Big Data Coursework Part 2

INM432

Pablo Quinoa

June 2020

Word count: 1656

Task 2: Parallelising the speed test with Spark in the cloud

2c) Improve cluster efficiency

i) Caching an RDD is very beneficial when we read multiple times from an RDD, so that we save processing the RDD each time. In this case at some point we have an RDD with the combination of parameters and its throughput (images per second read for that combination) and we want to create four RDDs, one for each parameter and its throughput. This means reading four times the same RDD, thus caching the first RDD saves processing the RDD more than one time.

ii) Before we parallelized our spark context, we could see how all the computation happens in up to two nodes simultaneously when we are running the spark job in a cluster with a master and 7 workers clearly not taking advantage of the cluster configuration. Then we used the second parameter (number of slices) in the parallelize call and set it to number of slices to 7 (to match the number of workers in the cluster). We could see how the CPU was now evenly spread across the 7 workers and the overall processing time was reduced.

Figure 1: Reading plain images without cache and without parallelizing in red (1), with cache and parallelizing in blue (2)

In figure 1 above and figure 2 below we can see how using cache and parallelization across all workers reduces the processing time (blue bit) and we observe how each worker, which is represented with a different colour, works at the same time.

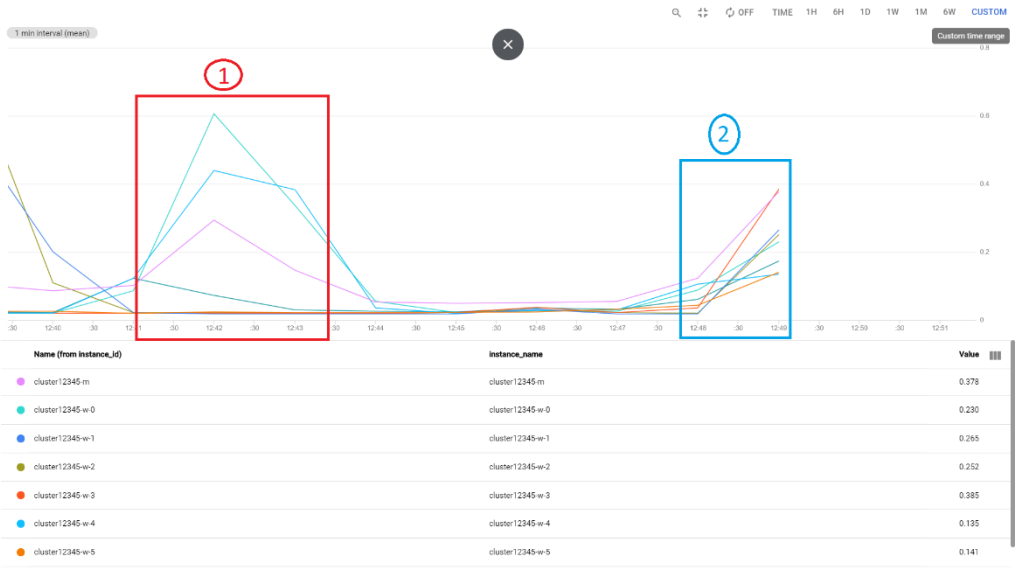

Figure 2: Reading images from TFRecord files without cache and without parallelizing in red (1), with cache and parallelizing in blue (2)

When we compare figure 2 (reading images from TFRecord files) with figure 1 (reading plain images) we can observe how when using TFRecord files reading images is at least twice as fast as reading the plain images.

2d) Retrieve, analyse and discuss the output

The relationship between each of the four parameters and the throughput (images per second) seems to be consistent across plain images and TFRecord files. In both cases we appreciate a positive big enough slope linear relationship for batch number, batch size and dataset size with respect to throughput. So, the bigger the batch size for example, the more images per second are retrieved. But for repetitions it looks like the linear relationship between number of repetitions and throughput is not there, so it takes approximately the same time to read the images a first time compared to reading various times.

In large-scale machine learning application with huge datasets and big processing required, the workload must be distributed across many servers (in some cases in distant data centres). Such applications would be reading millions of images compared to the 3000 we used for this experiment, this means that changes in batch size or batch number would have a massive effect in the throughput (the linear relationship multiplies almost proportionally with bigger datasets). Also, if we take a look at the latency numbers presented here https://gist.github.com/hellerbarde/2843375 we can observe how reading from memory is 100k times faster than reading from disk (100 ns vs 10M ns) thus configurations like the batch size will define how much is stored in memory and how much in disk, which will affect the processing time dramatically. It is also important to note that often the servers will be in distant physical locations performing the same job, and if we refer to the latency numbers again we can see how a round trip of data within the same data centre is 300 times faster (0.5ms vs 150ms) than when one server is in California and the other in Europe.

When we parallelized the speed test across a cluster of machines we are in fact not considering the bucket reading speed limit (as our experiments were always below the limit). At the start Google cloud offers 5000 object read requests per second, and as the bucket grows the cloud storage automatically scales/increases by distributing the request across more servers. But such autoscaling means distributing the load in new machines which takes time, so there is in fact a delay. Therefore, in bigger applications requiring bigger read loads the bucket at some point will have to auto scale meaning more delays and higher overall processing times which we did not consider in this experiment.

Task 3: Write TFRecord files to the cloud with Spark

3c) Experiment and discussion

i) In this experiment we studied the effects of distributing the spark job across four machines and compare it to just using a single machine (with the same total amount of resources).

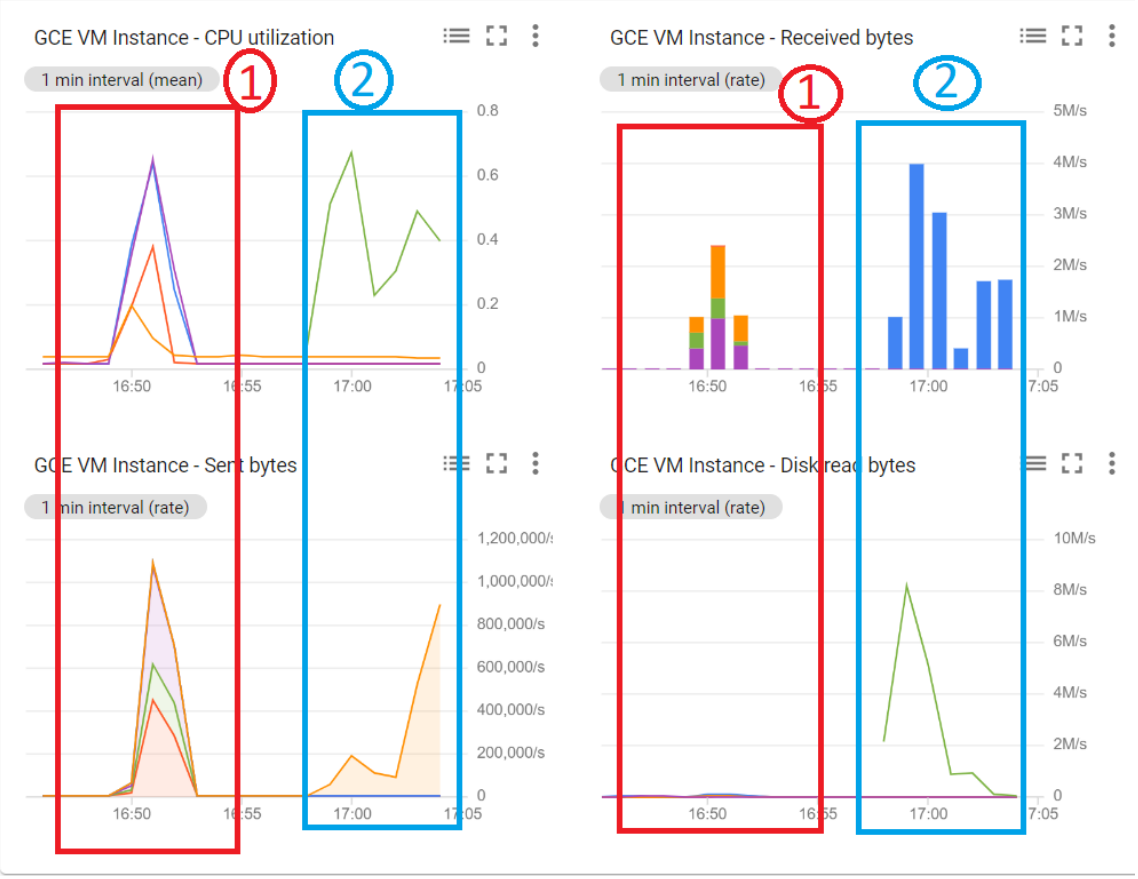

Figure 3: 4 machines cluster in red (1) vs a single machine in blue (2) performing the same spark job with the same amount of total resources

In figure 3 above is interesting to see how in the chart for disk read bytes (bottom right) when using a cluster of various machines (red) the disk read is almost 0 and this one is much higher when using a single machine (blue). This is because when the job is distributed across various

workers the data is probably basically only stored in main memory, but when using a single machine we rely on disk to store part of the data. In terms of network bandwidth we can observe in the received bytes chart (top right) how the network is more saturated or dense in the single machine scenario (blue), but if we look at the sent bytes chart (bottom left) we can see how the master node in orange has a bigger send traffic in the cluster of 4 machines than in a single machine. This is because in a distributed cluster the master node has an overhead send traffic distributing the load across each of the workers.

ii) In the labs we used Spark locally within our google collab instance, while in this exercise we submitted a spark job into a cluster of virtual machines in the cloud. This last exercise is in reality the purpose of using Spark and RDDs, as we are distributing the processing load over a cluster of machines (parallelizing).

Task 4: Machine learning in the cloud

4c) Distributed learning

No distributed training strategy	37.5 sec
Multiworker distributed mirrored strategy and batch size of 64	34.8 sec
Multiworker distributed mirrored strategy and batch size of 128	26 sec

When we did not use any distributed strategy we observed how the changes in the number of epoch has no effect on the accuracy, however when using a distributed mirrored strategy the accuracy changes with different epochs.

In terms of processing time, we can observe how using a batch size double the size decreases the processing time substantially (almost 30%).

Task 5: Discussion in context

5a) Contextualise

Alipourfard et al (2017) presents CherryPick, a system that uses Bayesian Optimization to find the most optimal cloud configuration for big data analytics. Like we observed in the tasks for this coursework changing the CPU, memory, disk, the number of instances in a cluster, and other configuration changes have a big impact in the time required to process like we observed in tasks 2 and 3, or the cost (e.g. a type of VM instance might cost three times more than another). According to Alipourfard et al, a bad cloud configuration can result in up to 12 times more cost for the same performance target. Therefore, it becomes apparent that having a way like CherryPick to find the most optimal big data cloud configuration across hundreds of possibilities is really going to have a big impact on operation costs of the companies performing such analytics. Nevertheless, using CherryPick (which tests around 60 different configurations in different cloud platforms) has an overall cost which only makes sense for those use cases where a big data analytics task is going to be recurrent, repeated a few times to justify the optimization search cost.

Smith et al (2018) presents a way to reduce the number of parameter updates required when training a model without having to decrease the learning rate, and this is achieved by increasing the batch size instead (with the same final performance results). In one of the experiments performed on this paper when training ImageNet, by increasing the batch size (images in one batch) they reduced the processing time in 15 minutes and they went from updating 14k parameters to just 6k. This theory corresponds with the results obtained in task 4 (machine learning in the cloud) where the processing time decayed from 35 seconds to 26 seconds when we increased the batch size from 64 to 128 with a multiworker mirrored strategy (with the same accuracy results).

5b) Strategise

In batch processing (or offline processing) an optimization search like CherryPick to find the most optimal cloud configuration to do the processing might not be beneficial. In offline processing reducing the processing time is not critical to justify the costs of running such overhead optimization. CherryPick is probably better suited for online real-time processing or streaming because the processing time might be critical for a real time application (e.g. a real time application is waiting for the output of the big data processing). Thus, makes sense to assume the overhead costs of running an optimization round with CherryPick to find the big data configuration that processes faster the data.

In terms of increasing the batch size as a way to reduce the number of parameter updates when training a model, this is a strategy better suited for batch/offline processing as we would be processing big enough datasets where we can increase the batch size during training. On the other hand, the extensively used gradient descent studied by Smith et al (2018) cannot satisfy the real-time learning needs of this type of data processing, thus we can´t apply such theory.