Class Number: CS6240 Paralleled Data Processing, Spr 2020

Name: Yihan Tian

Homework 2

All source codes with comments are zipped in the same package.

**Answer to the question of the key, value in Map function:**

As I read through the source code of WordCount example, in the TokenizerMapper class, it uses the Text value as one of the inputs. The function uses an iterator, and iterate through the string format of the value, and get each single word from this part. So I believe the value is the body of the input text file, and this part is easier to find.

For the key part, it is not used in the example code. Then I read through the source code and documentation of hadoop3.2.1, and find out that the Mapper class is designed as shown in the picture:

A close up of a logo

Description automatically generated

And the map function is as below:

A picture containing device

Description automatically generated

This function is an override from parent class, and requires a key, some value, and output. I checked through some online analysis and tutoring sources and found they say the Object key is a Mapper auto-generated key. So I understand the key as an auto-generated, unique identifier to each map task.

**AWS Execution and results**

Runtime report:

Used the time when Map 0%, Reduce 0% as the start time, and the task complete time as end time. This count corresponds with the elapsed time shown on EMR panel. Stored first run and second run in different charts, and also calculated average run of the two times. Illustrated a line chart according to the average run time.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| first-run | | | | | | | | |
|  | NoCombiner-6 | SiCombiner-6 | PerMapTally-6 | PerTaskTally-6 | NoCombiner-11 | SiCombiner-11 | PerMapTally-11 | PerTaskTally-11 |
| Run Time | 49 | 38 | 30 | 38 | 34 | 26 | 34 | 22 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| second-run | | | | | | | | |
|  | NoCombiner-6 | SiCombiner-6 | PerMapTally-6 | PerTaskTally-6 | NoCombiner-11 | SiCombiner-11 | PerMapTally-11 | PerTaskTally-11 |
| Run Time | 46 | 37 | 30 | 41 | 26 | 27 | 33 | 21 |
|  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| average | | | | | | | | |
|  | NoCombiner-6 | SiCombiner-6 | PerMapTally-6 | PerTaskTally-6 | NoCombiner-11 | SiCombiner-11 | PerMapTally-11 | PerTaskTally-11 |
| Run Time | 47.5 | 37.5 | 30 | 39.5 | 30 | 26.5 | 33.5 | 21.5 |

Answer to the questions:

1. Do you believe the combiner was called at all in program SiCombiner?

Yes, the combiners are called. In the syslog of SiCombiner, it shows as below:

A picture containing object, bottle

Description automatically generated

While at other syslog, such syslog of NoCombiner, the report shows:

A screenshot of a cell phone

Description automatically generated

So the combiners are called in the SiCombiner, but at other classes since the combiners are deactivated they are not called.

1. What difference did the use of a combiner make in SiCombiner compared to NoCombiner?

First obviously, using a combiner decreased run time of the whole program. Then by reading through the syslog, I found that the number of reduce tasks launched is decreased by 1 time in SiCombiner, and the time spent reduce dropped a lot. The reduce shuffle bytes also decreased significantly.

1. Was the local aggregation effective in PerMapTally compared to NoCombiner?

By using local aggregation didn’t improve the overall performance in PerMapTally significantly. The runtime was similar of the two, and according to the syslog the difference between the two are mainly on reduce input bytes, and the difference is slight. At the meantime the reduce shuffle bytes increased a lot. The other records of the two are pretty similar, so I think the local aggregation didn’t effectively improved NoCombiner.

1. What differences do you see between PerMapTally and PerTaskTally? Try to explain the reasons.

Many of the numbers in PerTaskTally dropped significantly compare to PerMapTally, for example the log of map outputs (records, bytes…), reduce shuffle bytes, reduce input records and also run time. I learned from Hadoop that in MapReduce, when setup() or cleanup() is override, these two functions both will be execute only once during the whole task. In my understanding this is like setting up a global hashmap for aggregation, and emit all k, v pairs after all aggregation is finished. So it will significantly reduce the records emit to reducers and decrease the cost of shuffle.

1. Which one is better: SiCombiner or PerTaskTally? Briefly justify your answer.

The record in syslog of these two didn’t differ a lot, numbers look similar, even for the memory PerTaskTally didn’t use too much than SiCombiner. The only great difference is the runtime, PerTaskTally runs faster than SiCombiner. In this example PerTaskTally is better.

1. Comparing the results for Configurations 1 and 2, do you believe this MapReduce program scales well to larger clusters? Briefly justify your answer.

Overall this program scales well, and by scaling performance improved a lot.