GROUP WORK IN THE COURSE BIG DATA / BIG DATA ANALYSIS

Team members:

Zervas Michalis (ics20015)

Drikos Christos (ics20046)

Contents

A MapReduce round was used for the implementation.

1. Algorithm we used to solve the problem

1. The mapper receives as input an order
2. It creates all possible unique product combinations from the specific order. The order of products does not play a role. Example F1 F3 combination is considered the same as F3 F1.
3. Mapper passes as a key every possible combination of products that has been produced and as a value for each of them the value 1. Example < [F1 F3] , 1 >
4. Combiners take as input the output of Mappers and produce as output: for key, each combination of products (obviously one at a time) and for value the sum of the values corresponding to each combination. E.g. <. [F1 F3] , 300 >
5. The Reducer, taking as input the results of the Combiner, totals for each product combination all the values it has received from the combiners and if the sum is greater than or equal to the threshold, the result is written as output with key the specific combination and value the number of times it appeared. If it is less than the threshold it ignores it. E.g. < [F1 F3] , 6000 >

2. Limitations and considerations of the solution

Restrictions: The combiner code had to be a bit different from the reducer code due to threshold control. We chose to compress the results produced by the mapper because this reduces the amount of data it produces; therefore, it is transferred faster to the network and reduces the I/O on disk. We chose Snappy compression to get the fastest compression possible, so that we don't get damaged in the end by the compression process.

Consideration of our original solution: Our original idea was that if an individual product appears fewer times than the threshold to

remove this and all other combinations in which it appears, thus saving unnecessary checks. But this was rejected because it made it difficult to implement. We considered that it would read the input files at least twice and write the intermediate results another two times as they would have to be transferred to the network. We considered this to be time consuming since we could implement the problem with one MapReduce cycle.

3. Additional packages that used

These packages were used in Mapper to create all possible combinations of products in an order.

- com.google.common.collect.sets
- com.google.common.collect.ImmutableSet

4. Parameters of the system

The additional parameters are the threshold which we give as args. Also the name of main does not need to be given.

5. Times tables and Graphs

2 nodes

Threshold 5000:

1. 00hrs, 07mins, 45sec job_1670238997489_0011

Average Map Time	1mins, 56sec
Average Shuffle Time	2mins, 30sec
Average Merge Time	47sec
Average Reduce Time	2mins, 29sec

2. 00hrs, 08mins, 12sec job_1670238997489_0019

Average Map Time	1mins, 35sec

Average shuffle time 2mins, 56sec
Average merge time 47sec
Average Reduce Time 2mins, 32sec

3. 00hrs, 09mins, 06sec job_1670238997489_0020

Average Map Time1mins , 39sec
Average Shuffle Time3mins , 41sec
Average Merge Time48sec
Average Reduce Time2mins , 31sec

4. 00hrs, 08mins, 08sec, job_1670238997489_0021

Average Map Time1mins , 47sec
Average Shuffle Time2mins , 49sec
Average Merge Time47sec
Average Reduce Time2mins , 34sec

5. 00hrs, 12mins, 47sec job_1670238997489_0022

Average Map Time1mins , 51sec
Average Shuffle Time7mins , 23sec
Average Merge Time47sec
Average Reduce Time2mins , 31sec

Threshold 10000:

1. 00hrs, 12mins, 45sec job_1670238997489_0024

Average Map Time1mins , 51sec **Average Shuffle Time** 7mins, 18sec **Average Merge** Time48sec

Average Reduce Time2mins	,
32sec	

2. 00hrs, 10mins, 34sec job_1670238997489_0025

Average Map Time	1mins, 44sec
Average Shuffle Time	5mins, 7sec
Average Merge	Time48sec

Average Reduce Time 2mins, 34sec

3. 00hrs, 08mins, 02sec job_1670238997489_0027

Average Map Time 1mins, 37sec
Average Shuffle Time2mins , 14sec
Average Merge Time47sec
Average Reduce Time 2mins, 33sec

4. 00hrs, 08mins, 12sec job_1670238997489_0028

Average Map Time1mins
, 35sec
Average Shuffle Time2mins
, 53sec
Average Merge Time48sec
Average Reduce Time2mins
, 33sec

5. 00hrs, 08mins, 02sec job_1670366157694_0006

Average Map Time1mins
, 47sec
Average Shuffle Time2mins
, 43sec
Average Merge Time52sec
Average Reduce Time2mins
, 31sec

Threshold 50000:

1. 00hrs, 12mins, 27sec job_1670238997489_0029

Average Map Time1mins , 51sec
Average Shuffle Time6mins , 57sec
Average Merge Time48sec
Average Reduce Time2mins , 34sec

2. 00hrs, 08mins, 06sec job_1670238997489_0030

Average Map Time1mins
, 46sec

Average Shuffle Time2mins

, 46sec

Average Merge Time49sec

Average Reduce Time2mins

, 33sec

3. 00hrs, 08mins, 02sec job_1670238997489_0032

Average Map Time1mins
, 38sec
Average Shuffle Time2mins
, 13sec
Average Merge Time48sec
Average Reduce Time2mins
, 33sec

4. 00hrs, 08mins, 16sec job_1670238997489_0033

Average Map Time1mins
, 35sec
Average Shuffle Time2mins
, 58sec
Average Merge Time48sec
Average Reduce Time2mins
, 32sec

5. 00hrs, 08mins, 09sec job_1670366157694_0007

Average Map Time 1mins, 38sec
Average shuffle time 2mins, 16sec
Average Merge Time 47sec
Average Reduce Time 2mins, 34sec

1 node

Threshold 5000:

1. 00hrs, 08mins, 36sec job_1670362094916_0001

Average Map Time1mins
, 26sec
Average Shuffle Time3mins
, 4sec
Average Merge Time48sec
Average Reduce Time2mins
, 34sec

2. 00hrs, 08mins, 35sec job_1670362094916_0002

Average Map Time1mins
, 26sec
Average Shuffle Time3mins

, 6sec

Average Merge Time47sec

Average Reduce Time2mins

, 31sec

3. 00hrs, 08mins, 37sec job_1670362094916_0003

Average Map Time1mins

, 26sec

Average Shuffle Time3mins

, 8sec

Average Merge Time48sec

Average Reduce Time2mins

, 31sec

4. 00hrs, 08mins, 39sec job_1670362094916_0004

Average Map Time1mins , 26sec
Average Shuffle Time3mins , 7sec
Average Merge Time48sec
Average Reduce Time2mins , 35sec

5. 00hrs, 08mins, 36sec job_1670362094916_0005

Average Map Time1mins , 26sec
Average Shuffle Time3mins , 2sec
Average Merge Time48sec
Average Reduce Time2mins , 34sec

Threshold 10000:

1. 00hrs, 08mins, 43sec job_1670366157694_0001

Average Map Time1mins , 27sec
Average Shuffle Time3mins , 11sec
Average Merge Time48sec
Average Reduce Time2mins , 32sec

2. 00hrs, 08mins, 39sec job_1670366157694_0002

Average Map Time1mins , 27sec
Average Shuffle Time3mins , 4sec
Average Merge Time 47sec
Average Reduce Time2mins , 34sec

3. 00hrs, 08mins, 20sec job_1670366157694_0003

Average Map Time1mins
, 27sec
Average Shuffle Time1mins
, 52sec
Average Merge Time47sec
Average Reduce Time2mins
, 30sec

4. 00hrs, 08mins, 32sec job_1670366157694_0004

Average Map Time1mins, 26sec
Average Shuffle Time3mins, 1sec
Average Merge Time46sec
Average Reduce Time2mins, 32sec

5. 00hrs, 08mins, 34sec job_1670366157694_0005

Average Map Time1mins, 27sec
Average Shuffle Time3mins, 7sec
Average Merge Time46sec
Average Reduce Time2mins, 31sec

Threshold 50000:

1. 00hrs, 08mins, 25sec job_1670366157694_0008

Average Map Time1mins, 27sec
Average Shuffle Time1mins, 57sec
Average Merge Time47sec
Average Reduce Time2mins, 34sec

2. 00hrs, 08mins, 39sec job_1670366157694_0009

Average Map Time1mins, 27sec
Average Shuffle Time3mins, 10sec
Average Merge Time48sec
Average Reduce Time2mins, 32sec

3. 00hrs, 08mins, 38sec job_1670366157694_0010

Average Map Time1mins, 27sec
Average Shuffle Time3mins

, 6sec

Average Merge Time47sec

Average Reduce Time2mins

, 32sec

4. 00hrs, 08mins, 40sec job_1670366157694_0011

Average Map Time1mins

, 26sec

Average Shuffle Time3mins

, 8sec

Average Merge Time48sec

Average Reduce Time2mins

, 33sec

5. 00hrs, 08mins, 28sec job_1670366157694_0012

Average Map Time 1mins, 27sec

Average shuffle time	1mins, 58sec
Average merge time	47sec
Average Reduce Time	2mins, 35sec

Average *Elapsed Time*

- 2 Nodes:
  threshold 5000: 00:08:29
  threshold 10000: 00:08:56
  threshold 50000: 00:08:10

- 1 Node:
  threshold 5000: 00:08:36
  threshold 10000: 00:08:35
  threshold 50000: 00:08:35

Average Map Time

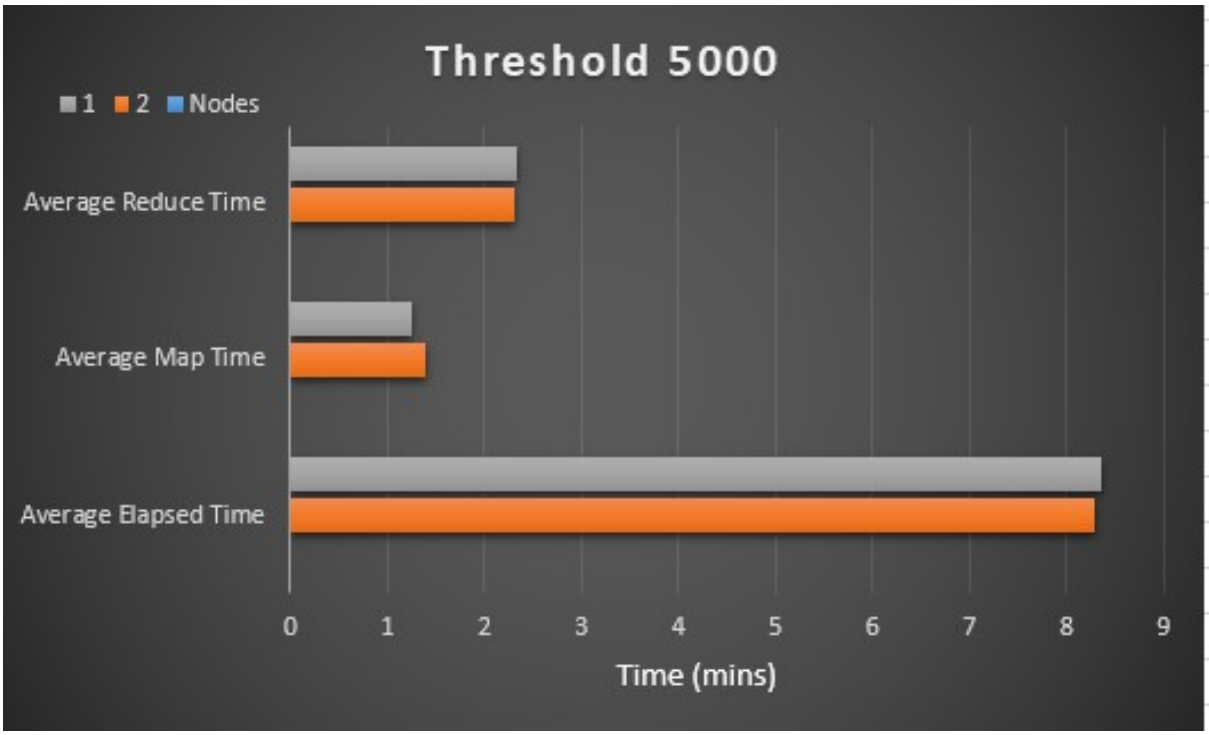
- 2 Nodes:
  threshold 5000: 00:01:40
  threshold 10000: 00:01:39
  threshold 50000: 00:01:40

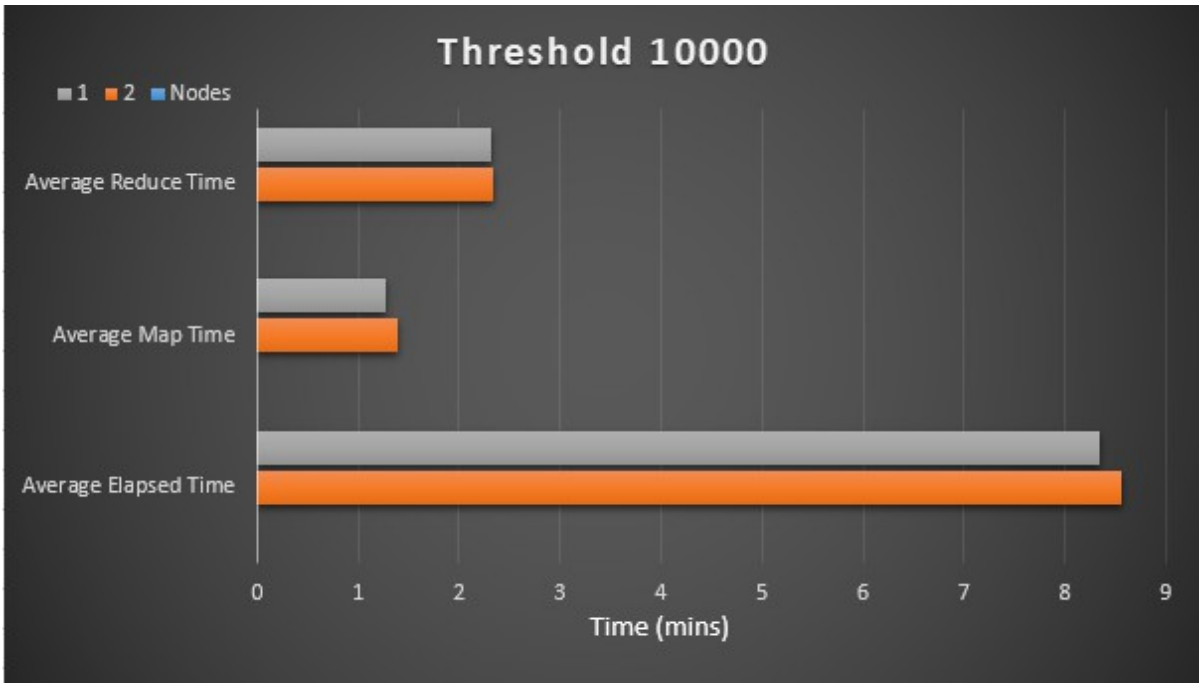
- 1 Node:
  threshold 5000: 00:01:26
  threshold 10000: 00:01:27
  threshold 50000: 00:01:27

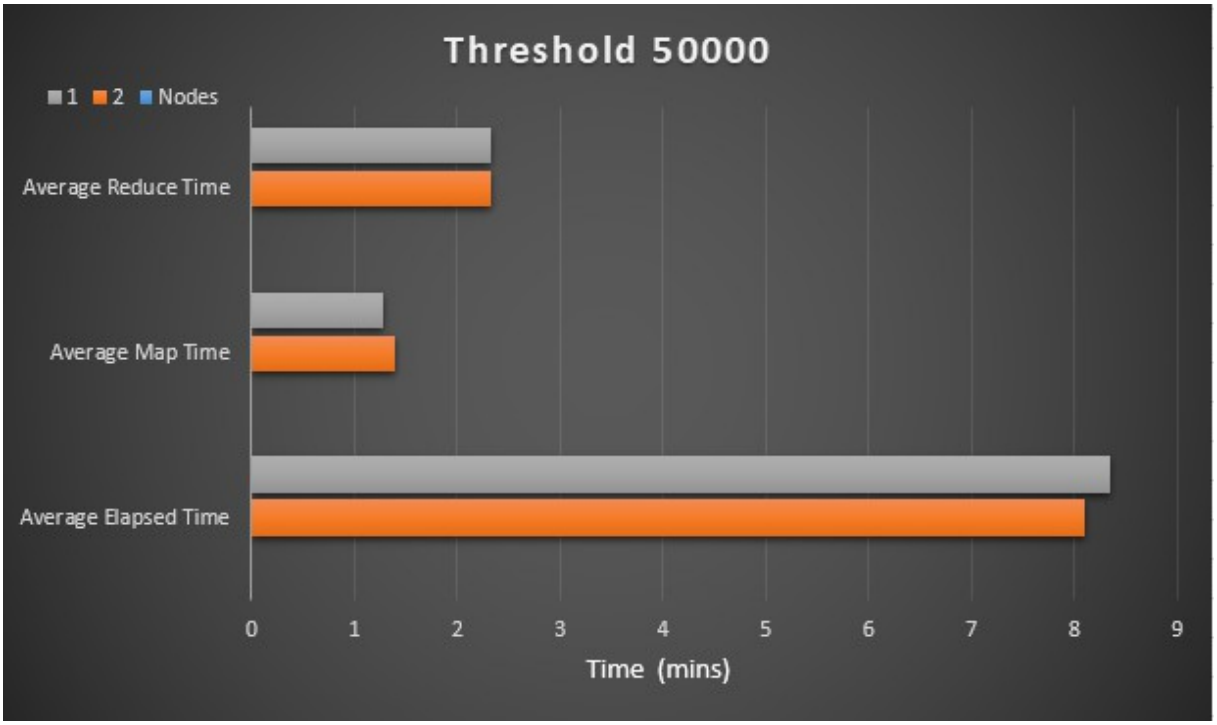
Average Reduce Time

- 2 Nodes:
  threshold 5000: 00:02:32
  threshold 10000: 00:02:33
  threshold 50000: 00:02:33

- 1 Node:
  threshold 5000: 00:02:33
  threshold 10000: 00:02:32
  threshold 50000: 00:02:33

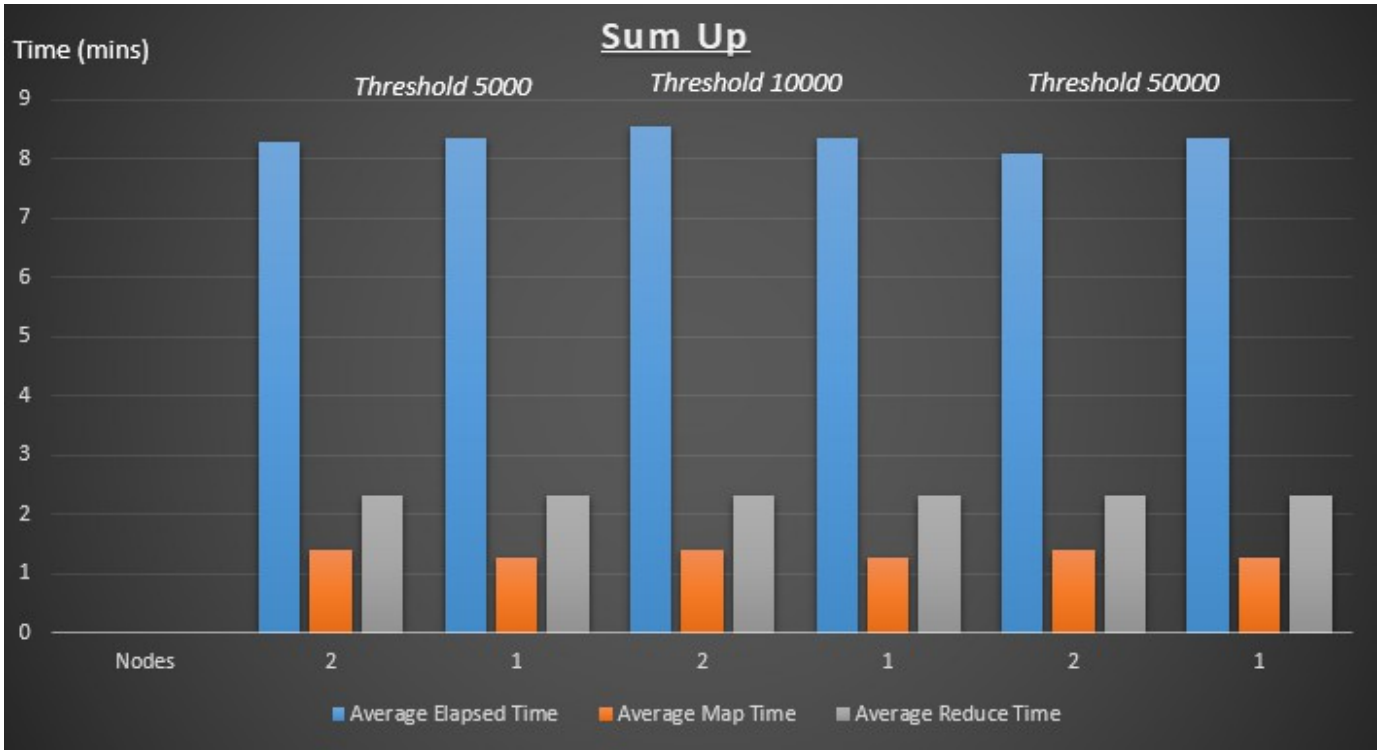

Threshold 5000
1
2
Nodes
Average Reduce Time
Average Map Time
Average Elapsed Time
0
1
2
3
4
5
6
7
8
9
Time (mins)


Threshold 10000
1
2
Nodes
Average Reduce Time
Average Map Time
Average Elapsed Time
0
1
2
3
4
5
6
7
8
9
Time (mins)



Overall

Nodes	Average Elapsed Time	Average Map Time	Average Reduce Time	Threshold
2	00:08:29	00:01:40	00:02:32	5000
1	00:08:36	00:01:26	00:02:33	5000
2	00:08:56	00:01:39	00:02:33	10000
1	00:08:35	00:01:27	00:02:32	10000
2	00:08:10	00:01:40	00:02:33	50000
1	00:08:35	00:01:27	00:02:33	50000



6. Notes for execution times

In general, we observe that there are no large variations in the times. Perhaps we would expect 2 knots to be considerably faster, but it didn't seem so. Furthermore, we notice that the Average Map Time came out a bit lower on average with 1 node; this might be due to the fact that with 2 nodes some extra time is required to transfer the data over the network. Also the Average Reduce Time is constant for any parameters. Finally, the measurements at 1 Node have smaller deviations between them, which is due to the fact that the network is not used.