CS 744 Group 15 Assignment 1

Team members: Yuhao Kang (yuhao.kang@wisc.edu), Jinmeng Rao (jinmeng.rao@wisc.edu), Xinyi Liu (xliu636@wisc.edu)

1.Environment Setup

We follow the instructions on the course website, setup the Apache Hadoop and Apache Spark on the three-node cluster machines.

To do so, we first setup three machines. The information of the machines is shown below (Figure 1):

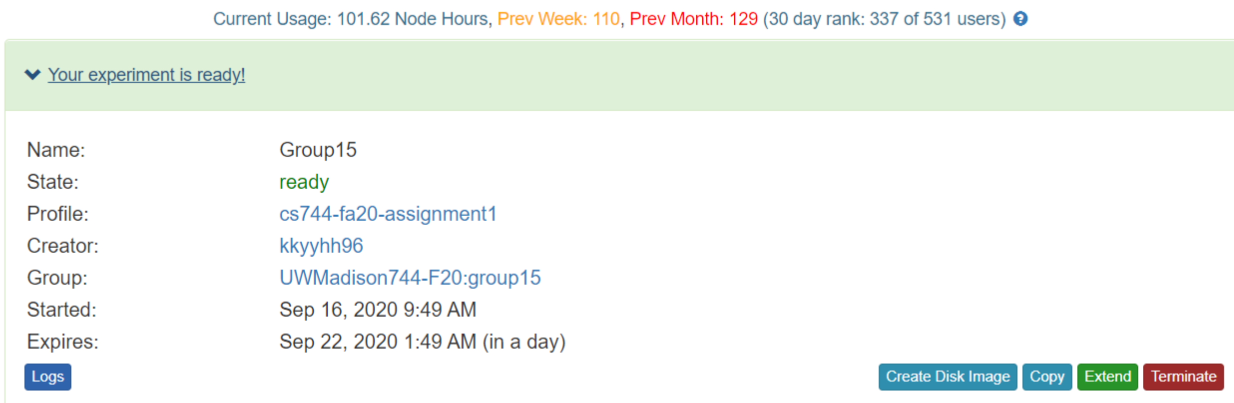


Figure 1 Information of our machines from CloudLab UI

The first step is to enable SSH service among nodes on the cluster. By syncing the master node public key to other slave nodes, master node can communicate with slave nodes.
We met an error as it showed no permission (public key). The reason is that there should be no password for the public key.

Then, we mounted the data disk so that it can store multiple the two datasets we used in the experiment.
We then edited the configuration files including *hadoop-3.1.2/etc/hadoop/core-site.xml*, *hadoop-3.1.2/etc/hadoop/hdfs-site.xml*, and added JAVA_HOME to *hadoop-3.1.2/etc/hadoop/hadoop-env.sh*. We met an error when editing the file *hadoop-3.1.2/etc/hadoop/workers*. After removing "localhost" in the file, it works well.
Apache Hadoop was setup successfully after all these procedures.

After installing Hadoop, we shift to setup the Apache Spark. We edited the configuration files such as *spark-2.4.7-bin-hadoop2.7/conf/slaves* and *spark-2.4.7-bin-hadoop2.7/conf/spark-env.sh*. We met an error and didn't figure it out. So we propose the question on the Piazza (https://piazza.com/class/kcnrt66wd4v4ot?cid=35) as shown in the Figure 2 below. We figured it out by adding setting SPARK_MASTER_HOST and SPARK_LOCAL_IP in *spark-2.4.6-bin-hadoop2.7/conf/spark-env.sh*.


question @35

stop following 79 views

Actions

Spark submit script error

We met a problem when running our Spark application following this:

```
In order to run your Spark application you need to submit it using spark-submit script from Spark's bin directory. More details on submitting applications could be found here.
```

We checked the documentation, and tried to run the following codes:

```
# Run a Python application on a Spark standalone cluster
./bin/spark-submit \
  --master spark://10.10.1.1:7077 \
  examples/src/main/python/pi.py \
  1000
```

But it failed with the following error:

```
INFO StandaloneAppClient$ClientEndpoint: Connecting to master spark://10.10.1.1:7077...
WARN StandaloneAppClient$ClientEndpoint: Failed to connect to master 10.10.1.1:7077
```

We have double-checked that Hadoop and Spark are indeed running. So how to deal with it? Thanks!

Figure 2 Question about Spark submit script error

We also edited the properties of the memory and CPU used by Spark. Then the Spark can be executed successfully.

2.A simple Spark application

In this part, we implement a simple Spark Application (sorting) and submit it to Spark to run. This part basically follow two steps:

1. Load the export.csv data from hdfs;
2. Run sorting algorithm and save output into HDFS as a csv file.

For the first step, we first need to copy the export.csv file to HDFS using hadoop fs -put command, and then we load the data from HDFS using the path as "hdfs://10.10.1.1:9000/export.csv".

For the second step, we implemented the sorting application using PySpark API (see code in part2 folder) as a .py file, and then we wrote a .sh file to submit the application .py file to Spark to run:

```
#!/bin/sh

../spark-2.4.7-bin-hadoop2.7/bin/spark-submit \
 --master spark://10.10.1.1:7077 \
 part2_py.py \
 1000
```

Note that the input and output file paths are written in the code. In order to run the code, you could simply run sh part2_run.sh in the part2 folder. The application will read the input data from HDFS, finish the sorting, and save the output into HDFS as ‘part2_output’.

3.PageRank

Task 1. Write a Scala/Python/Java Spark application that implements the PageRank algorithm. We write Python code for Page Rank algorithm which drops the RDD to memory or disk and takes input/output file paths as well as partitioning number from system input. The attached .py file named ‘part3_pagerank’ shows the memory example.

Figure 3 shows an example of the DAG visualization of lineage graph. Stage 0 stands for the data reading stage. Stage 1~10 stand for the iterations of the PageRank algorithm. Stage 12 stands for the data writing stage. Figure 4 shows the detailed DAG visualization of one stage. We can clearly see how the data are read, cached, grouped by key, etc.

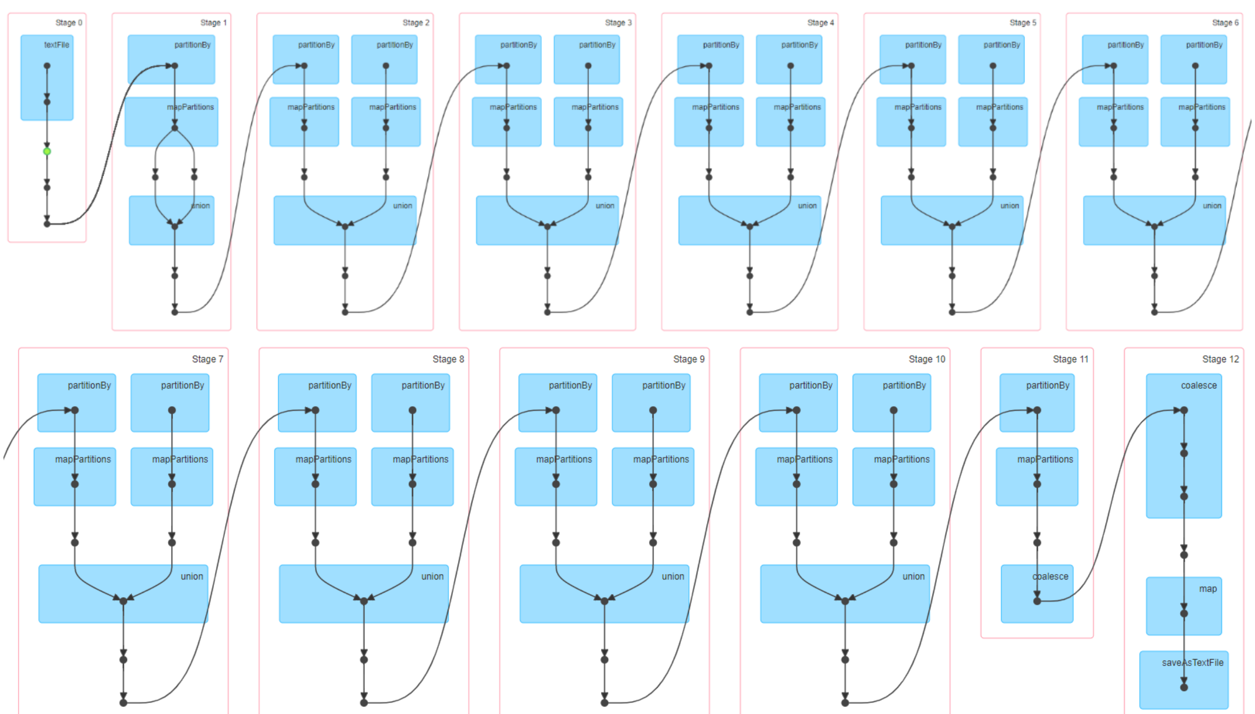


Figure 3 DAG Visualization of Lineage Graph

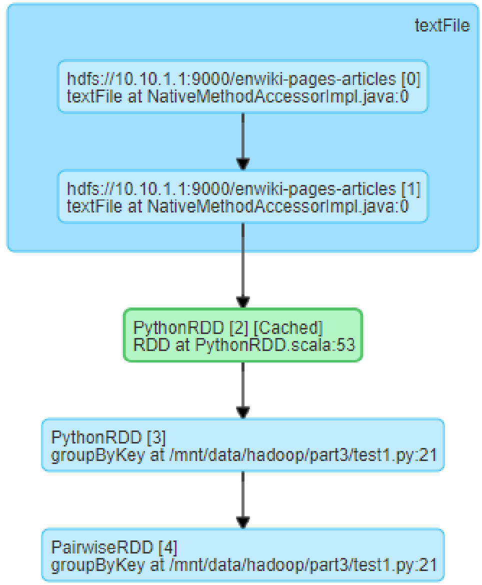


Figure 4 DAG Visualization of one stage (in detail)

Partitioning Analysis

Task 2. Add appropriate custom RDD partitioning and see what changes.
In this section, we set various partition numbers for PageRank algorithm and analyze the application execution time (Figure 5).

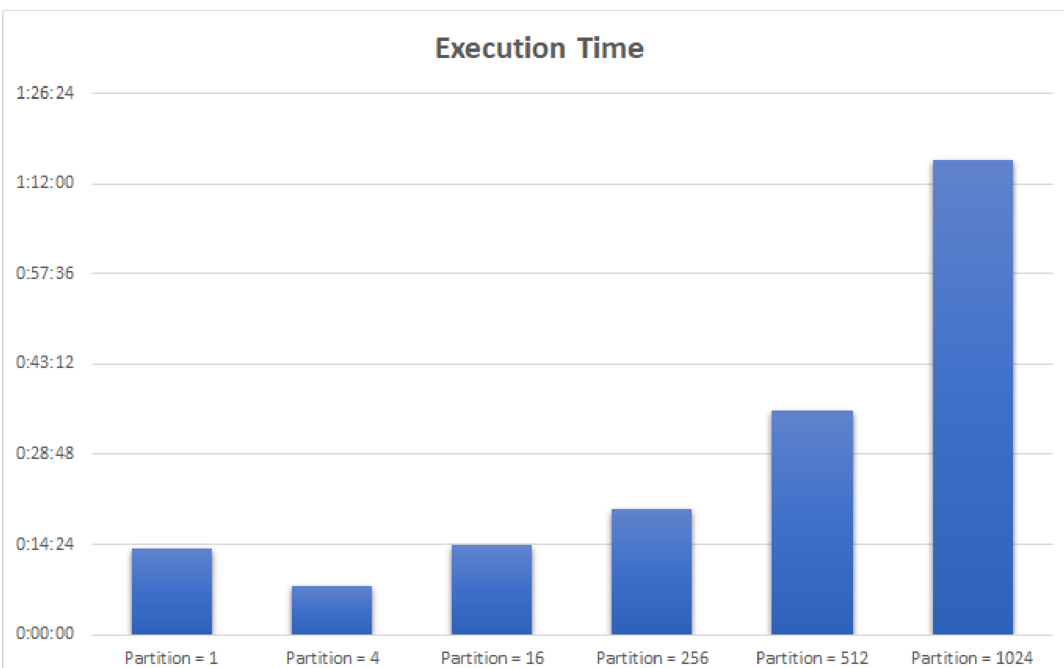


Figure 5 Execution Time for different Partition settings (Berkeley-Stanford web graph dataset)

As is shown in Figure 5, the execution time will first decrease then increase as the number of partition increases. When partition = 4, the execution time is the shortest. When the number of partitions is smaller than 4 or is larger than 4, the execution time will both increase. The possible reasons are that: too few partitions mean it would be very costly to swap data during the reduceByKey operations; too many partitions mean the cost for partitioning and merging operations would be significantly large so that the execution time would be very long.

After optimizing the implementation of Page Rank algorithm based on the RDD paper (shown in file part3_pagerank_update.py) and moving the dataset to hdfs (shown in file run_all_memory_large.sh), the number of tasks to be executed decreased to 292 per iteration. The optimized configuration is applied to a part of the experiment dataset (i.e., 1G data of the entire dataset of enwiki-pages-articles). We only used a part of the dataset because the execution time for a whole dataset is more than 1hour 20min, and we run out of time to finish all the tasks with the complete dataset. The execution time for different partition settings is shown in Figure 6.

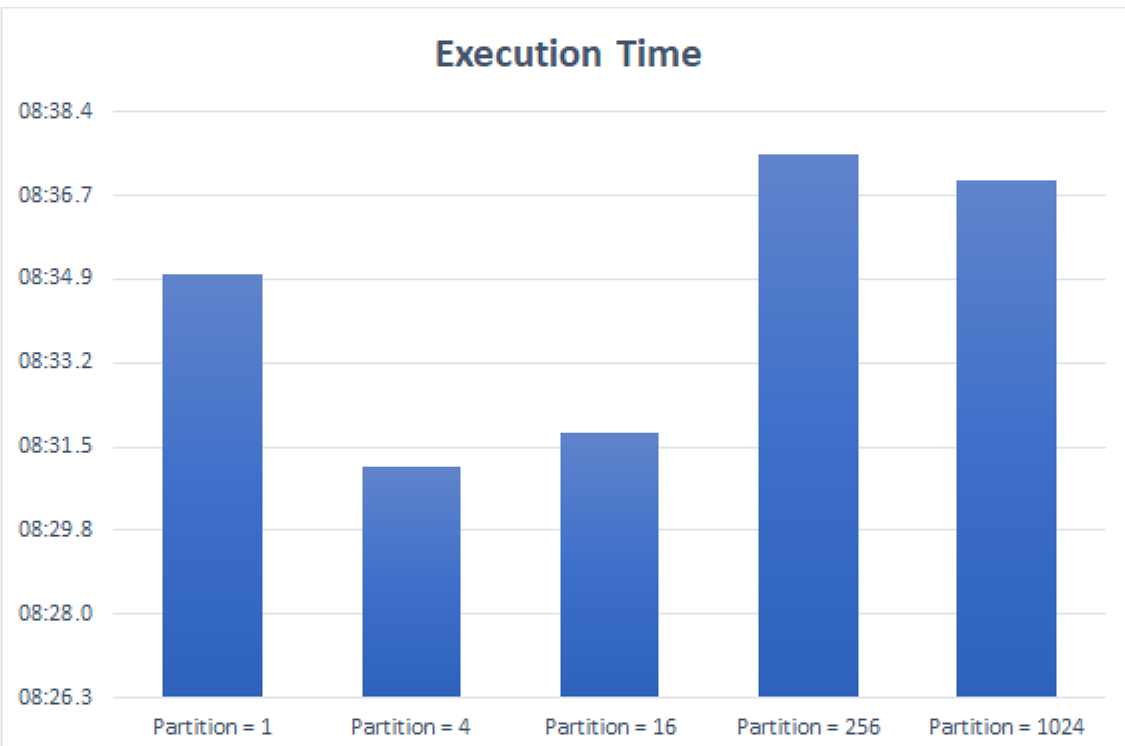
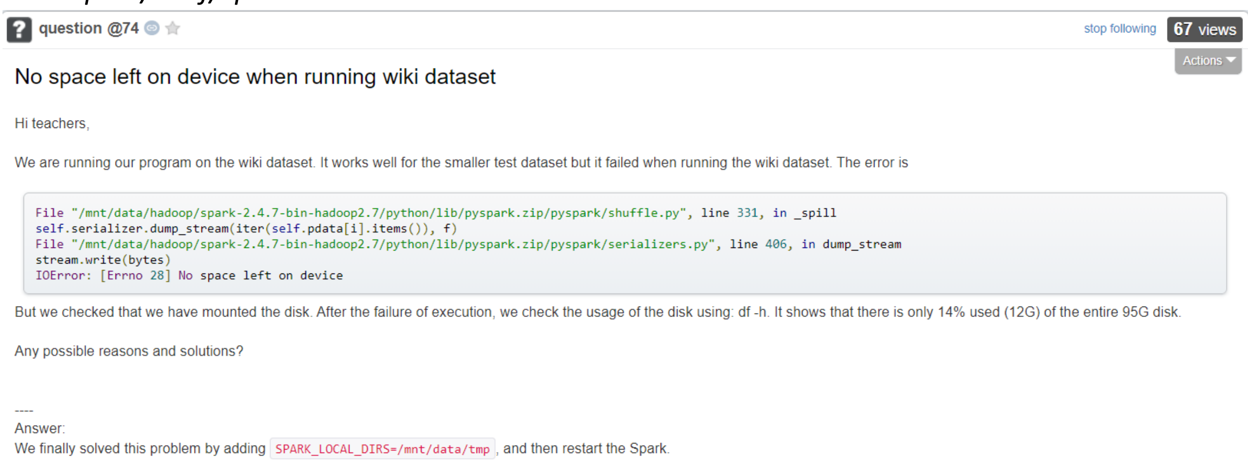


Figure 6 Execution Time for different Partition settings (wiki dataset)

Although tripling the size of the test dataset, the time to execute the experiment dataset decreased by around 40%. We can thus conclude that dropping the dataset to hdfs and optimizing the algorithm can greatly improve its performance by reducing the execution time at scale. Also, the variation of execution time for different partition settings follows the same trend as illustrated for the test dataset. We also executed the application on the whole experiment dataset, which took about 1hour 49min 43s with a partition number of 4.

During the implementation, we encountered several problems and we tried to solve them with the help of StackOverflow, piazza, and lab guide. For example, when we were executing the program on the larger dataset, there was no space left on device. So, we proposed a question on Piazza as shown in Figure 7. We added SPARK_LOCAL_DIRS in *spark-2.4.7-bin-hadoop2.7/conf/spark-env.sh*.


question @74

stop following 67 views

Actions

No space left on device when running wiki dataset

Hi teachers,

We are running our program on the wiki dataset. It works well for the smaller test dataset but it failed when running the wiki dataset. The error is

```
File "/mnt/data/hadoop/spark-2.4.7-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/shuffle.py", line 331, in _spill
self.serializer.dump_stream(iter(self.pdata[i].items()), f)
File "/mnt/data/hadoop/spark-2.4.7-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/serializers.py", line 406, in dump_stream
stream.write(bytes)
IOError: [Errno 28] No space left on device
```

But we checked that we have mounted the disk. After the failure of execution, we check the usage of the disk using: df -h. It shows that there is only 14% used (12G) of the entire 95G disk.

Any possible reasons and solutions?

Answer:
We finally solved this problem by adding `SPARK_LOCAL_DIRS=/mnt/data/tmp`, and then restart the Spark.

Figure 7 The question we posted regarding no space problem

Also, when we test our algorithm on the large dataset, we found that the execution time is very long. We first tried some solutions from Internet, but they didn't help. Then we post our question to piazza (Figure 8, https://piazza.com/class/kcnrt66wd4v4ot?cid=90) and reimplemented our algorithm following the original RDD paper closely, as suggested by professor. The execution time dropped a bit, while is still much more than we have expected. We think there might be some problems with other settings. We tried several solutions, but we haven't figured them out yet.

question @90

stop following 56 views

Actions

Expected execution time for PageRank algorithm?

We are testing our page rank algorithm on web-graph dataset (smaller dataset) and wiki article dataset (bigger dataset).

For the smaller dataset, it takes around 8 min to finish 10 iterations in total.
For the bigger dataset, it even takes around 1 hour to finish 1 iteration (it will take around 8 hours to finish 10 iterations).

Are they expected execution times? Or they should be faster? (partition number is set by default)

Also, we compared the execution time between the local dataset and the dataset stored to HDFS, and the latter is slower. Is this expected? I thought the dataset stored in HDFS should make the execution faster..

hw1

edit · undo good question | 1 Updated 23 hours ago by Jinmeng Rao (Anon. Poet to classmates)

the students' answer, *where students collectively construct a single answer*

I would think the more important part, unless we are actually being judged on overall performance, is to assess the difference in performance across the various tasks (so baseline, persist, custom partition, faults), holding your program/algorithm constant. At least that seems to be the larger goal of the assignment to me.

~ An instructor (Saurabh) endorsed this answer ~

edit · undo thanks | 2 Updated 23 hours ago by Matthew Gercz

the instructors' answer, *where instructors collectively construct a single answer*

These times are definitely higher than expected. In most cases this should ideally run under 60 minutes without much optimization. My suggestion is to follow the algorithm in the spark paper closely.

Edit: My apologies for previously saying 15 minutes. I was running on more workers with scala frontend.

undo thanks | 1 Updated 21 hours ago by Saurabh

followup discussions *for lingering questions and comments*

Resolved Unresolved

Anonymous Helix 22 hours ago So even the larger Wiki dataset should finish in 8-15 mins without any caching/persistence or other optimizations if we followed the paper algorithm closely?

helpful! | 0

Saurabh 22 hours ago
Yes somewhere close to that.

good comment | 0

Saurabh 21 hours ago
My apologies I just double checked, my previous numbers were for a different setup which used scala frontend. Around 60 minutes should be fine

undo good comment | 1

Reply to this followup discussion

Resolved Unresolved

Anonymous Poet Just now
Thank you professor. We reimplemented our algorithm following the RDD paper closely, and we are testing our algorithm on the wiki dataset. The time cost is still greater than expected. I think there might be some problems with other settings. We tried several solutions but we haven't figured them out yet... However, we did finish all the tasks and analyzed the results, and we think the results make sense and are expected. We decided to report them as well as our attempts to solve the problem.

helpful! | 0

Shivaram Venkataraman Just now Yep that sounds good. Thanks for posting -- As I said before, we will not penalize reports just because the performance number is worse than expected.

good comment | 0

Reply to this followup discussion

Figure 8 The question we posted regarding the expected execution time

Persistence analysis

Task 3. Persist the appropriate RDD as in-memory objects and see what changes. Read about RDD persistence.

When doing persistence analysis, we set partitioning number as 4 and keep all the other settings to be unchanged. Specifically, we set the storage level to be “MEMORY ONLY” to drop the RDD in memory before run the iteration, which caches all the pages and their attached lists of neighbors. The change of persistence configuration can be seen in Figure 3, which shows a green node in stage 0 and indicates the activation of cache mode. We compared the execution time between “DISK ONLY” mode and “MEMORY ONLY” mode to explore the difference between these two configurations. The execution time for these two persistence modes on the same dataset is recorded in Table 1.

Persistence Mode	**Execution Time**
DISK ONLY	7.8 min
MEMORY ONLY	7.8 min

Table 1 Execution Time for DISK ONLY mode and MEMORY ONLY mode

As Table 1 shows, the persistence mode doesn’t present a significant impact on execution time in this task. We further tested these persistence modes several times on different datasets (Berkeley-Stanford web graph and enwiki-20180601-pages-articles) and didn’t found significant difference. We also checked the execution time during each stage and found that the time for DISK ONLY mode and MEMORY ONLY mode is pretty much the same. Since MEMORY ONLY mode would help Spark reuse the data thus lower the computational cost and execution time, we expected that the execution time for MEMORY ONLY mode is significantly lower, while it didn’t happen in our observation.

Fault-tolerance analysis

Task 4. Kill a Worker process and see the changes. You should trigger the failure to a desired worker VM when the application reaches 25% and 75% of its lifetime:

1. Clear the memory cache using `sudo sh -c "sync; echo 3 > /proc/sys/vm/drop_caches"`.
2. Kill the Worker process.

In this section, we tested the fault-tolerance ability of Spark. We cleared the memory cache when the application reaches 25% and 75% first. Then, we killed one worker using kill -9 PID when the application reaches 25% and 75%.

Triggered Failure	Execution Time
Clear the memory cache at 25% of the lifetime	8min 0.82s
Clear the memory cache at 75% of the lifetime	8min 1.03s
Kill the Worker process at 25% of the lifetime	7min 52.54s
Kill the Worker process at 75% of the lifetime	7min 49.48s

Table 2 Execution Time Triggering Different Failures at either 25% or 75% of the application lifetime

After triggering each of the four failures, Spark can still resume working and eventually finish all the tasks correctly. However, we find that each of the four failures has dragged the execution time of the application. Specifically, killing a worker delays the process by around 4s (0.8%), and clearing the memory cache delays the process by around 13s (2.8%). Therefore, clearing the memory cache has more negative impact on the execution time compared with killing a worker for the test dataset. We also found that after killing a worker, it cannot be restarted automatically.

We didn’t get the time to measure the execution time by triggering different failures for the large dataset, which might give us different results concerning which failure drags the application the most. For the small dataset, the worker we killed might not take many tasks, so it didn’t consume lots of time to redo all the tasks when killing this worker. Therefore, to resume from clearing memory cache takes slightly longer time in our experiments.

4.Author Contributions

1. Environment setup: Mainly worked by Yuhao Kang. All team members participated the error solving discussion.
2. A simple Spark application: Mainly worked by Jinmeng Rao.
3. PageRank: Script writing and debugging by all team members. Mainly running by Xinyi Liu. All team members participated the analysis and discussion.