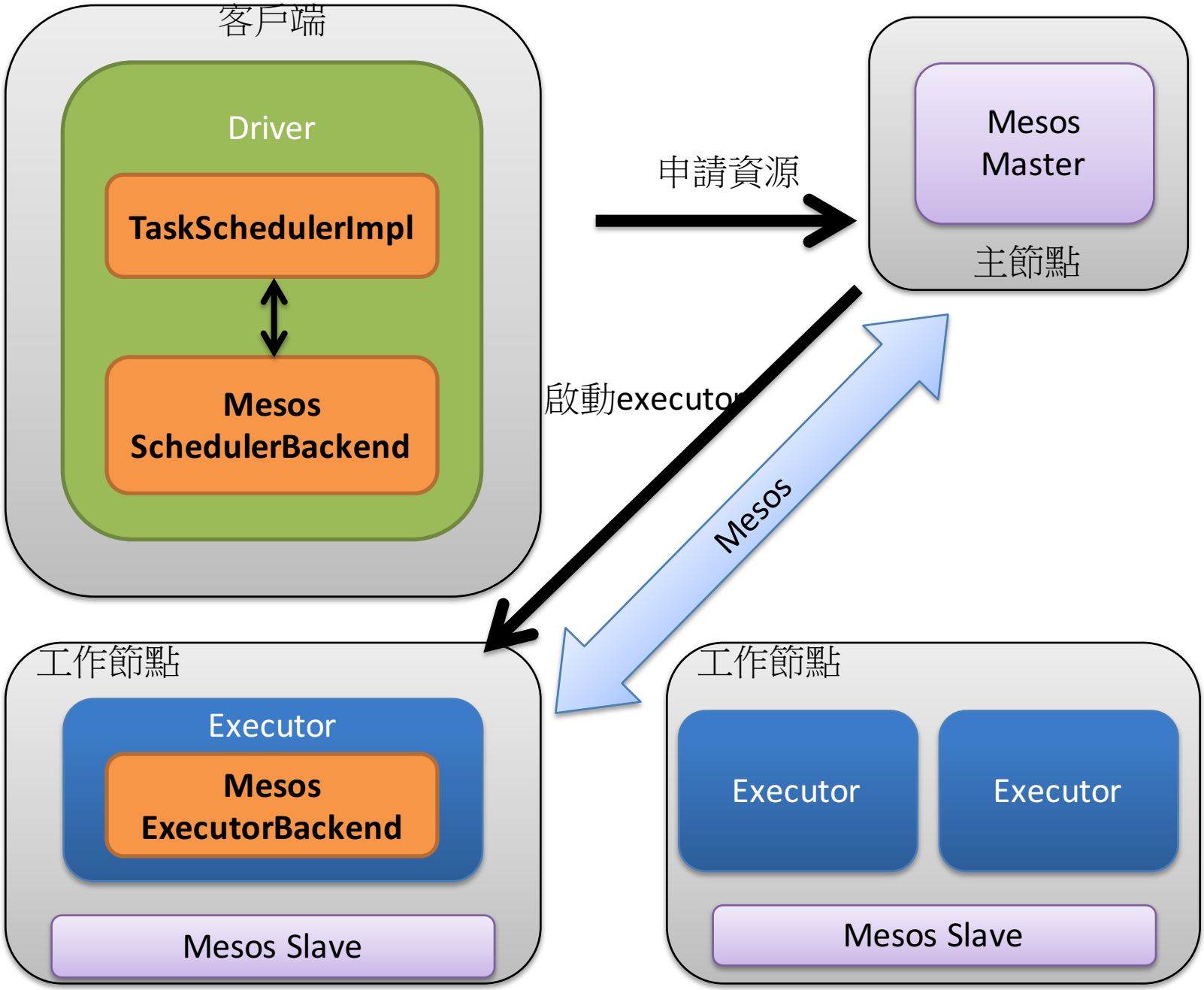




Mesos Coarse Grained



Mesos Fine Grained



Exercise iii-4: Setup Spark on Mesos Cluster

- Step 1: Install Mesos dependencies
  - $ sudo apt-get update
  - $ sudo apt-get install -y autoconf libtool
  - $ sudo apt-get -y install build-essential python-dev libcurl4-nss-dev libsasl2-dev libsasl2-modules maven libapr1-dev libsvn-dev

http://mesos.apache.org/documentation/latest/getting-started/
https://gist.github.com/ogre0403/6cd57ea7ee89b28221378536cf73c2b6

- Step 2: Build and Install Mesos from source
  - $ wget http://www.apache.org/dist/mesos/0.28.2/mesos-0.28.2.tar.gz
  - $ tar -zxf mesos-0.28.2.tar.gz
  - $ cd mesos-0.28.2
  - $ mkdir build ; cd build
  - $ ../configure; make ; make install
- Step 3: add mesos libraries to PATH
  - $ echo "export LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:/usr/local/lib" >> ~/.bashrc
- Step 4: Start Mesos Cluster
  - mesos-master --work_dir=/tmp/mesos_master
  - mesos-slave --master=master:5050 \
    --work_dir=/tmp/mesos_slave

- Step 5-1: copy spark binary into each slave
  - scp -r <SPARK_HOME> spark@slave:/home/spark

- Step 5-2: Only required on master
  - Put spark tarball into HDFS
  - Edit <SPARK_HOME>/conf/spark-defaults.conf
    - spark.executor.uri hdfs://master:9000/spark-1.6.2-bin-hadoop2.6.tgz

https://strat0sphere.wordpress.com/2014/10/30/spark-on-mesos-installation-guide/

- Step 6: connect to Mesos cluster
  - $ spark-shell --master mesos://master:5050 \
    --executor-memory 512m
  - $ spark-submit --master mesos://master:5050 \
    --executor-memory 512m \
    --class xx.yy.zz xyz.jar

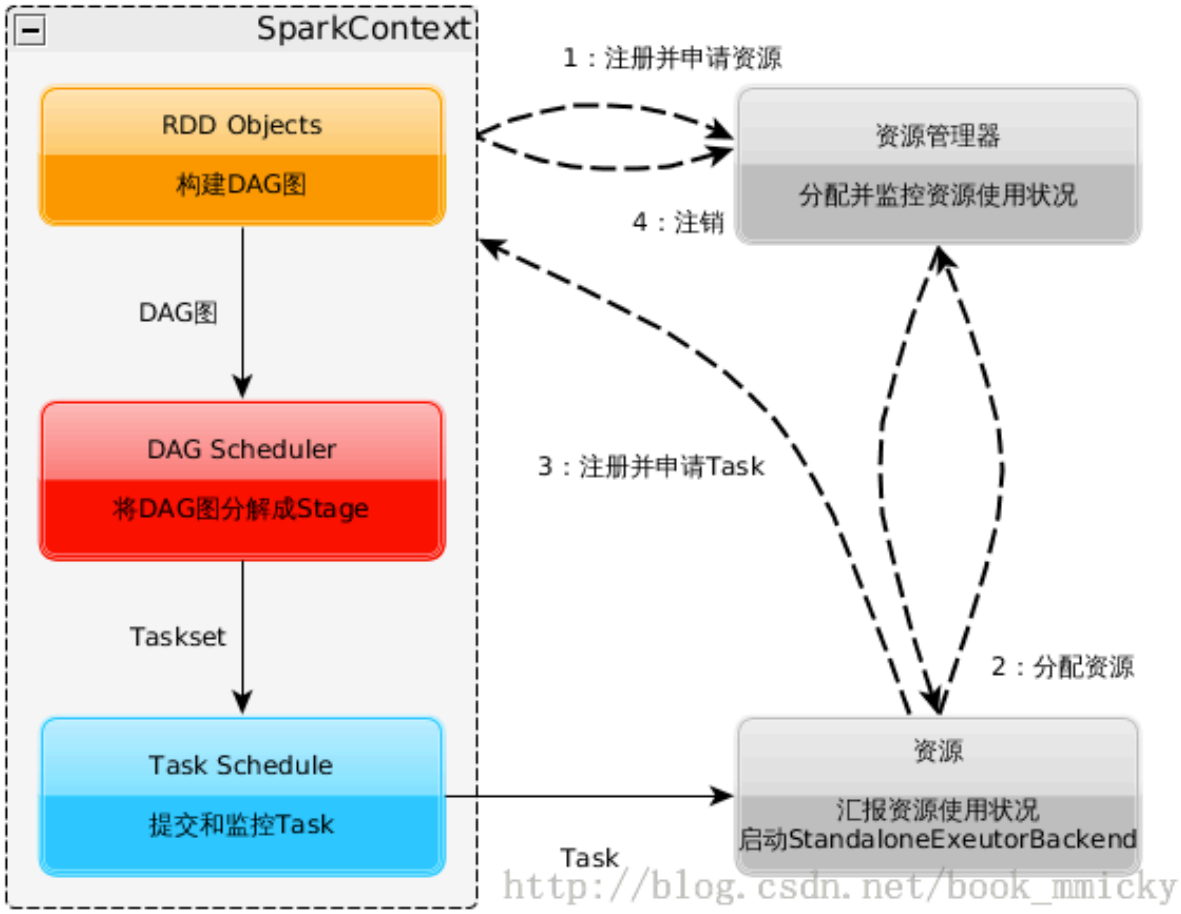
選擇那一種Cluster mode

- 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。

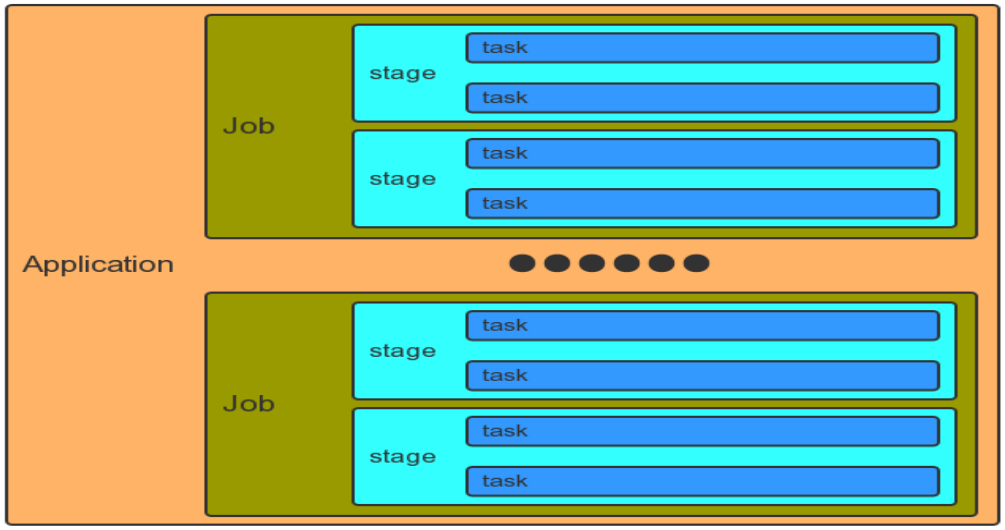
Hadoop application

- 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

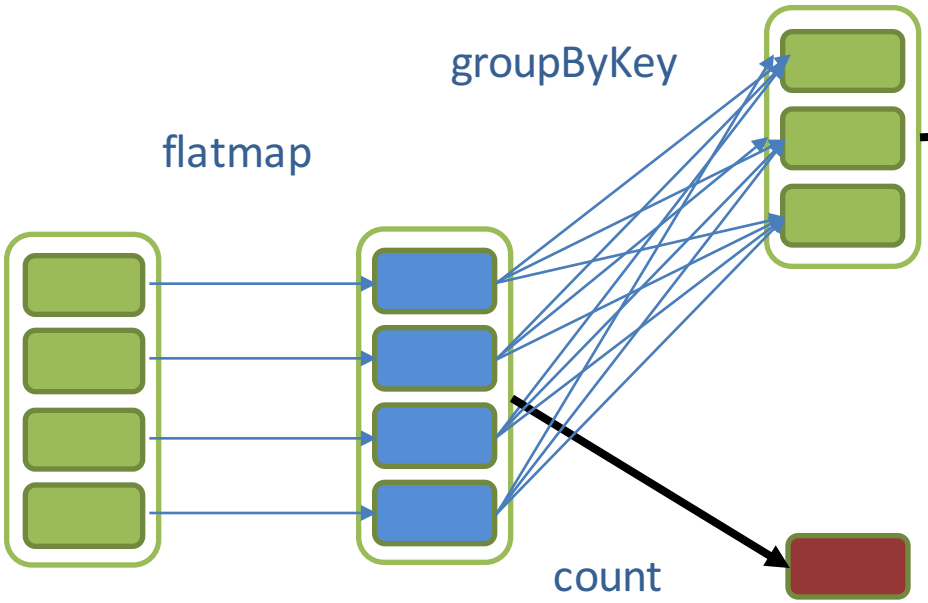
Spark Application

- 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

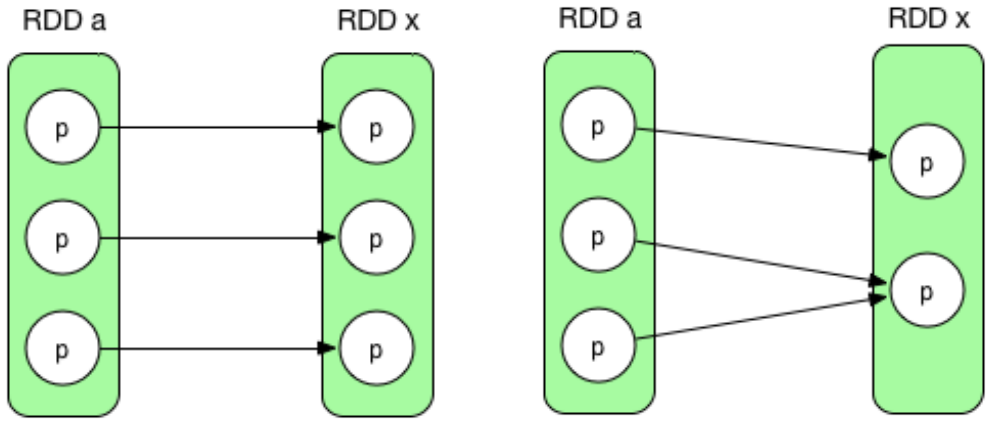
- application 可以包含多個job
- Driver 中每執行一個action，就會產生一個job

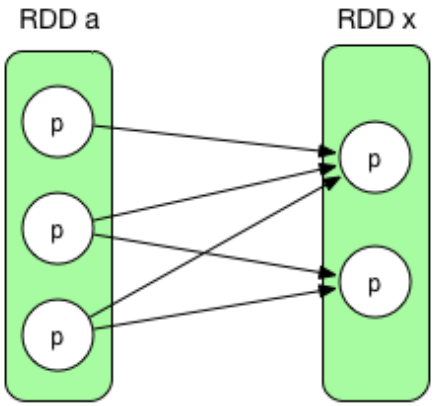



```
val pairs1 = sc.parallelize().flatMap {
            p =>  .....
}.cache

pairs1.count

pairs1.groupByKey(numReducers).count
```

Narrow Dependency

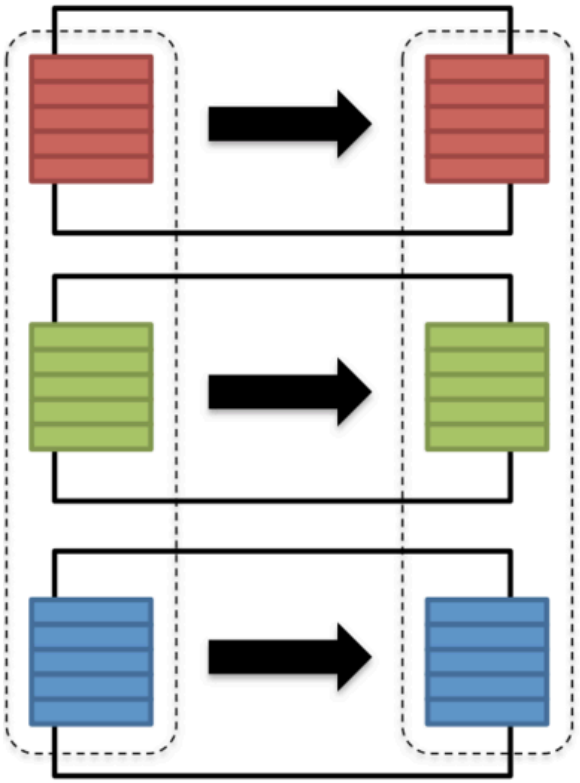
- depends only on data that is already residing in the partition and data shuffling is unnecessary
- Filter()、map()、…





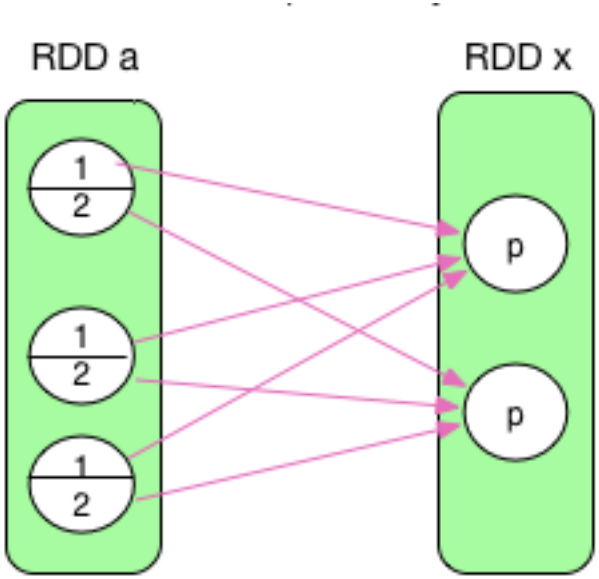
Narrow transformation

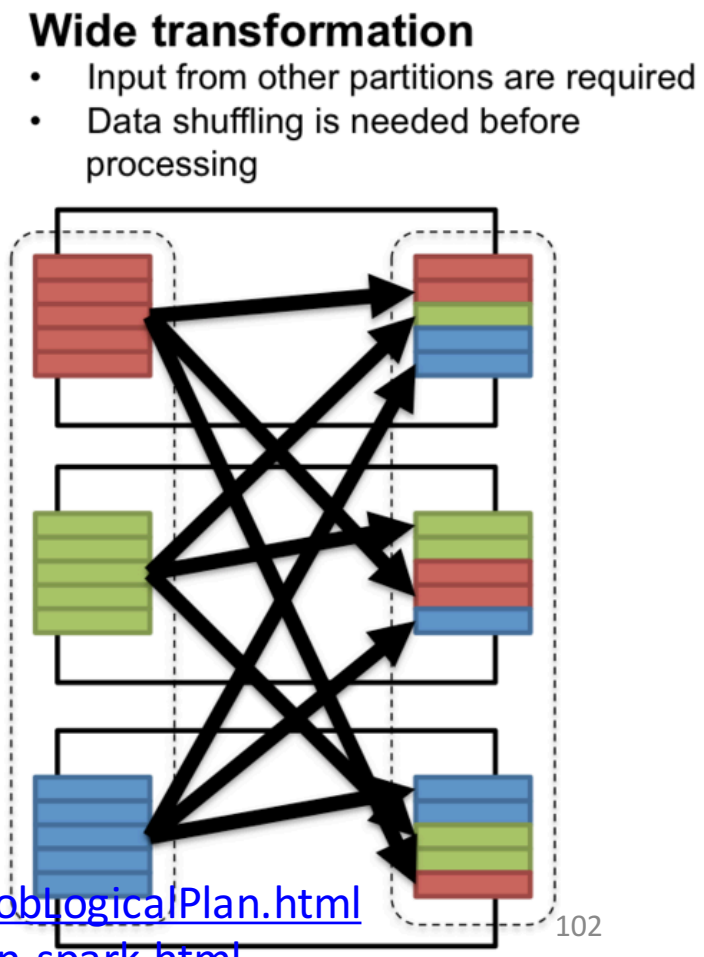
- Input and output stays in same partition
- No data movement is needed



Wide Dependency

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

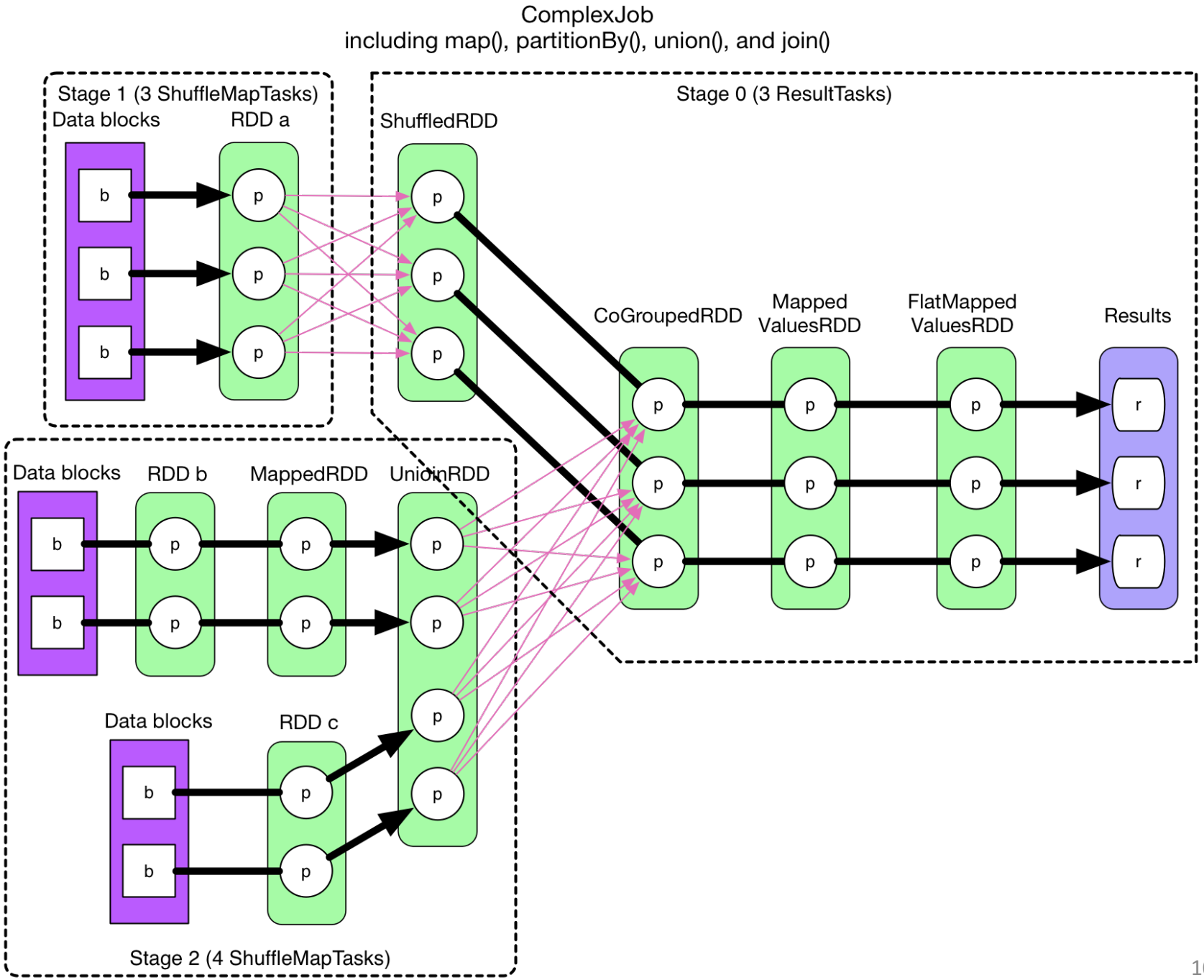




http://spark-internals.books.yourtion.com/markdown/2-JobLogicalPlan.html
http://horicky.blogspot.tw/2015/02/big-data-processing-in-spark.html

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

- Create local log directory
- Edit $SPARK_HOME/conf/spark-defaults.conf
  - 讓Executor 輸出event log
    - spark.eventLog.enabled true
    - spark.eventLog.dir file:///tmp/sparkLogs
  - 設定history server讀取目錄
    - spark.history.fs.logDirectory file:///tmp/sparkLogs
- Start history server
  - $SPARK_HOME/sbin/start-history-server.sh

Exercise iii-5：觀察Job、Stage、Task in UI

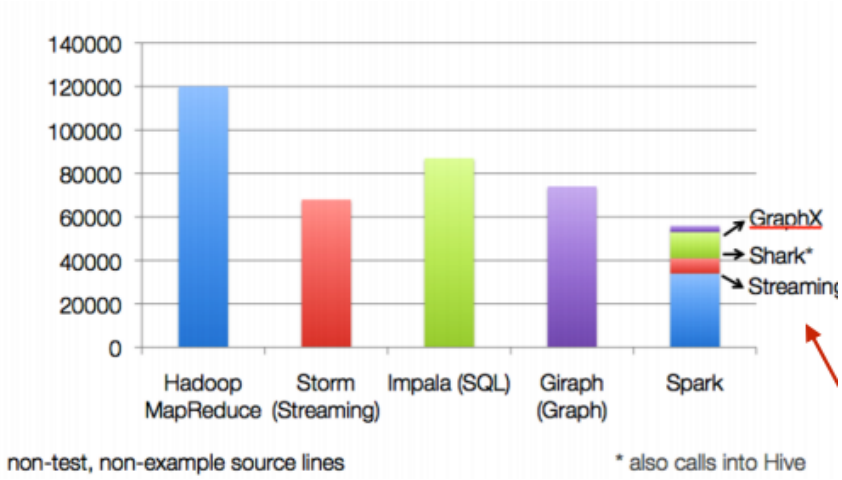
- 分別執行simpleJob與complexJob
- 在job-history-server上驗證job數、stage數、task數

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 Streaming
  - Spark SQL & DataFrames
  - SparkR
  - spark.mllib & spark.ml

Unified Programming Abstraction

Type	Hadoop	Spark
Batch	MapRduce	RDD
Interactive	Pig	Spark-shell
Streaming	Storm	Spark Streaming
Relational Query	Hive	Spark SQL
ML	Mahout	Mllib & ML

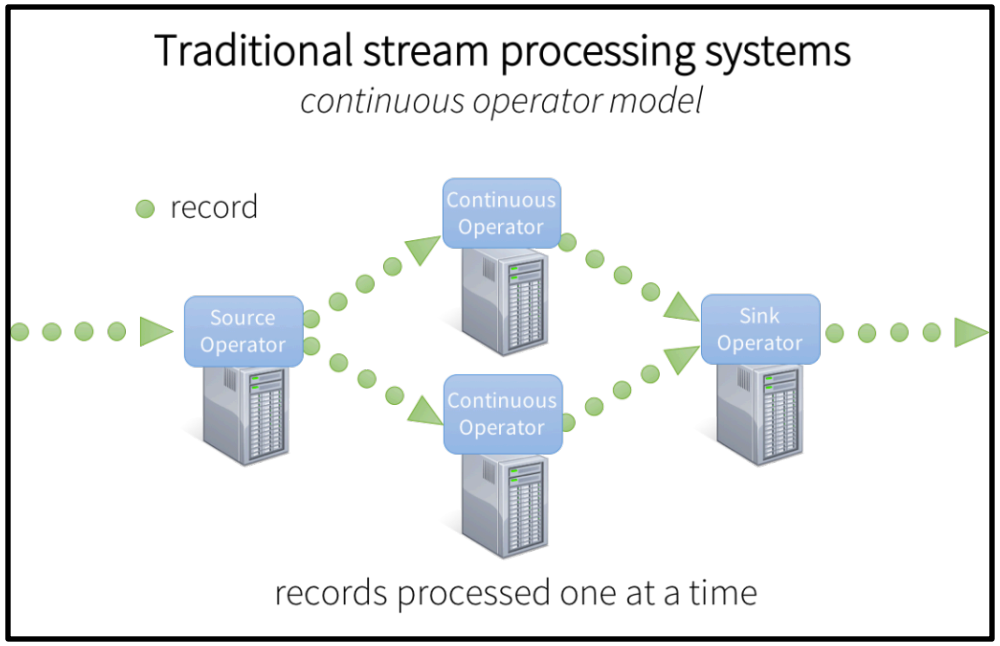


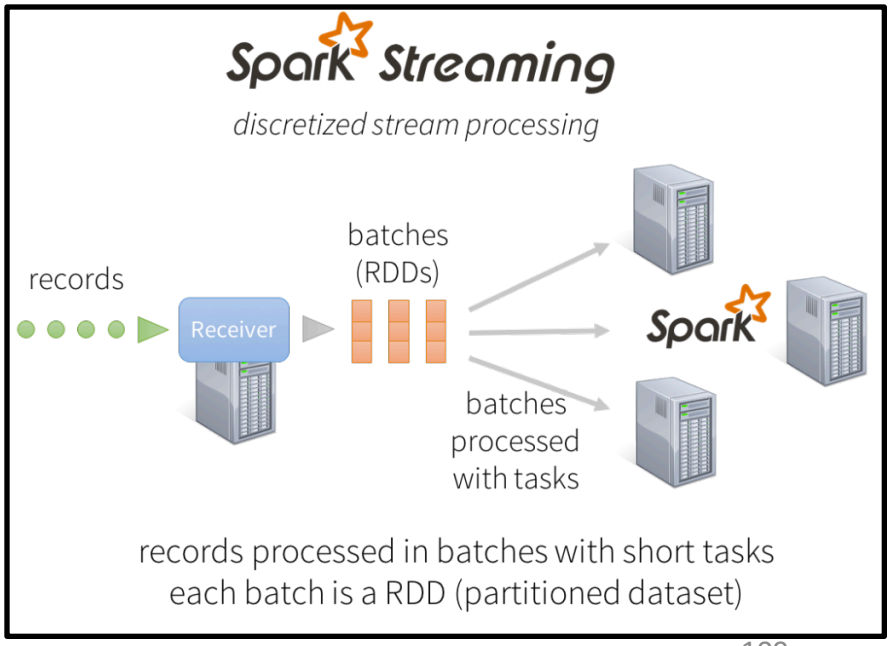
… This dissertation argues that, instead, we can design a **unified programming abstraction** that not only captures these disparate workloads, but enables new applications as well…

An Architecture for Fast and General Data Processing on Large Clusters
M Zaharia

Realtime vs. Near-realtime Streaming process

- Realtime streaming
  - Continues operation
  - Apache Strom
- Near-realtime
  - Micro-batch
  - Spark Streaming

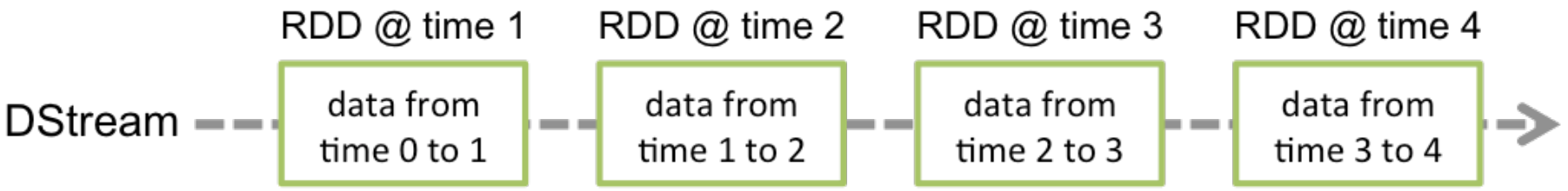




https://databricks.com/blog/2015/07/30/diving-into-spark-streamings-execution-model.html

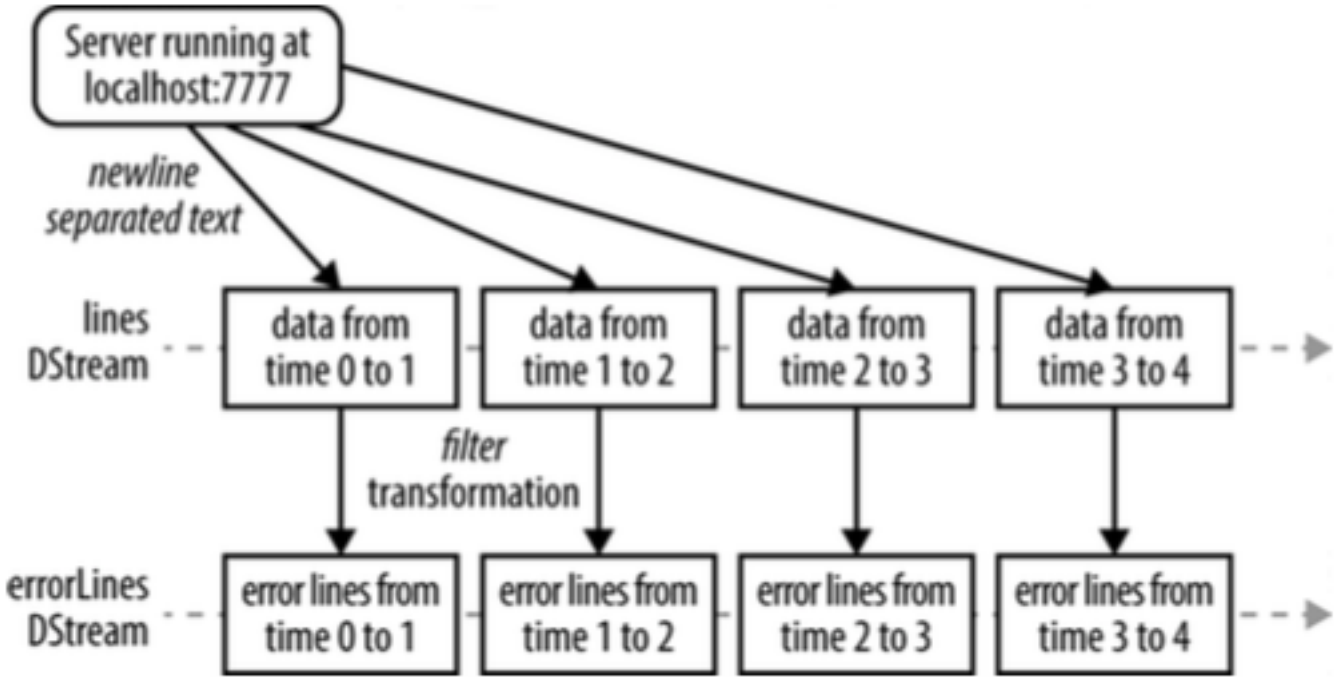
DStream

- Discretized Stream
  - DStream is represented by a continuous series of RDDs
  - Each RDD in a DStream contains data from a certain interval
    - One RDD contains data arrive in one interval



DStream

- Any operation applied on a DStream translates to operations on the underlying RDDs



Operations

- Transformation
  - Stateless transformation
    - applies separately to each RDD (ie. Time interval)
  - Stateful (or window)transformation
    - data from previous interval is used to generate the results
- Output
  - similar to RDD actions, write data to external
  - run periodically on each time step, producing output in batches

Stateless transformation

- Many RDD transformation is also available for DStream
- transform()
  - Apply any arbitrary RDD-to- RDD function to act on the DStream

Table 10-1. Examples of stateless DStream transformations (incomplete list)

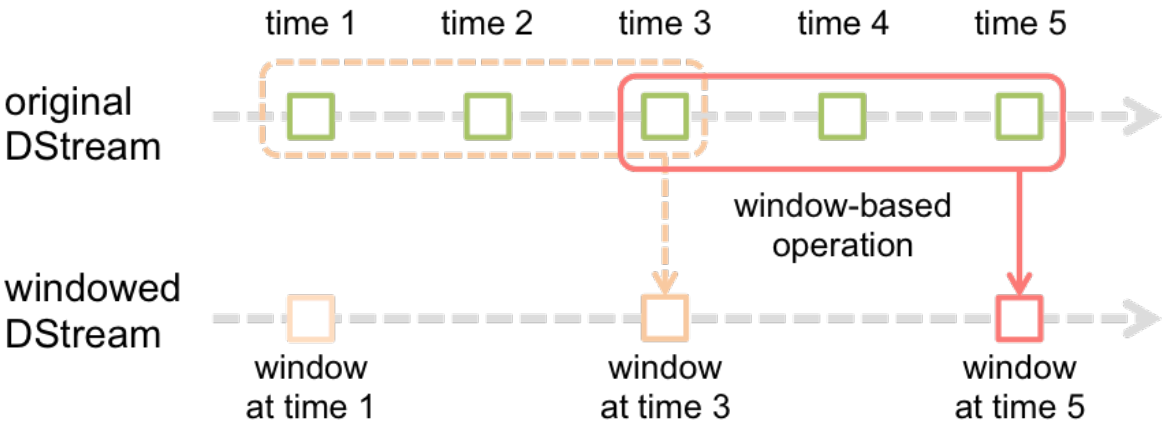
Function name	Purpose	Scala example	Signature of user-supplied function on DStream[T]
map()	Apply a function to each element in the DStream and return a DStream of the result.	ds.map(x => x + 1)	f: (T) → U
flatMap()	Apply a function to each element in the DStream and return a DStream of the contents of the iterators returned.	ds.flatMap(x => x.split(" "))	f: T → Iterable[U]
filter()	Return a DStream consisting of only elements that pass the condition passed to filter.	ds.filter(x => x != 1)	f: T → Boolean
repartition()	Change the number of partitions of the DStream.	ds.repartition(10)	N/A
reduceByKey()	Combine values with the same key in each batch.	ds.reduceByKey((x, y) => x + y)	f: T, T → T
groupByKey()	Group values with the same key in each batch.	ds.groupByKey()	N/A

Exercise iv-1: stateless transformation

- 產生日誌資料
  - $ ./loggen.sh
  - $ tail –f /tmp/fake.log | nc –lk 5555
- 篩選出日誌串流內容大小大於5000的資料
  - accessLogDStream.filter(…).filter(…).cache()
- 計算篩選出來的日誌大小總合
  - largerLog.map(…).reduce(…);

Window operation

- Batch interval
  - 將一段時間內的資料轉成micro batch
  - Window/sliding duration 皆為batch interval的倍數
- Window duration
  - 要包含過去的多少個interval
- Sliding duration
  - 多久執行一次Dstream計算



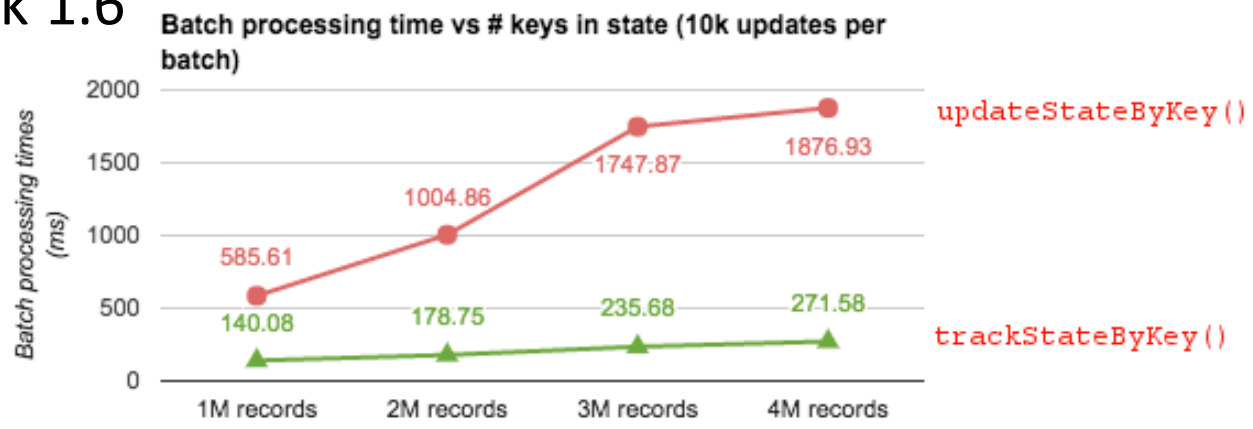
Window transformation	說明
winodw()	Return new DStream with the data for the requested window. We can then process with count(), transform(), and so on
reduceByWindow()	perform reduce function on each window
reduceByKeyAndWindow()	perform reduceByKey function on each window
countByWindow()	Return a DStream representing the number of elements in each window.
countByValueAndWindow()	Return a DStream with the counts for each value.

Exercise iv-2: window transformation

- Batch interval = 2sec
  - 每二個batch interval執行一次
    - slide = 4 sec
  - 每次統計包含過去三個batch interval
    - Window = 6 sec
- 每四秒算一次過去六秒內的日誌總數
  - countByWindow(WINDOW_LENGTH, SLIDE_INTERVAL)
- 每四秒算一次過去六秒內的日誌大小總合
  - reduceByWindow(… , WINDOW_LENGTH, SLIDE_INTERVAL)

updateStateByKey()

- Window operation
  - 計算 “過去一段” batch intervals 內的資料
- updateStateByKey()
  - 計算 “過去所有” batch interval 內相關的資料
- trackStateByKey()
  - Release in Spark 1.6



http://bit1129.iteye.com/blog/2198682

http://technicaltidbit.blogspot.tw/2015/11/spark-streaming-16-stop-using.html

- 定義全域的可更新變數，每個batch 得到的值都累計到之前batch的結果，以達到跨多個batch interval的統計
  - oldState
    - it might be missing if there was no previous state for the key
    - Generate newState based on oldState
  - Events
    - a list of events that arrived in the current batch (may be empty).
  - newState
    - returned by the function

```
.updateStateByKey(
    new Function2<List<Long>, Optional<Long>, Optional<Long>>() {
        @Override
        public Optional<Long> call(List<Long> events, Optional<Long> oldState)
            Long newState = oldState.or(0L);
            for(Long i:events)
                newState +=i;
            return Optional.of(newState);
        }
    }
);
```

Exercise iv-3:
統計過去全部的Code count

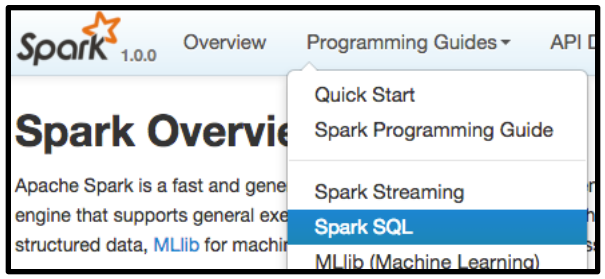
- responseCodeCountPairDStream.updateStateByKey(…);

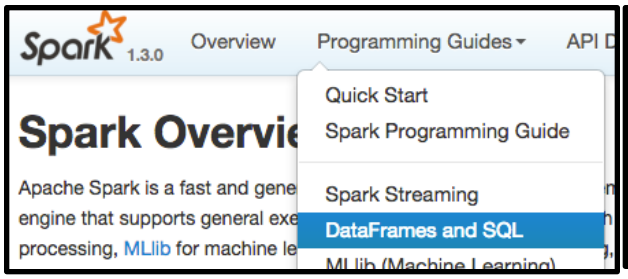
Outline

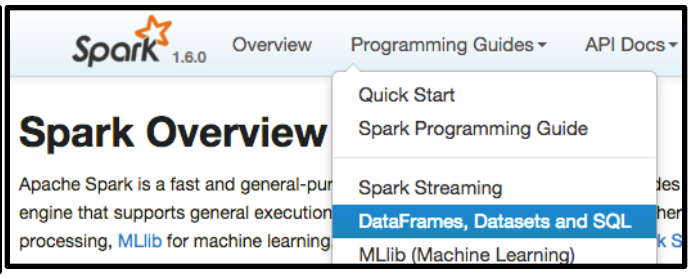
- Part III: Run on Cluster
  - Cluster Manager
    - Standalone, YARN, Mesos
  - Components of Execution
    - Application, Job, stage, task, RDD, partition, DAG
- Part IV: Play with Spark Ecosystem
  - Spark Streaming
  - Spark SQL & DataFrames
  - SparkR

Manipulating Structured Data in Spark

- Shark: Development End
- Spark SQL : new SQL engine
  - SchemaRDD: ver. 1.0 ~ ver. 1.2
    - RDD + Schema
  - **DataFrames**: since ver. 1.3
    - Inspired from R and Python panda
    - Abstraction for structured data in Spark
  - DataSet: release in ver. 1.6





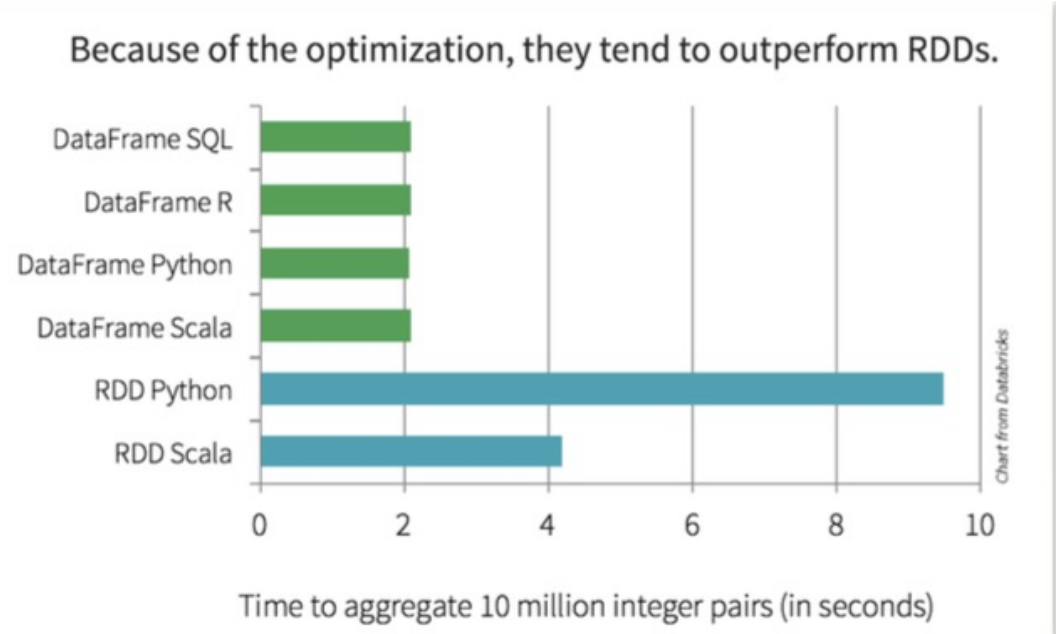


RDD v.s. DataFrame v.s. SparkSQL

- RDD:
  - 可直接讀取任何文字檔，需再進行加工
  - 需透過RDD operation進行程式開發
  - http://spark.apache.org/docs/2.1.0/api/python/pyspark.html#pyspark.RDD
- DataFrame:
  - 必須讀取結構化資料，讀進來後可以直接操作
  - 仍需使用DataFrame api進行操作
  - http://spark.apache.org/docs/2.1.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame

RDD v.s. DataFrame v.s. SparkSQL

- Saprk SQL
  - 由DataFrame產生tempTable
  - Spark 2.0 後，支援SQL:2003全部語法
- Performance



Creating DataFrames

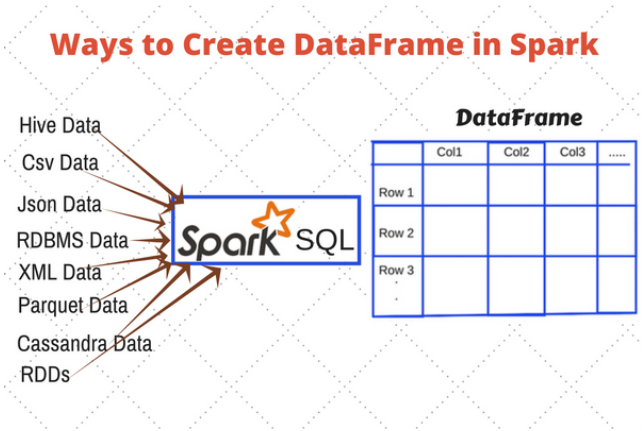
- From RDD

```
# Load a text file and convert each line to a Row.
lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))

# Infer the schema, and register the DataFrame as a table.
schemaPeople = spark.createDataFrame(people)
```

- From CSV/JSON … etc.

```
>>> df = spark.read.csv('python/test_support/sql/ages.csv')
>>> df.dtypes
[('_c0', 'string'), ('_c1', 'string')]
```

```
>>> df1 = spark.read.json('python/test_support/sql/people.json')
>>> df1.dtypes
[('age', 'bigint'), ('name', 'string')]
>>> rdd = sc.textFile('python/test_support/sql/people.json')
>>> df2 = spark.read.json(rdd)
>>> df2.dtypes
[('age', 'bigint'), ('name', 'string')]
```



DataFrames API

- Select

```
>>> df.select('*').collect()
[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]
>>> df.select('name', 'age').collect()
[Row(name=u'Alice', age=2), Row(name=u'Bob', age=5)]
>>> df.select(df.name, (df.age + 10).alias('age')).collect()
[Row(name=u'Alice', age=12), Row(name=u'Bob', age=15)]
```

- orderby

```
>>> df.sort(df.age.desc()).collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]
>>> df.sort("age", ascending=False).collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]
>>> df.orderBy(df.age.desc()).collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]
>>> from pyspark.sql.functions import *
>>> df.sort(asc("age")).collect()
[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]
>>> df.orderBy(desc("age"), "name").collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]
>>> df.orderBy(["age", "name"], ascending=[0, 1]).collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]
```

- groupBy

```
>>> df.groupBy().avg().collect()
[Row(avg(age)=3.5)]
>>> sorted(df.groupBy('name').agg({'age': 'mean'}).collect())
[Row(name=u'Alice', avg(age)=2.0), Row(name=u'Bob', avg(age)=5.0)]
>>> sorted(df.groupBy(df.name).avg().collect())
[Row(name=u'Alice', avg(age)=2.0), Row(name=u'Bob', avg(age)=5.0)]
>>> sorted(df.groupBy(['name', df.age]).count().collect())
[Row(name=u'Alice', age=2, count=1), Row(name=u'Bob', age=5, count=1)]
```

- filter

```
>>> df.filter(df.age > 3).collect()
[Row(age=5, name=u'Bob')]
>>> df.where(df.age == 2).collect()
[Row(age=2, name=u'Alice')]
```

```
>>> df.filter("age > 3").collect()
[Row(age=5, name=u'Bob')]
>>> df.where("age = 2").collect()
[Row(age=2, name=u'Alice')]
```

- join

```
>>> df.join(df2, df.name == df2.name, 'outer').select(df.name, df2.height).collect()
[Row(name=None, height=80), Row(name=u'Bob', height=85), Row(name=u'Alice', height=None)]
```

```
>>> df.join(df2, 'name').select(df.name, df2.height).collect()
[Row(name=u'Bob', height=85)]
```

Running SQL Queries

- Register DataFrames as a table first
  - <DF_NAME>.registerTempTable("<TABLE_NAME>")
- Run query
  - Ver. 2.0 : spark.sql("SELECT ….")
  - Ver. 1.6: sqlContext.sql(…)
- Result is returned as a DataFrame.

```
# Global temporary view is tied to a system preserved database `global_temp`
spark.sql("SELECT * FROM global_temp.people").show()
```

```
teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")
```

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 Streaming
  - Spark SQL & DataFrames
  - SparkR

What is SparkR

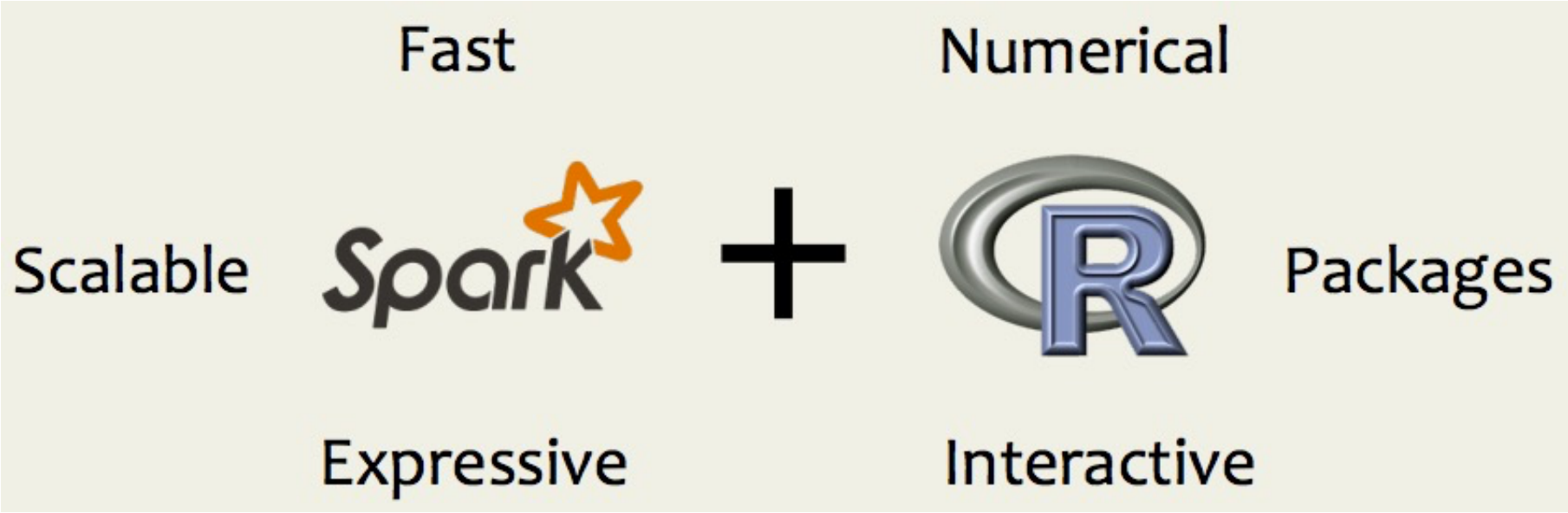
- light-weight frontend to use Spark from R
- provides R-like dataframe implementation for large dataset
- Integrated with Spark since Spark 1.4
  - Ver. 1.4: 66 functions
  - Ver. 1.5: 197 functions
  - Ver. 1.6: 225 functions

https://people.csail.mit.edu/matei/papers/2016/sigmod_sparkr.pdf
http://dataology.blogspot.tw/2016/05/sparkr.html

Run SparkR on Cluster

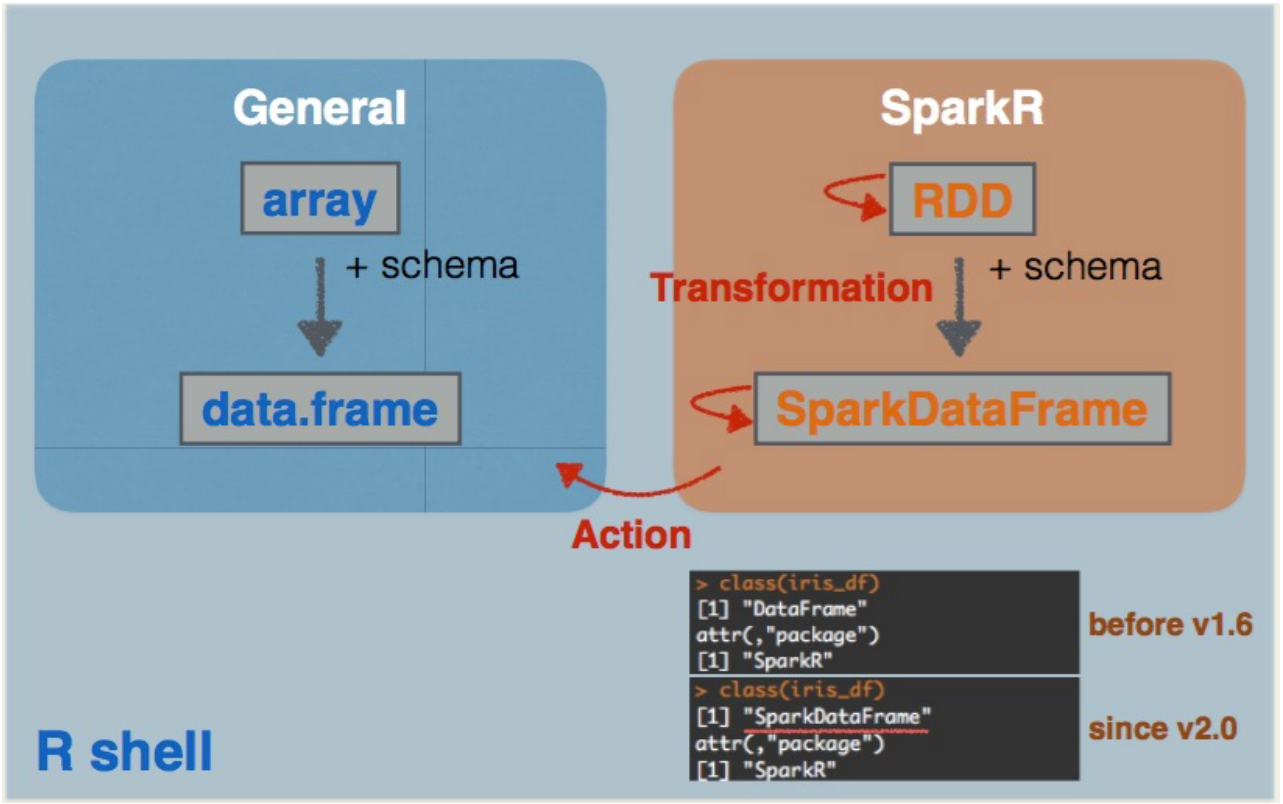
- Step 1: 每台worker node都安裝R
- Step 2: 依照Part-III啟動Spark Cluster (Standalone or YARN or MESOS)
- Step 3:
  - $ ./sparkR --master **spark://spark-master:7077**
  - $ ./sparkR --master **yarn**
  - $ ./sparkR --master **mesos://master:5050**

Key Advantages of Spark & R

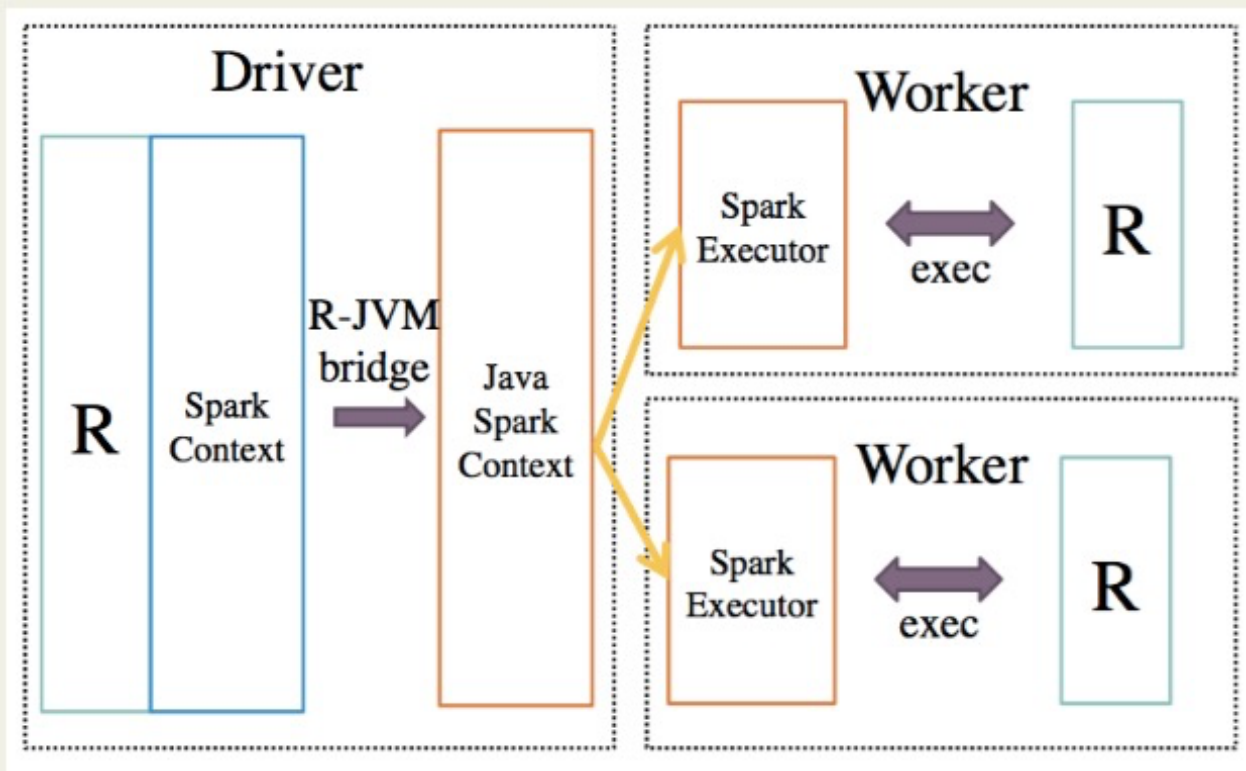


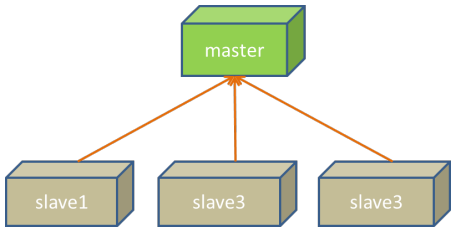
SparkR 互動介面

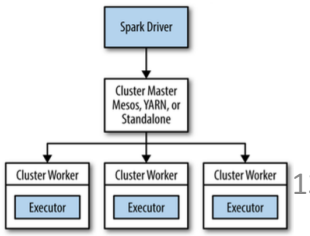
- 在單一的R Sehll (或 R Studio) 上同時處理SparkDataFrame與R datafrmae



How does SparkR works ?



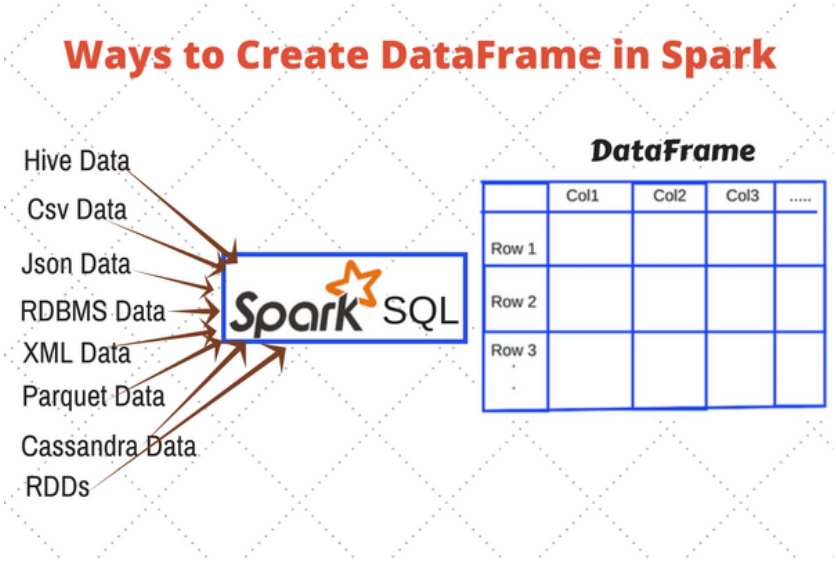




Creating SparkDataFrames

- From CSV/JSON … etc.

```
df <- read.df(csvPath, "csv", header = "true", inferSchema = "true", na.strings = "NA")
```



Basic SparkDataFrames API

- select

```
select(df, "*")
select(df, "col1", "col2")
select(df, df$name, df$age + 1)
select(df, c("col1", "col2"))
select(df, list(df$name, df$age + 1))
# Similar to R data frames columns can also be selected using $
df[,df$age]
```

- arrange

```
path <- "path/to/file.json"
df <- read.json(path)
arrange(df, df$col1)
arrange(df, asc(df$col1), desc(abs(df$col2)))
arrange(df, "col1", decreasing = TRUE)
```

- groupBy

```
# Compute the average for all numeric columns grouped by department.
avg(groupBy(df, "department"))

# Compute the max age and average salary, grouped by department and gender.
agg(groupBy(df, "department", "gender"), salary="avg", "age" -> "max")
```

- filter

```
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
filter(df, "col1 > 0")
filter(df, df$col2 != "abcdefg")
```

Running SQL Queries

- Register SparkDataFrames as a table first
  - createOrReplaceTempView(“${DF}”,“${TABLE}")
- Run query
  - Result <-sql(“SELECT …. ”)

```
# Register this SparkDataFrame as a temporary view.
createOrReplaceTempView(people, "people")

# SQL statements can be run by using the sql method
teenagers <- sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")
```

SparkR 線性迴歸模型: spark.glm()

- glm() in R

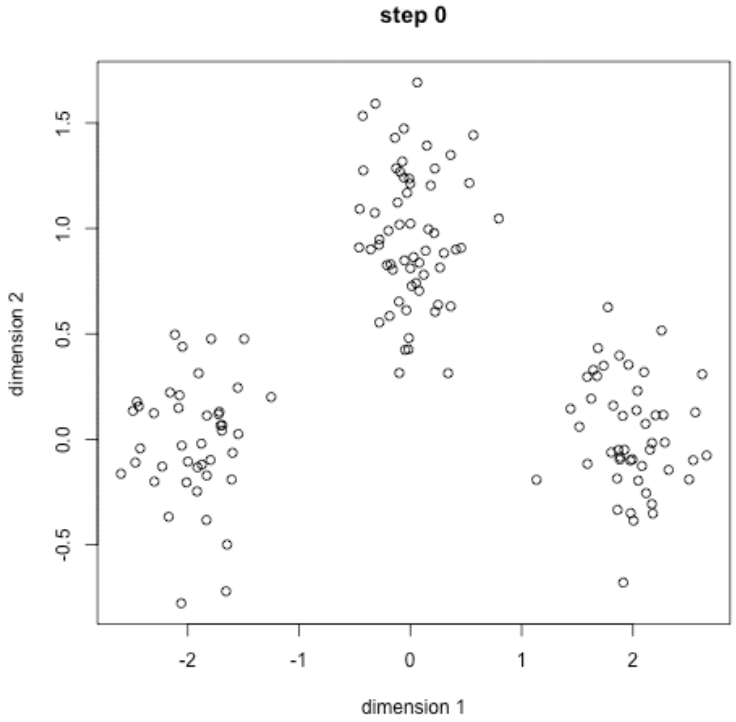
```
# Poisson Regression
# where count is a count and
# x1-x3 are continuous predictors
fit <- glm(count ~ x1+x2+x3, data=mydata, family=poisson())
summary(fit) display results
```

- glm() in SparkR

```
df <- createDataFrame(iris)
model <- spark.glm(df, Sepal_Length ~ Sepal_Width, family = "gaussian")
summary(model)
```

Lab : 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)



- 透過讀CSV檔產生DataFrame
- 用SparkR 的kmeans進行分群
  - Spark.kmeans API Sample
    - https://spark.apache.org/docs/2.0.0/sparkr.html#kmeans-model
- 畫出分群的結果

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 Streaming
  - Spark SQL & DataFrames
  - SparkR
  - spark.mllib & spark.ml

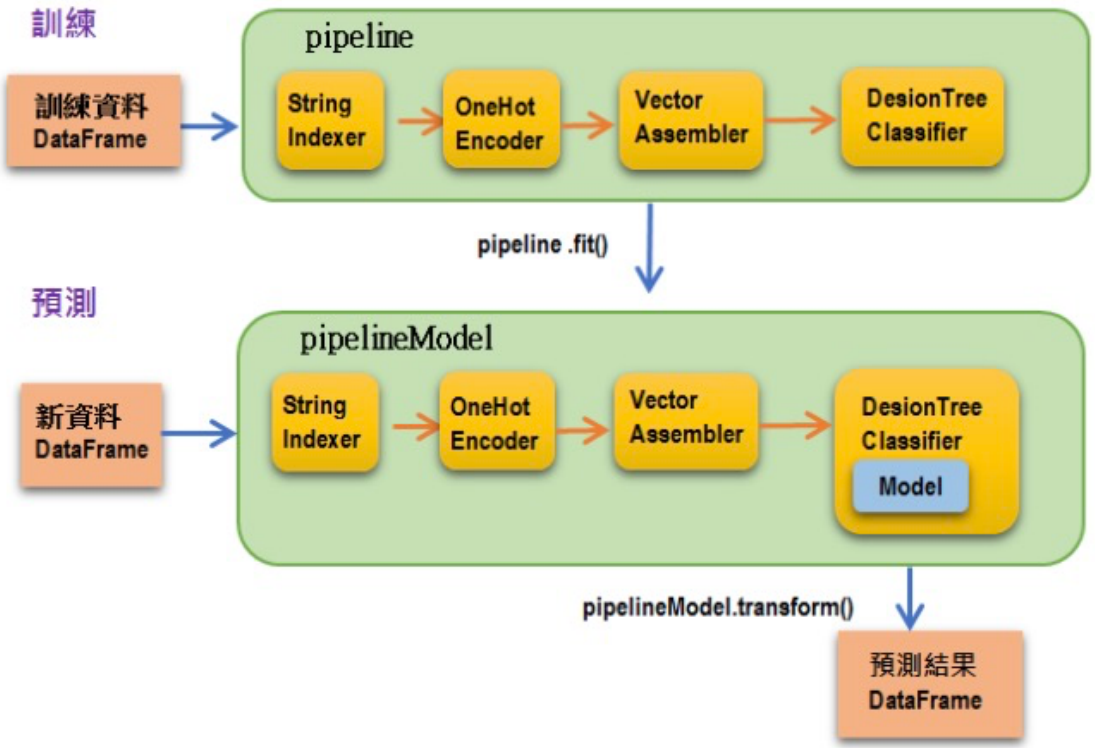
Spark Machine Learning Library

- MLlib RDD-based API is in maintenance mode
- ML DataFrame-based API is primary API
  - Pipeline: 建立ML的工作流程

	Spark.mllib	Spark.ml
Since	Ver. 0.8~	Ver. 1.2 ~
Datatype	RDD	DataFrame
API	contains the original API built on top of RDDs.	higher-level API built on top of DataFrames
優點	發展較早，演算法較多	結構化資料，資料操作較簡單

Spark.ML

- 以DataFrame為基礎的機器學習模組
  - Spark DataFrame: 受Pandas啟發的資料處理架構
  - Spark ML pipeline: 受Scikit-learn啟發的ML架構



資料前處理工具

- 不是每一種前處理都需要，視資料狀況混合使用
- StringIndexer
  - 將文字欄位，轉成數字
  - 男→0，女→1
- OneHotEncoder
  - 將數字欄位轉成多個欄位的向量
  - 0→[1,0]，1→[0,1]
- VectorAssembler
  - 將多個欄位組成一個特徵向量
  - 男，180cm → [1,0][180] → [1,0,180]

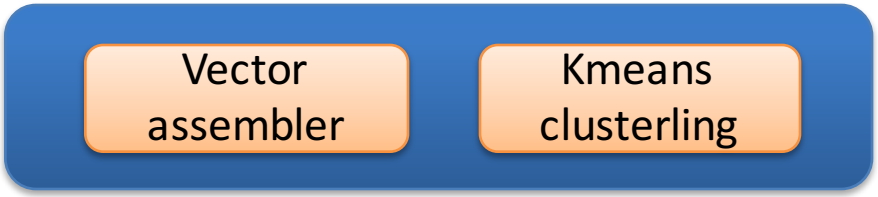
Pipeline

- 提供統一的訓練與預測語法
- pipeline使用fit()進行訓練，產生model
- model使用transform進行預測

Lab5: k-means pipeline

- 只需要vector assembler

```
+-------------------+--------------------+
|                  x|                   y|
+-------------------+--------------------+
|-1.0789262655979264|-0.29058890178264374|
|-1.5631374488292438|-0.01310586267402436|
|-1.2299535180277972|-0.10703168514429007|
|-1.2867842461628365| 0.45799739022887676|
|-2.9858188383793522|-0.15985872797137646|
+-------------------+--------------------+
```




```
+-------------------+--------------------+----------------------------------------+
|x                  |y                   |features                                |
+-------------------+--------------------+----------------------------------------+
|-1.0789262655979264|-0.29058890178264374|[-1.0789262655979264,-0.29058890178264374]|
|-1.5631374488292438|-0.01310586267402436|[-1.5631374488292438,-0.01310586267402436]|
|-1.2299535180277972|-0.10703168514429007|[-1.2299535180277972,-0.10703168514429007]|
|-1.2867842461628365|0.45799739022887676 |[-1.2867842461628365,0.45799739022887676] |
|-2.9858188383793522|-0.15985872797137646|[-2.9858188383793522,-0.15985872797137646]|
+-------------------+--------------------+----------------------------------------+
```

```
+-------------------+--------------------+----------------------------------------+----------+
|x                  |y                   |features                                |prediction|
+-------------------+--------------------+----------------------------------------+----------+
|-1.0789262655979264|-0.29058890178264374|[-1.0789262655979264,-0.29058890178264374]|0         |
|-1.5631374488292438|-0.01310586267402436|[-1.5631374488292438,-0.01310586267402436]|0         |
|-1.2299535180277972|-0.10703168514429007|[-1.2299535180277972,-0.10703168514429007]|0         |
|-1.2867842461628365|0.45799739022887676 |[-1.2867842461628365,0.45799739022887676] |0         |
|-2.9858188383793522|-0.15985872797137646|[-2.9858188383793522,-0.15985872797137646]|0         |
```

RDD-based mllib缺點

- 需要針對讀進來的RDD資料進行前處理
- 不同演算法的RDD[T]都不同
  - DecisionTree: RDD[LabeledPoint]
  - Kmean: RDD[array]

- 不同的演算法所用的訓練與預測api皆不同
  - DecisionTree.trainClassifier()
  - DecisionTree.trainRegressor()
  - Kmeans().train()

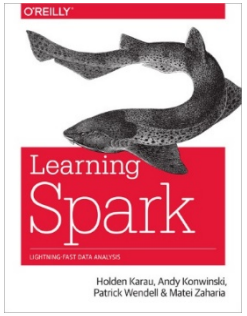
LabelPoint

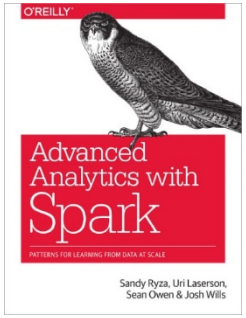
- 透過RDD.map()產生出演算法所需要用的資料結構

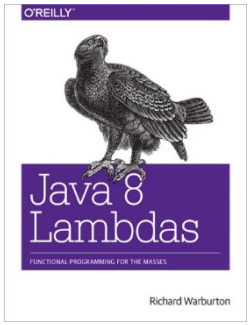
Lab4: k-means

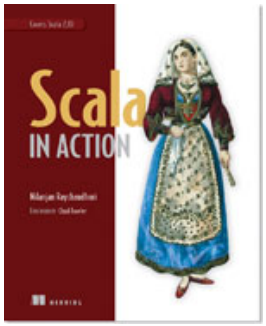
- 使用map()將RDD[tuple]轉化成RDD[array]
  - RDD[(x,y)] → RDD[[x,y]]
- 使用Kmean.train()進行訓練

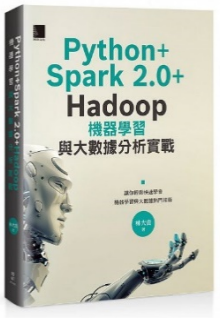
Reference











- 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
- PySpark
  - Python+Spark 2.0+Hadoop機器學習與大數據分析實戰