Class Average Hadoop Program Report

By Santipab Tipparach

Introduction

This program is designed to aggregate a data set by applying weights to 5 variables in each row and find the min, max, and average of each class/category via the Hadoop Zoidberg cluster.

Design & Implementation

The program is implemented in Java in the Eclipse IDE. The design is fairly simple. It takes advantage of the serialization interface in the Hadoop API called the 'Writable' interface. This interface is implemented in the 'Class Writable' class. It contains the various writable objects like IntWritable for the count of rows in each class, some DoubleWritables for the min, average, and max statistics, and the total value so far. The idea here is to allow the same class objects to merge, evaluating the statistics as we go. The key is the class name itself, this could be simplified to just an IntWritable, but I used a text since it looks nicer on the output.

The code flow is simple, all values are mapped to their class ID keys with the value being a ClassWritable object in the mapping step. The reduce step merges all the objects of the same class contained on the same node. The nodes are split in the main class, and it's also where the benchmark portion is being done.

Experimentation

The process of experimentation was straightforward. Using the file provided, I applied the weights by hardcoding them into the program.

The Hadoop commands were saved into a file, and executed via the SSH terminal "Putty". The execution command was run 10 times for each number of nodes specified.

```
hadoop jar ca.jar ClassAverage /user/tipparac/classaverage/input
/user/tipparac/classaverage/output /user/tipparac/classaverage/logs 2
```

The results were retrieved via this command

```
hadoop fs -copyToLocal /user/tipparac/classaverage/output /home/tipparac/output
```

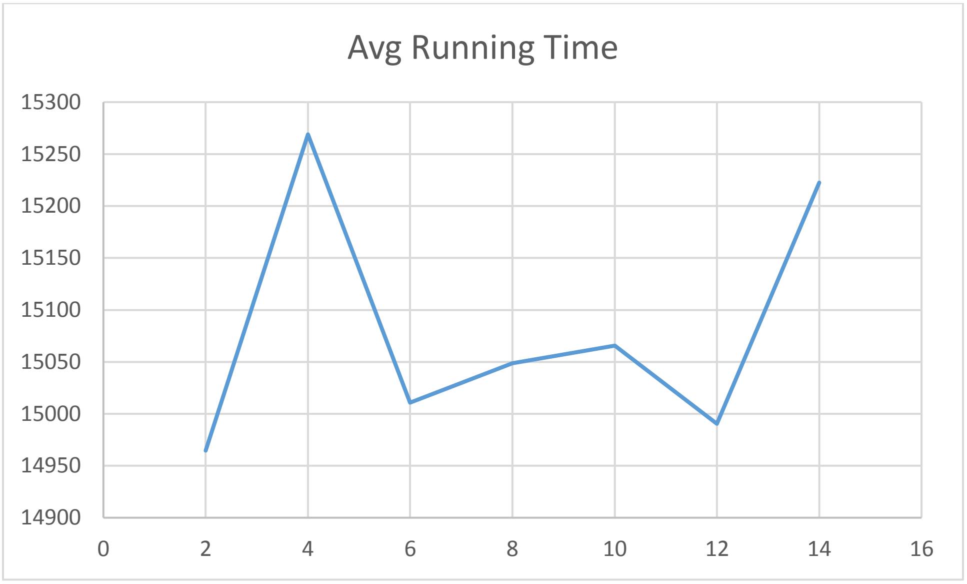
or could be simply read using this:

```
hadoop fs -cat /user/tipparac/classaverage/output/part-r-00000
```

Result

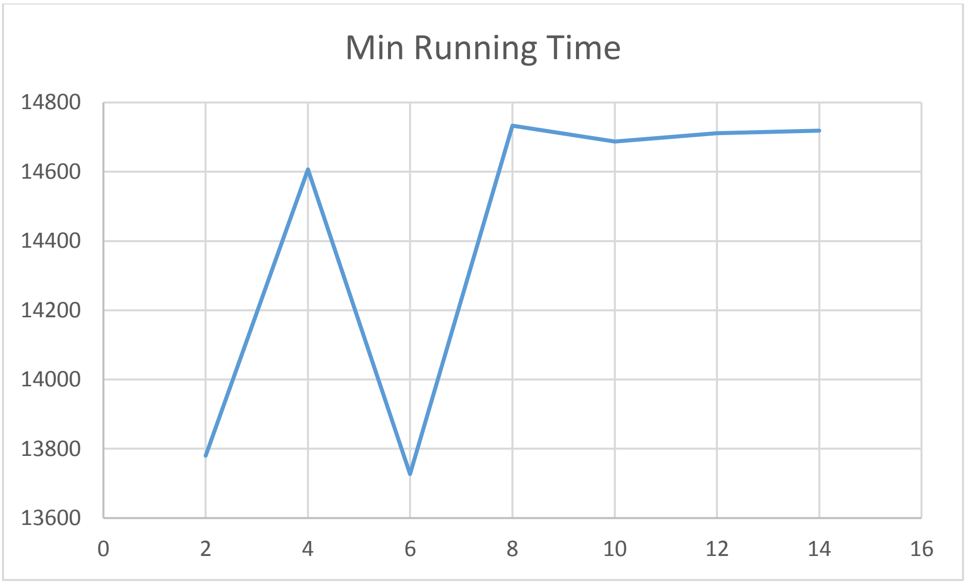
The results of this experimentation is in the graphs provided below, along with an excel sheet that will be attached, and a set of sample output files.

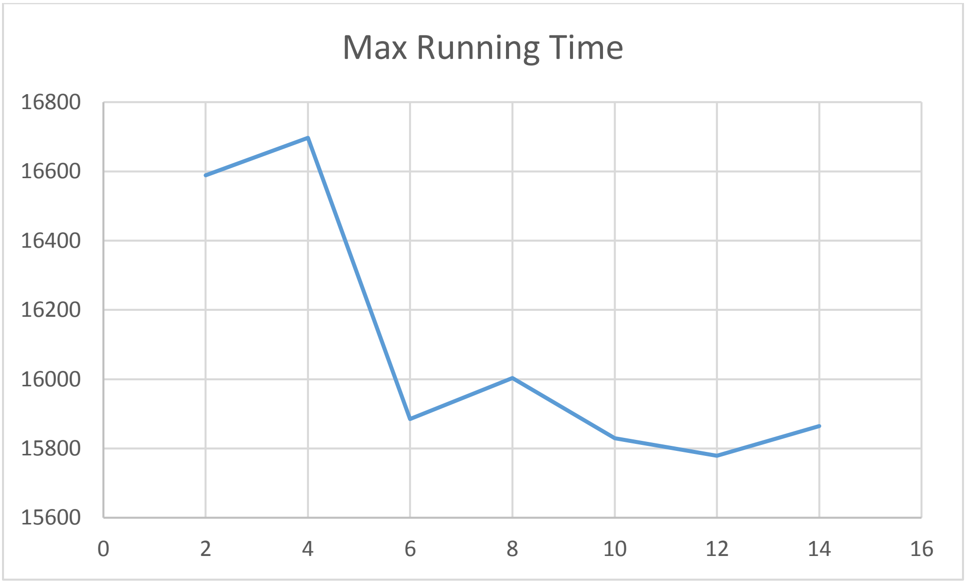
Here we can begin analyzing and making assertions as to what's going on.



In this graph, the average time it takes to run is bench marked. Overall, we don't see much of a change as we scale upward since this value only fluctuates by about 250 milliseconds. We seem to get quite a random sample when we run every time, this is most likely due to the time it takes to facilitate communication between all the nodes. The conclusion I can draw here, is that the first run might've been an anomaly (2 nodes), and ran faster than subsequent test cases. It appears that in the middle around 6-12 nodes also seem to be optimal, whereas after 12 nodes, the performance begins to climb.

It appears though, the scaling doesn't seem to have an issue since all the work of categorization is done in the mapping phase, and the reduction phases simply perform the calculation distributed among the nodes.





The maximum and minimum running time, however, do seem to have some sort of pattern to them. It appears that as the number of nodes increase, the values become more and more stable across each run, the max time gradually decreases, and the minimum time gradually decreases. This could be caused due to the redundant nature of having multiple nodes, allowing less random time to occur since the nodes do less work each.

Conclusion

The dataset was rather small for a problem designed to benchmark code or test out the performance and usage of the Hadoop cluster. I don't think I can yet say anything concrete about the data, only that it looks to be more stable as the number of nodes increased. If I had more data to work with, perhaps I would've seen more of a performance change between the test cases.

I did however, learn a lot about how to program a Map Reduce application using Hadoop in Java. I have some background in parallel programming from my previous jobs, so that came in handy. One feature I really liked was the ability to serialize data by implementing the “Writable” interface, which made my job way easier, and made it clearer to me as to how the reduction step worked.

This project was a fun exercise in parallel programming, perhaps I should look forward to creating more parallel programs for the Hadoop cluster for fun in the future.