Politecnico di Milano

Middleware Technologies for Distributed Systems

Project Report

Analysis of COVID-19 Data

Author:
Fabio Losavio
Matteo Velati
Niccolò Zangrando

Supervisor:
Dr. Alessandro Margara

April 11, 2021

1 Introduction

1.1 Project description

We have implemented a program that analyzes open datasets to study the evolution of the COVID-19 situation worldwide. The program starts from the dataset of new reported cases for each country daily and computes the following queries:

1. Seven days moving average of new reported cases, for each country and for each day
2. Percentage increase (with respect to the day before) of the seven days moving average, for each country and for each day
3. Top 10 countries with the highest percentage increase of the seven days moving average, for each day

1.2 Technology used

We have decided to use the *Apache Spark* technology. In particular, given the nature of the project, we decided to use the *Spark SQL* engine.
In the following sections, we are going to describe how we have exploited the three different queries, and then we will show and comment the different performance results we have obtained with different configurations.

2 Implementation

As mentioned above, given the nature of the problem and the format in .csv of the dataset [1], the solution has been implemented using the *SQL engine of Apache Spark*.

2.1 Query 1: Seven days moving average of new reported cases, for each country and for each day

The moving average can be defined as:

$$movingAverage = \frac{1}{n}\sum_{t=-n}^{1} a_t \tag{1}$$

where *a=value at period t*, and *n=number of time periods*.

Since we have a dataset of new reported cases for each country daily, we can split the computation of the seven days moving average for each country simply using the SQL *"partition by" Window function*, order the rows by date and then calculate the average between each row and its six previous. The query will be:

```
final Dataset<Row> dateCountry_ma = covidData
    .select(col("dateRep"), col("countriesAndTerritories"),
        avg("cases").over(Window
            .partitionBy("countriesAndTerritories")
        .orderBy("dateRep").rowsBetween(-6, 0))
            .as("7days_moving_average"))
    .orderBy("countriesAndTerritories", "dateRep");
```

Listing 1: *Query 1 solution*

where *covidData* is the dataframe created from the .csv dataset.

[1]https://opendata.ecdc.europa.eu/covid19/casedistribution/csv/data.csv

2.2 Query 2: Percentage increase (with respect to the day before) of the seven days moving average, for each country and for each day

The percentage increase (with respect to the day before) of the seven days moving average can be calculated as:

$$increase = 100 * \frac{7ma_x}{7ma_{x-1}} - 100 \tag{2}$$

where

$7ma_x$=*is the seven days moving average at day x*, and $7ma_{x-1}$=*is the seven days moving average at day x-1.*

The approach is very similar to the previous one. Using the first query solution [Listing 1], we can split the computation for each country using the the SQL *"partition by" Window function*, order the rows by date and then calculate the ratio between each row and its previous one.
The query will be:

```java
final Dataset<Row> dateCountry_increase = date_Country_ma
    .select(col("dateRep"), col("countriesAndTerritories"),
        col("7days_moving_average").divide(lag(
        col("7_days_moving_average"), 1)
            .over(Window
                .partitionBy("countriesAndTerritories")
            .orderBy("dateRep"))).multiply(100)
                .minus(100).as("increase_day_before"))
    .orderBy("countriesAndTerritories", "dateRep")
    .na().fill(0);
```

Listing 2: *Query 2 solution*

2.3 Query 3: Top 10 countries with the highest percentage increase of the seven days moving average, for each day

In this case, starting from the query 2 solution [Listing 2] we can use the SQL *"partition by" Window function* for partitioning the dataset by date. Then, for each day, we can order the countries by the percentage increase

(from the biggest value) and keep only the first 10 rows.
The query will be:

```java
final Dataset<Row> topCountry_increase =
    date_Country_increase
    .select(col("dateRep"), col("countriesAndTerritories"),
        col("increase_day_before"), row_number().over(Window
            .partitionBy("dateRep").orderBy(
                desc("increase_day_before")))
                .alias("rank"))
    .filter(col("rank").leq(10))
    .orderBy(desc("dateRep"), desc("increase_day_before"));
```

Listing 3: Query 3 solution

3 Analysis and Testing

In this section we are going to describe the analysis and tests we have done with different configurations.

3.1 Hardware

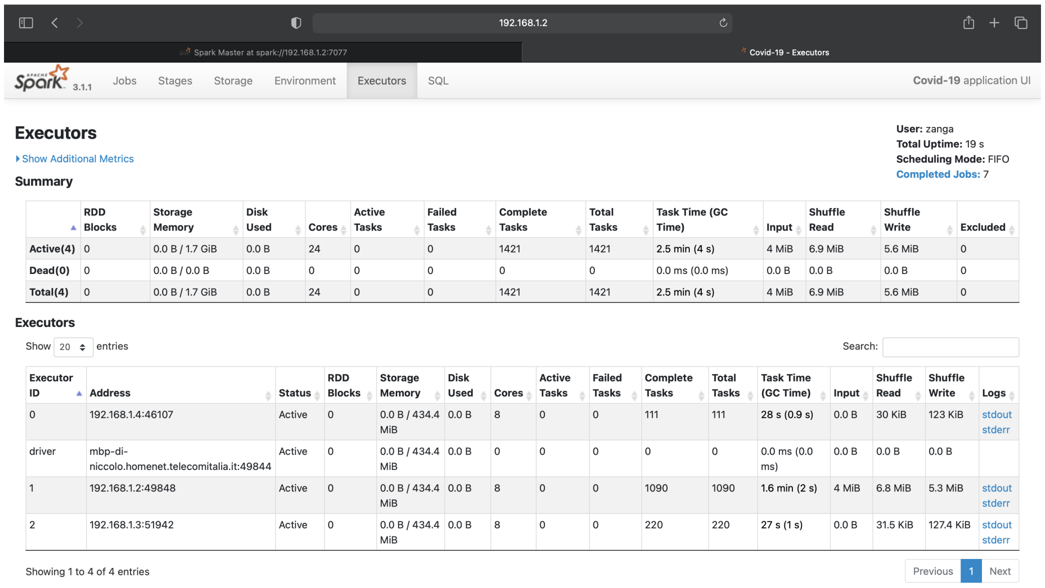
For all the tests we have used 3 machines connected to the same local network, with 1 Master and 3 Workers. The specifications of each machine are as follows:

- Machine A:
  - CPU: Intel Core i5-8259U @ 2.30GHz
  - RAM: 8GB
- Machine B:
  - CPU: Intel Core i5-8259U @ 2.30GHz
  - RAM: 8GB
- Machine C:
  - CPU: Intel Core i7-8550U @ 1.80GHz
  - RAM: 8GB

3.2 Tests

3.2.1 Test 1

In this test we have submitted the program to the spark cluster with the default parameters using all the 24 available cores (8 for each machine). Here are shown the results obtained:



Executors

Show Additional Metrics

User: zanga
Total Uptime: 19 s
Scheduling Mode: FIFO
Completed Jobs: 7

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(4)	0	0.0 B / 1.7 GiB	0.0 B	24	0	0	1421	1421	2.5 min (4 s)	4 MiB	6.9 MiB	5.6 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(4)	0	0.0 B / 1.7 GiB	0.0 B	24	0	0	1421	1421	2.5 min (4 s)	4 MiB	6.9 MiB	5.6 MiB	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
0	192.168.1.4:46107	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	111	111	28 s (0.9 s)	0.0 B	30 KiB	123 KiB	stdout stderr
driver	mbp-di-niccolo.homenet.telecomitalia.it:49844	Active	0	0.0 B / 434.4 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	
1	192.168.1.2:49848	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	1090	1090	1.6 min (2 s)	4 MiB	6.8 MiB	5.3 MiB	stdout stderr
2	192.168.1.3:51942	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	220	220	27 s (1 s)	0.0 B	31.5 KiB	127.4 KiB	stdout stderr

Showing 1 to 4 of 4 entries

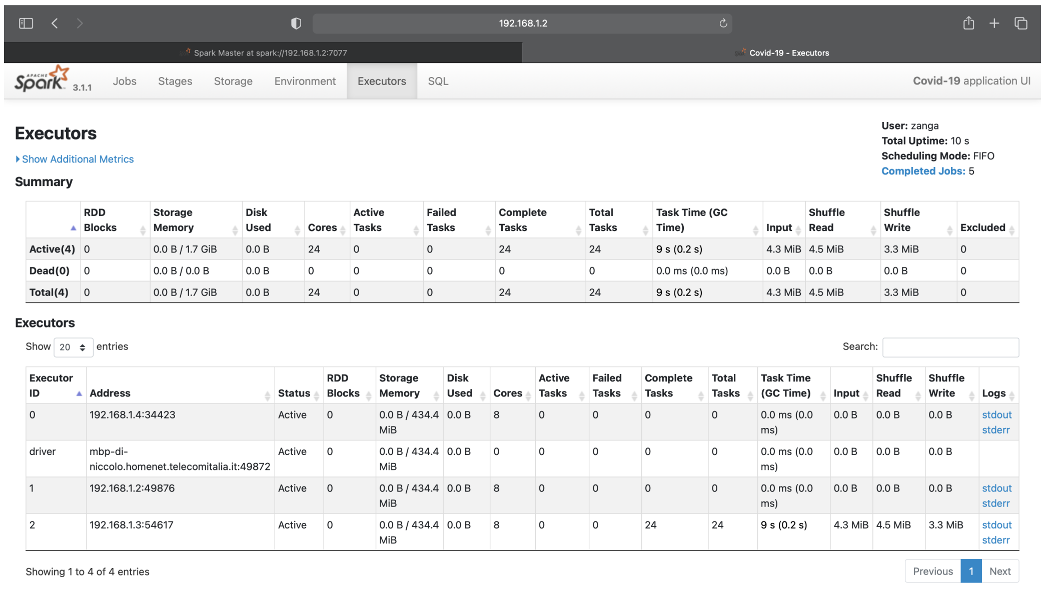
Previous 1 Next

Figure 1: test 1 results

As you can see, 1421 tasks have been executed in 19 seconds and the workload has not been evenly divided among the Executors.

3.2.2 Test 2

In this test we have tried to increase the performace. We know that the number of tasks depends also on the number partitions. By setting the configuration parameter *"spark.sql.shuffle.partitions=3"* we have obtained the results shown in Figure 2.



Executors

Show Additional Metrics

User: zanga
Total Uptime: 10 s
Scheduling Mode: FIFO
Completed Jobs: 5

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(4)	0	0.0 B / 1.7 GiB	0.0 B	24	0	0	24	24	9 s (0.2 s)	4.3 MiB	4.5 MiB	3.3 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(4)	0	0.0 B / 1.7 GiB	0.0 B	24	0	0	24	24	9 s (0.2 s)	4.3 MiB	4.5 MiB	3.3 MiB	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
0	192.168.1.4:34423	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr
driver	mbp-di-niccolo.homenet.telecomitalia.it:49872	Active	0	0.0 B / 434.4 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	
1	192.168.1.2:49876	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr
2	192.168.1.3:54617	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	24	24	9 s (0.2 s)	4.3 MiB	4.5 MiB	3.3 MiB	stdout stderr

Showing 1 to 4 of 4 entries

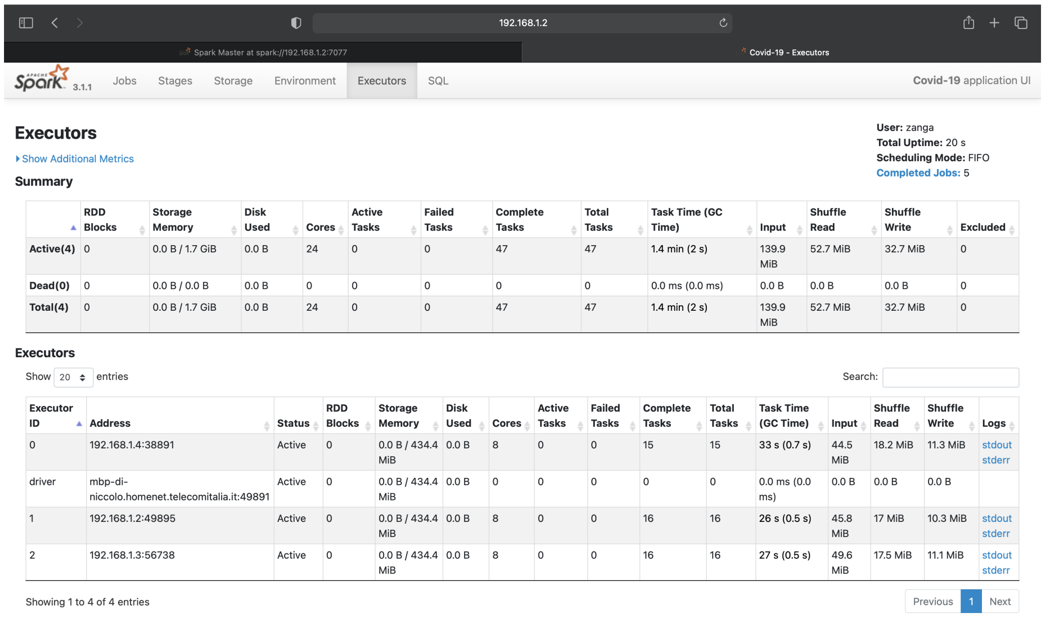
Previous 1 Next

Figure 2: test 2 results

In this test 24 tasks have been executed in 10 seconds. We observed that all the tasks have been executed by one Executor and the execution time has been halved. This behavior is justified by the Apache Spark architecture, in fact since the computation workload is very small (the dataset is only 4MB), Apache Spark try not to split the work among different nodes.

3.3 Test 3

Since the original dataset is only 4MB, we have increased the dataset size by simply duplicating many times the original entries. Now the dataset size is about 130MB. By setting *"spark.sql.shuffle.partitions=3"* we have obtained the results shown in Figure 3.



Executors

User: zanga
Total Uptime: 20 s
Scheduling Mode: FIFO
Completed Jobs: 5

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(4)	0	0.0 B / 1.7 GiB	0.0 B	24	0	0	47	47	1.4 min (2 s)	139.9 MiB	52.7 MiB	32.7 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(4)	0	0.0 B / 1.7 GiB	0.0 B	24	0	0	47	47	1.4 min (2 s)	139.9 MiB	52.7 MiB	32.7 MiB	0

Executors

Show 20 entries   Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
0	192.168.1.4:38891	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	15	15	33 s (0.7 s)	44.5 MiB	18.2 MiB	11.3 MiB	stdout stderr
driver	mbp-di-niccolo.homenet.telecomitalia.it:49891	Active	0	0.0 B / 434.4 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	
1	192.168.1.2:49895	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	16	16	26 s (0.5 s)	45.8 MiB	17 MiB	10.3 MiB	stdout stderr
2	192.168.1.3:56738	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	16	16	27 s (0.5 s)	49.6 MiB	17.5 MiB	11.1 MiB	stdout stderr

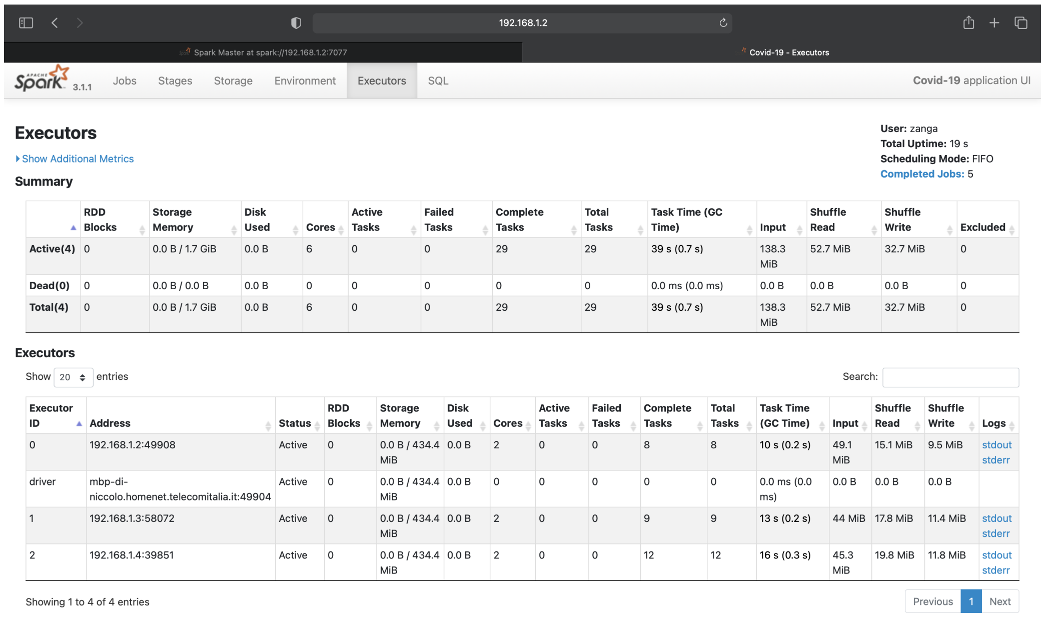
Showing 1 to 4 of 4 entries

Figure 3: test 3 results

In this case the 47 tasks have been equally divided among all the Executors and the computation finished in 20 seconds.

3.4 Test 4

For this test we have used the "big" dataset with a different configuration. In fact we have used only 2 cores per Worker, but we have set *"spark.sql.shuffle.partitions=3"* as before. In [Fig. 4] we can see that has been processed 29 tasks in 19 seconds. The execution time is more or less the same despite lower computing power (6 cores). On the other side the execution has been divided in less tasks than *Test 3* [Fig.3].



Executors

Show Additional Metrics

User: zanga
Total Uptime: 19 s
Scheduling Mode: FIFO
Completed Jobs: 5

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(4)	0	0.0 B / 1.7 GiB	0.0 B	6	0	0	29	29	39 s (0.7 s)	138.3 MiB	52.7 MiB	32.7 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(4)	0	0.0 B / 1.7 GiB	0.0 B	6	0	0	29	29	39 s (0.7 s)	138.3 MiB	52.7 MiB	32.7 MiB	0

Executors

Show 20 entries

Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
0	192.168.1.2:49908	Active	0	0.0 B / 434.4 MiB	0.0 B	2	0	0	8	8	10 s (0.2 s)	49.1 MiB	15.1 MiB	9.5 MiB	stdout stderr
driver	mbp-di-niccolo.homenet.telecomitalia.it:49904	Active	0	0.0 B / 434.4 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	
1	192.168.1.3:58072	Active	0	0.0 B / 434.4 MiB	0.0 B	2	0	0	9	9	13 s (0.2 s)	44 MiB	17.8 MiB	11.4 MiB	stdout stderr
2	192.168.1.4:39851	Active	0	0.0 B / 434.4 MiB	0.0 B	2	0	0	12	12	16 s (0.3 s)	45.3 MiB	19.8 MiB	11.8 MiB	stdout stderr

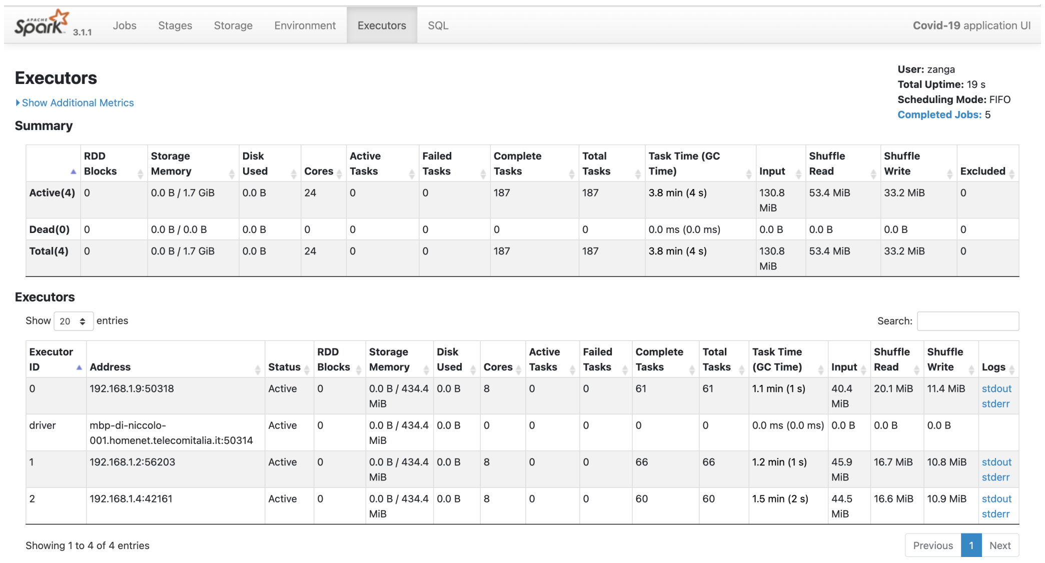
Showing 1 to 4 of 4 entries

Previous 1 Next

Figure 4: test 4 results

3.5 Test 5

Since in *Test 4* (Fig.4) we have reached better performances than *Test 3* (Fig.3), we have changed again the configuration. We set *"spark.sql.shuffle.partitions=24"* and we used all 24 available cores.



Executors

User: zanga
Total Uptime: 19 s
Scheduling Mode: FIFO
Completed Jobs: 5

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(4)	0	0.0 B / 1.7 GiB	0.0 B	24	0	0	187	187	3.8 min (4 s)	130.8 MiB	53.4 MiB	33.2 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(4)	0	0.0 B / 1.7 GiB	0.0 B	24	0	0	187	187	3.8 min (4 s)	130.8 MiB	53.4 MiB	33.2 MiB	0

Executors

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
0	192.168.1.9:50318	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	61	61	1.1 min (1 s)	40.4 MiB	20.1 MiB	11.4 MiB	stdout stderr
driver	mbp-di-niccolo-001.homenet.telecomitalia.it:50314	Active	0	0.0 B / 434.4 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	
1	192.168.1.2:56203	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	66	66	1.2 min (1 s)	45.9 MiB	16.7 MiB	10.8 MiB	stdout stderr
2	192.168.1.4:42161	Active	0	0.0 B / 434.4 MiB	0.0 B	8	0	0	60	60	1.5 min (2 s)	44.5 MiB	16.6 MiB	10.9 MiB	stdout stderr

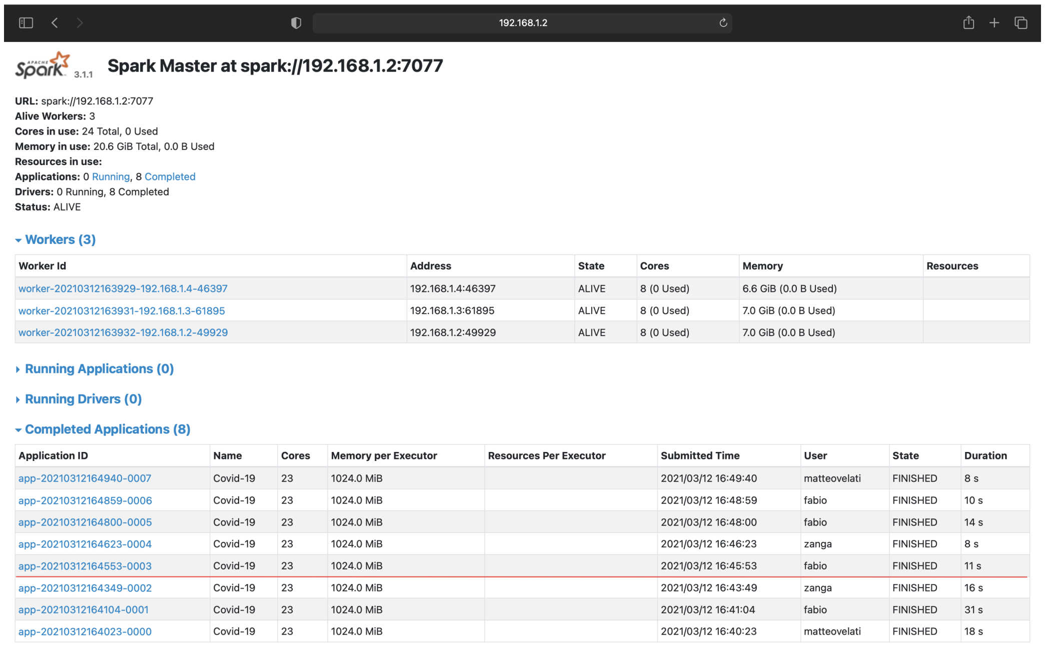
Showing 1 to 4 of 4 entries

Figure 5: test 5 results

4 Conclusion

Is very important to find the correct configuration, based on the submitted application, of the Apache Spark cluster. For example, for small computations like in *Test 1* (Fig.1) and *Test 2* (Fig.2), is useless and lowers the performance creating a huge quantity of tasks. On the other side also creating too few tasks can be useless because, in this case, we would not be making full use of parallelization (Fig. 3). Of course this technology improves performance for big computations, but if we run the application in local mode we can see the benefits. In fact, if we use 1 core with *"spark.sql.shuffle.partitions=1"* we have the solutions in 7 seconds; with 4 cores, we can reach 5,8 seconds.

In the figure below are shown other tests in cluster mode with different drivers (For the applications under the red line we have used the "big" dataset, for the ones above the red line we have used the original dataset).


192.168.1.2

3.1.1 Spark Master at spark://192.168.1.2:7077

URL: spark://192.168.1.2:7077
Alive Workers: 3
Cores in use: 24 Total, 0 Used
Memory in use: 20.6 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 8 Completed
Drivers: 0 Running, 8 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory	Resources
worker-20210312163929-192.168.1.4-46397	192.168.1.4:46397	ALIVE	8 (0 Used)	6.6 GiB (0.0 B Used)	
worker-20210312163931-192.168.1.3-61895	192.168.1.3:61895	ALIVE	8 (0 Used)	7.0 GiB (0.0 B Used)	
worker-20210312163932-192.168.1.2-49929	192.168.1.2:49929	ALIVE	8 (0 Used)	7.0 GiB (0.0 B Used)	

Running Applications (0)

Running Drivers (0)

Completed Applications (8)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20210312164940-0007	Covid-19	23	1024.0 MiB		2021/03/12 16:49:40	matteovelati	FINISHED	8 s
app-20210312164859-0006	Covid-19	23	1024.0 MiB		2021/03/12 16:48:59	fabio	FINISHED	10 s
app-20210312164800-0005	Covid-19	23	1024.0 MiB		2021/03/12 16:48:00	fabio	FINISHED	14 s
app-20210312164623-0004	Covid-19	23	1024.0 MiB		2021/03/12 16:46:23	zanga	FINISHED	8 s
app-20210312164553-0003	Covid-19	23	1024.0 MiB		2021/03/12 16:45:53	fabio	FINISHED	11 s
app-20210312164349-0002	Covid-19	23	1024.0 MiB		2021/03/12 16:43:49	zanga	FINISHED	16 s
app-20210312164104-0001	Covid-19	23	1024.0 MiB		2021/03/12 16:41:04	fabio	FINISHED	31 s
app-20210312164023-0000	Covid-19	23	1024.0 MiB		2021/03/12 16:40:23	matteovelati	FINISHED	18 s

Figure 6: tests results