SLR207 - Technologies de calcul parallèle à grande échelle

MapReduce from Scratch

ZHAO Yuanjie

June 28, 2024

Contents

List of Tables

List of Figures

1 Brief Introduction

In this project, our goal is to write the MapReduce algorithm from scratch.

2 The way to build, execute and deploy the project

2.1 Project Structure

```
javaftp/
|--- myftpclient/  # client directory
|    |--- src/main/java/rs/MyClient.java  # java src code
|    |    |--- target/  # target directory for .jar and dependencies
|    |    |    |--- myclient-1-jar-with-dependencies.jar  # executable java file
|    |    |    |--- ...
|    |    |--- pom.xml  # MAVEN pom file
|--- myftpserver/
|    |--- src/main/
|    |    |--- java/rs/MyServer.java  # java src code
|    |    |--- resources/log4J.properties  # style for logs
|    |    |--- target/  # target directory for .jar and dependencies
|    |    |    |--- myserver-1-jar-with-dependencies.jar
|    |    |    |--- ...
|    |    |--- pom.xml  # MAVEN pom file
```

2.2 Build the project

In order to build the project, use Maven. For example, to build the client file:

```
cd myftpclient
mvn clean compile assembly:single
```

To build the server file:

```
cd myftpserver
mvn clean compile assembly:single
```

2.3 Execute the project

After using Maven to build the project, you can use this line to execute the server:

```
java -jar myserver-1-jar-with-dependencies.jar
```

After running the server file, use this line to execute the client:

```
java -jar myclient-1-jar-with-dependencies.jar
```

If you want to change the sample file which is used as the one to be split into `xxx.txt`, using this command:

```
java -jar myclient-1-jar-with-dependencies.jar xxx.txt
```

The result can be seen in the `Result` folder generated by the program.

Also note that, if you want to change the servers used in this project into the servers that you want, you need to modify `String[] server = {"tp-4b01-04", "tp-4b01-06", "tp-4b01-07"};` in both the server program and the client program.

2.4 Deploy the project

You need to use the command `scp` to upload the `.jar` file on each computer (3 computers with the server `.jar`, 1 with the client `.jar`). The target directory should be `/tmp/yuanjie`.

3 Details of the project

In this project, I try to write the MapReduce algorithm from scratch. I use 4 computers of Télécom-Paris: one as the client, and the other three as servers. The client need to split a huge file into 3 and send one split file to each server. And for the splitting, I tried to control the line number of each split file to be almost the same. I start a FTP server on each server computer, and start a FTP client on the client computer to send the split files. The client and the servers communicate via socket.

After sending the file, it's the MapReduce procedure. The first phase is the Map phase where each server computes the number of words in the given file and does a hash to each word. Note that, there are languages like Chinese, Japanese and Korean where words are not separated by spaces. For these languages (CJK), we just separate the sentence by each character. For other languages, we separate the sentences by spaces. Then we start the Shuffle phase where the words and their occurrence times are put into a file according to their hash value, and the files are redistributed to each server. After that it's the Reduce phase where each server computes the occurrence times in the new files sent to them.

Then we do a second MapReduce procedure. Each server counts the maximal and minimal number of occurrence times of words in their word list, and returns this value to the client. The client uses the Zipf's law to generate three intervals. (Zipf's Law is an empirical law that describes the frequency distribution of words in natural languages, if $f(r)$ is the frequency of the word with rank r, then $f(r) \propto \frac{1}{r}$.) We then do the Shuffle phase: The words whose appearing times is within the first interval are sent to the first server, and so on. After that it's the reduce phase where each server computes the appearing times of each word. We finally do a sort according to the occurrence times of each word.

The problem remaining for this project is that, I did not write all the settings into the configuration file, so to modify the settings, you need to directly edit the program. Moreover, I only do the test on 3 computers due to limited time.

The maximum size of the dataset that I achieved to compute is 329M. In fact, it can be bigger, it's 329M because the file that I used to test my program is of this size. Also, all the sample files on the server are all of similar sizes (a bit more than 300M) and I don't have better examples, so I only test the first sample file.

Then I run the code for 10 times to get an average executing time. The result can be seen below. Except those phase mentioned above, Count phase is for calculating the maximal and minimal appearing times of words, and Request phase is for generating the result file. Also, I merged the Shuffle phase and Reduce phase together because they can be put together in the code. In fact, when a server receives the shuffled words, at the same time it can do the counting.

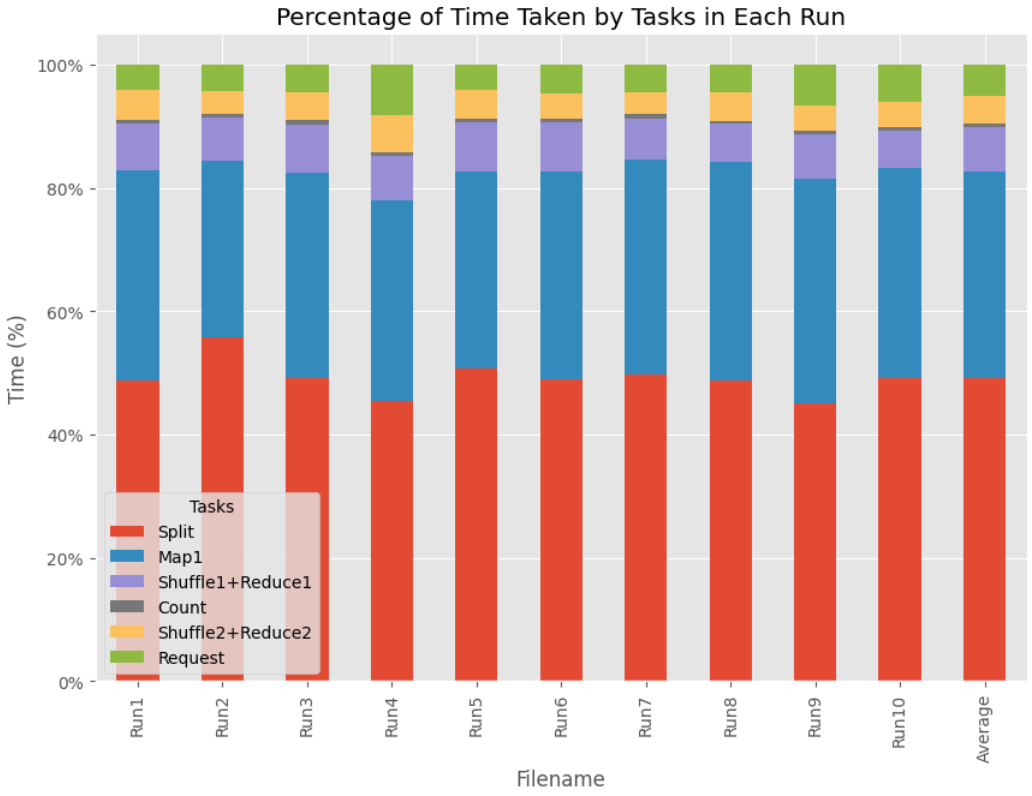

Figure 1: Percentage of time taken by tasks in each run

Runs	**Split**	**Map1**	**Shuffle1+Reduce1**	**Count**	**Shuffle2+Reduce2**	**Request**
Run1	10113	7060	1578	138	1005	839
Run2	12712	6521	1625	103	843	991
Run3	10107	6874	1610	138	928	919
Run4	9893	7081	1580	137	1327	1759
Run5	10200	6436	1610	139	940	801
Run6	10040	6950	1629	104	867	935
Run7	10002	6996	1350	139	701	900
Run8	10111	7357	1307	95	971	912
Run9	9780	7903	1589	134	852	1453
Run10	10019	6954	1203	138	830	1221
Average	10397.7	7013.2	1508.1	126.5	926.4	1073

Table 1: Time taken by tasks in each run (in ms)

In order to validate Amdahl's law, we need to change the number of servers to see what happens to the MapReduce procedure. For a given text, if we increase the number of servers, the time for split part will not change because the number of bytes sent by the client does not change. But for the MapReduce part, the size of data on which each server works will be smaller because it's a highly parallelized procedure. Theoretically, the time should be inversely proportional to the number of nodes. Let's look at the formula of Amdahl's law $S = \frac{1}{(1-P)+P/N}$. In the MapReduce problem, S is the speedup, N is the number of nodes, P should be the fraction of MapReduce, thus $1 - P$ is the fraction of Split.

I use many threads in my program. For example, for the socket part, the server should handle each communication by a thread. I also used thread for the FTP server because it should run with the socket server at the same time.

For the optimization that I've done, I used Zipf's law to get the split interval for the second MapReduce instead of evenly splitting. In this way, the number of words will likely to be more close for the three intervals.

For further optimization, because I use hard disk for storage of intermediary results, it can be optimized if we use sockets to transmit the bytes.