MBIT School
Madrid Business Intelligence Technology

Apache Spark

Diego J. Bodas Sagi

Índice

MBIT School
Madrid Business Intelligence Technology

- Introdución a Spark
- Características básicas
- Spark vs Hadoop
- Scala y Java
- RDDs
- Arquitectura
- Código
- Resumen

¿Qué necesidad tenemos?

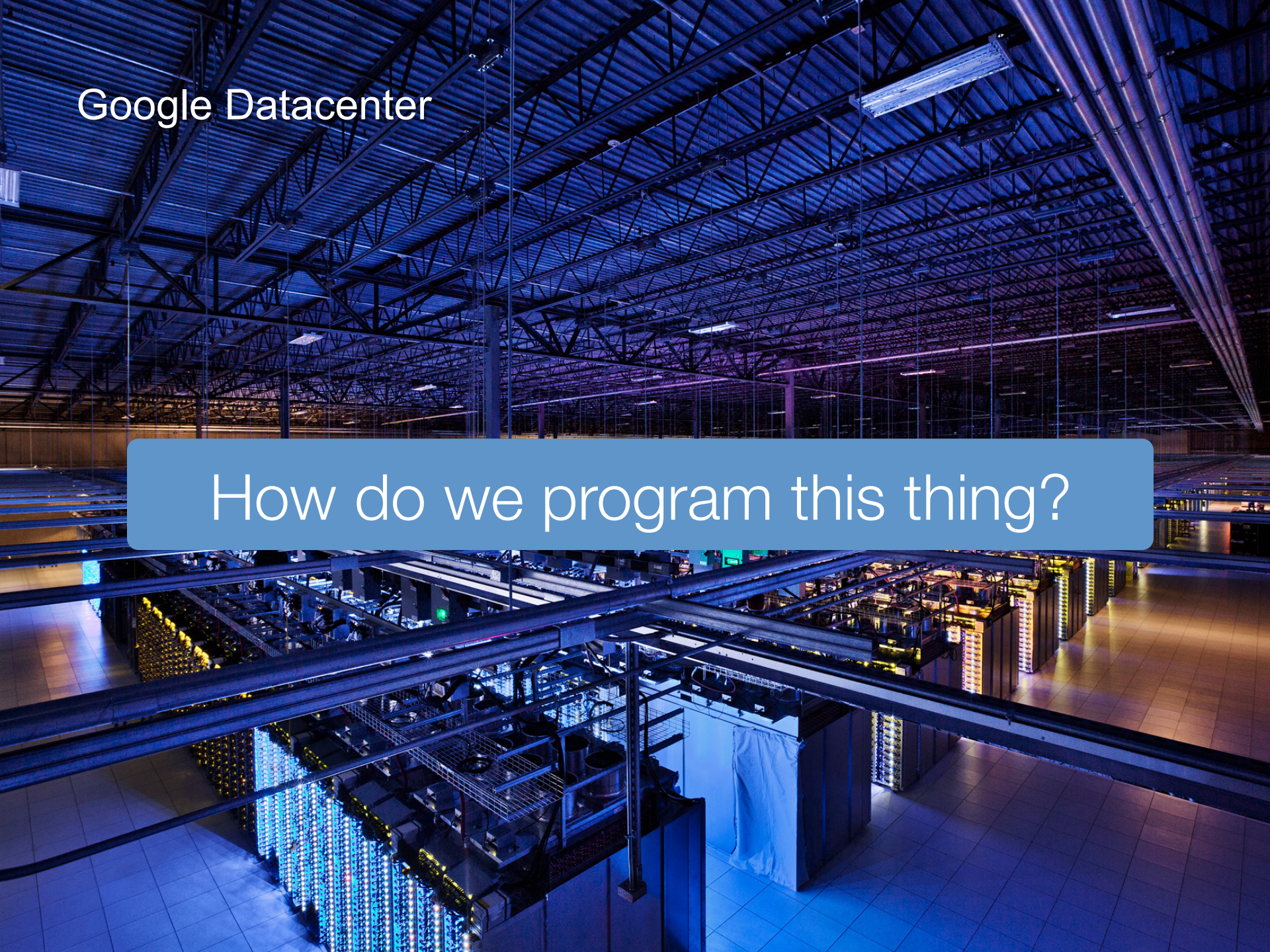
MBIT School
Madrid Business Intelligence Technology

Extender el modelo de MapReduce para dotarlo de más potencia fomentando el uso de:

- ✓ Iterative algorithms (machine learning, graph)
- ✓ Interactive data mining

Mejorar en entorno de programación:

- ✓ Integrate into Scala programming language
- ✓ Allow interactive use from Scala interpreter


Google Datacenter
How do we program this thing?

Situación previa

Most current cluster programming models are based on directed acyclic data flow from stable storage to stable storage

Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures

Situación previa

MBIT School
Madrid Business Intelligence Technology

Este tipo de esquema es ineficiente en contextos donde se usa de forma iterativa el mismo conjunto de datos:

- ✓ Iterative algorithms (machine learning, graphs)
- ✓ Interactive data mining tools (R, Excel, Python)

En el modelo hadoop es preciso cargar el dataset para cada Job (aunque parte de la información esté en memoria caché)

Propuesta

MBIT School
Madrid Business Intelligence Technology

Berkeley 2009
UNIVERSITY OF CALIFORNIA

spark.incubator.apache.org

github.com/apache/incubator-spark

Spark

user@spark.incubator.apache.org

Generalizing the MapReduce Framework

¿Qué es Spark?

MBIT School
Madrid Business Intelligence Technology

Fast and **Expressive** Cluster Computing System
Compatible with Apache Hadoop

Up to **10x** faster on disk,
100x in memory

2-5x less code

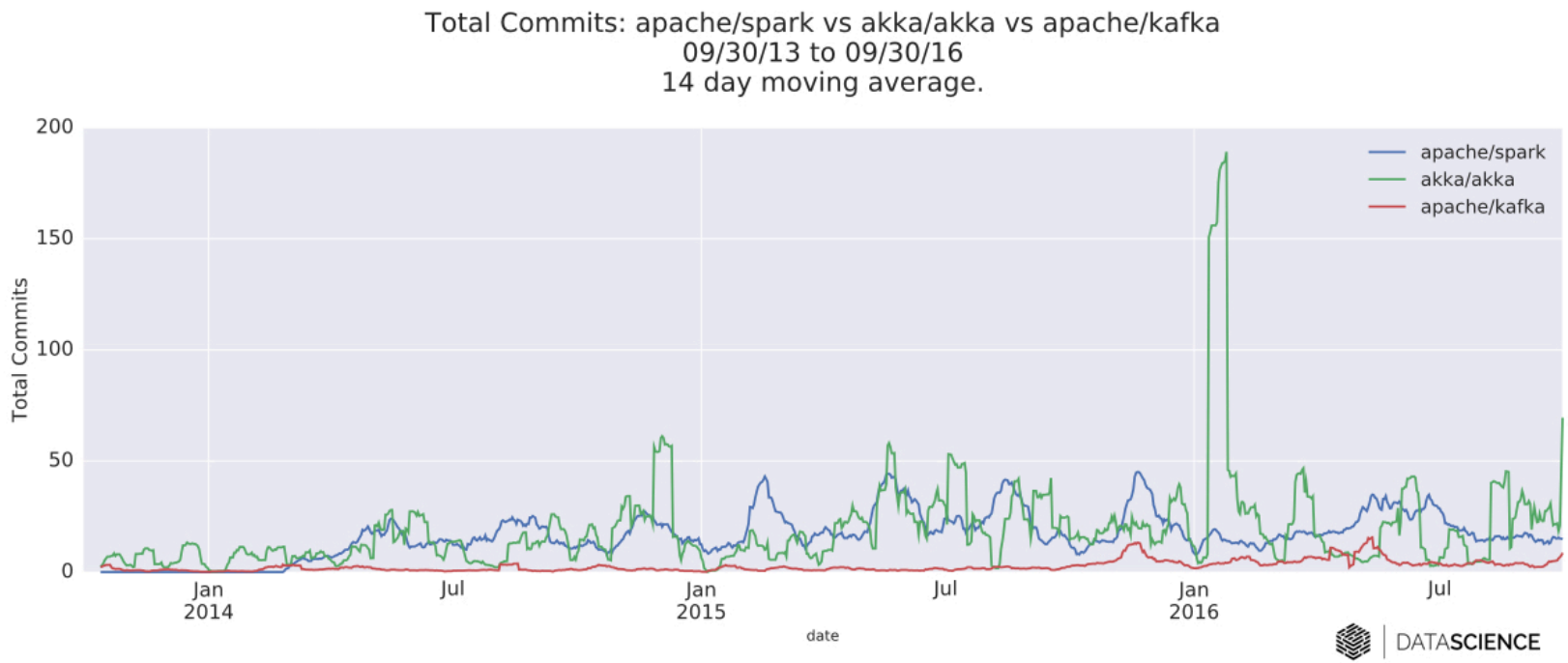
Efficient

- General execution graphs
- In-memory storage

Usable

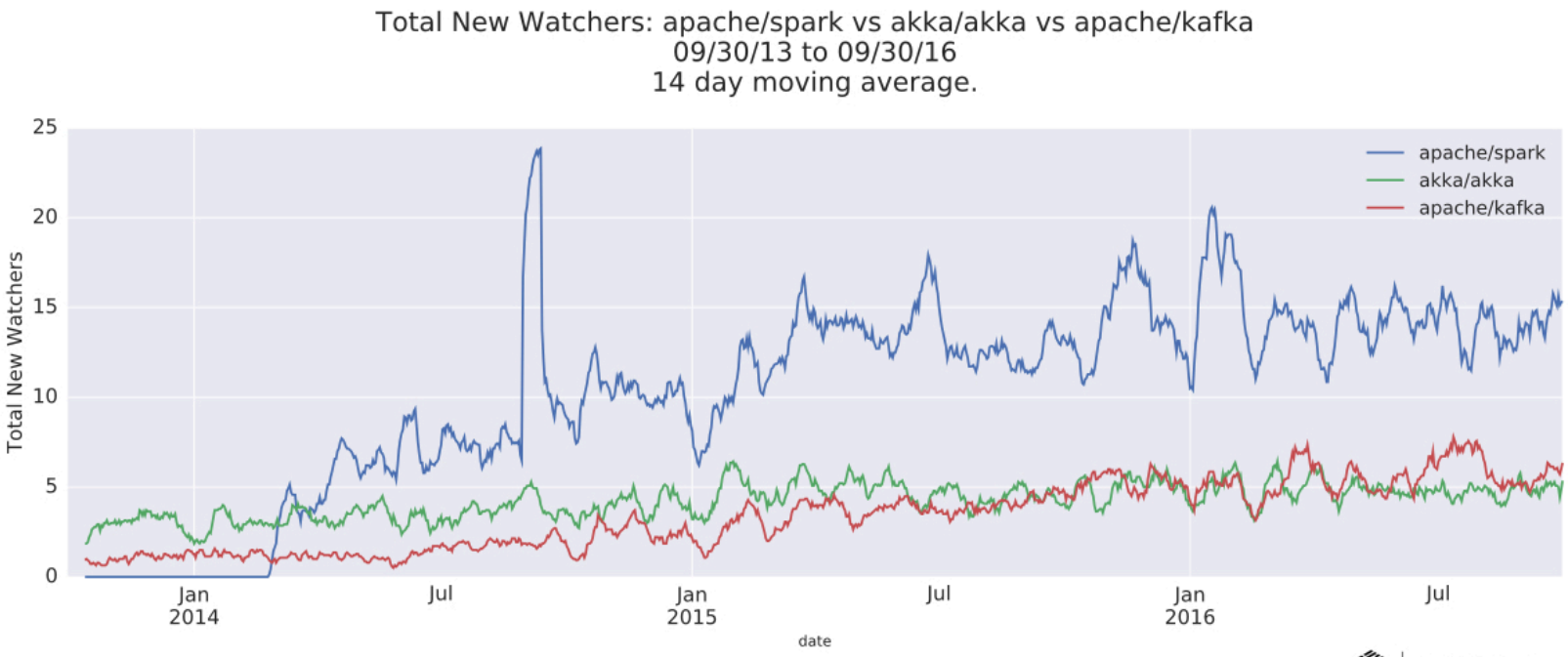
- Rich APIs in Java, Scala, Python
- Interactive shell

The Spark Community



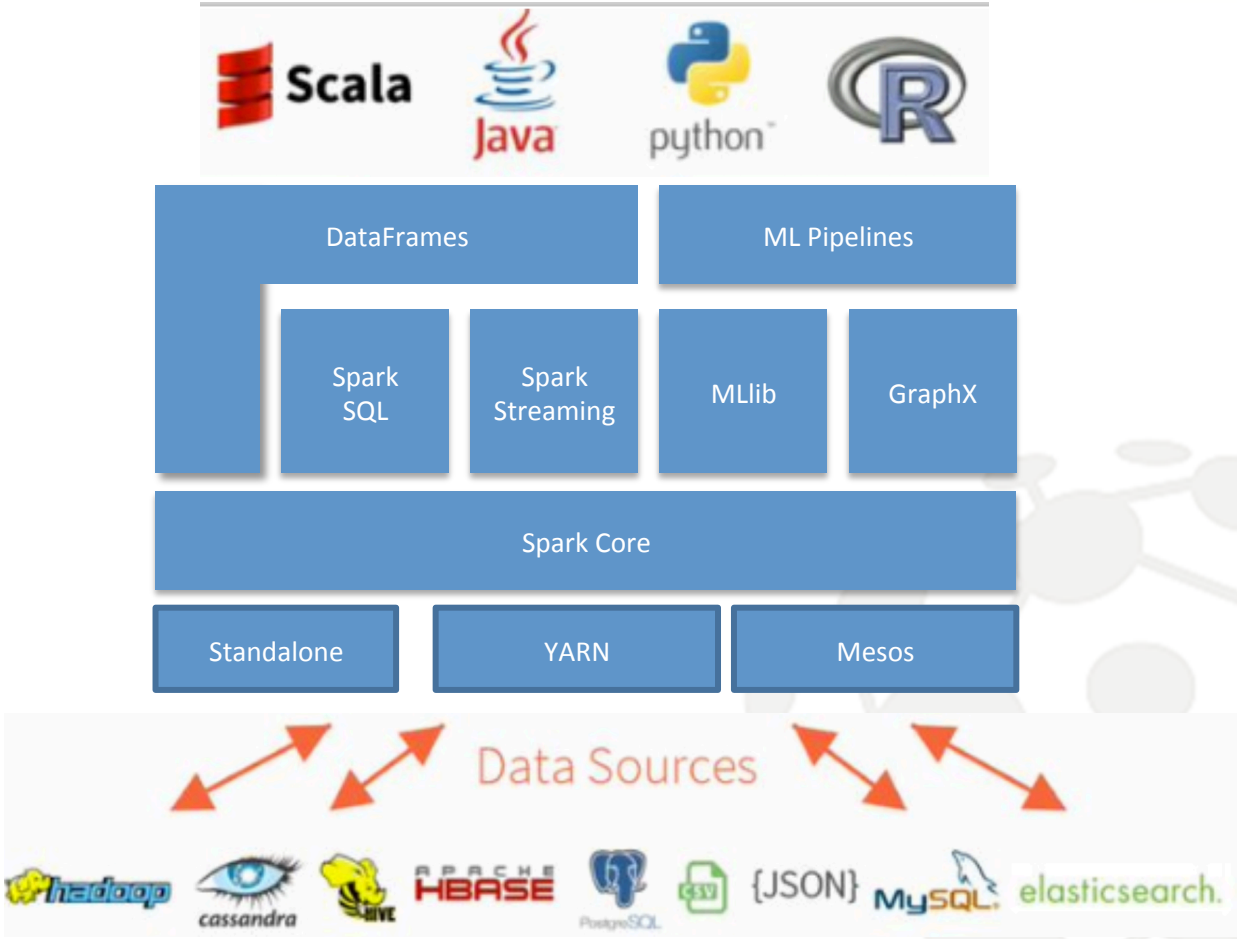
Apache Spark

Interés



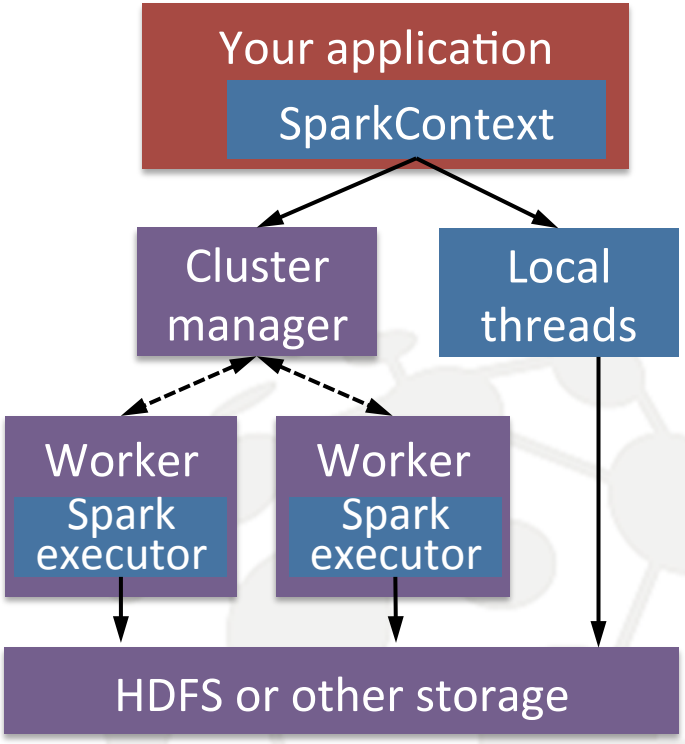
DATASCIENCE

Ecosistema

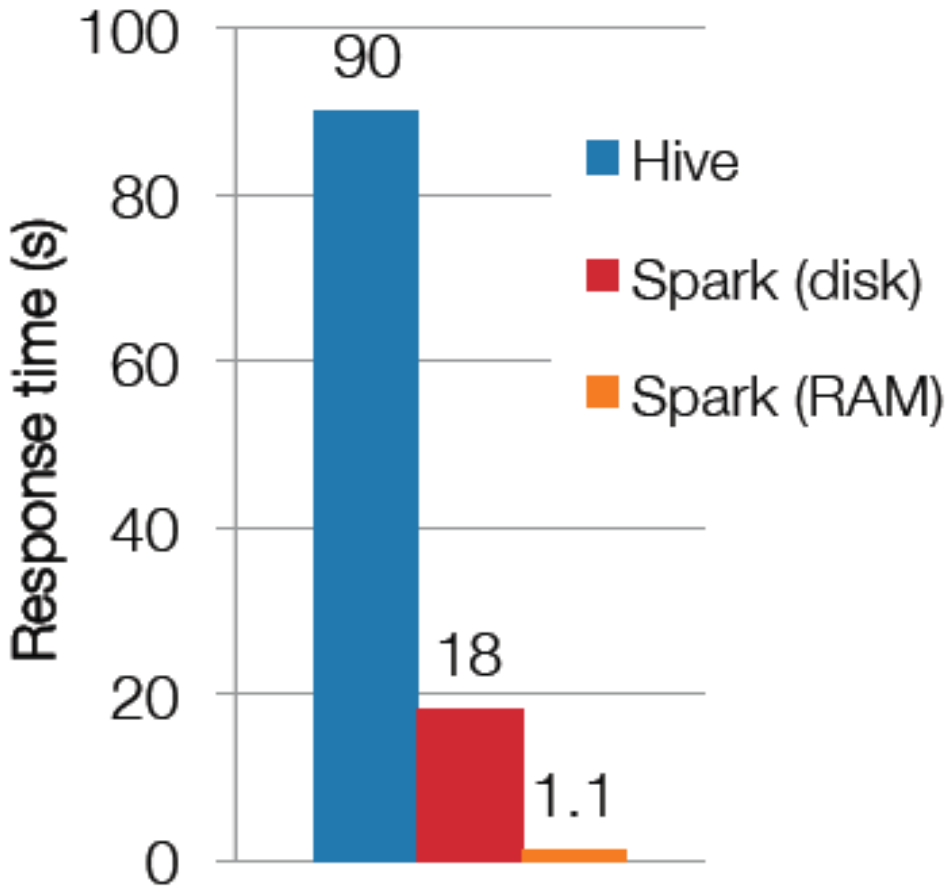

Scala
Java
python
R
DataFrames
ML Pipelines
Spark SQL
Spark Streaming
MLlib
GraphX
Spark Core
Standalone
YARN
Mesos
Data Sources
hadoop
cassandra
HBASE
{JSON}
MySQL
elasticsearch.

Arquitectura

- Repositorios posibles:
  - HDFS
  - Hbase
  - S3
  - …

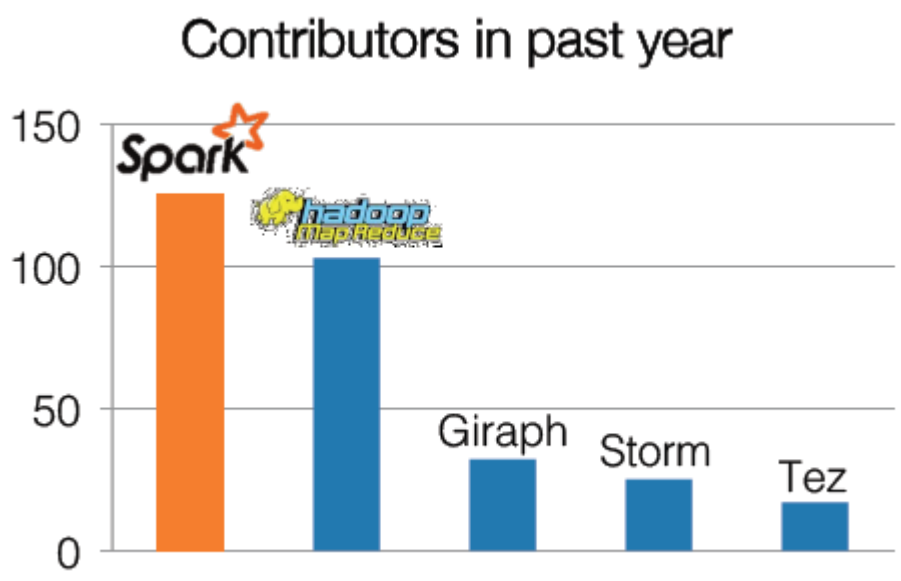


Shark vs Hive

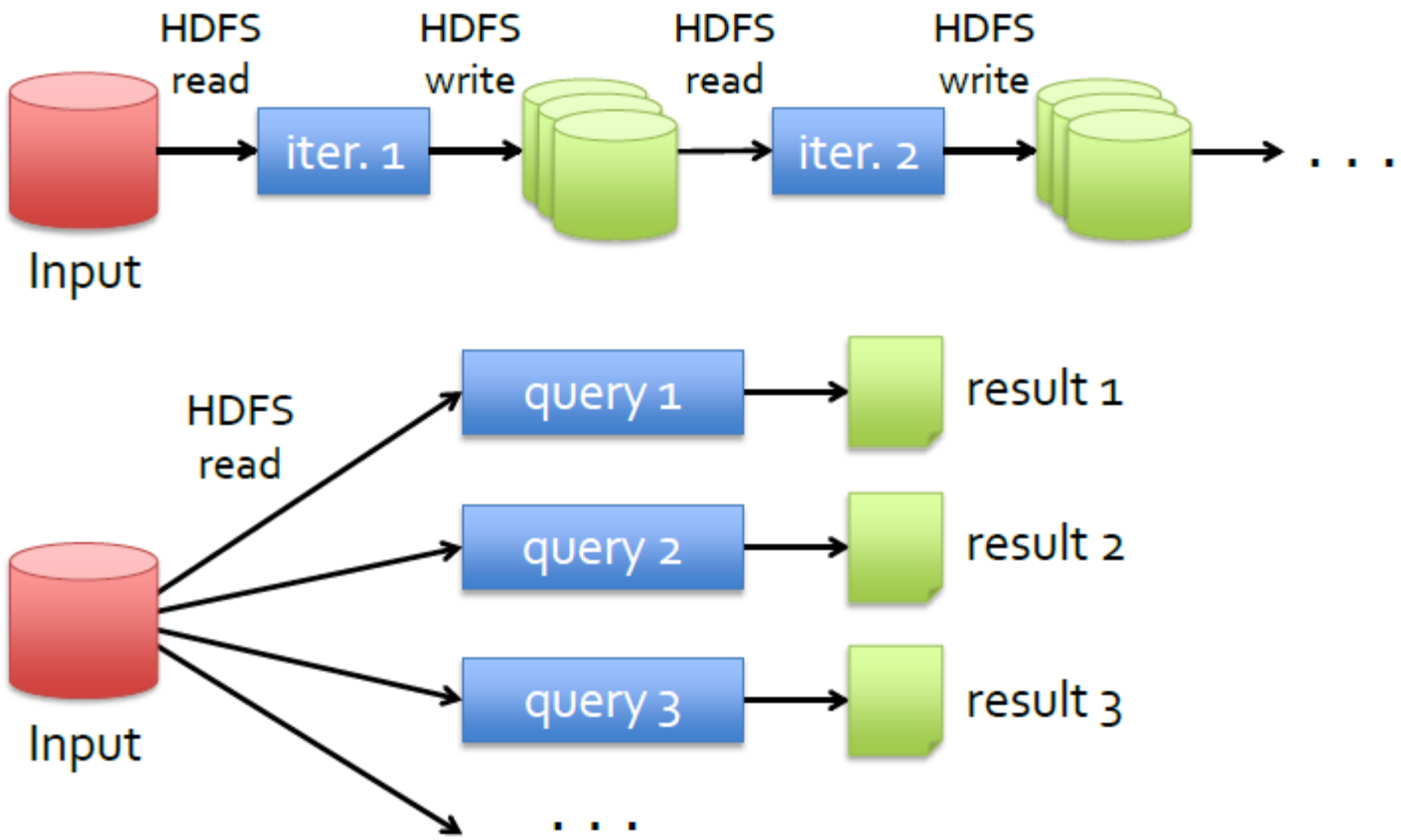


Spark vs Hadoop

- Aprox.
  - 100 veces más rápido para procesos en memoria
  - 10 veces más rápido en disco
- Proyecto que crece rápidamente

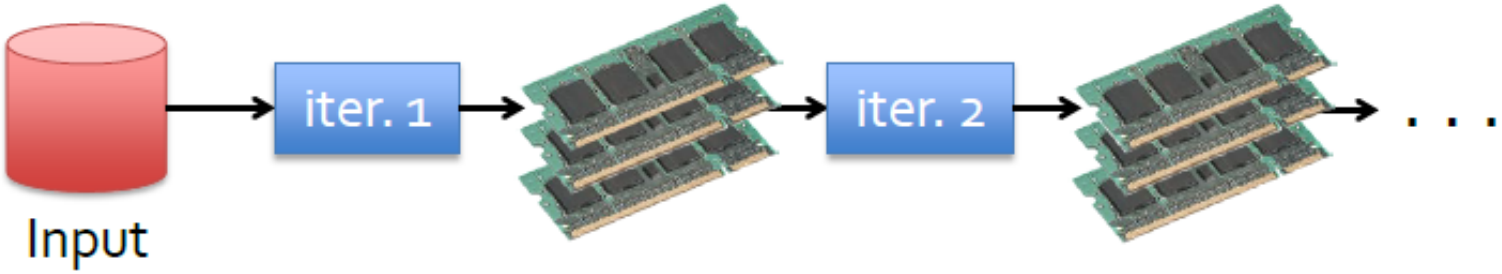


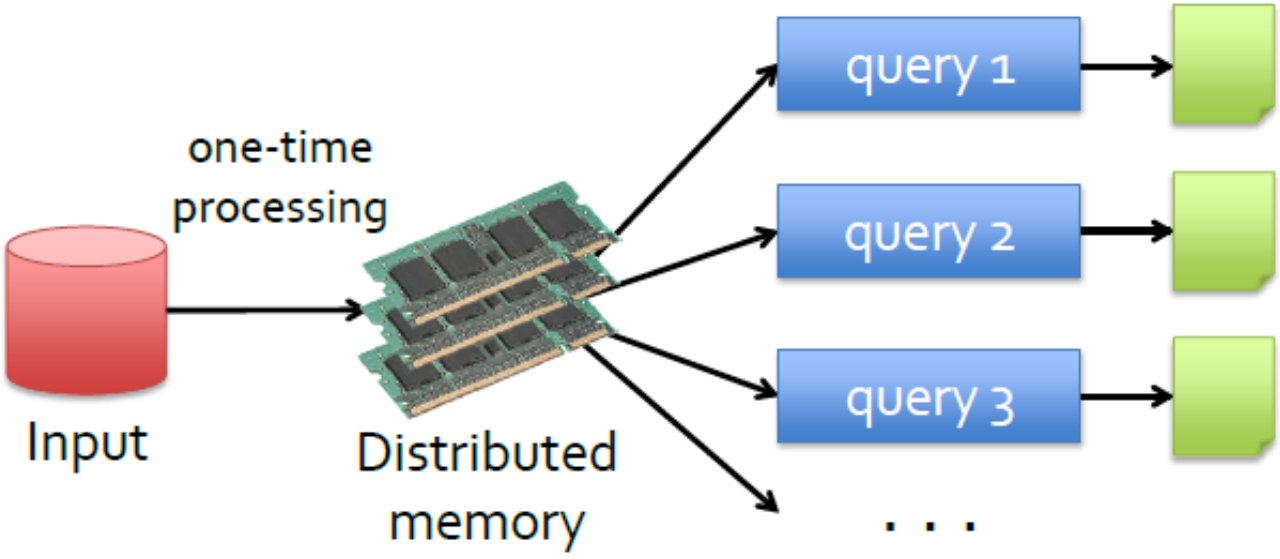
Gestión de datos en Hadoop



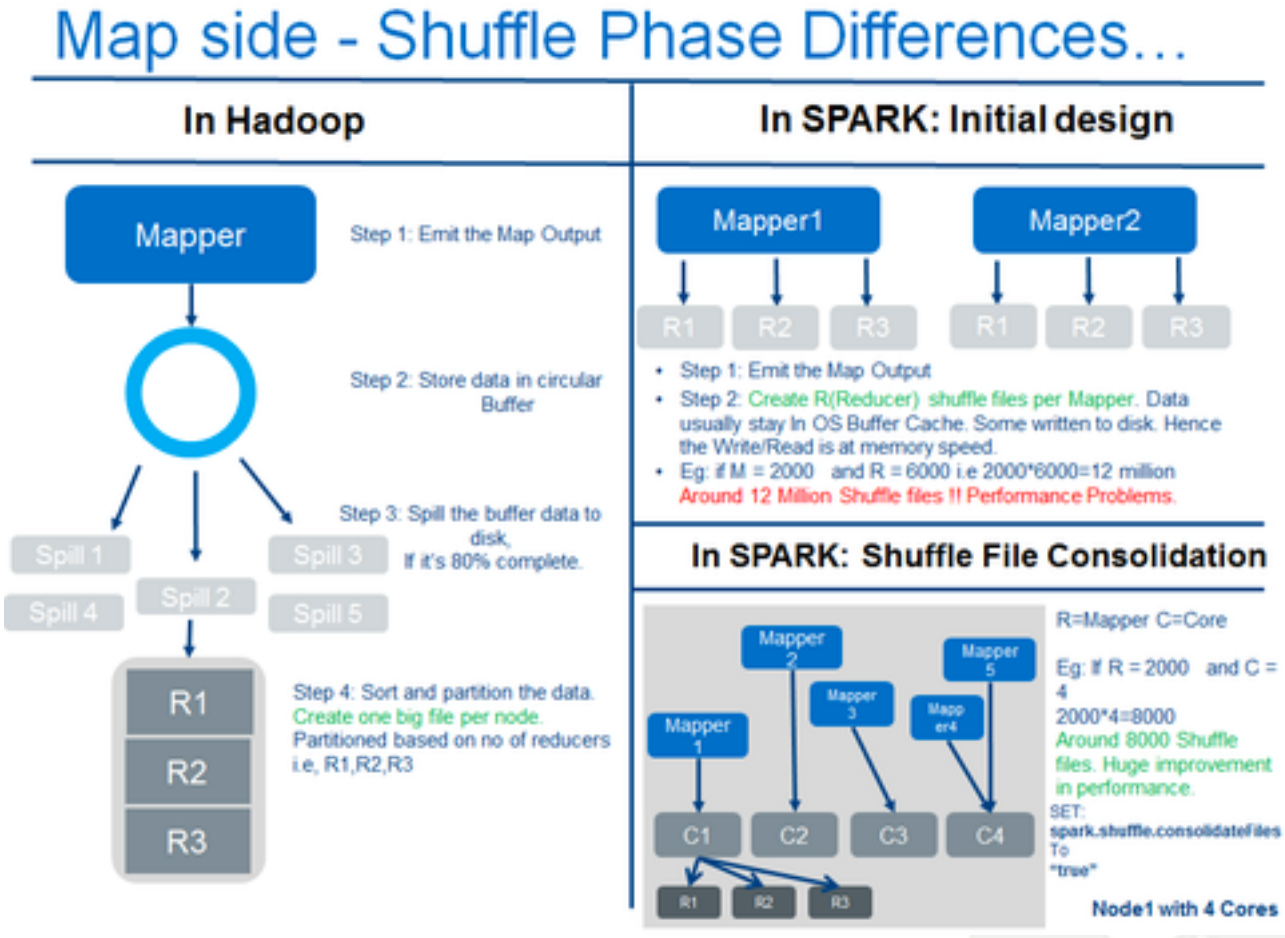
Cuellos de botella: replicación, serialización, operaciones IO

Gestión de datos en Spark

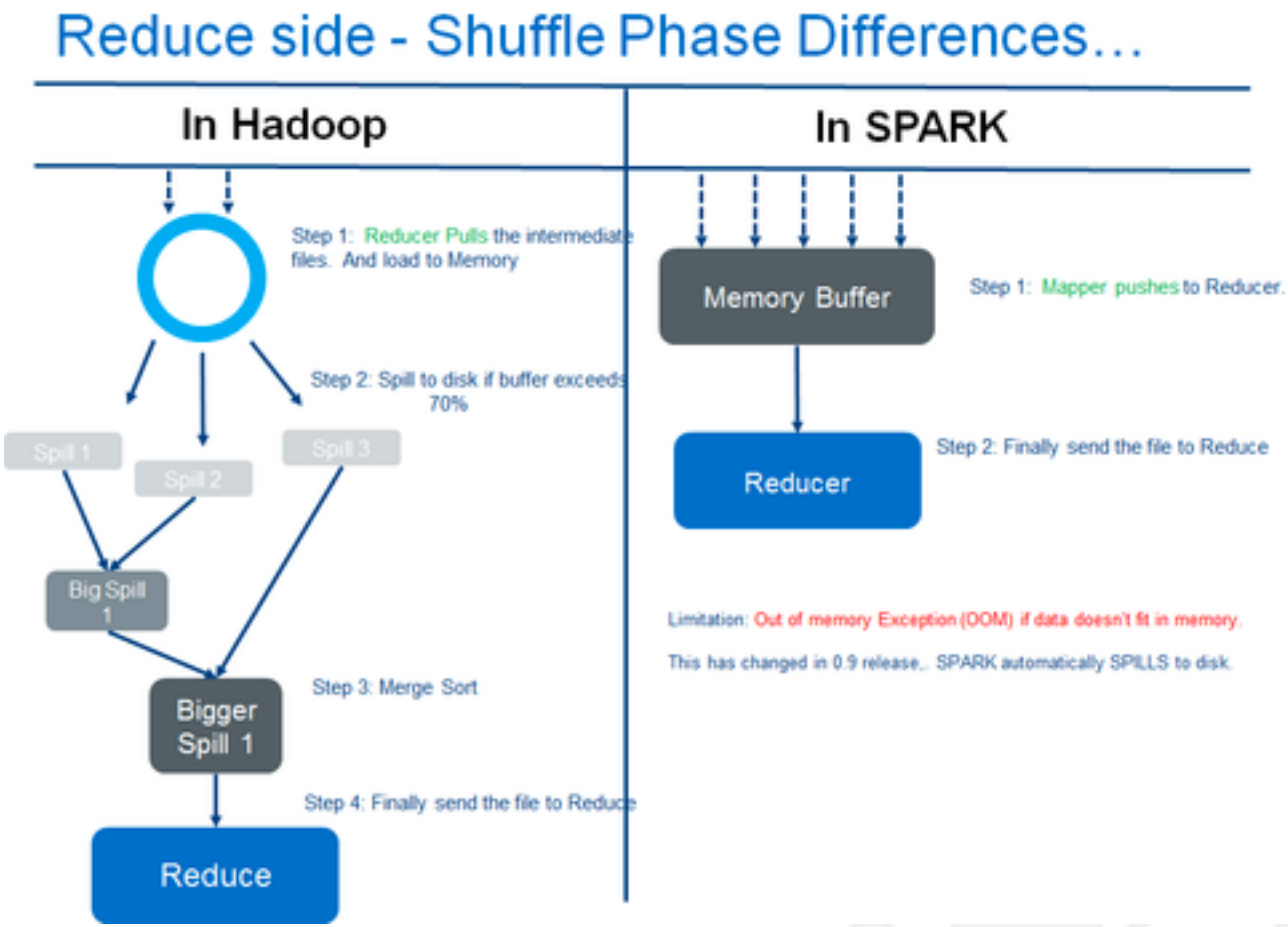




Fase Mapper


Map side - Shuffle Phase Differences…
In Hadoop
Mapper
Step 1: Emit the Map Output
Step 2: Store data in circular Buffer
Step 3: Spill the buffer data to disk, If it's 80% complete.
Spill 1
Spill 2
Spill 3
Spill 4
Spill 5
R1
R2
R3
Step 4: Sort and partition the data. Create one big file per node. Partitioned based on no of reducers i.e, R1,R2,R3
In SPARK: Initial design
Mapper1
Mapper2
R1
R2
R3
R1
R2
R3
Step 1: Emit the Map Output
Step 2: Create R(Reducer) shuffle files per Mapper. Data usually stay In OS Buffer Cache. Some written to disk. Hence the Write/Read is at memory speed.
Eg: if M = 2000 and R = 6000 i.e 2000*6000=12 million
Around 12 Million Shuffle files !! Performance Problems.
In SPARK: Shuffle File Consolidation
Mapper 1
Mapper 2
Mapper 3
Mapper 4
Mapper 5
C1
C2
C3
C4
R1
R2
R3
R=Mapper C=Core
Eg: If R = 2000 and C = 4
2000*4=8000
Around 8000 Shuffle files. Huge improvement in performance.
SET: spark.shuffle.consolidateFiles To "true"
Node1 with 4 Cores

Fase Reducer



Pero recuerda…

MBIT School
Madrid Business Intelligence Technology

La tecnología no es la respuesta
Es una herramienta que cambia constantemente

Casos de uso

- Yahoo: personalización de la web
  - Modelos para encontrar los artículos que interesan a cada usuario
  - Continuamente se están generando nuevas publicaciones
  - 120 líneas en Scala vs 15 mil líneas de código en C++
  - En 30 minutos ejecutan 100 millones de muestras

Conviva

- Gestiona más de 4 billones de vídeo streams al mes
- Selección dinámica de fuentes a optimizar
- Depende mucho del factor tiempo, el usuario no quiere tener que esperar a que se vayan cargando los extractos del vídeo

Scala vs Java

MBIT School
Madrid Business Intelligence Technology

Similitudes

- Lenguajes basados en la JVM, ambos producen el mismo byte code
- Sencilla integración Java – Scala y viceversa
- Soporte en multitud de IDEs
- Orientados a objetos

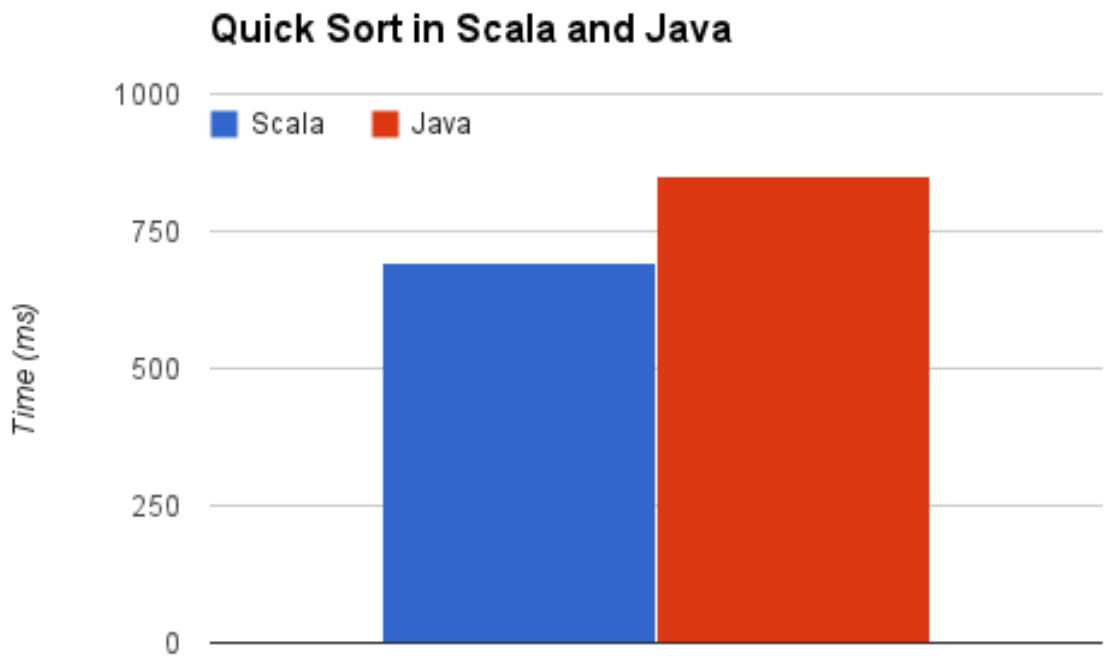
Diferencias

- En Scala el código es más breve y conciso
- Scala está pensado para implementar de forma sencilla los principales patrones de diseño
- Diferente curva de aprendizaje, los programadores Java necesitan un tiempo para acostumbrarse al nuevo entorno

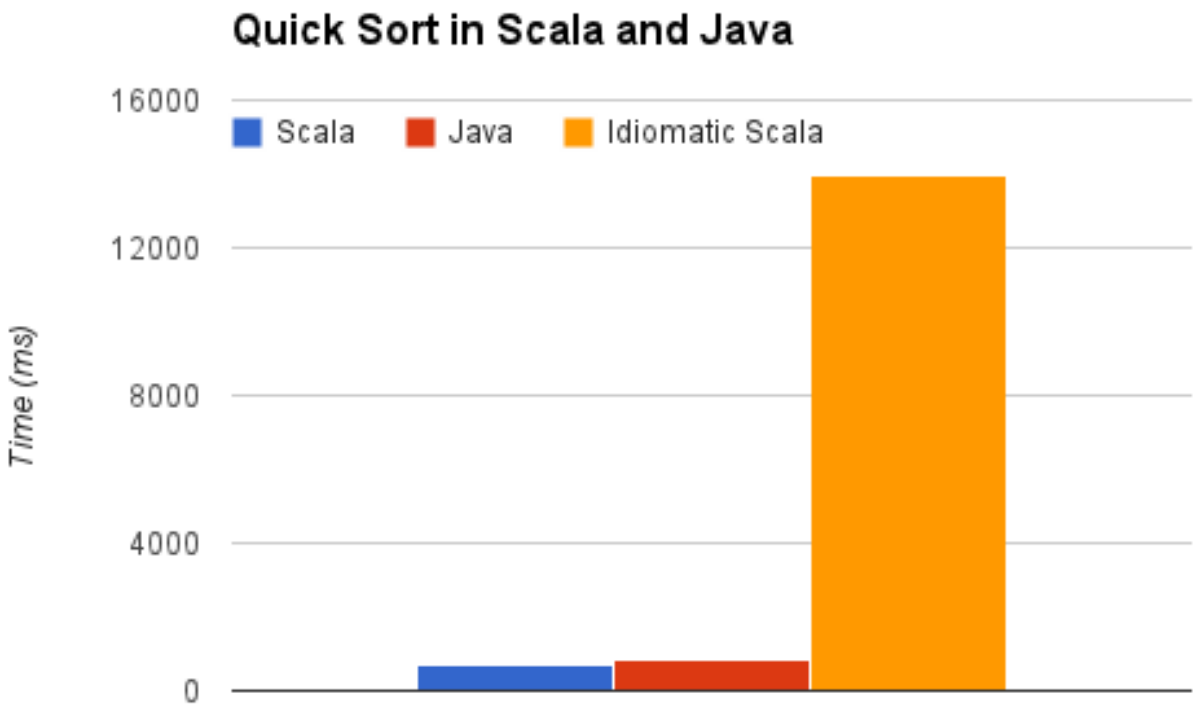
Más diferencias

- Scala emplea evaluación perezosa (lazy evaluation)
  - Permite reducir el tiempo de cómputo
  - Retrasa el cálculo de una expresión hasta que su valor sea necesario
  - Evita repetir la evaluación compartiendo el cálculo
- En ocasiones, Java parece un lenguaje de programación más claro que Scala
- Java no proporciona total libertad para sobrecargar el comportamiento de algunos operadores (como los aritméticos), Scala sí

Respecto al rendimiento



¡Cuidado que hay trampa!



Código Scala: optimizado "común"

Conclusión: Scala vs Java

MBIT School
Madrid Business Intelligence Technology

- Si eres un buen programador, Scala puede ayudar a mejorar de forma importante el tiempo de ejecución de tu código
- En caso contrario, mejor con Java

Conceptos clave

MBIT School
Madrid Business Intelligence Technology

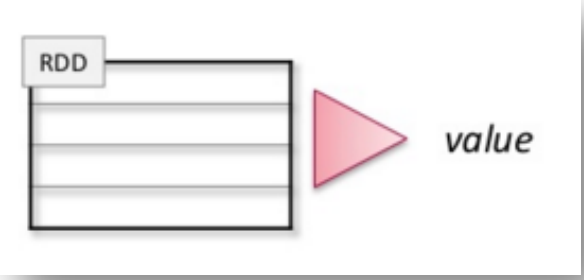
Resilient Distributed Datasets

- Collections of objects spread across a cluster, stored in RAM or on Disk
- Built through parallel transformations
- ***Automatically rebuilt on failure***

Operations

- Transformations (e.g. map, filter, groupBy)
- Actions (e.g. count, collect, save)

Operations



count, function(n)

Filters, maps

RDD Operations

MBIT School
Madrid Business Intelligence Technology

Transformations

>> create a new dataset from an existing one

Actions

>> Return a value to the driver program

Transformations are *lazy, they don't compute right away. Just remember the transformations applied to datasets(lineage). Only compute when an action require.*

Transformations

MBIT School
Madrid Business Intelligence Technology

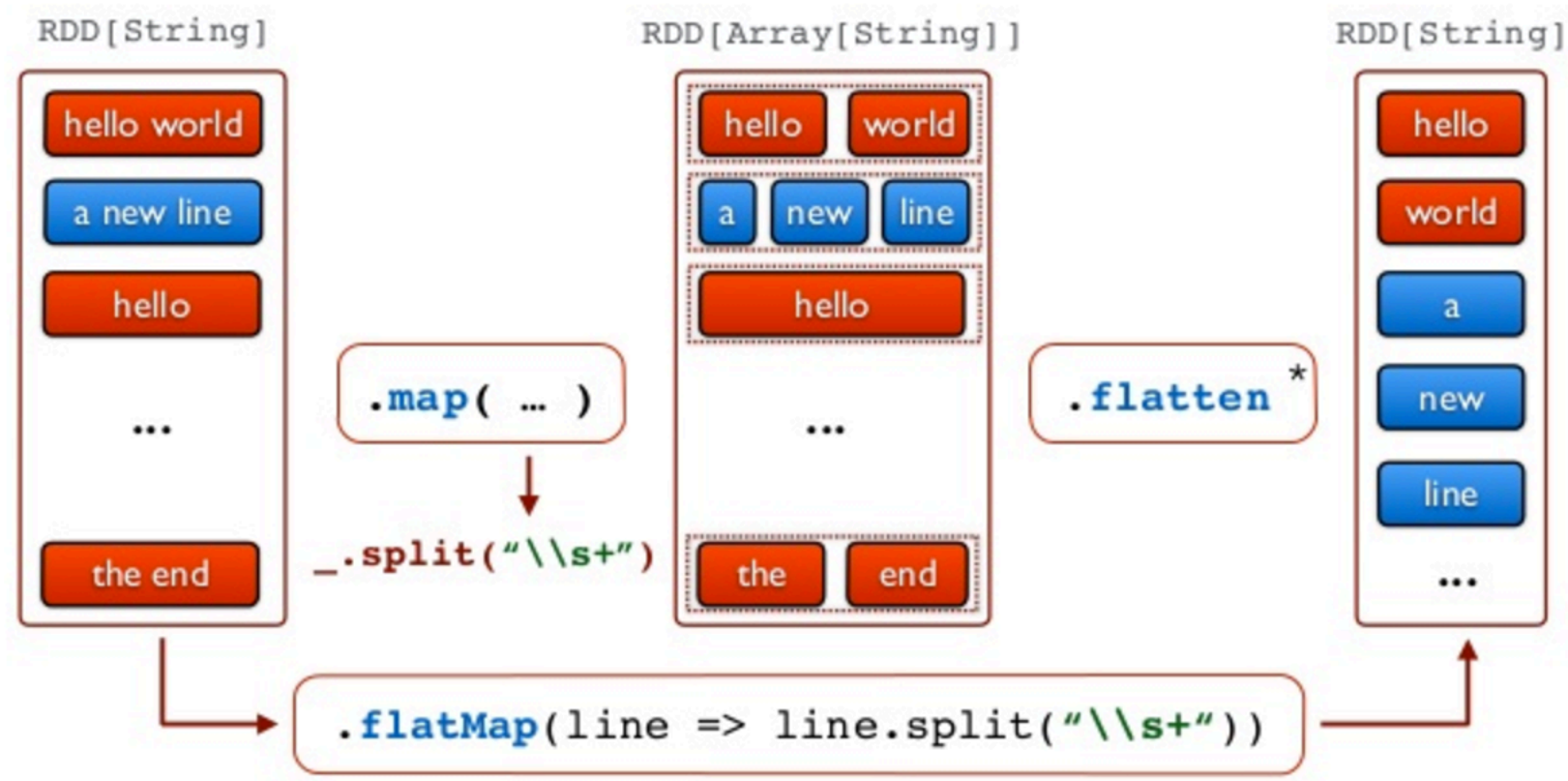
Transformations	Meaning
map(func)	Return a new distributed dataset formed by passing each element of the source through a function *func*
flatMap(func)	Return a new datasets formed by selecting those elements of the source on which *func* returns true
union(otherDateset)	Return a new dataset that contains the union of the elements in the source dataset and the argument
…	…

map vs flatmap

```
scala> val l = List(1,2,3,4,5)

scala> l.map( x => x*2 )
res60: List[Int] = List(2, 4, 6, 8, 10)
```

```
scala> def g(v:Int) = List(v-1, v, v+1)
g: (v: Int)List[Int]

scala> l.map(x => g(x))
res64: List[List[Int]] = List(List(0, 1, 2), List(1, 2, 3), List(2, 3, 4), List(3, 4, 5), L

scala> l.flatMap(x => g(x))
res65: List[Int] = List(0, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4, 5, 4, 5, 6)
```

map vs flatmap

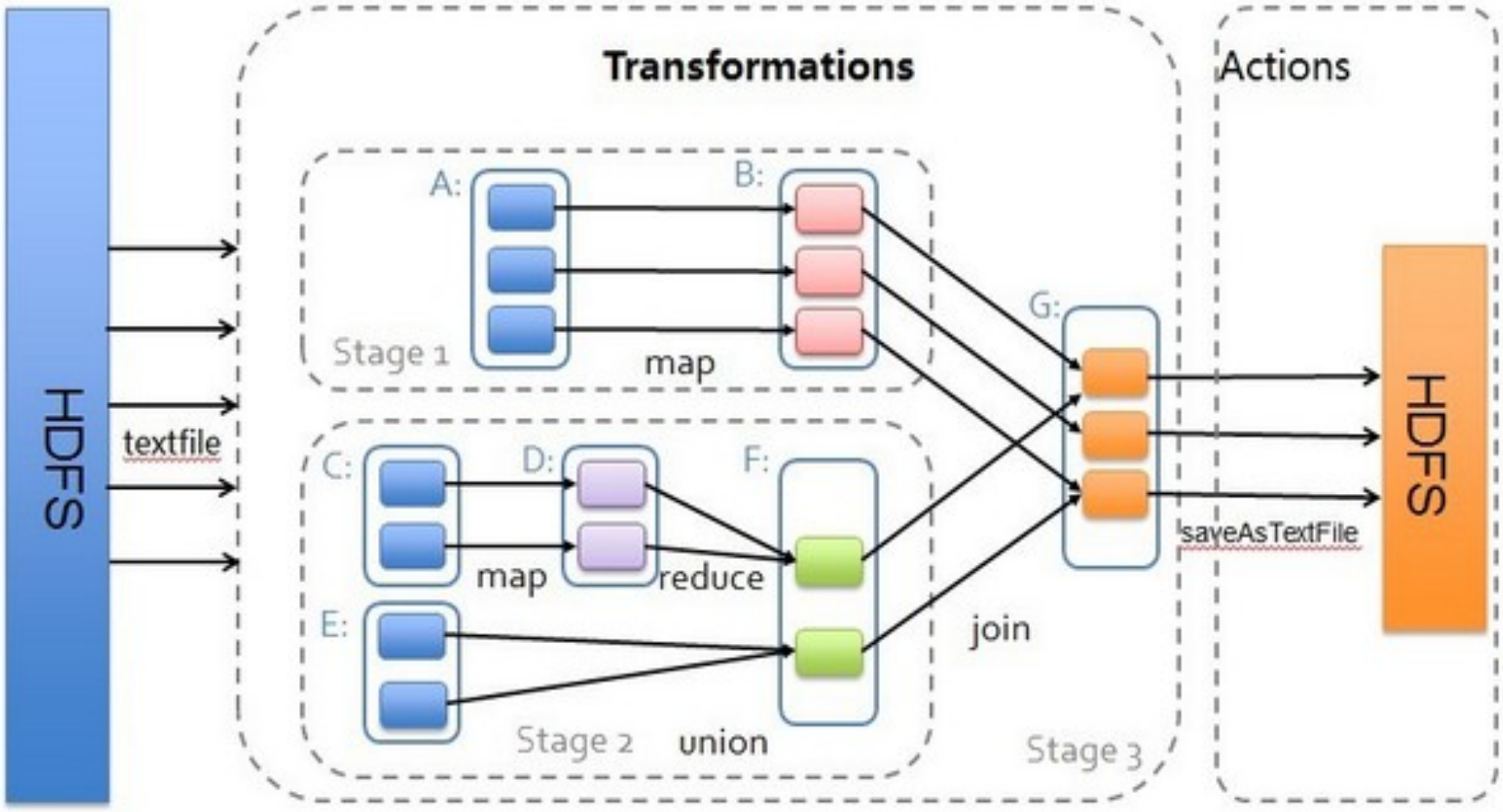



```
// Step 3 - Split lines into words
val words  = lower.flatMap(line => line.split("\\s+"))
```

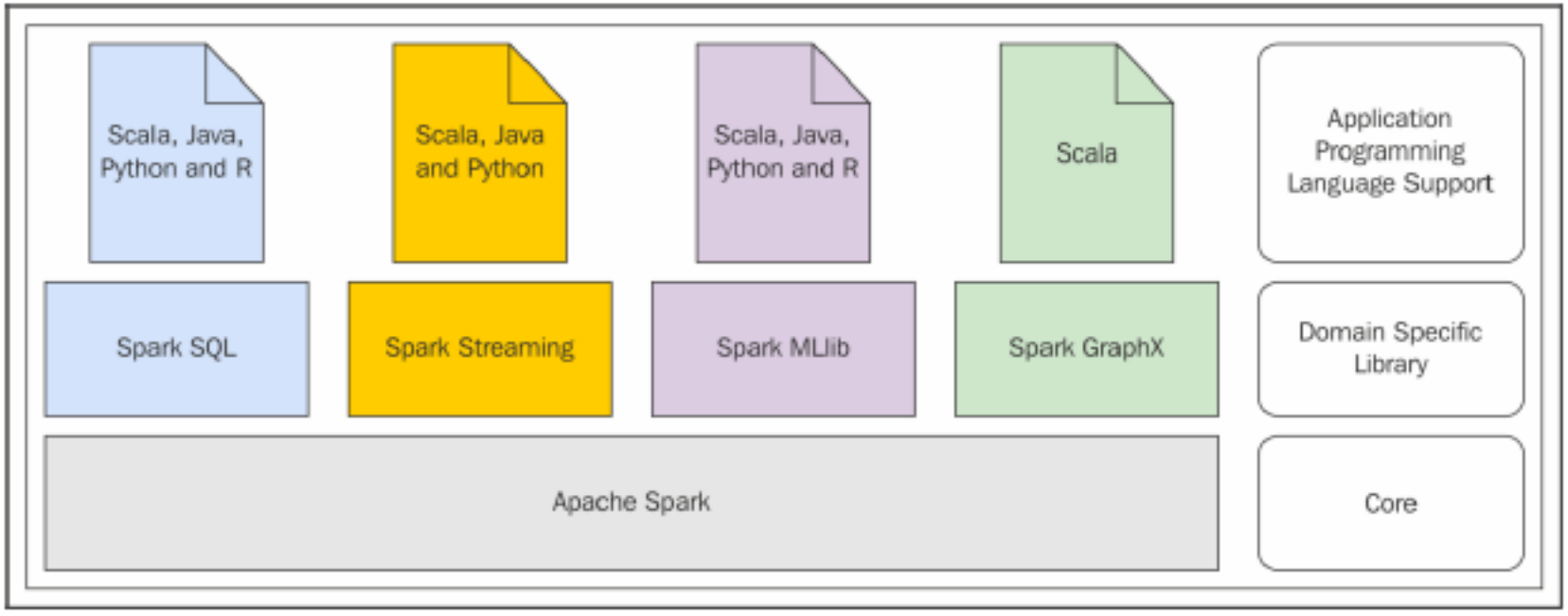
Actions

Actions	Meaning
reduce(func)	Aggregate the elements of the dataset using a function *func*
collect()	Return all the elements of the dataset as an array at the driver program
count()	Return the number of elements in dataset
first()	Return the first element of the dataset
saveAsTextFile(path)	Write the elements of the dataset as text file (or set of text file) in a given dir in the local file system, HDFS or any other Hadoop-supported file system
…..	……

Transformations & Actions



Lenguage APIs



Code examples: WordCount

```
void map (String doc_id, String text):
  for each word w in segment(text):
    emit(w, "1");



void reduce (String word, Iterator group):
  int count = 0;

  for each pc in group:
    count += Int(pc);

  emit(word, String(count));
```

Code Examples (II)

MBIT School
Madrid Business Intelligence Technology

Scala:

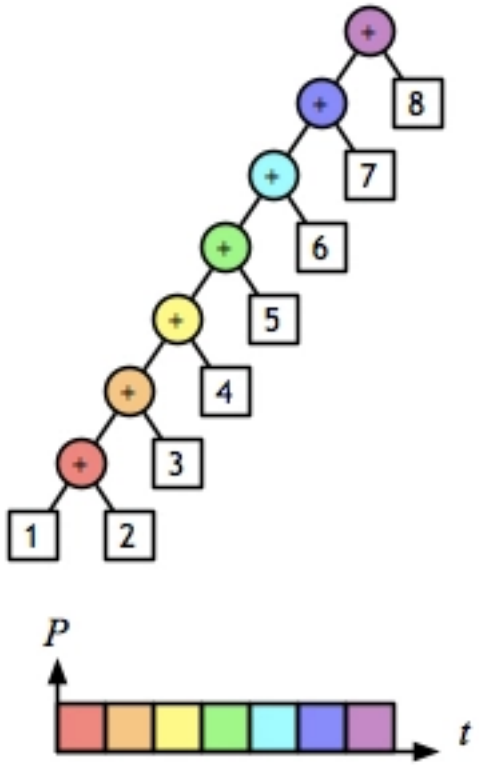
```
val f = sc.textFile("README.md")
val wc = f.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
wc.saveAsTextFile("wc_out")
```

Python:

```
from operator import add
f = sc.textFile("README.md")
wc = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).reduceByKey(add)
wc.saveAsTextFile("wc_out")
```

Entender el reduce

- Basado en el concepto *`reduceLeft`* de Scala
- Concepto típico de programación funcional




```
scala> val a = Array(20, 12, 6, 15, 2, 9)
a: Array[Int] = Array(20, 12, 6, 15, 2, 9)
```

```
scala> a.reduceLeft(_ + _)
res0: Int = 64

scala> a.reduceLeft(_ * _)
res1: Int = 388800

scala> a.reduceLeft(_ min _)
res2: Int = 2

scala> a.reduceLeft(_ max _)
res3: Int = 20
```

ReduceLeft

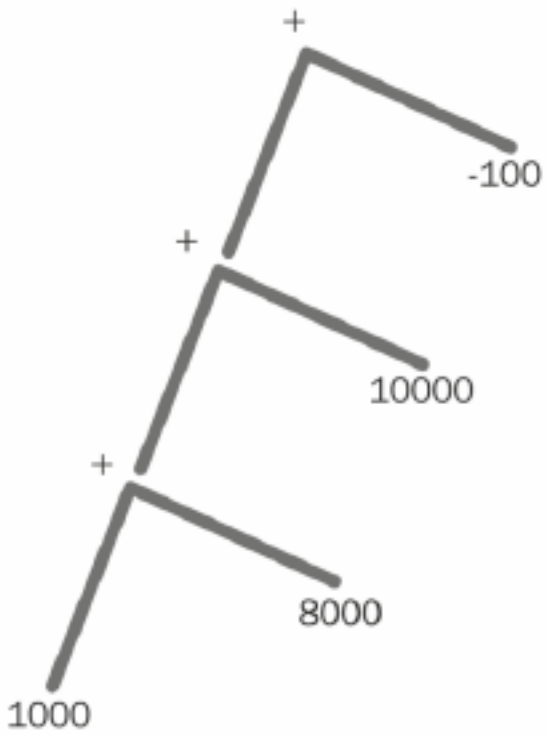
((SB10001, 1000), (SB10001, 8000), (SB10001, 10000), (SB10001, -100)).**reduceByKey**(_+_)

↓

(SB10001, Array(1000, 8000, 10000, -100).**reduceLeft**(_+_))

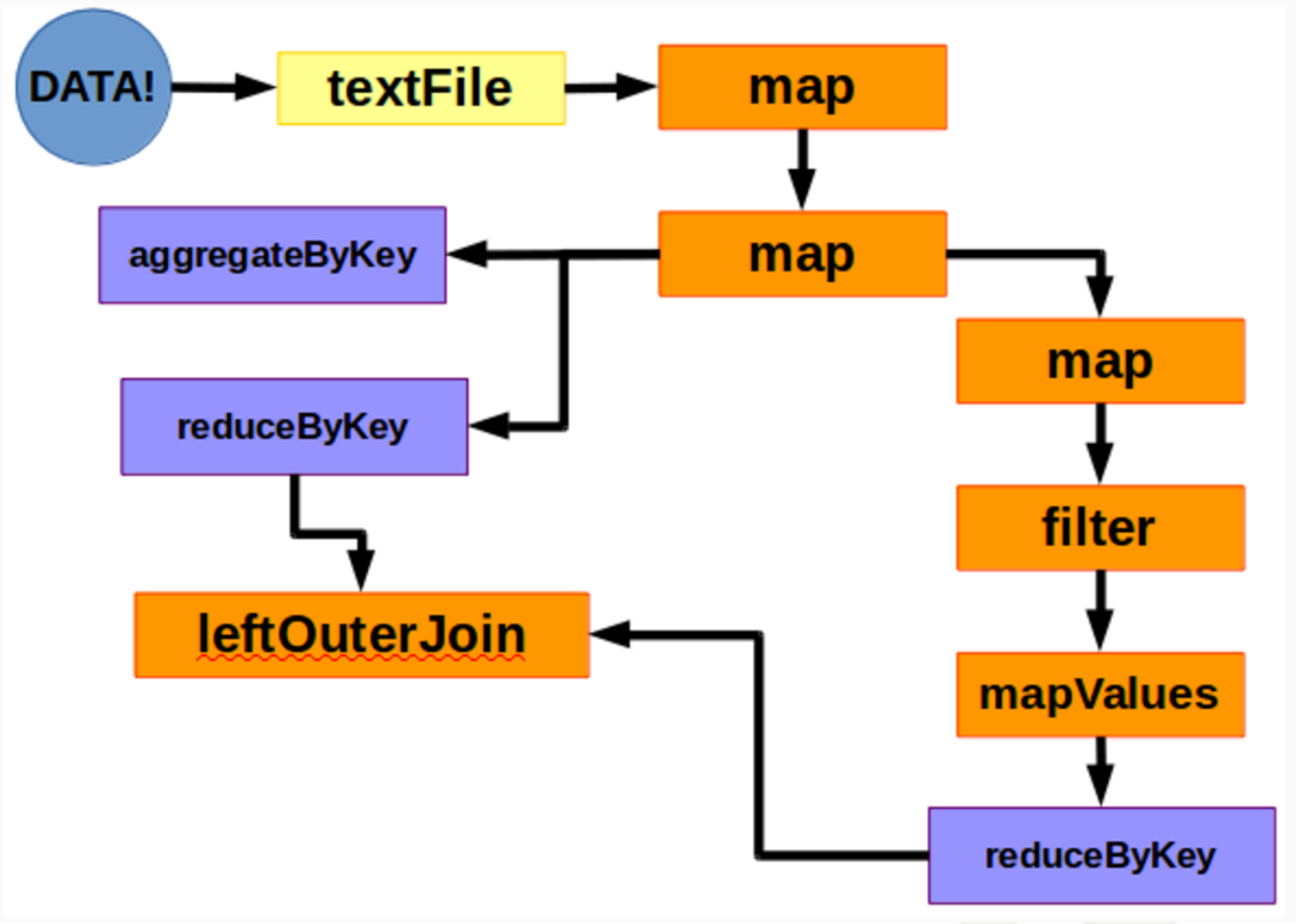
↓

(SB10001, 18900)



((1000+8000)+10000)-100=18900

Esquema funcional



Representing RDDs

MBIT School
Madrid Business Intelligence Technology

Challenge: choosing a representation for RDDs that can track lineage across transformations

Each RDD include:

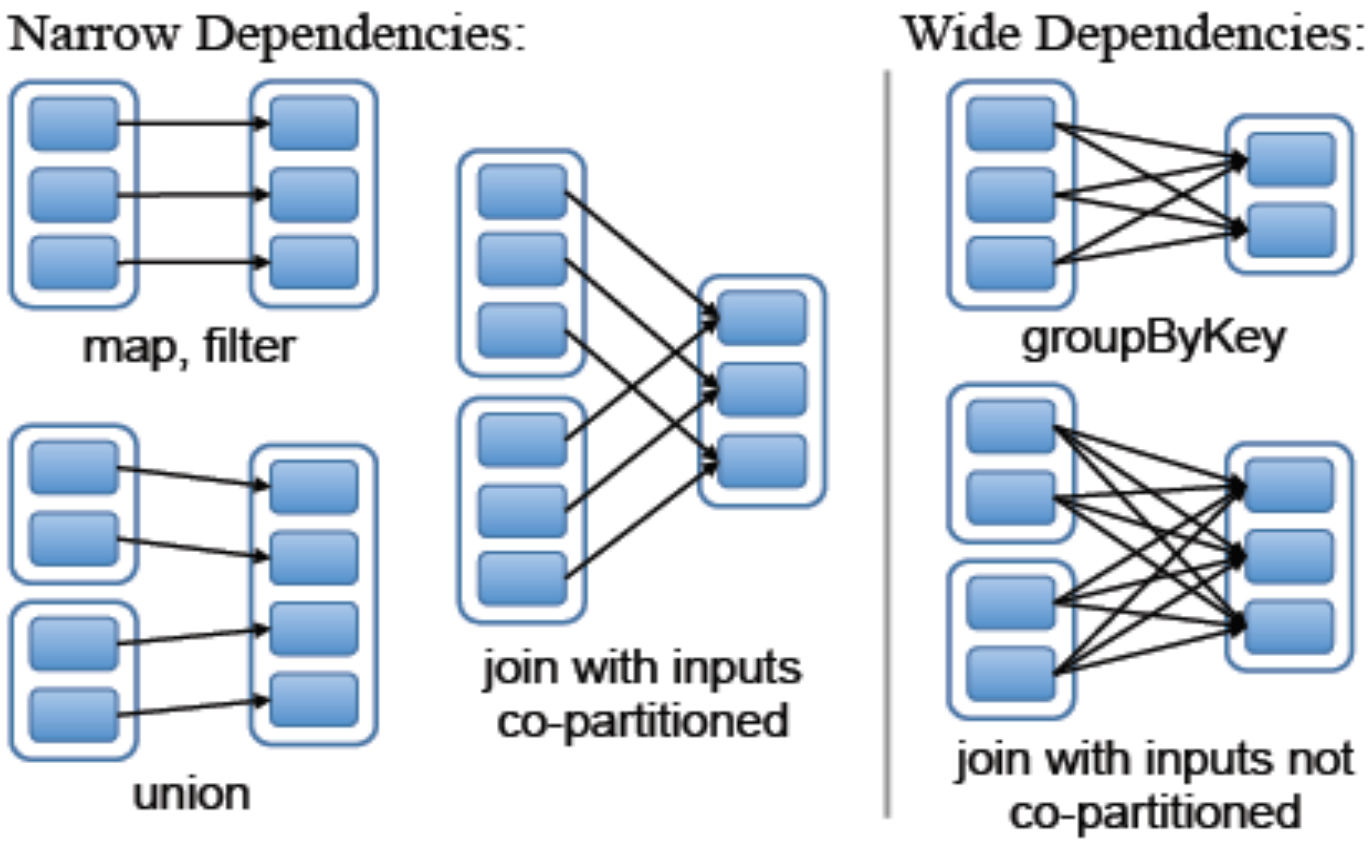
1) A set of partitions(atomic pieces of datasets)
2) A set of dependencies on parent RDDs
3) A function for computing the dataset based its parents
4) Metadata about its partitioning scheme
5) Data placement

Interface used to represent RDDs

MBIT School
Madrid Business Intelligence Technology

Operation	Meaning
partitons()	Return s list of partition objects
preferredLocations(p)	List nodes where partition p can be accessed faster due to data locality
dependencies()	Return a list of dependencies
iterator(p, parenetIters)	Compute the elements of partition p given iterators for its parent partitions
partitioner()	Return metadata specifying whether the RDD is hash/range partitioned

RDD Dependencies



Each box is an RDD, with partitions shown as shaded rectangles

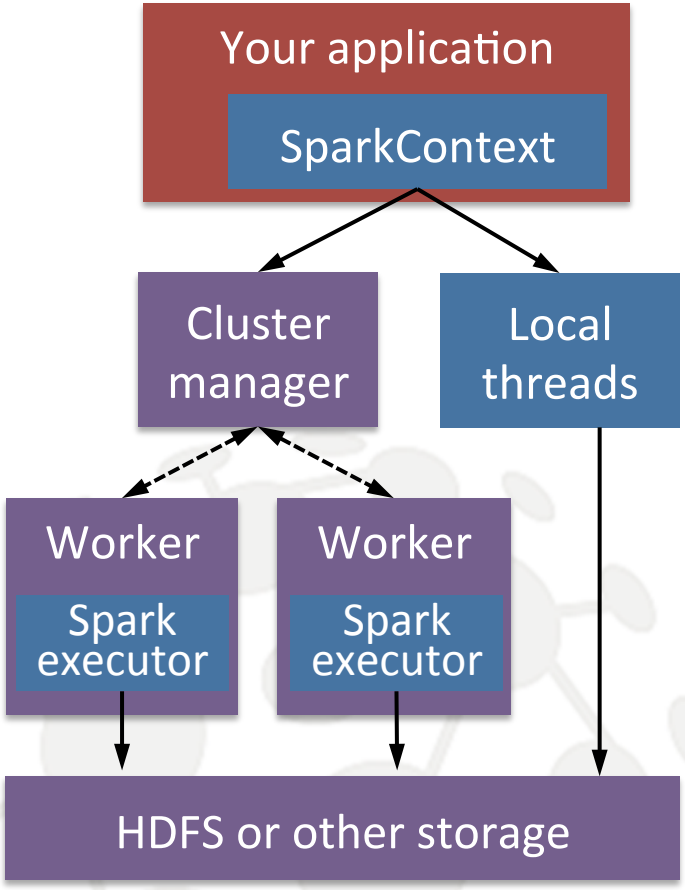
RDDs vs. Distributed Shared Memory

MBIT School
Madrid Business Intelligence Technology

Aspect	RDDs	DSM
Reads	Coarse- or fine-grained	Fine-grained
Writes	Coarse-grained	Fine-grained
Consistency	Trivial(immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using backup tasks	Difficult
Work placement	Automatic based on data locality	Up to app (runtimes aim for transparency)
Behavior if not enough RAM	Similar to existing data flow systems	Poor performance(swapping ?)

Software Components & Execution Flow

- Spark runs as a library in your program (1 instance per app)
- Runs tasks locally or on cluster
  - Mesos, YARN or standalone mode
- Accesses storage systems via Hadoop InputFormat API
  - Can use HBase, HDFS, S3, …



Terminology

MBIT School
Madrid Business Intelligence Technology

- **Application Jar**
  - User Program and its dependencies except Hadoop & Spark Jars bundled into a Jar file
- **Driver Program**
  - The process to start the execution (main() function)
- **Cluster Manager**
  - An external service to manage resources on the cluster (standalone manager, YARN, Apache Mesos)
- **Deploy Mode**
  - **cluster** : Driver inside the cluster
  - **client** : Driver outside of Cluster

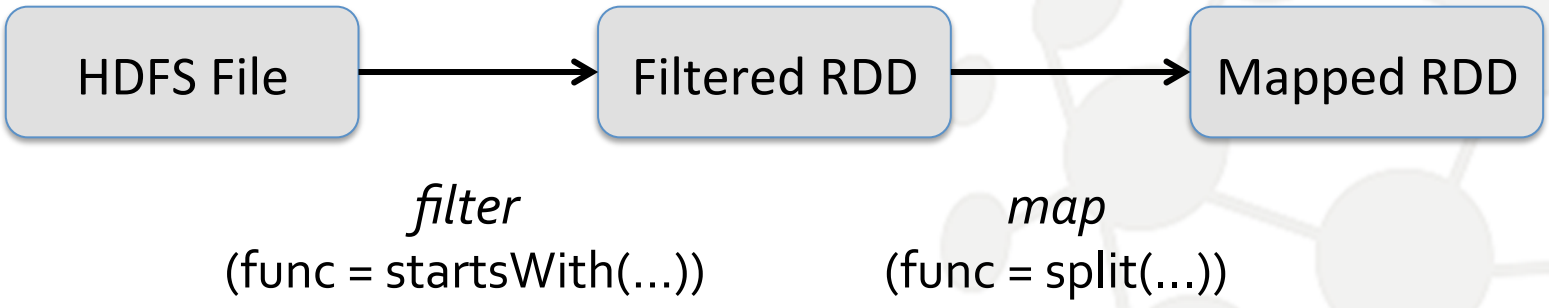
Terminology

- **Worker Node :** Node that run the application program in cluster
- **Executor**
  - Process launched on a worker node, that runs the Tasks
  - Keep data in memory or disk storage
- **Task :** A unit of work that will be sent to executor
- **Job**
  - Consists multiple tasks
  - Created based on a Action
- **Stage :** Each Job is divided into smaller set of tasks called Stages that is sequential and depend on each other
- **SparkContext :**
  - represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

Fault Recovery

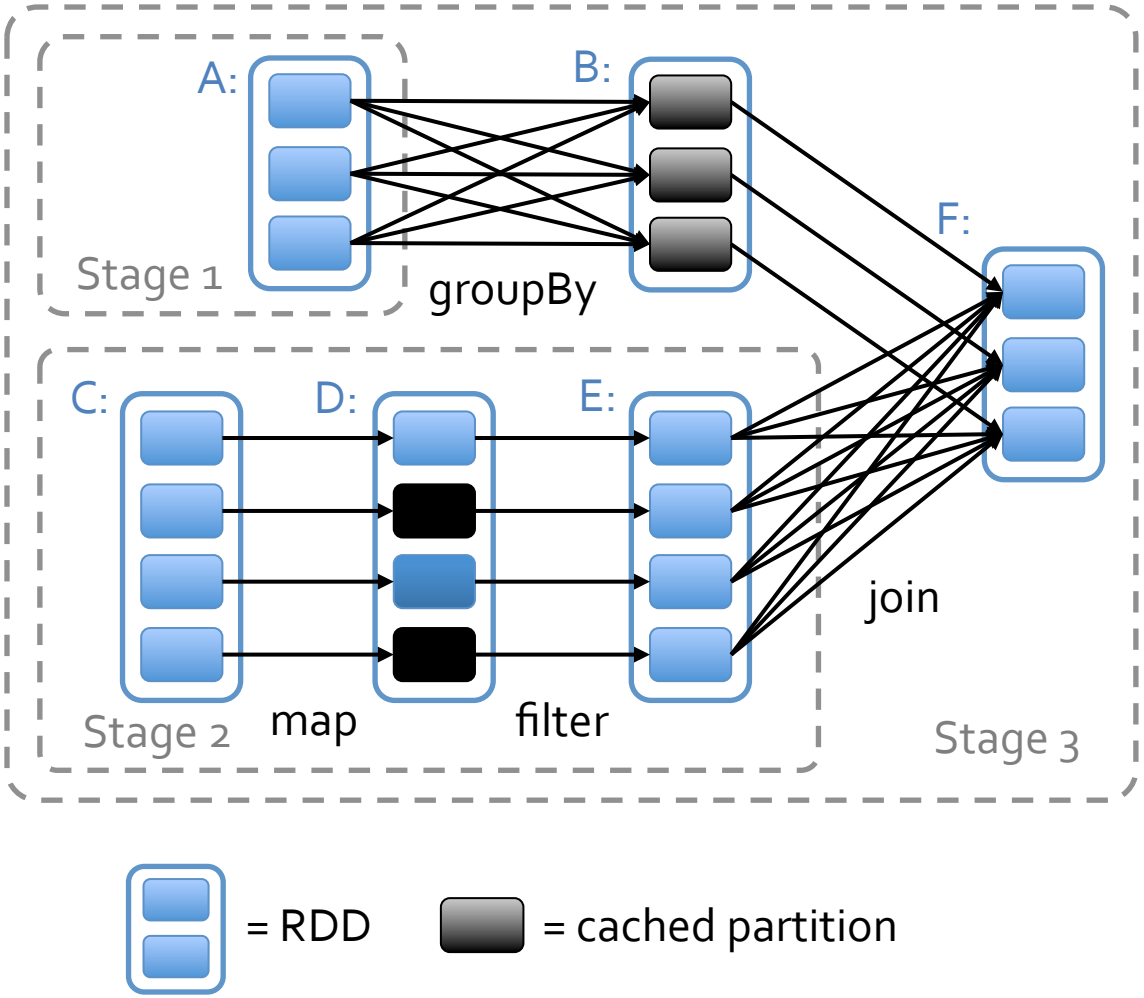
RDDs track *lineage* information that can be used to efficiently recompute lost data

```
msgs = textFile.filter(lambda s: s.startswith("ERROR"))
               .map(lambda s: s.split("\t")[2])
```



Task Scheduler

- General task graphs
- Automatically pipelines functions
- Data locality aware
- Partitioning aware to avoid shuffles



Advanced Features

- Controllable partitioning
  - Speed up joins against a dataset
- Controllable storage formats
  - Keep data serialized for efficiency, replicate to multiple nodes, cache on disk
- Shared variables: broadcasts, accumulators
- See online docs for details!

SPARK ESSENTIALS

Coding

Apache® Spark™ 2.0 Technical Preview

EASIER

FASTER

SMARTER

Sorry, cloudera QuickStart version is 1.6

PRE: intro to Scala (install)

MBIT School
Madrid Business Intelligence Technology

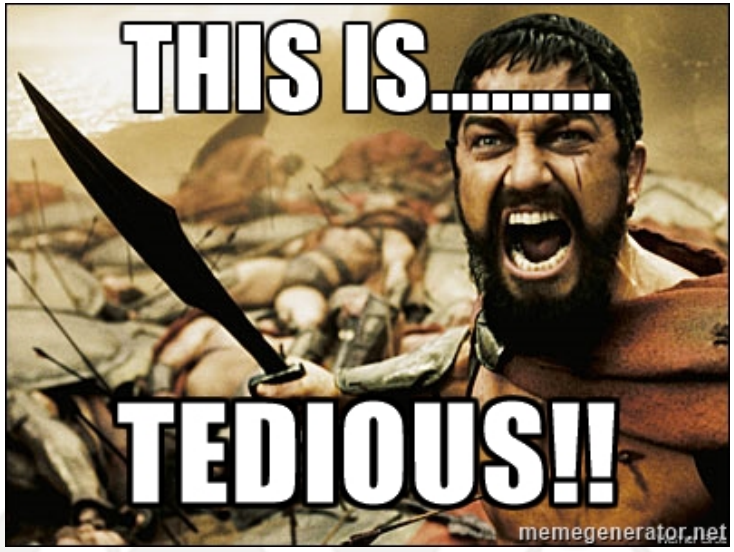
- Check java
  - $ java –version
- Download and install scala 2.11.1 (Java 7!!!)

```
$ cd Downloads
$ wget http://downloads.lightbend.com/scala/2.11.1/scala-2.11.1.rpm
$ sudo yum install scala-2.11.1.rpm
$ scala –version
$ scala
```

```
scala> 44 + 55
res0: Int = 99

scala> print("Hellooooo")
Hellooooo
scala>
```

Hello World!

```
scala> object HelloWorld {
     | def main(args: Array[String]): Unit = {
     |    println("Hello world!")
     | }
     | }
defined object HelloWorld

scala> HelloWorld.
asInstanceOf   isInstanceOf   main   toString

scala> HelloWorld.main(Array())
Hello world!

scala> :q
```



Una mejor solución

MBIT School
Madrid Business Intelligence Technology

IntelliJ IDEA

What's New 2017.1 EAP Features Learn Buy

Version: 2016.3.4

Build: 163.12024.16

Released: January 30, 2017

Download IntelliJ IDEA

macOS Windows Linux

Community

Java, Groovy, Scala and Android development

DOWNLOAD .TAR.GZ

Free download, 392 MB

Ultimate

Web, mobile and enterprise development

DOWNLOAD .TAR.GZ

Free 30-day trial, 572 MB

SparkContext

MBIT School
Madrid Business Intelligence Technology

Scala:

```
scala> sc
res: spark.SparkContext = spark.SparkContext@470d1f30
```

Python:

```
>>> sc
<pyspark.context.SparkContext object at 0x7f7570783350>
```

RDD

Scala:

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val distData = sc.parallelize(data)
distData: spark.RDD[Int] = spark.ParallelCollection@10d13e3e
```

Python:

```
>>> data = [1, 2, 3, 4, 5]
>>> data
[1, 2, 3, 4, 5]

>>> distData = sc.parallelize(data)
>>> distData
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:229
```

RDD & Files

Scala:

```
scala> val distFile = sc.textFile("README.md")
distFile: spark.RDD[String] = spark.HadoopRDD@1d4cee08
```

Python:

```
>>> distFile = sc.textFile("README.md")
14/04/19 23:42:40 INFO storage.MemoryStore: ensureFreeSpace(36827) called
with curMem=0, maxMem=318111744
14/04/19 23:42:40 INFO storage.MemoryStore: Block broadcast_0 stored as
values to memory (estimated size 36.0 KB, free 303.3 MB)
>>> distFile
MappedRDD[2] at textFile at NativeMethodAccessorImpl.java:-2
```

Transformations (review)

transformation	*description*
map(*func***)**	return a new distributed dataset formed by passing each element of the source through a function *func*
filter(*func***)**	return a new dataset formed by selecting those elements of the source on which *func* returns true
flatMap(*func***)**	similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item)
sample(*withReplacement, fraction, seed***)**	sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed*
union(*otherDataset***)**	return a new dataset that contains the union of the elements in the source dataset and the argument
distinct([*numTasks***]))**	return a new dataset that contains the distinct elements of the source dataset

Transformations (review)

transformation	*description*
groupByKey([*numTasks*])	when called on a dataset of `(K, V)` pairs, returns a dataset of `(K, Seq[V])` pairs
reduceByKey(*func*, [*numTasks*])	when called on a dataset of `(K, V)` pairs, returns a dataset of `(K, V)` pairs where the values for each key are aggregated using the given reduce function
sortByKey([*ascending*], [*numTasks*])	when called on a dataset of `(K, V)` pairs where `K` implements `Ordered`, returns a dataset of `(K, V)` pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument
join(*otherDataset*, [*numTasks*])	when called on datasets of type `(K, V)` and `(K, W)`, returns a dataset of `(K, (V, W))` pairs with all pairs of elements for each key
cogroup(*otherDataset*, [*numTasks*])	when called on datasets of type `(K, V)` and `(K, W)`, returns a dataset of `(K, Seq[V], Seq[W])` tuples – also called `groupWith`
cartesian(*otherDataset*)	when called on datasets of types `T` and `U`, returns a dataset of `(T, U)` pairs (all pairs of elements)

Actions

action	*description*
reduce(*func*)	aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one), and should also be commutative and associative so that it can be computed correctly in parallel
collect()	return all the elements of the dataset as an array at the driver program – usually useful after a filter or other operation that returns a sufficiently small subset of the data
count()	return the number of elements in the dataset
first()	return the first element of the dataset – similar to *take(1)*
take(*n*)	return an array with the first *n* elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
takeSample(*withReplacement, fraction, seed*)	return an array with a random sample of *num* elements of the dataset, with or without replacement, using the given random number generator seed

Actions (II)

action	*description*
`saveAsTextFile(`*`path`*`)`	write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call `toString` on each element to convert it to a line of text in the file
`saveAsSequenceFile(`*`path`*`)`	write the elements of the dataset as a Hadoop `SequenceFile` in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's `Writable` interface or are implicitly convertible to `Writable` (Spark includes conversions for basic types like `Int`, `Double`, `String`, etc).
`countByKey()`	only available on RDDs of type `(K, V)`. Returns a `Map` of `(K, Int)` pairs with the count of each key
`foreach(`*`func`*`)`	run a function *func* on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems

Persistence

MBIT School
Madrid Business Intelligence Technology

Scala:

```
val f = sc.textFile("README.md")
val w = f.flatMap(l => l.split(" ")).map(word => (word, 1)).cache()
w.reduceByKey(_ + _).collect.foreach(println)
```

Python:

```
from operator import add
f = sc.textFile("README.md")
w = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).cache()
w.reduceByKey(add).collect()
```

Broadcast variables

- Read only variables cached on each machine

Scala:

```
val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar.value
```

Python:

```
broadcastVar = sc.broadcast(list(range(1, 4)))
broadcastVar.value
```

Accumulators

Scala:

```
val accum = sc.accumulator(0)
sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)

accum.value
```

Python:

```
accum = sc.accumulator(0)
rdd = sc.parallelize([1, 2, 3, 4])
def f(x):
    global accum
    accum += x

rdd.foreach(f)

accum.value
```

(K, V) pairs

MBIT School
Madrid Business Intelligence Technology

Scala:

```scala
val pair = (a, b)

    pair._1 // => a
    pair._2 // => b
```

Python:

```python
pair = (a, b)

    pair[0] # => a
    pair[1] # => b
```

Java:

```java
Tuple2 pair = new Tuple2(a, b);

    pair._1 // => a
    pair._2 // => b
```

Resumen

MBIT School
Madrid Business Intelligence Technology

- Apache Spark es el entorno ideal cuando se quiere escribir programas que trabajen con datos compartidos a través de una memoria distribuida
- Posibilidad de interactuar con HDFS y otros DFS
- Se puede trabajar en Java, Scala, R y Python
- Scala es un lenguaje recomendado si se quiere insistir mucho en la mejora de rendimiento
- Hemos repasado los conceptos básicos de programación en Spark