Big Data Systems (CS744)

University of Wisconsin-Madison

CS744

Assignment 2

Submitted By :
Chakshu Ahuja
Sushma K N
Vibhor Goel

Contents

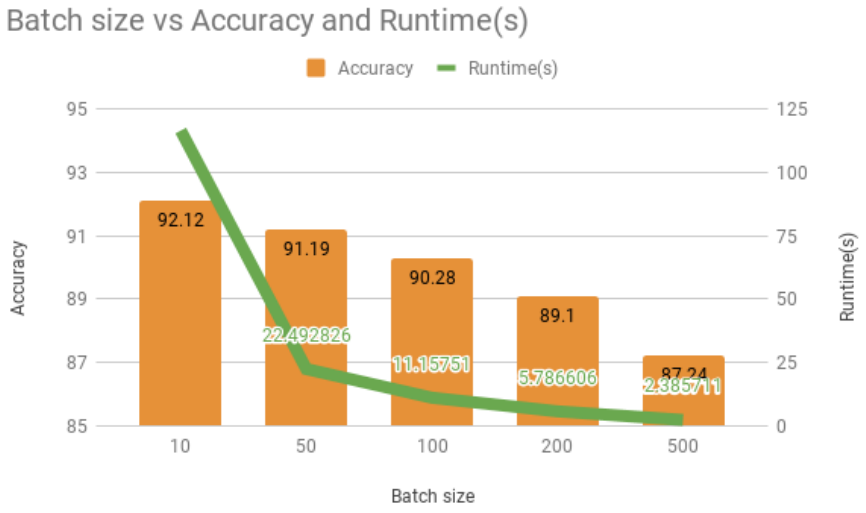
1 Logistic Regression

1.1 Single Machine

We first present our analysis of basic logistic regression code using tensorflow on a single node. Using the MNIST dataset, we train the model by creating a single graph. The model is run in multiple epochs with fixed learning rate and small batches. A session is run for each batch, after which gradients are updated.

We compare the results for varying different batch sizes in terms of accuracy and runtime below. We have kept the learning rate constant as 0.01 and number of epochs as 10. As we can see the accuracy decreases slightly as the batch size increases, similarly with a sharp decrease in runtime. The decrease in runtime from batch size of 10 to 50 can be explained by the fact that increase in batch size, reduces the number of updates in inversely proportional manner.

Figure 1: Batch size comparison for single node



We further show the effect of changing number of epochs on single node in figure 2. The number of epochs as expected increases the accuracy in a proportional manner. However, the rate of this increase becomes very less after a certain stage (200 epochs). The runtime can be seen to increase in proportional manner.

The effect of changing learning rate is shown in figure 6. The change in learning rate shows similar effects as changing the number of epochs.

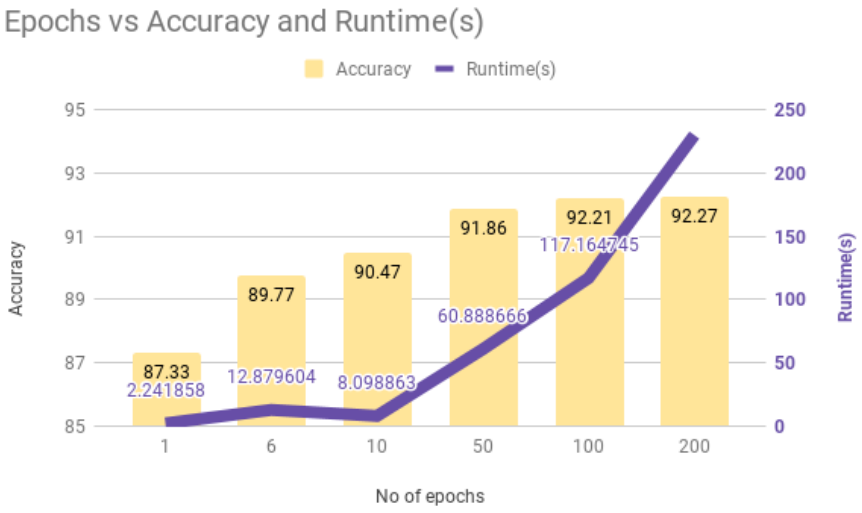
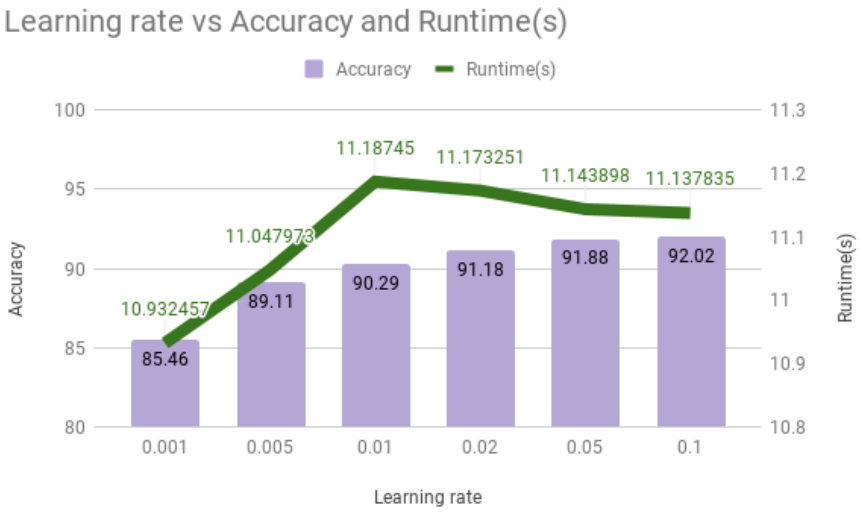


Figure 2: Epochs comparison for single node

Figure 3: Learning rate comparison for single node



Tensorboard

We have also added summary writing using tensorflow into checkpoints for visualization. Using this, we show the computation graph for our model. We can also visualize, the change in accuracy, loss and predictions as we go along through the learning. For recording on tensorboard, we are adding a summary after each batch

gets completed (basically after each session is run).

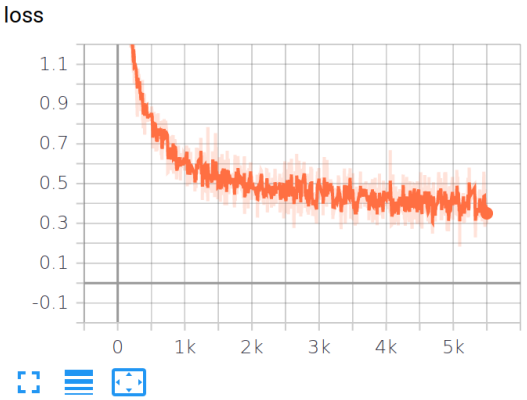


Figure 4: Single node loss (tensorboard)

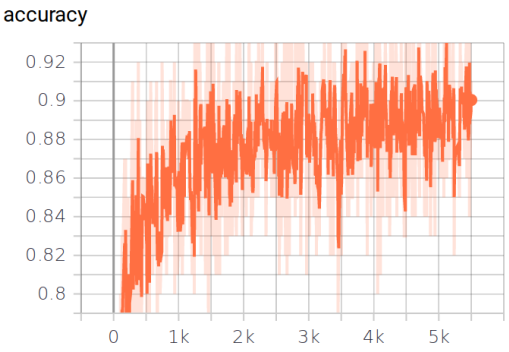
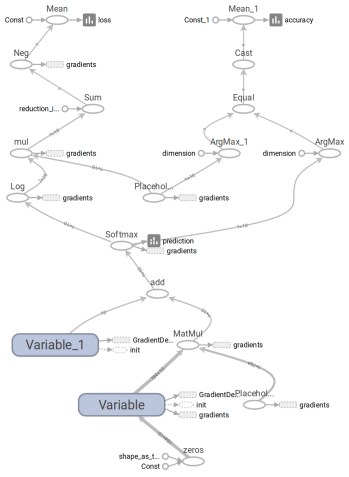


Figure 5: Single node Accuracy (tensorboard)

Figure 6: Tensorboard graph for single node



1.2 Async vs Sync SGD

In this part we implement and compare the performance for two distributed training methods: asynchronous and synchronous. The distributed training can occur in two ways: model parallel manner or data parallel manner. We are testing only for the data parallel manner here. In the data parallel training, the parameters are placed on a parameter server. The data is distributed among workers, each of which computes

gradients and sends them to parameter server. The two differ in the aspect that the updates to the parameters in case of synchronous method are aggregated for workers, and applied together. While in case of asynchronous model, each individual pushes its updates randomly to the parameter server.

Performance Evaluation

From table 1, we can see the performance for asynchronous training on single node vs cluster of 2 nodes vs cluster of 3 nodes. As we can see, the runtime decreases with increase in the nodes, as we are dividing the data between them. There is a slight drop in accuracy, however, that is negligible.

Next, we see the comparison for synchronous training in table 2, for single node vs two node cluster and three node cluster. As we can see there is slightly more decrease in accuracy than asynchronous learning, which is quite surprising. We found that the reason for this is because **updates in the SyncReplicasOptimizer (optimizer used for aggregating synch updates) are averaged** for workers, instead of getting summed. Hence, the progress is slow. The time reduction compared to single node is expected due to the sharing of data between workers.

Cluster Type	Accuracy	Time (in sec)
Single	0.9031	14.302971
Cluster	0.9025	10.002409
Cluster 2	0.9019	6.972539

Table 1: Performance comparison of training Async SGD on single node, cluster of 2 workers and cluster of 3 workers

Cluster Type	Accuracy	Time (in sec)
Single	0.903	11.544702
Cluster	0.891	9.48555
Cluster 2	0.8843	6.620273

Table 2: Performance comparison of training Sync SGD on single node, cluster of 2 workers and cluster of 3 workers

We also see the accuracy and decrease in loss for async training through tensorboard in figure 13 and 10. Both accuracy and loss follow the same pattern when we add clusters, with addition of noise along with faster convergence.

From table 3, we can see the CPU performance and memory utilization for both the processes. As we can see, the CPU utilization and memory usage is not significant

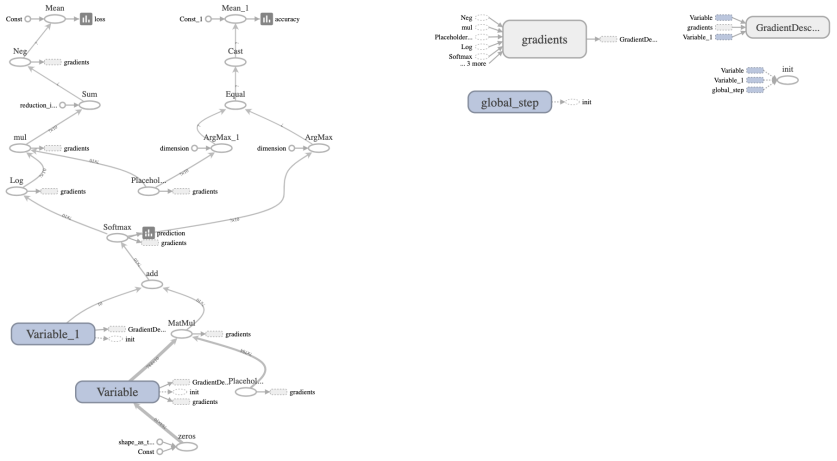


Figure 7: DAG of running Async SGD on Single node cluster

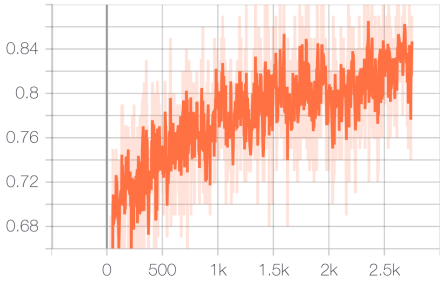


Figure 8: Single node cluster

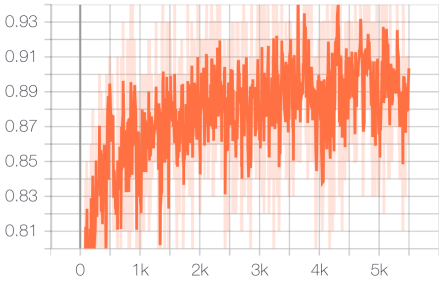


Figure 9: Cluster of 2 workers

Figure 10: Accuracy (from tensorboard) by running Async SGD

and in accordance with single cluster node. The network transfer however has increased and acts as the bottleneck. We also saw that the amount of data received on node 1 is almost thrice that sent from node 2 which should be the ideal case when we run on 3 node cluster, as the first node receives all the parameter updates.
Comparing with the single node statistics, we can see that the cpu usage for the master node in both synch and asynch learning reduces, which is again expected. However, memory usage does not get reduced, as it is still getting data from other

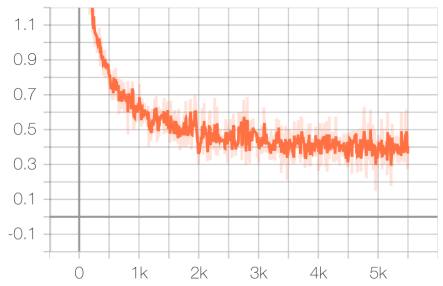


Figure 11: Single node cluster

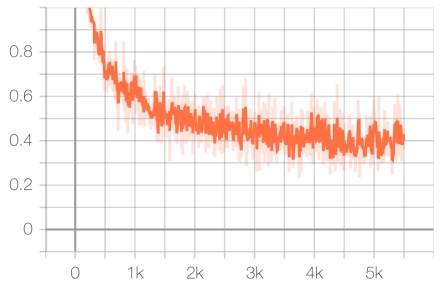


Figure 12: Cluster of 2 workers

Figure 13: Loss (from tensorboard) by running Async SGD

nodes.

We also show the workerwise runtime comparison for the two training methods [Figure 14]. As, we can see since each worker independently updates in the asynchronous method, the finish time for each worker varies. However, in case of synchronized learning, the workers have to wait constantly for stragglers, and coordinate with each other with respect to time. Though, we have not displayed here, we also saw the test accuracy after each epoch for asynchronous and synchronous training workerwise. There was exact match in accuracies at workers in case of synchronous model, while asynchronous accuracy was slightly different for workers.

Experiment Type	Avg CPU Usage	Avg Memory	Network received	Network sent
Single node	11.8	685M	-	-
Async SGD	5.3	674M	20M	21M
Sync SGD	6.3	707M	22M	22M

Table 3: Performance comparison in terms of CPU/Memory/Network received/sent of training Sync SGD and Async SGD on cluster of 3 workers

We also compare the performance for network usage among different workers for asynchronous and synchronous learning on three nodes. From here, we can see that the network usage is almost double on node 2, which is expected as it is getting data from the two other nodes.

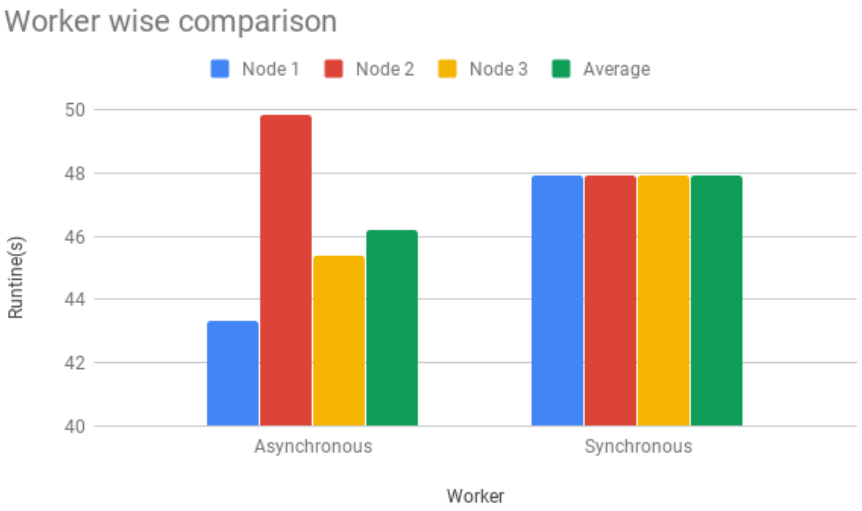


Figure 14: Worker runtime comparison for synchronous and asynchronous training

	Node 1		Node 2	
Experiment Type	Network rec	Network sent	Network rec	Network sent
Async SGD	29M	28M	14M	14M
Sync SGD	22M	26M	13M	13M

Table 4: Performance comparison of network usage among different workers in synch and asynch learning

1.3 Batch Performance Evaluation

In this section, we compare the performance of synchronous [Table 6] vs asynchronous [Table 5] on cluster of three nodes with single node [Figure 1] for different natch sizes. As we can see, upon increasing the batch size, the runtime decreases in an inversely proportional manner for both synch and asynchronous methods. There is also a decrease in accuracy noted, because increasing the batch size leads to less updates giving slower convergence.

Similarly, the decrease in case of single node from batch size 10 to 50 is there [116 to 22]. However, the single node takes more time than synch and async trainings which can be attributed to parallelization.
In both synchronous ad asynchronous, we report out results keeping the global steps as constant (each global step corresponds to processing of one batch each by nodes.)

Batch Size	Accuracy	Time (in sec)
10	0.9232	50.749356
50	0.9112	11.972627
100	0.9019	6.972539
200	0.8916	4.32506
500	0.8736	2.561592

Table 5: Performance analysis of training Async SGD on a cluster of 3 workers with different batch sizes

,

Batch Size	Accuracy	Time (in sec)
10	0.9172	47.61287
50	0.8970	11.36118
100	0.8837	6.67819
200	0.8678	4.1216
500	0.8394	2.44617

Table 6: Performance analysis of training Sync SGD on a cluster of 2 workers with different batch sizes

2 AlexNet

AlexNet is a very famous convolutional neural network. It consists of five convolutional layers followed by three fully connected layers. It uses ReLU activation function instead of Tanh to add non-linearity. It is designed to classify images.

In this part, we analysed the performance of AlexNet by training the model on a single machine and also in a distributed environment. We compared the performance of training the model by varying batch-size and number of workers in a distributed environment.

We trained the model in data parallelism fashion using Tensorflow framework and for both single and cluster setup, we followed the synchronous approach. All the experiments were done on fake dataset given in the assignment

2.1 Single Machine

In this section, we will experiment on training AlexNet model on a single machine and see how does the training proceed with different batch-sizes of training instance/examples. we will analyse different aspects of training the model like CPU

and memory utilization.

Performance Evaluation

Batch Size	Examples/sec	Time (in sec)/batch
32	14.0	2.29
64	26.7	2.40
128	41.4	3.10
256	42.3	6.06
512	42.9	11.97

Table 7: Performance analysis of training AlexNet on a single machine

The table 7 shows run-time for every batch and the number of training instances processed per second. We can clearly observe that the number of units processed per second has almost doubled starting from batch-size 32 to 128 while the batch run-time shows a very slight increase. This stands with the most obvious case where the resources are underutilized initially. The batch instances are processed very quickly until the point where the resource is completely utilization allocated on a single machine (which we can observe from the batch-size 128). We can notice that the batch is processing at a very slow rate from the batch-size 128 with more examples per batch. This supports our argument that the computation with the higher batch size is taking time because of complete utilization of resources, the run-time is almost double with the slightly same number of training examples processed per second between batch-size 128 and 512.

As the batch-size increases, the increase in batch processing speed is not linear with the number of processed instances. This is because of many intermediate tensors computation (as each batch involves many examples).

CPU/Memory Utilization

The figure 15 shows the CPU utilization with different batch sizes. As evident to our explanation on analysing the processing speed, we can observe that the CPU utilization has a substantial growth for initial batches between 32 and 128, after which there is very slight variation in CPU utilization as the resource is at it's most efficient utilization.

The figure 16 shows the highest memory(RAM) utilization at a particular instance of time for different batch sizes. Intuitively, for larger batch size there would be greater memory utilization, so does the above graph represents.

Batch Size	CPU(in Percentage)	Memory (in MB)
32	57	3366
64	71	4081
128	88	4446
256	94	5191
512	98	6057

Table 8: CPU/Memory utilization of training AlexNet

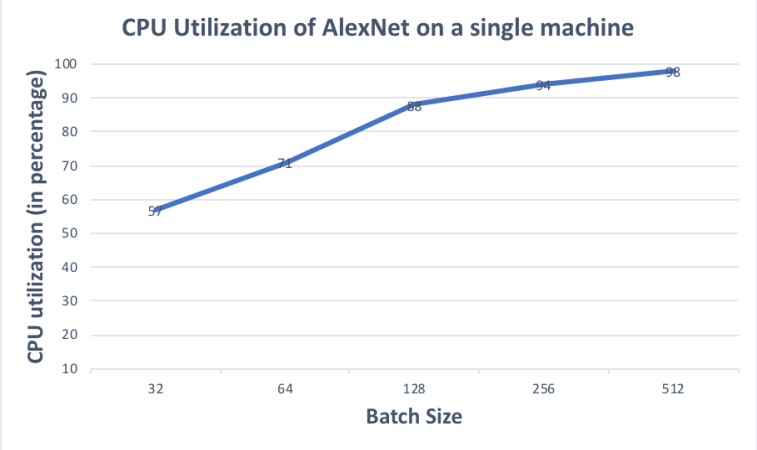


Figure 15: CPU analysis of training AlexNet on single node

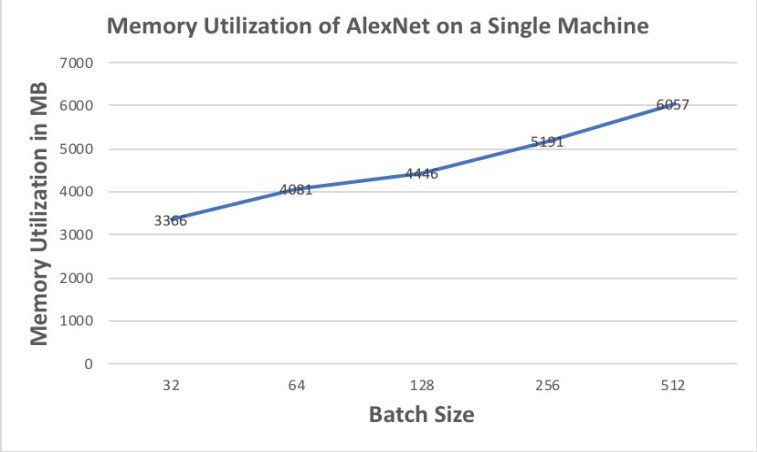


Figure 16: Memory analysis of training AlexNet on single node

2.2 Cluster

In this section, we started training the AlexNet model in a distributed environment which is a cluster of workers and a parameter server. The conducted experiments

exploit on the CPU, memory and network usage while training the model with different batch-size of training instances.

Note:

- Cluster-1 is a cluster of two workers and a parameter server and
- Cluster-2 is a cluster of three workers and a parameter server (one of the will perform both tasks of worker and parameter server)
- All experiments are done using fake data.
- All the experiments are executed in a synchronous fashion.

Performance Evaluation

Batch Size	Cluster-1		Cluster-2	
	Examples/sec	Time (in sec)/batch	Examples/sec	Time (in sec)/batch
32	27.9	2.30	40.5	2.39
64	54.9	2.34	78.0	2.48
128	67.9	3.78	93.3	4.13
256	71.8	7.14	99.2	7.75

Table 9: Performance analysis of training AlexNet on cluster

The table 9 compares the performance of training AlexNet model in cluster-1 and cluster-2. As expected we can see a substantial increase in run-time with the increase in batch-size. Cluster-2 with more number of workers will process more examples per second than cluster-1 and will take more run-time to complete a batch of data. The slightly greater run-time in cluster-2 is the synchronization cost (more workers to synchronize).

The figure 17 shows the CPU utilization comparison of Parameter Server node for Cluster-1 and Cluster-2. We can observe for all the batch-size there is a slight increase in the CPU utilization for cluster-2. This is because the Parameter Server has to do little more processing in terms of accumulating gradients.

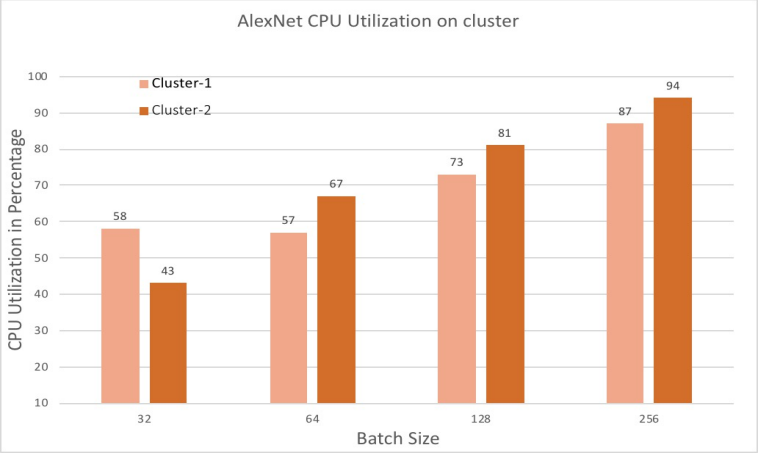


Figure 17: AlexNet CPU utilization comparison on clusters

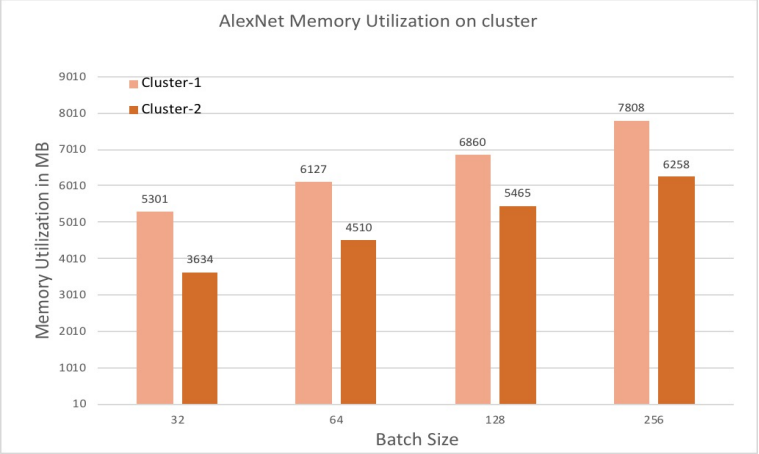


Figure 18: AlexNet Memory utilization comparison on clusters

We also show the comparison of Alexnet memory utilization [Figure 18] on the Parameter Server for clusters of two and three nodes using different batch sizes. From here, we can see the memory usage on cluster 2 is always less which is expected as data gets divided across three workers. Further, an increase in batch size leads to higher in-memory utilization because more data is loaded in each batch.

The figure 19 shows the network usage at the Parameter Server for both cluster-1 and cluster-2.

All model parameters are aggregated and updated at the Parameter Server. Ideally, the network traffic should increase at the Parameter Server as we increase the number of workers. This is due to the reason that the model parameters has to be

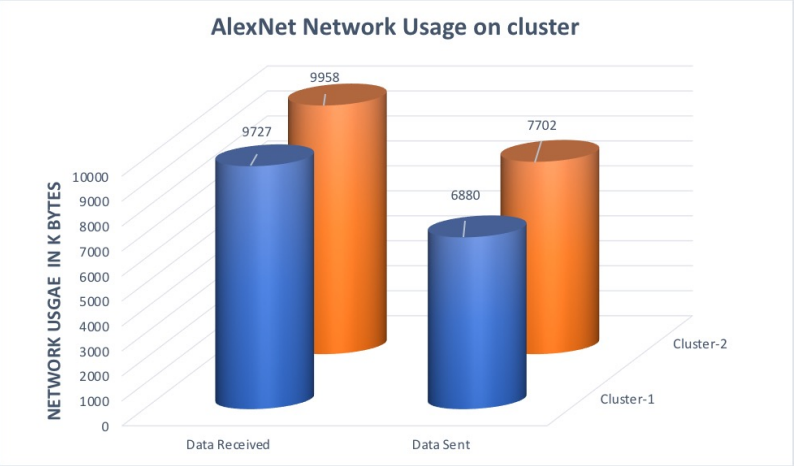


Figure 19: AlexNet Network utilization comparison on clusters

sent/receive from all workers. Above plot supports this intuition.

As the Parameter Server will receive all the parameters updates and will send the updated parameters along with data to the workers, we ideally should expect more data being received than sent at Parameter Server. For both the setup, we observe this pattern in the above plot.