Analyzing Data with Spark

Thomas Ropars, Francieli Zanon Boito

2018

Please read all instructions carefully before starting the lab session.

1 Installing Spark

To install Spark on your home account do the following:

1. download the latest version at `https://www.apache.org/dyn/closer.lua/spark/spark-2.3.2/spark-2.3.2-bin-hadoop2.7.tgz`.
2. Extract the archive with the command `tar zxvf spark-2.3.2-bin-hadoop2.7.tgz`.
3. Modify your .bashrc to include the lines

```
export SPARK_HOME=PATH_TO_DIR/spark-2.3.2-bin-hadoop2.7
export PYTHONPATH="${SPARK_HOME}/python/:$PYTHONPATH"
export PYTHONPATH="${SPARK_HOME}/python/lib/py4j-0.10.7-src.zip:$PYTHONPATH"
export PATH=${SPARK_HOME}/bin:$PATH
```

Now you can interactively use Spark with Python `pyspark` or Scala `spark-shell`. To submit whole programs for execution (instead of using the shell), provide a .jar (Scala) or .py to `spark-submit`.

We ask you to use the Scala language for this lab session.

1.1 Creating a .jar from a source code in Scala

There are multiple ways of doing this. One of them is using *sbt*.

1. Download sbt at `https://www.scala-sbt.org/download.html`.
2. Extract the archive with the command `tar zxvf sbt-1-2-7.tgz`
3. Create a folder for your program (we can name it as you wish, in the following we assume its whole path is PROGRAM_FOLDER). inside it create (with *mkdir*) the folders *project/*, *target/*, and *src/main/scala*. Put your .scala source files into *src/main/scala* .

4. In PROGRAM_FOLDER, create a build.sbt file with these contents (you are free do modify the provided name):

```
name := "GoogleTracesAnalysis"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.3.2"
```

5. Still from PROGRAM_FOLDER, run the following command. You will need to either provide the **full path** to the extracted sbt folder (created in step 2) *sbt-1-2-7/bin/* , or add it to your PATH variable as done when installing Spark. If it is successful, the .jar file will be inside *target/scala-2.11/* .

```
sbt package
```

2 The dataset

2.1 About the dataset

The data we will study in this lab have been released by Google in 2011. It represents 29 days of activity of a large-scale Google machine (a cluster) featuring about 12.5k machines. It includes information about the jobs executed on this cluster during that period as well as information about the corresponding resource usage.

A documentation including a detailed description of the dataset written by people from Google can be found in this document: `https://drive.google.com/open?id=0B5g07T_gRDg9Z0lsSTEtTWtpOW8`. We refer to this document as the *Google Documentation* in the following.

We give additional information about the dataset in Section 2.3.

2.2 Downloading the data

The data we are going to manipulate are publicly available on Google Storage. We have to use the tool `GSUtil` to download them:

- Information about `GSUtil` can be found here: `https://cloud.google.com/storage/docs/gsutil`
- We recommend you to install `GSUtil` by directly downloading the archive, as described here: `https://cloud.google.com/storage/docs/gsutil_install#alt-install`
- The *Google Documentation* includes a section that describes how to download the data (see Section "Downloading the data" at the end of the document). In a few words:
  - `gsutil ls gs://clusterdata-2011-2/` allows you to see the available files

- `gsutil cp gs://clusterdata-2011-2/machine_events/part-00000-of-00001.csv.gz ./` will copy the file `part-00000-of-00001.csv.gz`, from the machine_events table, to your current directory.

Important comment: The total size of the dataset is huge (41 GB of data). **Do not copy all the data at once**. Large data *tables* have been divided into multiple files. Each file has all data from a timestamp range. It allows you to copy just one file for one *table* and to test your programs on *small* sub-parts of the data.

2.3 Description of the dataset

As already mentioned, a detailed description of the data we are studying is available in the Google Documentation. We provide you with an overview of the data in the following. We only discuss the tables you will need for this lab session.

Important : The file `schema.csv` available on the data repository describes each of the field included in all CSV files comprised in the dataset. In case of doubt, always refer to this file.

When working with Spark in Scala, notice some IDs used in the dataset cannot be converted to Integers. They have to be converted to `Long` Integers instead.

2.3.1 About the machines:

The table about "machine_events" gives a description of the machines available in the system:

- Each machine is identified with a unique ID
- The table gives us information about the resources available on each machine: CPU and Memory. These data are normalized between 0 and 1. It means that the machines with the value 1 for the amount of CPUs are the machines including the more CPUs. A machine with the value 0.4 for the number of CPUs include a number of CPUs that corresponds to 40% of the maximum number
- The table gives us information about the machines that went offline and reconnected during the period that the data covers (field *event type*).

2.3.2 About jobs and tasks:

The programs that are executed on the machines are called *jobs*. A job can be composed of multiple *tasks*.

Jobs and tasks go through different states during their life cycle. A simple life cycle would be something like this: a job is SUBMITted and gets put into a pending queue; soon afterwards, it is SCHEDULEd onto a machine and starts running; some time later it FINISHes successfully. A task or a job might also be EVICTed, for instance if high priority tasks have to be executed instead. A detailed description of the possible states is available in the *Google Documentation*, page 6.

Job events table This table ("job_events") gives mostly information about the state changes of the jobs executing on the cluster (field *event type*) and the time when these changes occur (field *timestamp*). Each job is identified with a unique ID.

Task events table This table ("task_events") includes a large set of information about each task, including:

- The ID of the task (only unique among tasks of the same job);
- The ID of the job it belongs to;
- The ID of the machine on which it is scheduled;
- The priority of the task (A value between 0 and 11, 0 and 1 being the lowest priority);
- The amount of CPU cores and Memory space requested for the task (value normalized between 0 and 1, as explained previously).

3 Analyses to be conducted

Use Spark to answer the following questions. For help with the Spark API, consult the RDD documentation: `https://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-operations`. To help you get started, we provide an example of scala code in Analyzer.tgz.

1. This dataset contains a few anomalies. Notably, some tables (such as machine_events) contain some entries with fewer columns than defined in schema.csv. Before manipulating a table, be sure to filter out those entries.

   How many entries were filtered out from the machine_events table? They come from how many different machines?

2. What is the distribution of the machines according to their CPU capacity?
3. Print all "priority" values tasks may have.
4. How many different tasks were executed?
5. On average, how many tasks compose a job?
6. What is the percentage of jobs/tasks that got killed or evicted?
7. On average, how many tasks get killed or evicted per job?
8. Do tasks with low priority (0 or 1) have a higher probability of being evicted?
9. Is there a relation between the priority of a task and the amount of resources available on the machine it runs on?

We suggest you interact with Spark by its shell to analyze data and find out how to answer the questions. By the end of this part, combine all your code into a single program that answers these questions in this order.

4 Performance Analysis

Draw the DAG (directed acyclic graph) that represents your program, identifying all narrow and wide dependencies and the actions. Use it to identify intermediate RDDs that are reused multiple times, then use the "persist" function with adequate arguments to cache some of them in memory.

Evaluate the performance with and without the persist calls. To measure time, instead of using the shell, submit the whole program to Spark.

Repeat this analysis a few times progressively increasing the amount of data (number of files from each table) being treated. Does the observed impact from caching RDDs remain the same as we increase the amount of data?