PA2-Tutorial

Section #1

Hadoop

PA2

- AWS registering
- AWS starting an Instance
- Hadoop
- Spark

AWS register

- Create an account in AWS
- Give your details including credit card
- Get your Account ID from AWS account

  Name→My Account→Account Settings→AccountID
- Go to https://aws.amazon.com/education/awseducate/
- Give your details and account ID
- Wait for Promo Code
- Enter Promo Code in

  Name→Credits→Promo Code

Aws Starting and Connecting to an Instance

- Services → Compute→ EC2→ Create Instance→ Launch Instance
- Select AMI
- Select Instance type
- Select or create key
- Launch
- Chmod 400 <keyname>.pem
- Connect

Hadoop

- Download and configure Hadoop
- Configure the system
- Start the services
- Run a Job

Download and Configure Hadoop

- Download Hadoop

wget http://apache.claz.org/hadoop/common/hadoop-2.7.4/hadoop-2.7.4.tar.gz

- Untar and rename to Hadoop
- Go to ./hadoop/etc/Hadoop/
- Add properties to
    - hdfs-site.xml
    - core-site.xml
    - yarn-site.xml
    - mapred-site.xml
    - slaves

Core-site.xml

- fs.default.name
- hadoop.tmp.dir

Hdfs-site.xml

- dfs.replication
- dfs.namenode.name.dir
- datanode.data.dir

Mapred-site.xml

- mapreduce.framework.name
- mapreduce.jobtracker.address
- mapred.tasktracker.map.tasks
- mapred.tasktracker.reduce.tasks

Yarn-site.xml

- yarn.resourcemanage.hostname
- yarn.nodemanager.aux-services
- yarn.nodemanager.aux-services.mapreduce.shuffle.class
- yarn.resourcemanager.scheduler.address
- yarn.resourcemanager.address
- yarn.resourcemanager.webapp.address
- yarn.resourcemanager.resource-tracker.address
- yarn.resourcemanager.admin.address

Slaves

- Specify the hostnames of all the slaves in this file

Configure the instance

- Update the system
  - apt-get update
  - apt-get upgrade
- Install Java
  - apt-get install default-jdk
- Update the bashrc file
- Source .bashrc
- Setup passwordless ssh
  - ssh-keygen –t rsa

Start the services

- Format the namenode
    - hdfs namenode –format
- Start Datanode and namenode
    - start-dfs.sh (to stop stop-dfs.sh)
- Start Resource manage
    - start-yarn.sh (to stop stop-yarn.sh)
- JPS to check whether all services are running
    - NameNode
    - DataNode
    - NodeManager
    - ResourceManager
    - SecondaryNameNode

Run a Job

- Create a file a move to HDFS
- Create a folder in hdfs
  - hdfs dfs –mkdir /input
  - Hdfs dfs –mkdir /output
- Make sure the file has been created
  - hdfs dfs –ls /
- Move the file to the folder
  - Hdfs dfs –put <file> /input

Run a job..

- Create mapreduce code
- Compile the code
  - bin/hadoop com.sun.tools.javac.Main WordCount.java
- Make a jar
  - jar cf wc.jar WordCount*.class
- Submit the job
  - hadoop jar wc.jar WordCount <inputfile> <outputfile>
- Get the output
  - hdfs dfs –get <file>

Tips

- log files
- scripts and other tools eg pssh
- Understand properties

Section #2

Apache Spark

General steps

1. Configure Hadoop
2. Configure Spark
3. Develop your Spark program
4. Deploy your program on Spark

Installation

- Install and configure Hadoop
  - http://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-2.8.1/hadoop-2.8.1.tar.gz
  - https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html

- Download and install Spark
  - Java 8 + if using Java (anonymous function & lambda expression)
  - https://spark.apache.org/downloads.html
  - https://spark.apache.org/docs/latest/running-on-yarn.html

- Deploying Spark on Hadoop
  - http://www.informit.com/articles/article.aspx?p=2755929&seqNum=5

Spark Configuration

- Cluster modes:
  - Standalone (Simplest)
  - Mesos
  - YARN (Recommended)
- Depends on what resource scheduler or manager you want to use.

- Does Spark depend on Hadoop?
  - https://stackoverflow.com/questions/32022334/can-apache-spark-run-without-hadoop
- Also, difference between client and cluster deploy mode?

RDD – Resilient Distributed Dataset

- Can be created from:
  - Parallelized collection

```
List<Integer> data = Arrays.asList(1, 2, 3, 4, 5);
JavaRDD<Integer> distData = sc.parallelize(data);
```

  - External Datasets

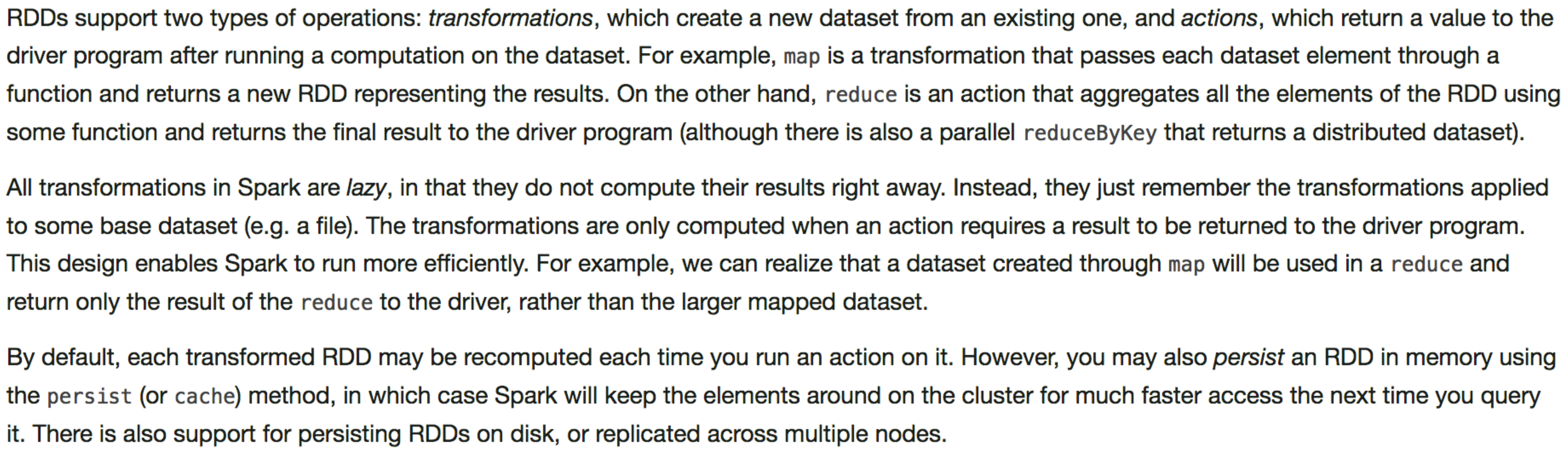
```
JavaRDD<String> distFile = sc.textFile("data.txt");
```

- Also Support Key-value pair

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaPairRDD<String, Integer> pairs = lines.mapToPair(s -> new Tuple2(s, 1));
JavaPairRDD<String, Integer> counts = pairs.reduceByKey((a, b) -> a + b);
```

RDD operations

http://spark.apache.org/docs/2.1.1/programming-guide.html

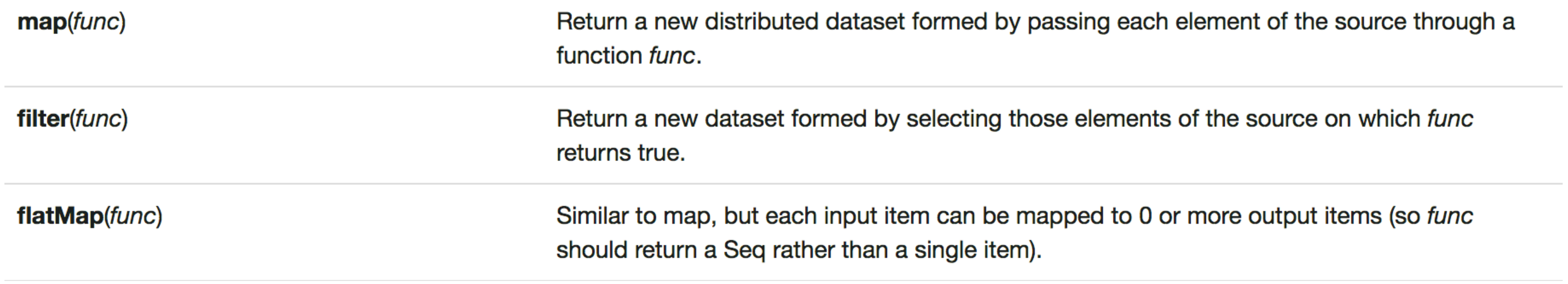


RDDs support two types of operations: *transformations*, which create a new dataset from an existing one, and *actions*, which return a value to the driver program after running a computation on the dataset. For example, `map` is a transformation that passes each dataset element through a function and returns a new RDD representing the results. On the other hand, `reduce` is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel `reduceByKey` that returns a distributed dataset).

All transformations in Spark are *lazy*, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently. For example, we can realize that a dataset created through `map` will be used in a `reduce` and return only the result of the `reduce` to the driver, rather than the larger mapped dataset.

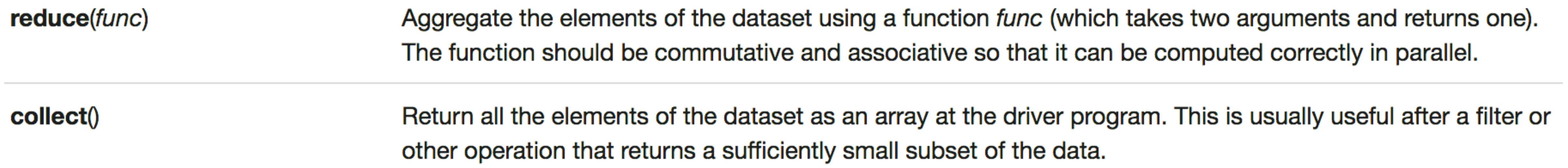
By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also *persist* an RDD in memory using the `persist` (or `cache`) method, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

- Transformation (Happens in memory)



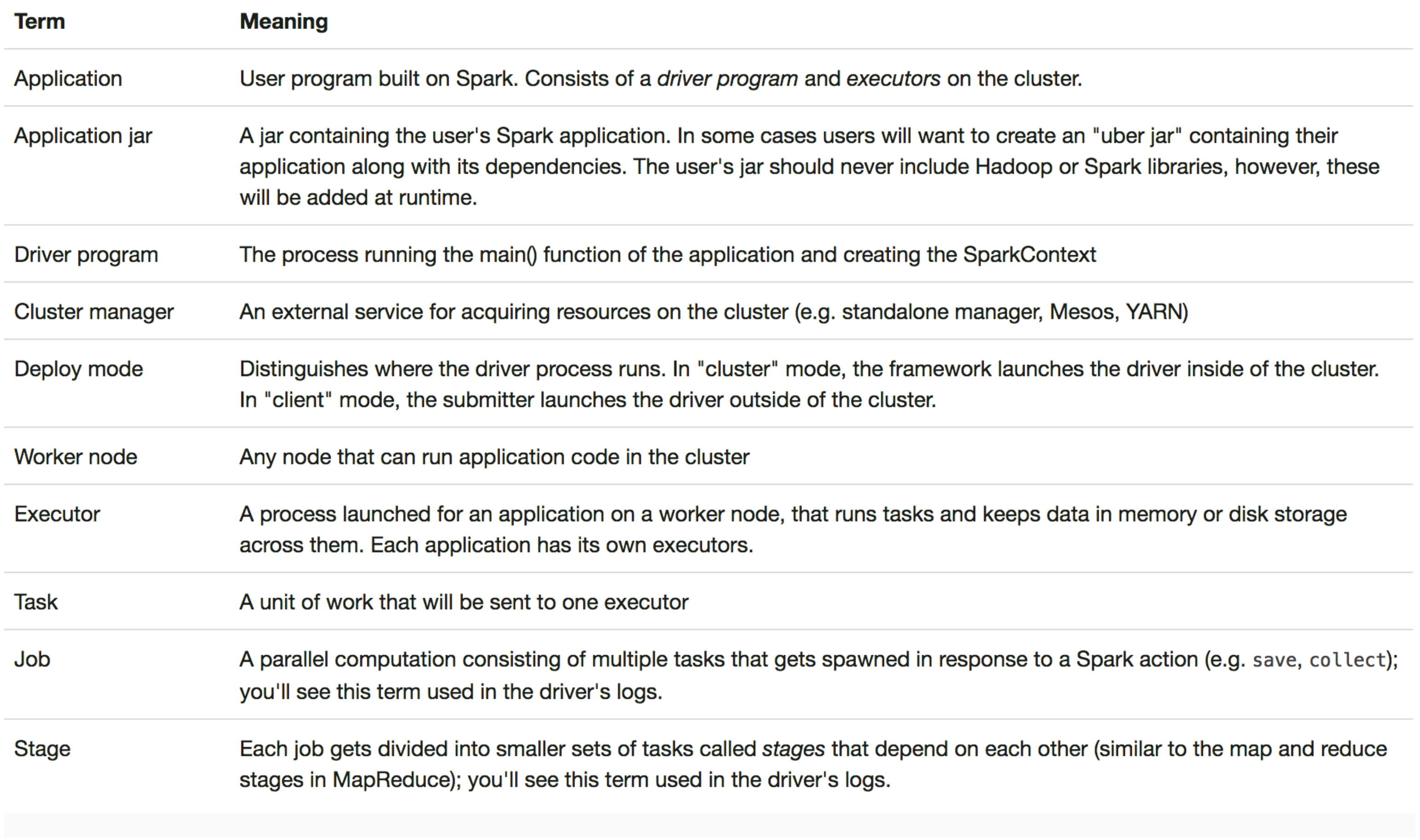
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).

- Action (Write to disk)



reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should 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. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

Terms and Concepts



Term	Meaning
Application	User program built on Spark. Consists of a *driver program* and *executors* on the cluster.
Application jar	A jar containing the user's Spark application. In some cases users will want to create an "uber jar" containing their application along with its dependencies. The user's jar should never include Hadoop or Spark libraries, however, these will be added at runtime.
Driver program	The process running the main() function of the application and creating the SparkContext
Cluster manager	An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN)
Deploy mode	Distinguishes where the driver process runs. In "cluster" mode, the framework launches the driver inside of the cluster. In "client" mode, the submitter launches the driver outside of the cluster.
Worker node	Any node that can run application code in the cluster
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
Task	A unit of work that will be sent to one executor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. `save`, `collect`); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called *stages* that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.

Developing and Deploying Program

- Very good examples at:
  - https://www.cloudera.com/documentation/enterprise/5-6-x/topics/spark_develop_run.html
  - Scala, Python and Java included.

- Two ways:
  1. Write commands in Spark Shell and directly run it. (Python, Scala)
  2. Write a program, deploy it using *spark-submit*. (Python, Scala, Java)

Debugging

- Debugging methods in different modes are different
- Important and not trivial

- Standalone mode
    - http://apache-spark-developers-list.1001551.n3.nabble.com/Debugging-Spark-itself-in-standalone-cluster-mode-td18139.html
- YARN mode
    - http://spark.apache.org/docs/latest/running-on-yarn.html
- Memos
    - http://spark.apache.org/docs/latest/running-on-mesos.html#troubleshooting-and-debugging