Ask Bigger Questions

CAP - Developing with Spark and Hadoop:

Homework Assignment Guide for Students

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Homework: View Jobs and Stages in the Spark Application UI

Files and Data Used in This Homework:

Exercise Directory: `$DEV1/exercises/spark-stages`

Data files (HDFS):

`/loudacre/weblogs/*`

`/loudacre/accounts/*`

Test Scripts:

`SparkStages.pyspark`

`SparkStages.scalaspark`

In this Exercise you will use the Spark Application UI to view the execution stages for a job.

In a previous exercise, you wrote a script in the Spark Shell to join data from the accounts dataset with the weblogs dataset, in order to determine the total number of web hits for every account. Now you will explore the stages and tasks involved in that job.

Explore Partitioning of file-based RDDs

1. Start (or restart, if necessary) the Spark Shell. Although you would typically run a Spark application on a cluster, your course VM cluster has only a single worker node that can support only a single executor. To simulate a more realistic multi-node cluster, run in local mode with 2 threads:

```
$ pyspark --master local[2]
```

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

```
$ spark-shell --master local[2]
```

2. Review the accounts dataset (`/loudacre/accounts/`) using Hue or command line. Take note of the number of files.

3. Create an RDD based on a *single file* in the dataset, e.g. `/loudacre/accounts/part-m-00000` and then call `toDebugString` on the RDD, which displays the number of partitions in parentheses `()` before the RDD id. How many partitions are in the resulting RDD?

```
pyspark> accounts=sc. \
   textFile("/loudacre/accounts/part-m-00000")
pyspark> print accounts.toDebugString()
```

```
scala> var accounts=sc.
   textFile("/loudacre/accounts/part-m-00000")
scala> accounts.toDebugString
```

4. Repeat this process, but specify a minimum of three partitions: `sc.textFile(filename,3)`

   Does the RDD correctly have three partitions?

5. Finally, create an RDD based on *all the files* in the accounts dataset. How does the number of files in the dataset compare to the number of partitions in the RDD?

6. Bonus: use `foreachPartition` to print out the first record of each partition.

Set up the job

7. First, create an RDD of accounts, keyed by ID and with `first name, last name` for the value.

```
pyspark> accountsByID = accounts \
```

cloudera® © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

```
   .map(lambda s: s.split(',')) \
   .map(lambda values: \
      (values[0],values[4] + ',' + values[3]))
```

```
scala> var accountsByID = accounts.
  map(line => line.split(',')).
  map(values => (values(0),values(4)+','+values(3)))
```

8. The first step is to construct a `userreqs` RDD with the total number of web hits for each user ID:

   Tip: In this exercise you will be reducing and joining large datasets, which can take a lot of time running on a single machine, as you are using in the course. Therefore, rather than use all the web log files in the dataset, specify a subset of web log files using a wildcard, e.g. `textFile("/loudacre/weblogs/*6")`.

```
pyspark> userreqs = sc \
   .textFile("/loudacre/weblogs/*6") \
   .map(lambda line: line.split()) \
   .map(lambda words: (words[2],1)) \
   .reduceByKey(lambda v1,v2: v1+v2)
```

```
scala> var userreqs = sc.
   textFile("/loudacre/weblogs/*6").
   map(line => line.split(' ')).
   map(words => (words(2),1)).
   reduceByKey((v1,v2) => v1 + v2)
```

9. Then join the two RDDs by user ID, and construct a new RDD based on first name, last name and total hits:

```
pyspark> accounthits = accountsByID.join(userreqs)\
```

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

```
  .values()
```

```
scala> var accounthits =
  accountsByID.join(userreqs).map(pair => pair._2)
```

10. Print the results of `accounthits.toDebugString` and review the output. Based on this, see if you can determine

a. How many stages are in this job?

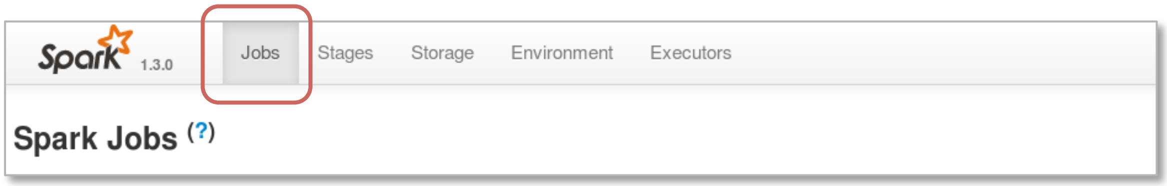
b. Which stages are dependent on which?

c. How many tasks will each stage consist of?

Run the job and review it in the Spark Application UI

11. In your browser, visiting the Spark Application UI by using the provided toolbar bookmark, or visiting URL `http://localhost:4040`

12. In the Spark UI, make sure the **Jobs** tab is selected. No jobs are yet running so the list will be empty.



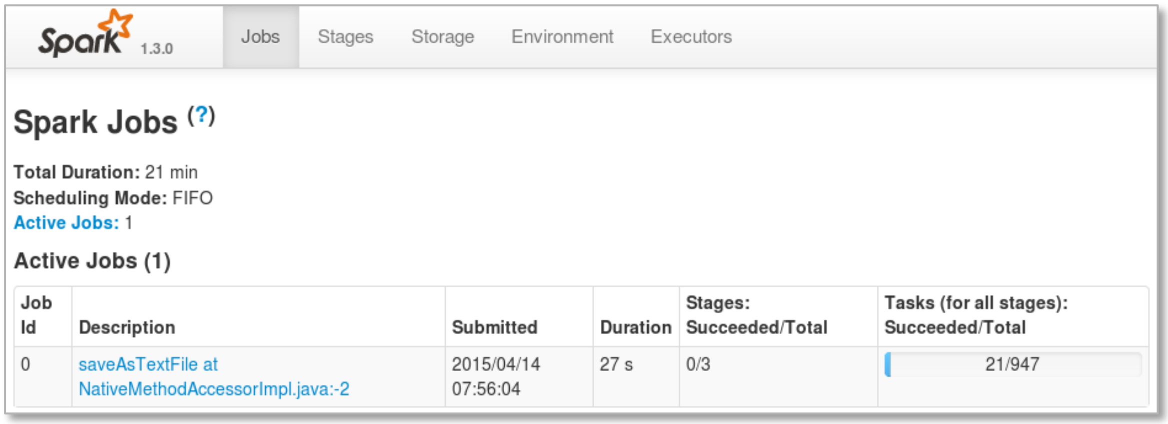
13. Return to the shell and start the job by executing an action (`saveAsTextFile`):

```
pyspark> accounthits.\
  saveAsTextFile("/loudacre/userreqs")
```

```
scala> accounthits.
  saveAsTextFile("/loudacre/userreqs")
```

cloudera
© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

14. Reload the Spark UI Jobs page in your browser. Your job will appear in the Active Jobs list until it completes, and then it will display in the Completed Jobs List.



Spark Jobs (?)

Total Duration: 21 min
Scheduling Mode: FIFO
Active Jobs: 1

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	saveAsTextFile at NativeMethodAccessorImpl.java:-2	2015/04/14 07:56:04	27 s	0/3	21/947

15. Click on the job description (which is the last action in the job) to see the stages. As the job progresses you may want to refresh the page a few times.

Things to note:

a. How many stages are in the job? Does it match the number you expected from the RDD's `toDebugString` output?

b. The stages are numbered, but numbers do not relate to the order of execution. Note the times the stages were submitted to determine the order. Does the order match what you expected based on RDD dependency?

c. How many tasks are in each stage? The number of tasks in the first stages correspond to the number of partitions, which for this example corresponds to the number of files processed.

d. The Shuffle Read and Shuffle Write columns indicate how much data was copied between tasks. This is useful to know because copying too much data across the network can cause performance issues.

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

2. Click on the stages to view details about that stage. Things to note:

    a. The Summary Metrics area shows you how much time was spend on various steps. This can help you narrow down performance problems.

    b. The Tasks area lists each task. The Locality Level column indicates whether the process ran on the same node where the partition was physically stored or not. Remember that Spark will attempt to always run tasks where the data is, but may not always be able to, if the node is busy.

    c. In a real-world cluster, the executor column in the Task area would display the different worker nodes that ran the tasks. (In this single-node cluster, all tasks run on the same host: localhost.)

3. When the job is complete, return to the **Jobs** tab to see the final statistics for the number of tasks executed and the time the job took.

4. *Optional*: Try re-running the last action. (You will need to either delete the `saveAsTextFile` output directory in HDFS, or specify a different directory name.) You will probably find that the job completes much faster, and that several stages (and the tasks in them) show as "skipped".

    Bonus question: Which tasks were skipped and why?

This is the end of the Homework