Big Data Coursework Report

Evaluation, and Theoretical Reflection of Tasks

Yasmin Buzari

INM432

Notebook Online Link:
https://colab.research.google.com/drive/11uDPiH5BJwR2CTKAKqNT3rHx1dPMUMDY?usp=sharing

---

Task 2c)

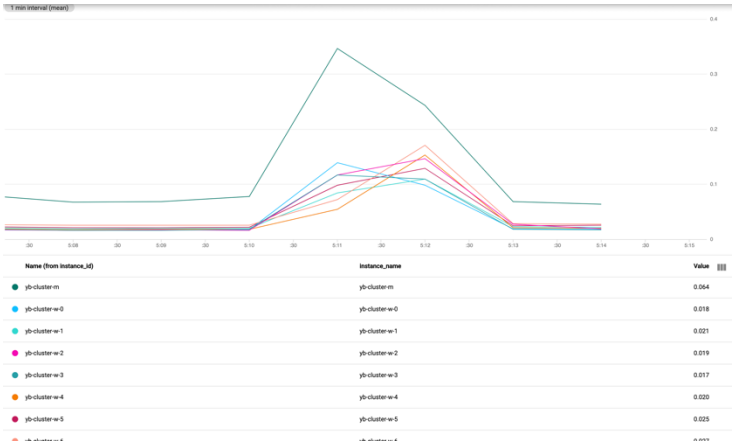
An optimization method is to cache the resulting RDD that has passed through the time function for measurements of parameter values. Caching is useful because it saves interim results to be used in later stages of the script. Thus, the result is kept in a disk and saves computation time because instead of rerunning the function each time to obtain the result across all machines, the cache() added to resulting RDD will store the resulting RDD, across all virtual machines.

Furthermore, optimizing the spark job by numSlices() when creating the RDD parameter list, will denote the number of partitions that the data would be parallelized to, therefore distributing the data processes across. In the task we have set 7 and this will create 7 partitions in the RDD which is distributed. This is another method of optimizing the job than if it was not set. In this case, spark would set N number of partitions based on spark cluster that the program is running on, which would not be ideal if the processing job contains higher parameter value combinations, and more complex processing. Ideally, we adapt the number of partitions to parallelize based on the requirements of the job itself, and what the data processing would benefit most from.

The screenshots below show CPU utilization processing the spark job for Plain images, and TF records to show effects of Caching and Parallelisation.

In Figure 1 we see the CPU utilization when processing the spark job in the box. Without the cache and parallelisation, only 3 workers are processing the job with higher peak/ shorter pyramids. One thing to note, because our parameter combinations are small in these experiments (due to crashing on higher combination) the job was processed relatively fast, and so it is also difficult in this scenario to argue the true effects of caching and parallelisation. However, from this we can get an idea of how the work is distributed by default of the cluster.

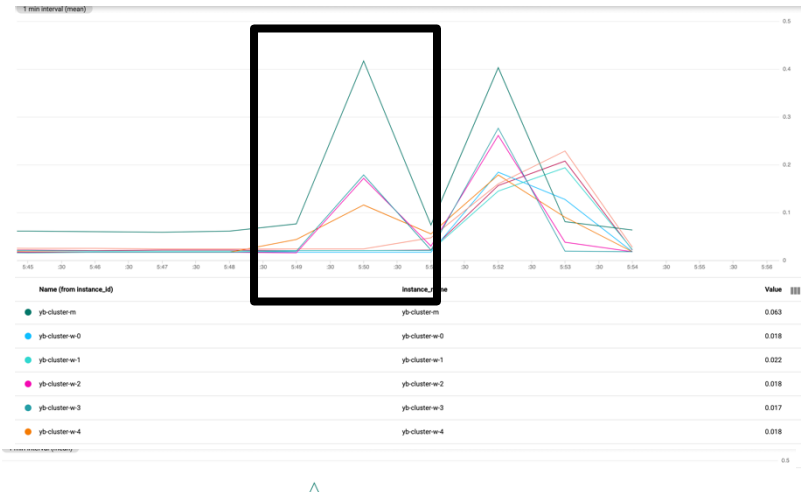
Figure 1: Figure 1: CPU utilization process Plain images, without Cache, NumSlices



In Figure 2: We see the effects of caching and parallelisation when we cache the resulting RDD result, and specify the numSlices() in the parameter RDD list. We can see how 7 workers have converged together, and the process is distributed. Thus, the optimization techniques of caching and parallelisation improve the job processing, resulting in more efficiency. However, because of the small parameter combinations, we cannot conclude here the extent to which parallelisation and caching improve CPU utilization.

Figure 2: Figure 2: CPU utilization processing Plain Images with Cache, NumSlices

It could be that a processing job will not improve by increasing the number of workers for smaller processes because it had plateaued. In conclusion, we see that the spark job in Figure 2: is more parallelized, but the caching is not noticeable because of the few combinations.



In Figure 3: We see the CPU utilization processing the Tensor Flow dataset without caching and parallelization. Similar to Figure 1, the high pyramid spikes/smaller pyramid width show the distribution between 3 workers. If we run the experiment on more parameter combinations, it should show even less convergence of the workers/higher peaks.

Figure 3: CPU utilization for TF Records Job without Cache, NumSlices

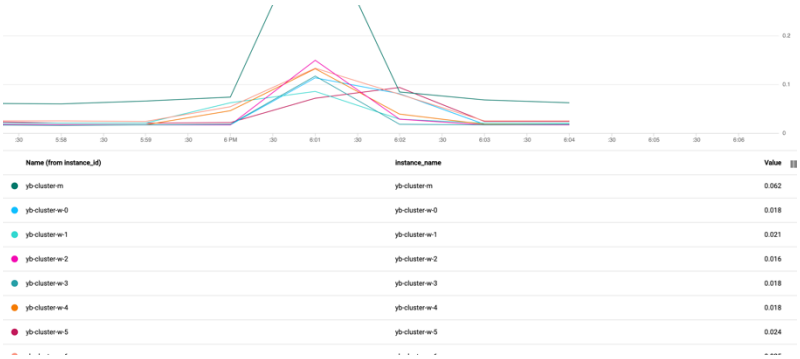


Figure 4 has a similar result of Figure 2 with convergence of workers and the work distributed better.

Figure 4: CPU utilization for TF Records job with Cache, NumSlices

Task 2d)

When performing speed tests in parallel, how the network runs (network latency) can affect processing times, in addition to having access to API's, disks and memory allocations. Other factors that affect times (giving higher or lower speed measurements for running processes) can be the amount of CPU-bound work required. This means, that the program must be big enough so that it can be performed in parallel. Local issues can also cause differences in time measurements as well as task granularity. To get the best accurate picture of timing processes, multiple processes should be run and averaged to account for all these factors that affect timings. Thus, speed timings of certain processes run in the cloud are highly dependent on external factors, and this is a big consideration when forming an assumption on how long a certain process takes to run in the cloud.

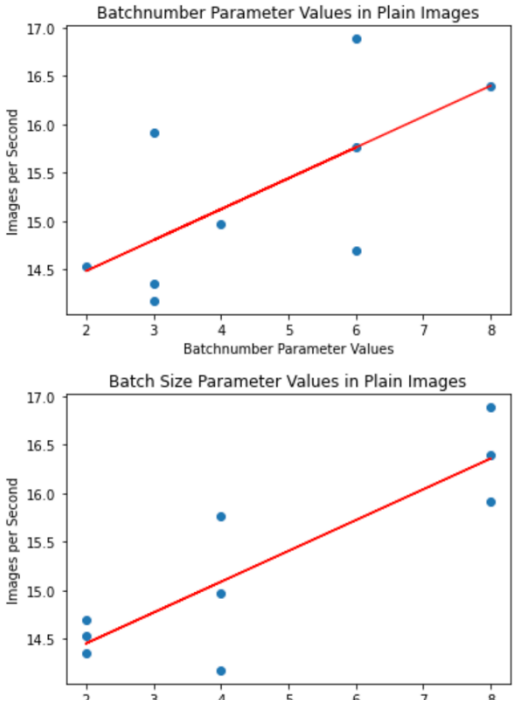


Figure 5 shows parameter associativity of batchsize, batchnumber on images/second. Because of small parameter combinations, it is difficult to accurately say at which point increasing batchnumber and batchsize values will lead to a plateu of images/second processed. Higher combinations and using repetitions as a parameter could not be observed due to program crashes and so only a hypothesis can be made for more variety of parameter combinations. However, from the coefficient value's in the table below, the p value suggests a strong linear relationship between images/second processed and batchsize/number.

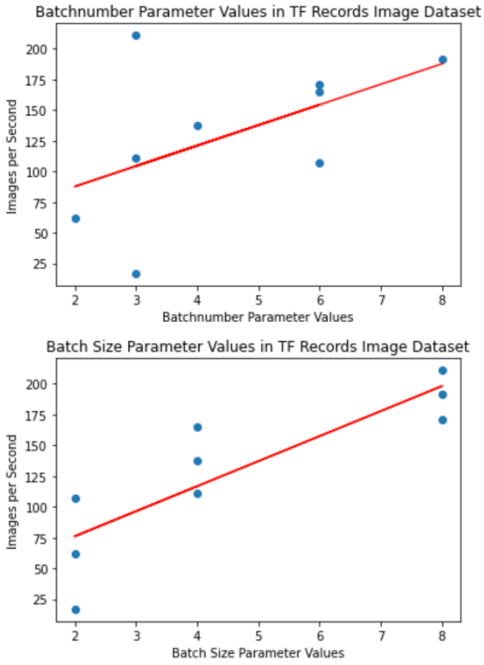
Figure 5: Linear Regression of Parameter values BatchNumber and BatchSize

```
 [8.56077431e-11 2.32785715e-05]

 Summary Table for Linear Regression
Residuals:
    Min      1Q  Median      3Q     Max
-0.6751 -0.2501 -0.0392  0.1177  0.9106


Coefficients:
             Estimate  Std. Error  t value   p value
_intercept  13.818718     0.31747  43.5276  0.000000
x1           0.317391     0.03638   8.7244  0.000023
---
R-squared:  0.75238,    Adjusted R-squared:  0.71700
F-statistic: 21.27 on 1 features
```

Table 1: Statistical Summary of Linear Regression for Plain Images



We can compare here the images/second difference where the TF datasets performs quite higher than the plain images dataset, meaning that the image processing done in tensorflow is computationally more efficient. Similarly to the plain images dataset, there is a strong correlation with increase in parameter values to number of images processed. From this, it is difficult to denote the true relationship with increasing batch size/batch number when the combinations in these experiments are small. We can hypothesize that with more parameter combinations, caching can improve the timings of processing in the cloud when run multiple times as the cache allows processing to work faster after the first initial run. Parallelization as discussed above can also work better with more combinations.

Figure 6: Linear Regression of Parameter values BatchNumber and BatchSize

```
Coefficient P Value:
 [1.38824647e-01 8.70912714e-06]

 Summary Table for Linear Regression
Residuals:
     Min       1Q  Median       3Q      Max
-48.2403 -14.2228   3.4068  27.7994  36.0311


Coefficients:
             Estimate  Std. Error  t value   p value
_intercept  30.621872   18.627873   1.6439  0.138825
x1          21.272811    2.134617   9.9656  0.000009
---
R-squared:  0.79857,    Adjusted R-squared:  0.76979
F-statistic: 27.75 on 1 features
```

Table 2: Statistical Summary of Linear Regression for TF Records

Task 3c)

Here, we tested the experiments of the spark RDD and tensor flower processing of the flower's dataset using different configurations in the cloud. In Figure 7 we see the peak of CPU utilization with 2 workers to run the job. In Figure 9, there isn't a huge difference between resource changes, and this is because of the lower parameter combination. In terms of network bandwidth, we see the bytes sent in Figure 8 and 10, with similar distribution between the two. A small difference in Figure 10 is the higher peak with 7 workers. Also, the relationship with bandwidth and network latency is a re-occurring factor that can affect memory allocation, especially when processing large datasets. The difference between spark and most standard applications that process locally, is that the efficiency of spark relies on additional configurations in the cloud, and more external factors which can alter the times of processes. Whereas locally, these external factors do not impact processing times as storage and disk allocation are all accessed locally in a more straightforward process.

Figure 7: TF Records Spark job with 2 Workers CPU utilization

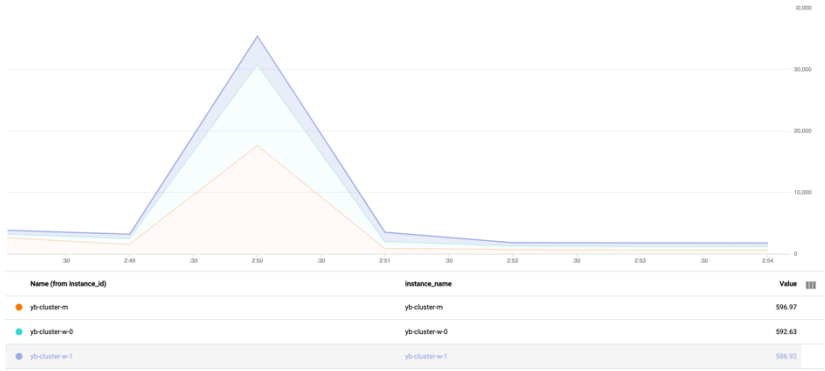


Figure 8: TF Records spark job with 2 workers Bytes Sent

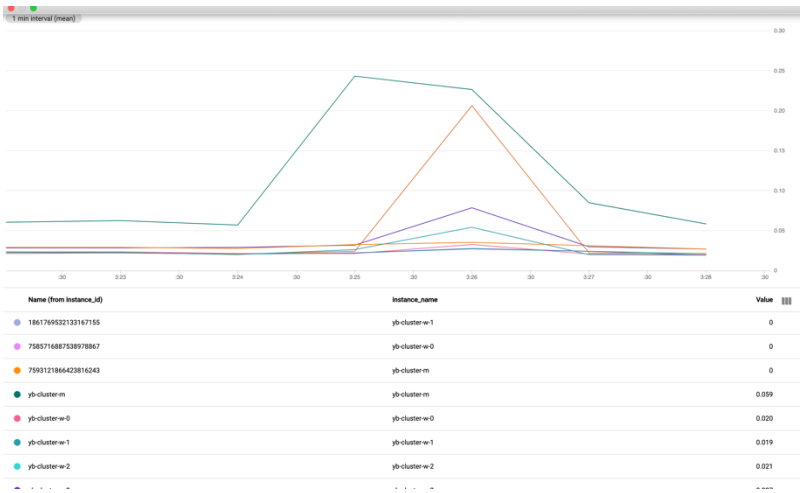
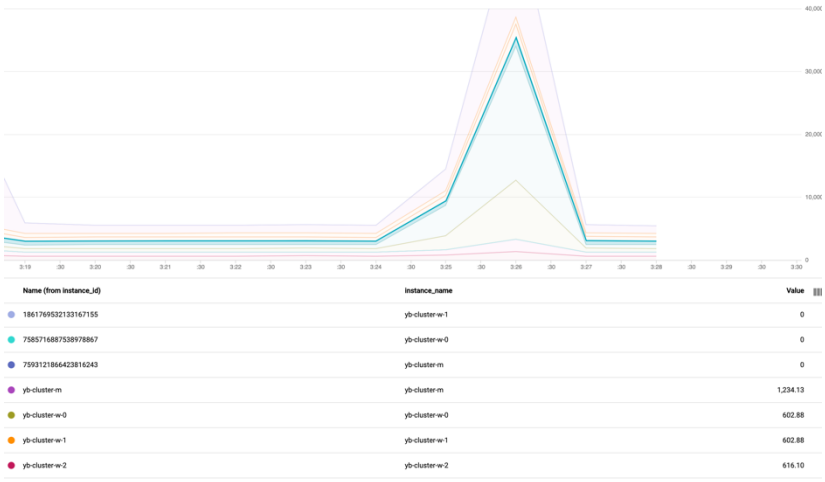


Figure 9: TF Records spark job with 7 workers CPU utilization



Name (from instance_id)	instance_name	Value
1861769532133167155	yb-cluster-w-1	0
7585716887538978867	yb-cluster-w-0	0
7593121866423816243	yb-cluster-m	0
yb-cluster-m	yb-cluster-m	1,234.13
yb-cluster-w-0	yb-cluster-w-0	602.88
yb-cluster-w-1	yb-cluster-w-1	602.88
yb-cluster-w-2	yb-cluster-w-2	616.10

Figure 10: TF Records spark job with 7 workers Byes sent

Section 5a)

One of the paper's discuss the capability of batch size to reduce the effort of concurrent parameter updates resulting in less computation time. This also means there is less focus on hyper-parameter tuning, to result in high model performances in the context of a stochastic gradient descent. We can observe this concept in task 2, where we calculated time measurements for different parameters of batch number and batch size. The idea is that by manipulating batch size's of a dataset and parallelizing the work across a number of virtual machines, the processing time becomes less and more efficient in the cloud. This also depends on the size of the job and dataset. The methodology used in the paper increased batch size during training (the same that we tested in task 2) and compared to decaying the learning rate, and found that the result for model performances are close in both. Thus, increasing batch size is a computationally more efficient way of data processing as opposed to decaying the learning rate and tuning hyperparameter. In task 2 CPU utilization graphs and linear regression models, our findings support this concept. There were also differences seen in CPU utilization when more distribution occurred in parallel for processing the job across virtual machines, which showed higher number of images processed/second.

The second paper focused on optimal cloud configurations, and this is important because on a wider perspective, if you process an analytics job on its most optimal configurations, the results of your analysis can be improved further given the time frame. Conversely, if you run a job on poor configurations, the time spent trying to adjust and reconfigure will eat up the time spent on improving the actual analytics work and results outcome of a job. The paper also argues that, because of different cases of data processing, it is difficult to create a unified configuration set up. Analytic jobs will each require different resources, and this can also be applied to job distribution. In task 3 we got a general idea that distributing work across several worker machines is more efficient, and faster at computing images/second. We can hypothesize that with more parameter combinations, we can see a bigger difference in memory allocation, and network distributions among cloud resources. Because the task tested times for pre-processing the flowers dataset with tensor flow in parallel, using different parameter combinations, the main factor to consider was the distribution of machines when submitting the spark job. Thus, this is the individual requirement of this task and the configuration was focused to address this.

Section 5b)

Generally, batch (offline) processing deals with a large dataset offline, which is the most simple scenario amongst the three. Because the information in the dataset does not change, there is room to experiment with batch size, and cloud configuration to reach an optimal result faster. This gives more room to speed up pre-processing. For example, in Task 2 we saw that pre-processing using tensor flow instead of manually processing the images, resulted in higher throughput (images/second). Therefore, the strategy to create a more efficient analytics job can be focused on improving techniques on the dataset itself (altering batch size, parameter values, and methods of

pre-processing and testing). Caching can also be a technique to use in this case, because the underlying dataset does not change.

Online processing will be different, because it occurs in real time where data is either increasing or changing in nature. This means that the strategy is focused more on cloud configurations to account for this, such as memory allocation, and work distribution. This ties in with external factors that affect processing times, and overall job results such as network latency. Because of these external issues, and the wide spectrum of customizable configurations that Gcloud allow us to utilize, it becomes a challenge to reach to a close optimal configuration. That is where system's like CherryPick, use different frameworks to search for the best configurations for individual job requirements. This reduces computation cost as well as time in analytics projects.

Finally, Stream processing means that new data is processed in intervals, and with different sizes. In a way it can be considered as real time due to its nature of new data in intervals, but also as a batched dataset because the data points remain the same for a longer period of time. Stream processing would be the most difficult to manage in comparison to batch and online processing, because we would need to consider both concepts in the papers to find the best solution. Factors that affect online processing would also affect the job in this way, as well as the factors that affect offline processing such as dealing with dataset size, and processing times. Caching could also be a strategy here, depending on the time window between each interval.

Word Count: 1,993