University of St.Gallen

Institute of Computer Science

Distributed Systems, Fall Semester 2023
Sanjiv Jha, Jérémy Lemée, & Andrei Ciortea
{sanjiv.jha, jeremy.lemee, andrei.ciortea}@unisg.ch

Assignment 3: Spark Cluster using Raspberry Pi (7pt)

Deadline: Nov 14, 2023; 23:59 CET

In the first assignment, we looked at MapReduce and its drawbacks. We also simulated a distributed system on our local machines to compute word counts with MapReduce. In this assignment, you will simulate an actual cluster using your local machine and a Raspberry Pi to implement the word count algorithm using Apache Spark. To this end, we will use Java 17 and 8[1], Apache Spark 3.5.0[2] and Hadoop 3.3.1[3].

① **(2pt)** Your first task is to **implement the word count algorithm using Spark** and run it on your local machine. To do this, follow the comments given in the project template `WordCount.java`[4] and optionally, test your solution using the provided test file `CheckOutput.java`. Note that the test file expects the output as key-value pairs, in the format `(word,frequency)`. For example, `(the,83)`. Save the output file on your local device and print it on the terminal of the Spark Master. Your implementation might generate multiple output part files depending on the number of partitions Spark uses. Use output consolidation methods to consolidate your output into a single output file. Please also help us understand your code through inline comments.

To run the project, First, set up the Spark and Hadoop environments on your local machine. Please use the steps given in the `README` to set the environment and use Java 17 to build this task. Use the commands associated with this task in the `README`. Update the assignment package and submit the `output-task1.txt` in the assignment package. You can compare your output with the given sample `output.txt`.

② **(4pt)** In the first assignment, you simulated a distributed system using RPC for communication among multiple processes running on your machine. In this task, you will **create a small cluster for computing word counts using your machine and a Raspberry Pi (RPi)**. To do this, follow the setup guide in the project `README` and build a Spark cluster using your local machine as the master, a worker node (localhost) running on your local machine, and RPi as another worker node. The cluster involves more than one machine. Therefore, we use the Hadoop File System (HDFS) to make the required files (including input and output) accessible to all the machines (the master and the worker nodes).

Once the setup is done, you can link the workers (RPi and the worker process running on your local machine) to the master–your local machine. Update the project template from *Task1* to run on the cluster. To do so, take your implementation from *Task1* and update the file input and output paths with HDFS file paths, and the code if needed (no need to make separate files for the two tasks; just update the modified `WordCount.java` for this task). Execute the given commands (for this task in the `README`) from the master node. The output sample can be seen on the master UI (see Figure 1:Left). The output should be the same as in *Task 1*. However, the number of output files generated may vary depending on the number of executors. For example, the two workers (running on your local machine and the RPi) would produce two output part files. If you

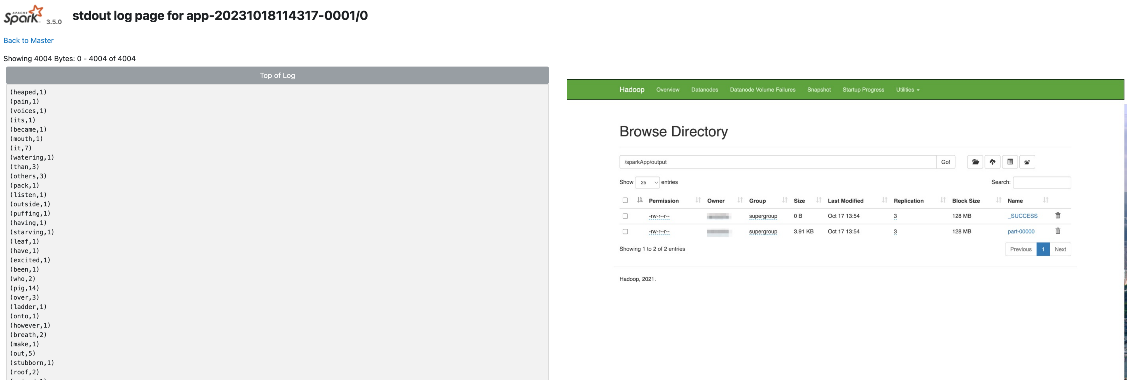
[1] Tutorial for installing Java https://www.geeksforgeeks.org/download-and-install-java-development-kit-jdk-on-windows-mac-and-linux/ (for Hadoop only), https://www.oracle.com/java/technologies/downloads/

[2] https://spark.apache.org/docs/latest/

[3] https://hadoop.apache.org/release/3.3.1.html

[4] https://github.com/HSG-DS-HS23/Assignment3

have not already consolidated output for *Task 1* and it is necessary, update your code to combine the output into a single file (see Figure 1:Right).



stdout log page for app-20231018114317-0001/0

Back to Master

Showing 4004 Bytes: 0 - 4004 of 4004

Top of Log

```
(heaped,1)
(pain,1)
(voices,1)
(its,1)
(became,1)
(mouth,1)
(it,7)
(watering,1)
(than,3)
(others,3)
(pack,1)
(listen,1)
(outside,1)
(puffing,1)
(having,1)
(starving,1)
(leaf,1)
(have,1)
(excited,1)
(been,1)
(who,2)
(pig,14)
(over,3)
(ladder,1)
(onto,1)
(however,1)
(breath,2)
(make,1)
(out,5)
(stubborn,1)
(roof,2)
```

Browse Directory

/sparkApp/output

Show 25 entries

Search:

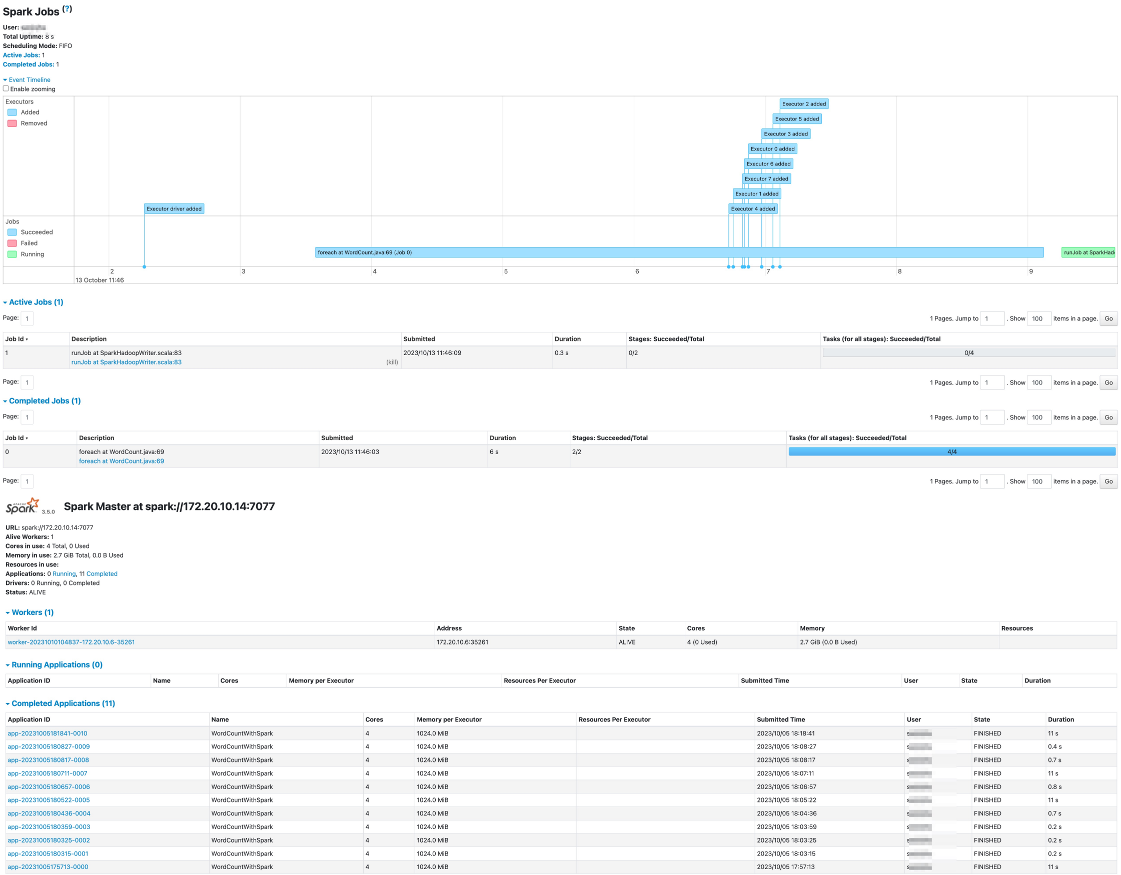
□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	[redacted]	supergroup	0 B	Oct 17 13:54	3	128 MB	_SUCCESS
□	-rw-r--r--	[redacted]	supergroup	3.91 KB	Oct 17 13:54	3	128 MB	part-00000

Showing 1 to 2 of 2 entries

Hadoop, 2021.

Figure 1: *Left*: Master UI showing the output from running a single RPi worker; *Right*: Output files saved on HDFS using two working executors

Finally, to complete the task, in the `Report.md` file, explain your arguments about the performance of this setup (performing the word count algorithm using cluster mode) in comparison to the local implementation (client mode) that you performed in *Task 1*. For the submission, update the project template to run on the cluster, submit a screenshot of the master UI showing all the connected workers (for example, one worker node can be seen connected in Figure 2), the RPi itself, and a zip of the output files (download it from the HDFS – see Figure 1:Right).



Spark Jobs (?)

User: [redacted]
Total Uptime: 8 s
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 1

▾ Event Timeline
□ Enable zooming

▾ Active Jobs (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	runJob at SparkHadoopWriter.scala:83 runJob at SparkHadoopWriter.scala:83 (kill)	2023/10/13 11:46:09	0.3 s	0/2	0/4

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

▾ Completed Jobs (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	foreach at WordCount.java:69 foreach at WordCount.java:69	2023/10/13 11:46:03	6 s	2/2	4/4

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Spark 3.5.0 Spark Master at spark://172.20.10.14:7077

URL: spark://172.20.10.14:7077
Alive Workers: 1
Cores in use: 4 Total, 0 Used
Memory in use: 2.7 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 11 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

▾ Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20231010104837-172.20.10.6-35261	172.20.10.6:35261	ALIVE	4 (0 Used)	2.7 GiB (0.0 B Used)	

▾ Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

▾ Completed Applications (11)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20231005181841-0010	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:18:41	[redacted]	FINISHED	11 s
app-20231005180827-0009	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:08:27	[redacted]	FINISHED	0.4 s
app-20231005180817-0008	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:08:17	[redacted]	FINISHED	0.7 s
app-20231005180711-0007	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:07:11	[redacted]	FINISHED	11 s
app-20231005180657-0006	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:06:57	[redacted]	FINISHED	0.8 s
app-20231005180522-0005	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:05:22	[redacted]	FINISHED	11 s
app-20231005180436-0004	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:04:36	[redacted]	FINISHED	0.7 s
app-20231005180359-0003	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:03:59	[redacted]	FINISHED	0.2 s
app-20231005180325-0002	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:03:25	[redacted]	FINISHED	0.2 s
app-20231005180315-0001	WordCountWithSpark	4	1024.0 MiB		2023/10/05 18:03:15	[redacted]	FINISHED	0.2 s
app-20231005175713-0000	WordCountWithSpark	4	1024.0 MiB		2023/10/05 17:57:13	[redacted]	FINISHED	11 s

Figure 2: Master UI showing the executed tasks

③ (1pt)

1.) How does Spark optimize its file access compared to the file access in MapReduce?

2.) In your implementation of WordCount (*Task 1*), did you use the ReduceByKey or groupByKey method? What does your preferred method do in your implementation? What are the differences between the two methods in Spark?

3.) Explain what a *Resilient Distributed Dataset (RDD)* is and the benefits it brings to the classic MapReduce model.

4.) Imagine that you have a large dataset that needs to be processed in parallel. How would you partition the dataset efficiently and keep control over the number of outputs created at the end of the execution? If a task is stuck on the Spark cluster due to a network issue that the cluster had during execution, which methods can be used to retry or restart the task execution on a node?

Hand-in Instructions By the deadline, you should hand in a single **zip** file via Canvas upload. The name of this file should start with `a3` and contain the last names of all team members separated by underscores (e.g., `a3_jha_lemee_ciortea.zip`). It should contain the following files:

- All answers to the assignment questions in the given `REPORT.md`; if you wish to submit your solution code via GitHub, please include a link to your GitHub repository as well
- Task1: Output from Task 1 as `output-task1.txt`
- Task2: Output from Task 2 as a screenshot of the master UI showing all the connected workers `masterUI-task2.png`, the updated project template without build files (if submitting through canvas), and a zip of the output downloaded from the HDFS.
- Please return the RPi package after solving the assignment.

Across all tasks in this and the other assignments in this course, you are **required to declare** any support that you received from others and, within reasonable bounds,[5] any support tools that you were using while solving the assignment.

[5] It is not required that you declare that you were using text-editing software with orthographic correction; it is however required to declare if you were using any non-standard tools such as generative machine learning models such as GPT