Getting started on Apache Spark

TANG Gen

Outline

- Online ressources
- Installation
- Text mining example
    - Spark shell
    - RDD operator graph
    - How it works?
- Spark compile and submit

Online ressources

/Training materials

- Spark docs

  https://spark.apache.org/docs/latest/

- Spark Q&A

  http://stackoverflow.com/questions/tagged/apache-spark

  http://spark.apache.org/community.html

- Spark packages

  http://spark-packages.org/

- Course ressources

  http://databricks.com/spark/developer-resources#

Installation

/Step 1: Install Java JDK 6/7

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

- Follow the license agreement instructions
- Then click the download for your OS
- need JDK instead of JRE (for Maven, etc.)

Installation

/Step 2: Download Spark

- we will use spark 1.3.0

    1. http://spark.apache.org/downloads.html

    2. Choose spark release 1.3.0 and *Pre-built for Hadoop 2.4 and later*

    3. Download and unzip spark

    4. Follow the same instructions to download *Source Code*

Installation

/Optional downloads: Python

- For python 2.7, check out Anaconda by Continuum Analytics for a full featured platform:

https://store.continuum.io/cshop/anaconda/

Installation

/Optional downloads: Maven

- Java builds later require Maven, which can download at

http://maven.apache.org/download.cgi

Installation

/Optional downloads: SBT

- SBT provides quick build on Scala
- You can copy the ./sbt and ./project directoires from

https://github.com/GenTang/spark_hbase

- It uses sbt-spark-package plugin to facilitate spark applications building

Text mining example

/Run Spark Shell

- First of all, let's run Spark's interactive shell
  - There are two Spark shell: Scala and Python
- Enter the "Spark" directory, run

./bin/spark-shell or ./bin/pyspark

- We will use spark-shell in the following training, but also provide the code in pyspark

Small tips for spark-shell: tap "tab" to show all the namespace available

Text mining example

/Create an RDD

- Create some data

val data = 1 to 10000

- Create an **RDD** based on that data

val rdd = sc.parallelize(data, 10) or val rdd = sc.parallelize(data)

- Some transformations on created RDD

*rdd.map(_ * 2).filter(_ < 20).collect()*

- We use ***map(_ * 2)*** to multiply every element by 2 and then use ***filter(_ < 20)*** to select values less than 20. At last, we use ***collect()*** to send the data from slaves to master

Text mining example

/Create an RDD

- Create some data

data = range(1,10000)

- Create an **RDD** based on that data

rdd = sc.parallelize(data, 10) or rdd = sc.parallelize(data)

- Some transformations on created RDD

*rdd.map(lambda a: a * 2).filter(lambda a: a < 20).collect()*

- We use ***map(lambda a: a * 2)*** to multiply every element by 2 and then use ***filter(lambda a: a < 20)*** to select values less than 20. At last, we use ***collect()*** to send the data from slaves to master

Text mining example

/CHANGES.txt mining

```
1 /*
2  * load the CHANGES.txt in the spark directory.
3  * Then put it into memory and do some interactive
4  * search for various patterns
5 */
6
7 //Created an RDD
8 val docs = sc.textFile("file:/Users/tgbaggio/spark/spark-1.3.0-bin-hadoop2.4/CHANGES.txt")
9
10 //Transformation on RDD
11 val sparkChange = docs.filter(_.contains("SPARK"))
12
13 //Put data in memory
14 sparkChange.cache()
15
16 //First action
17 sparkChange.count()
18
19 //Second action
20 sparkChange.filter(_.contains("PySpark")).collect().foreach(println)
```

Text mining example

/CHANGES.txt mining

- RDDs support two types of operations: *Transformations* and *Actions*
  1. **Transformations** are operations on RDDs that return a new RDD
  2. **Actions** are operations that return a final value to the driver program or write data to an external storage system

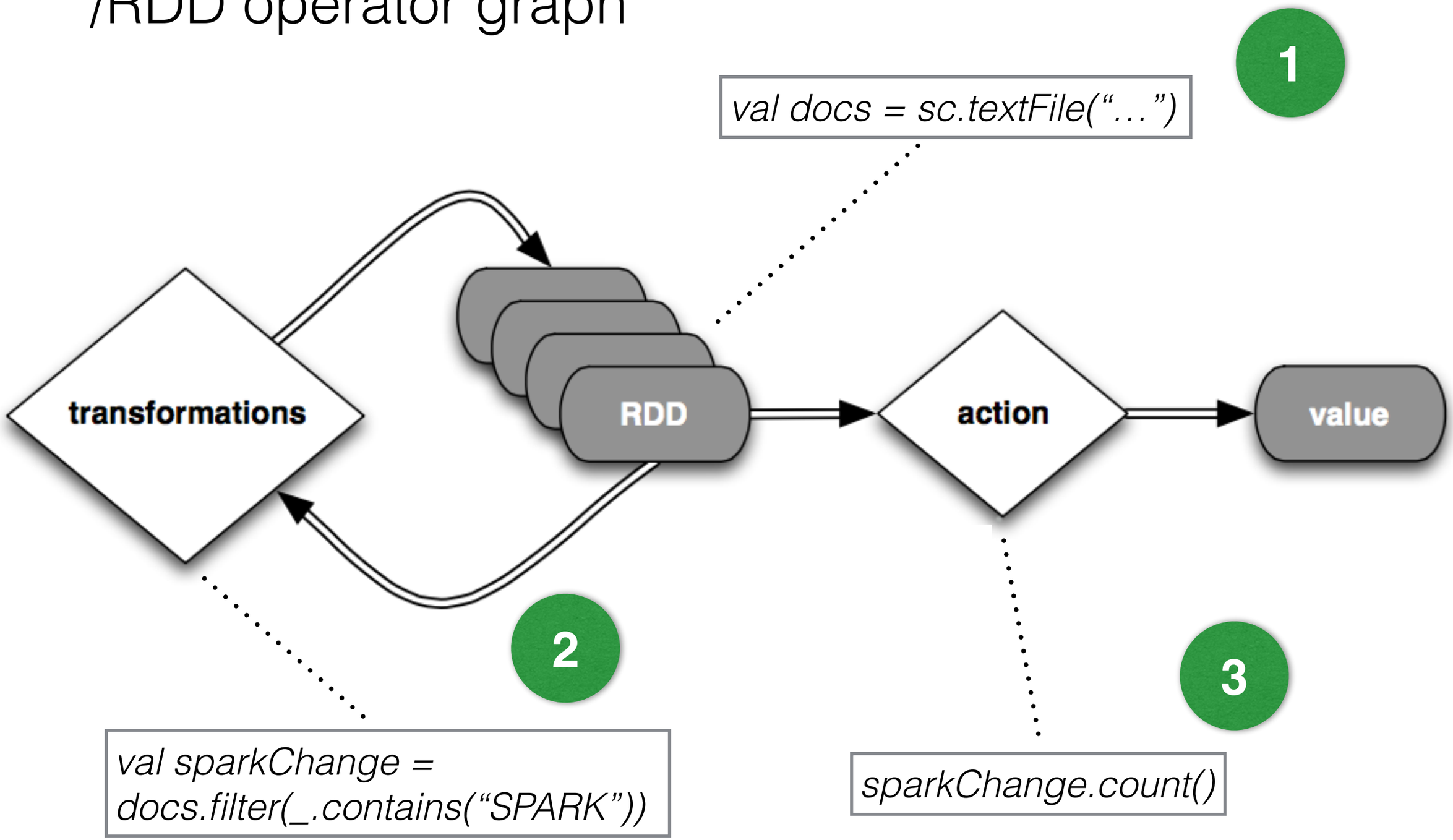
```
1 /*
2  * load the CHANGES.txt in the spark directory.
3  * Then put it into memory and do some interactive
4  * search for various patterns
5 */
6
7 //Created an RDD
8 val docs = sc.textFile("file:/Users/tgbaggio/spark/spark-1.3.0-bin-hadoop2.4/CHANGES.txt")
9
10 //Transformation on RDD
11 val sparkChange = docs.filter(_.contains("SPARK"))
12
13 //Put data in memory
14 sparkChange.cache()
15
16 //First action
17 sparkChange.count()
18
19 //Second action
20 sparkChange.filter(_.contains("PySpark")).collect().foreach(println)
```

Transformation

Action

Text mining example

/RDD operator graph



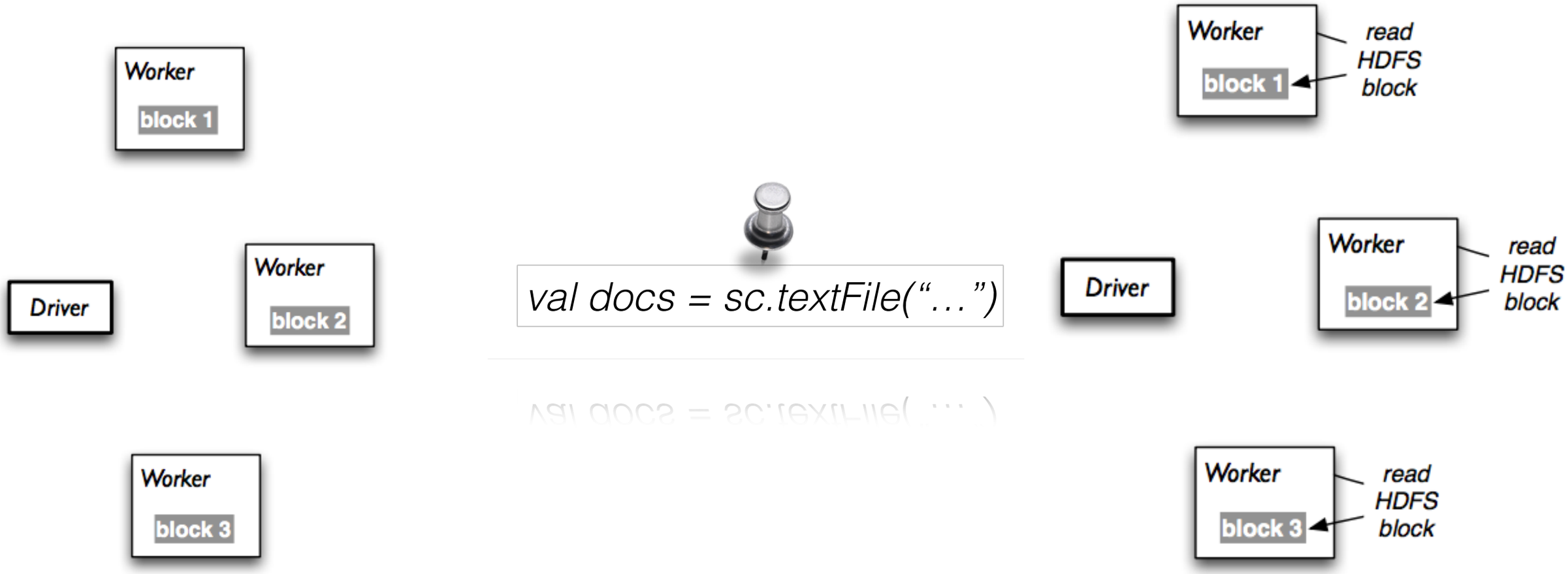
Text mining example

/RDD operator graph

```
scala> sparkChange.filter(_.contains("PySpark")).toDebugString
res24: String =
 | MapPartitionsRDD[23] at filter at <console>:26 []
 |     MapPartitionsRDD[20] at filter at <console>:23 []
 |         CachedPartitions: 2; MemorySize: 193.7 KB; TachyonSize: 0.0 B; DiskSize: 0.0 B
 |             file:/Users/tgbaggio/spark/spark-1.3.0-bin-hadoop2.4/CHANGES.txt
MapPartitionsRDD[19] at textFile at <console>:21 []
 |                 file:/Users/tgbaggio/spark/spark-1.3.0-bin-hadoop2.4/CHANGES.txt
HadoopRDD[18] at textFile at <console>:21 []
```

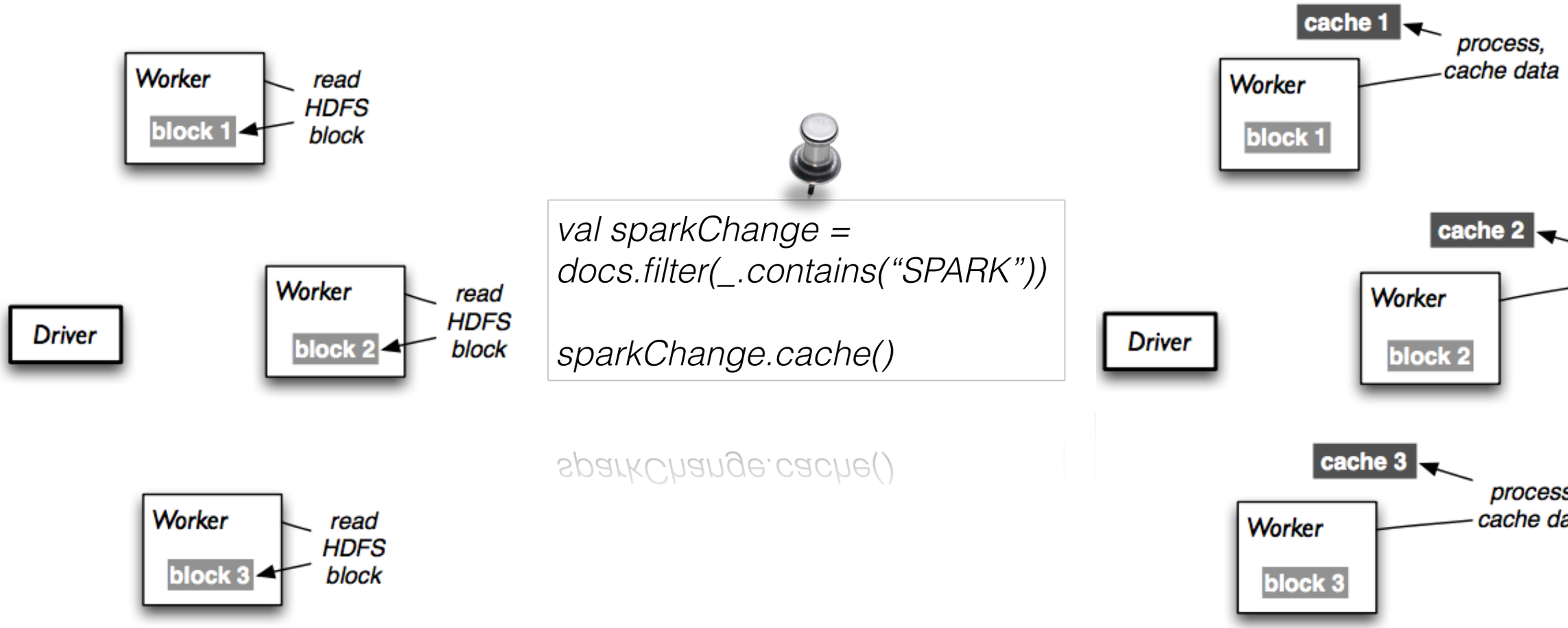
Text mining example

/How it works?



Text mining example

/How it works?



Text mining example

/How it works?

Spark compile and submit

/Spark on Scala

- Follow the structure of https://github.com/GenTang/spark_hbase
- Copy ./sbt and ./project directory

```scala
package spark.training

import org.apache.spark.{SparkConf, SparkContext}

object TextMining extends Serializable {
  def main(args: Array[String]): Unit = {

    //Created SparkContext
    val sc = new SparkContext(new SparkConf().setAppName("TextMining"))

    //Created an RDD
    val docs = sc.textFile("file:/Users/tgbaggio/spark/spark-1.3.0-bin-hadoop2.4/CHANGES.txt")

    //Transformation on RDD
    val sparkChange = docs.filter(_.contains("SPARK"))

    //Put data in memory
    sparkChange.cache()

    //First action
    println("There are " + sparkChange.count().toString + " lines containing SPARK")

    //Second action
    sparkChange.filter(_.contains("PySpark")).collect().foreach(println)
  }
}
```

Spark compile and submit

/Spark on Scala

- We use sbt (and sbt-spark-package) to compile spark applications

1. Created build.sbt under the directory of the application

```
lazy val root = (project in file(".")).
  settings(
    name := "text_mining",
    version := "1.0",
    scalaVersion := "2.10.5",
    sparkVersion := "1.3.0"
  )

sparkComponents ++= Seq("core")

// change output path
target := baseDirectory.value / "sbt_target"
```

2. run ./sbt/sbt clean assembly

3. The compiled file (jar) will be in ./sbt_target directory.

- We can also use maven to build application, but it is more boring.

Spark compile and submit

/Spark on Scala

- Once we have built the application, we can submit it to spark by “spark-submit”

1. Enter “Spark” directory

2. Run ./bin/spark-submit \
   --class spark.training.TextMining \
   <path to application>/sbt_target/scala-2.10/text_mining-assembly-1.0.jar

Spark compile and submit

/Spark on Scala

- The more complete version of submit command is as follows

1. Enter “Spark” directory

2. Run ./bin/spark-submit \
   --class spark.training.TextMining \
   --master local[4] \
   --conf spark.app.name=“text_mining” \
   <path to application>/sbt_target/scala-2.10/text_mining-assembly-1.0.jar

- This submit is under localhost and under other deployment, the command has similar structure

Spark compile and submit

/Spark on Python

- Python **doesn't need compile!!!**

```
1 from pyspark import SparkContext, SparkConf
2
3 def textMining():
4     conf = SparkConf().set("spark.app.name", "text_mining")
5     sc = SparkContext(conf = conf)
6
7     rdd = sc.textFile("file:/Users/tgbaggio/spark/spark-1.3.0-bin-hadoop2.4/CHANGES.txt")
8     sparkChange = rdd.filter(lambda line: "SPARK" in line)
9
10    sparkChange.cache()
11
12    print("There are %s lines containing SPARK" % sparkChange.count())
13
14    for i in sparkChange.filter(lambda line: "PySpark" in line).collect():
15        print i
16
17
18 if __name__ == "__main__":
19     textMining()
```

Spark compile and submit

/Spark on Python

- The submit of pyspark is quite simple

    1. Enter “Spark” directory

    2. Run ./bin/spark-submit <path to python file>

- When we use other **python projects** as your application dependencies, you should use ***--py-files*** to distribute your dependencies with your application.

- When we use other **non-python projects** as dependencies, the life is becoming bitter, we will cover this later.