TensorFlow Evaluation

GUO Suibing
20387947
sguoah@connect.ust.hk

LIN Shangyu
20401901
slinaq@connect.ust.hk

YU Mengxue
20406559
myuaf@connect.ust.hk

QIU Feng
20398324
fqiuaa@connect.ust.hk

1. INTRODUCTION

TensorFlow is a platform in making use of machine learning algorithms, and an implementation for executing these algorithms. It is also a flexible system, which can be used to express a variety of machine learning models, such as neural networks, with which users can train them on large datasets and put them into production. It has been used for conducting research and for deploying machine learning systems into production across many areas of computer science and other fields, including speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and computational drug discovery [1]. It was at first explored by researchers and engineers in Google Brain Team applying in very-large-scale deep neural networks and conducting machine learning, but it is also able to widely use in other domains as well.

TensorFlow uses data flow graphs for numerical computation. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. [2]

The publishing of distributed TensorFlow is convenient for users to train models in parallel on multiple servers in order to reach the best data scale and fault tolerance. In the meantime, it encourages researchers to dig into the distributed strategy for more deep learning models of large quantity of reality.

The current distributed scheme need conduct several machines all together by manual configuration. This can basically meet the need of small-scale distributed computing, but need more manual configuration. It requires us to put the script and data distribution on different machines and assign different tasks on different servers. It is still the original distributed scheme. For the present distributed solution, there is no specific high performance parameters to the server. The overall efficiency is not able to reach a relatively advanced level. Another solution is to combine Docker and Kubernetes to carry out large-scale distributed deployment. This project doesn't bring into use the solution because of small scale experiments.

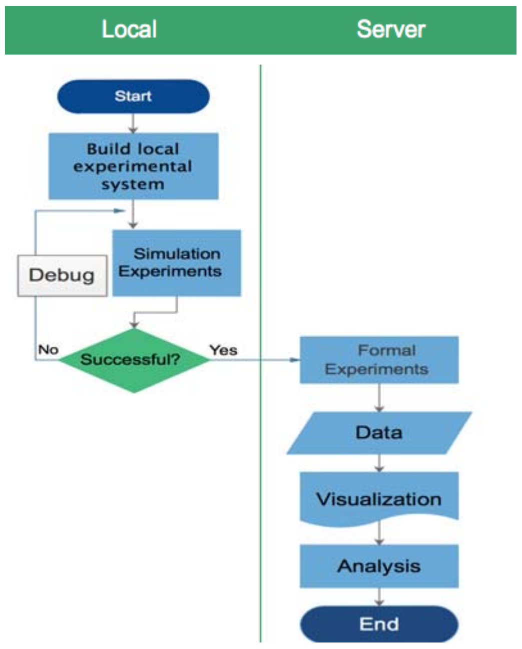
This project is an evaluation of the open-sourced TensorFlow as a software system. By modifying system parameters, the performance of distributed system (including time complexity, computing results and utilization efficiency of hardware) contributed by various parameters can be measured. Our project is going to identify the performance defects, bottlenecks, and inefficiencies in the framework. We are going to use control variate method in our project to make comparisons among different algorithms and to see what's the differences among them. We want to see the bottlenecks of different algorithms. By making an observation, it is obvious to know what's the need of resources of each algorithm. The only fixed parameter is the number of servers. Our purpose is to choose different schemes according to disparate situations when constructing distributed TensorFlow system.

2. DESIGN AND IMPLEMENTATION

2.1 Methodology

Regard the performance of single machine model with the initial dataset as the comparison baseline. By using the method of control variate to test the distributed model (AutoEncoder and Transformer) with three system variables, different performances of the model with different parameters combination have been record to evaluate the TensorFlow.

The following flow chart shows the overall experimental steps. The details of the formal experiment will be illustrated in the Implementation Details part.



In order to guarantee the safety of the server, we build the local experimental system and install two 'Debian' systems in a virtual machine as two servers. As a result, the code of all the model with all system parameters have been debugged on the local environment.

2.2 Implementation Details

2.2.1 Experimental environment construction

Install TensorFlow based on the Virtualenv on the server environment. And then setup monitoring system Collectl on every server and use a shell script to collect data.

2.2.2 Baseline test

Run the single machine model with original dataset and record their performance as the baseline.

2.2.2.1 Model: AutoEncoder -- various autoencoders

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs [3].

Dataset: MNIST.

2.2.2.2 Model: Transformer -- spatial transformer network, which allows the spatial manipulation of data within the network

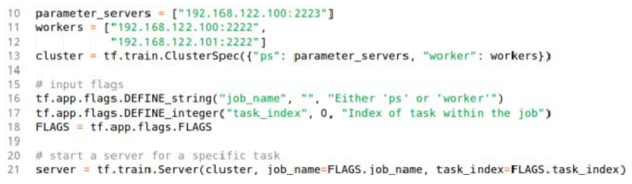
Dataset: MNIST.

2.2.3 Model modification

2.2.3.1 Transform the tested single machine model into distributed model

There are different ways to train a network in a distributed fashion. The simplest approach is to share all the model parameters across all workers while parallelizing data and gradient updates [4]. The following code blocks show our major steps to share parameters among all servers.

First, the following code blocks show our major steps to share parameters among all servers.

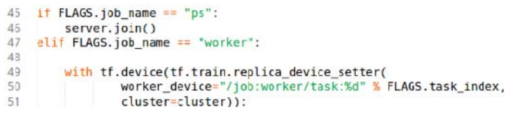


```
parameter_servers = ["192.168.122.100:2223"]
workers = ["192.168.122.100:2222",
           "192.168.122.101:2222"]
cluster = tf.train.ClusterSpec({"ps": parameter_servers, "worker": workers})

# input flags
tf.app.flags.DEFINE_string("job_name", "", "Either 'ps' or 'worker'")
tf.app.flags.DEFINE_integer("task_index", 0, "Index of task within the job")
FLAGS = tf.app.flags.FLAGS

# start a server for a specific task
server = tf.train.Server(cluster, job_name=FLAGS.job_name, task_index=FLAGS.task_index)
```

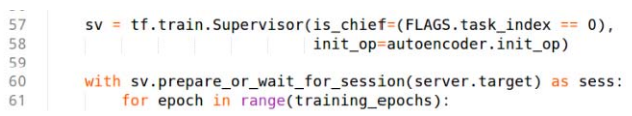
Second,



```
if FLAGS.job_name == "ps":
    server.join()
elif FLAGS.job_name == "worker":

    with tf.device(tf.train.replica_device_setter(
            worker_device="/job:worker/task:%d" % FLAGS.task_index,
            cluster=cluster)):
```

Third,



```
    sv = tf.train.Supervisor(is_chief=(FLAGS.task_index == 0),
                             init_op=autoencoder.init_op)

    with sv.prepare_or_wait_for_session(server.target) as sess:
        for epoch in range(training_epochs):
```

Two servers were set up, one parameter server and two workers.

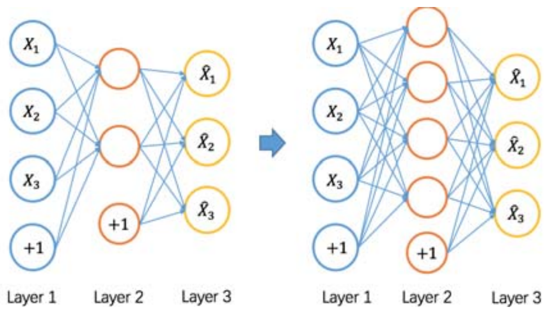
If it shows PS, we add server in. If it's worker, we execute the following steps and initialize variables. Finally, we wait for all the workers to get in.

2.2.3.2 System Parameter Modification

In order to observe the influence of different parameter combinations on system performance, we need to modify the system parameters.

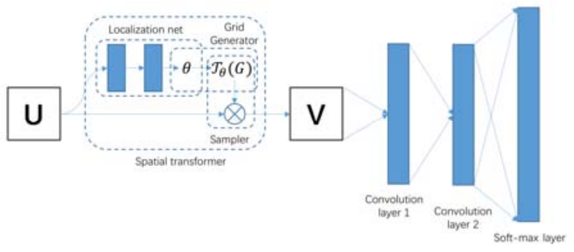
2.2.3.2.1 Model size of AutoEncoder

It is a method of model complication, which increase the number of hidden layer neurons to layer 2.

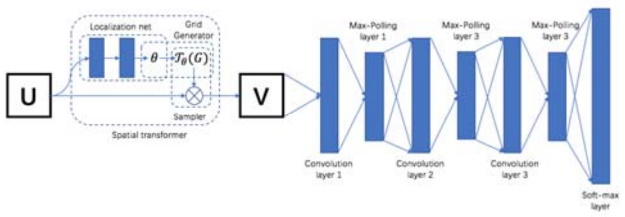


2.2.3.2.2 Model size of Transformer

There are two convolution layers after the Spatial Transformer Network in the initial Transformer model.



In the modified Transformer model, we add one down-sampling layer to every convolution layer, then add a new convolution layer and a down-sampling layer before the soft-max layer.



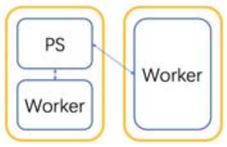
2.2.3.2.3 Dataset size

As for the quantity of the datasets, MNIST is the dataset for both of the models, but the format of these two datasets are different. Fortunately, both of algorithms have provided the specific methods to modify the size of datasets. We only need to change the settings of those methods, and then we can get the different size of datasets.

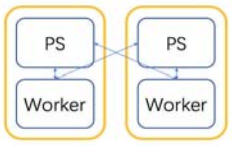
2.2.3.2.4 Cluster structure

Three cluster structures are used to test the distributed models.

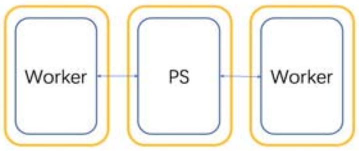
In the first structure, let a parameter server and a worker in one machine. This mode is called PS-W-W in the following context.



Then let the other worker in another machine. So, there will be communication between two machines. It is called PW-PW for short.



In the third structure, we put the parameter server and worker in different machines. Then all the workers should send their updated parameters to the parameter server. This pattern is named by PS-W-W.



2.2.4 *Run Distributed model and record data*

To run the distributed model and collect system information by using the Collectl, we use the shell script to simplify the setup steps.

Since there are 2 kinds of models (large and small), 2 kinds of datasets (large and original) and 3 different cluster structures, 12 different parameter combinations should be tested. For each combination, we test three sets of data.

2.2.5 *Visualization and Analysis*

To simplify the analysis, we make the visualization by using the library matplotlib in Python.

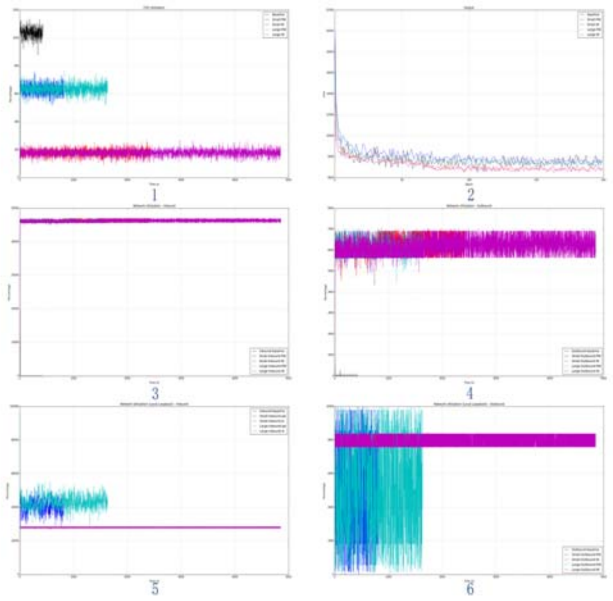
In order to analysis the condition of server, put all the system parameters which need to be analyzed together.

3. EVALUATION

For concisely, in the following passages, PS represents parameter server, PW stands for the parameter and worker server, and W indicates the worker server.

3.1 Changing the size of dataset and comparing the performance of the model

3.1.1 Model: AutoEncoder



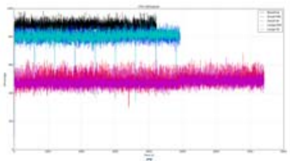
The CPU utilization of the AutoEncoder model running on single server is represented by the black line in picture 1. It takes short time for single machine to train the dataset since the CPU utilization is really high. The CPU utilization of parameter server with small dataset is shown by the blue line, while the one with large dataset is the green line. Red line reveals the CPU situation of the worker server with small dataset, and the magenta line is the one with large dataset. In the case of data quantity increases, the running time becomes longer, but each server's CPU utilization remains the same.

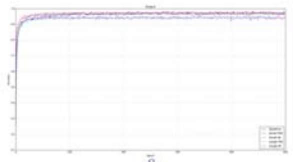
Graph 3 and graph 4 demonstrate the network utilization of outbound and inbound respectively. Because of the network utilization of the parameter server and worker server are similar, the lines of different cases are lapped. Hence the trend is demonstrated shown by the magenta line, which is the longest since it used the largest dataset. The usage to communicate with each other in the network utilization remains the same, either the outbound or the inbound.

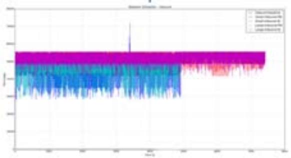
According to graph 5 and graph 6, the network utilization of local loopback inbound is lower than others, maybe because the master server has other work which draw down the inbound network utilization. The local loopback network utilization remains the same, either the outbound or the inbound

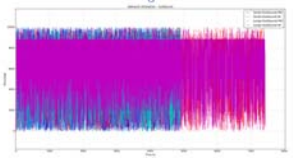
It is shown in graph 2 that the loss of training results in different cases are basically the same. After 100 epochs, the loss decreases slightly.

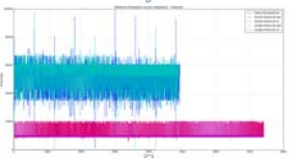
3.1.2 *Model: Transformer*

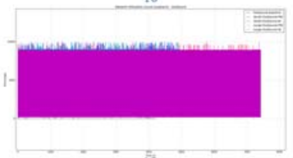

7


8


9


10


11


12

In graph 7, the CPU utilization baseline of the Transformer model is represented by the black line. The green line and the blue line present the CPU utilization of the parameter server for the small dataset and the large dataset respectively. There is no big change between them. In the meantime, there is no significant difference between the red line and the magenta line, which are the CPU utilization of small and large dataset for the worker server. The CPU utilization of distributed TensorFlow is lower than the utilization of single machine, since cooperation and dispatch are needed between servers. While, the utilization of small and large dataset is almost the same.

So, increasing the amount of data to the CPU utilization makes little or no difference.

As the size of datasets differ, the inbound of network utilization changes inconspicuously comparing graph 9 and 10. While, the inbound network utilization of worker server is higher than parameter worker server. The parameter server has to distribute tasks for itself and other worker servers. Therefore, it takes more time for worker server to get the dataset and tasks from the parameter server. Here, the load of the parameter server is the heaviest. What's more, the parameter server distributes task to itself use the local loopback network, which needs the most of time, maybe a bottleneck of this model.

The time cost of outbound network utilization of the worker server is higher than the parameter server. But the outbound flow seems similar.

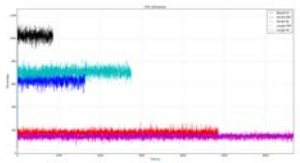
3.1.3 *Conclusion of changing dataset*

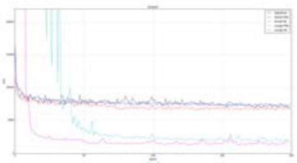
Increasing the amount of data, the model is the same and there is little influence on the performance of the whole system.

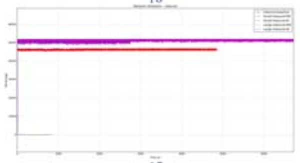
As for AutoEncoder, the cost of time becomes longer. And there is no influence on Transformer.

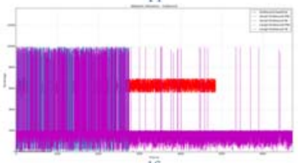
3.2 Changing the complexity of model and comparing their performance

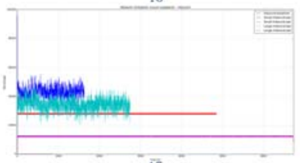
3.2.1 *Model: AutoEncoder*

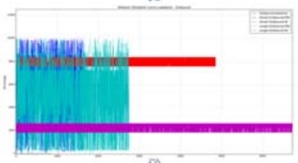

13


14


15


16


17

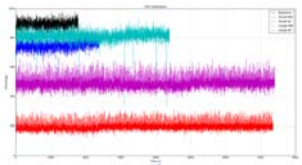

18

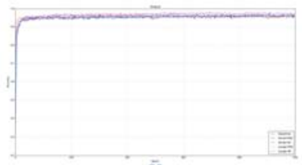
When complicating the model, it will make the CPU utilization of master increase, which is presented in graph 13, because we need to deal with more parameters. However, the utilization of slave server will decline. And in picture 16 and 18, the complicate model will increase the outbound peak and network instability. But when the worker of PS end of work, it will reduce overall utilization.

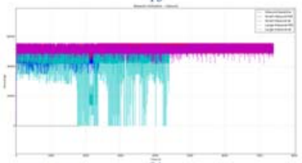
When complicating model, the pressure of inbound network is increasing, while the outbound network utilization of the local loopback is reduced. It will reduce the inbound network utilization of the local loopback as well.

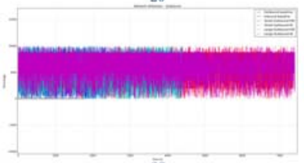
On account of this, complicating model will significantly reduce the loss.

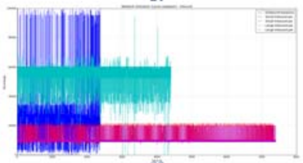
3.2.2 *Model: Transformer*

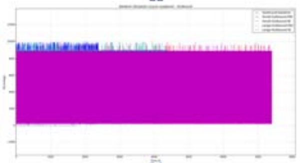

19


20


21


22

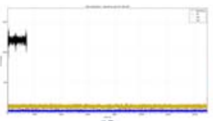

23

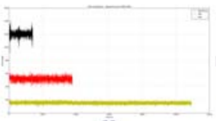

24

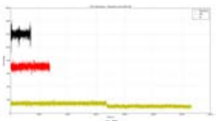
When complicating the model, we can see from graph 19 that the CPU utilization and the using time of PW are increasing. From graph 22 and graph 24, the outbound network utilization remain the same. From graph 21, it shows us the network instability increasing of the PS server. Graph 23 gives us the increasing of local loopback inbound utilization. And graph 20 indicates there is a little bit increasing in accuracy.

3.3 Changing task of servers and comparing their performance of different models

3.3.1 Model: AutoEncoder

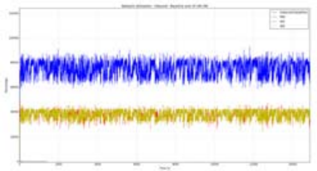

25

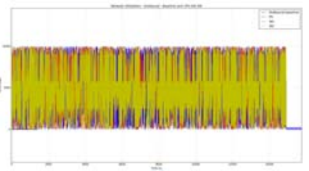

26

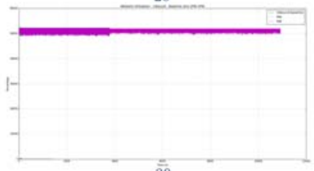

27

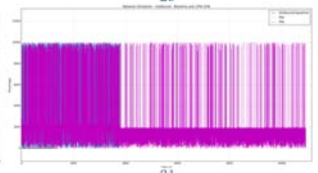
Observing the mode of PS-W-W in figure 25, the CPU utilization of the two workers is similar, which is also low, because the servers need to pass parameters to each other.

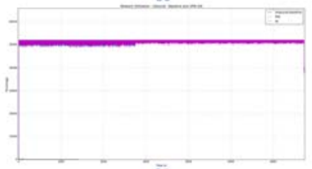
Comparing to picture 26 and 27, the CPU utilization of the pattern PW-PW and the pattern PW-W is similar, but PW-W is a little bit higher. Therefore, the time cost of mode PW-W is shorter.

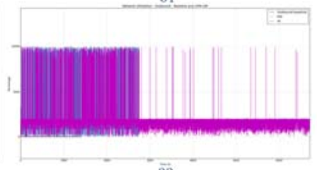

28


29

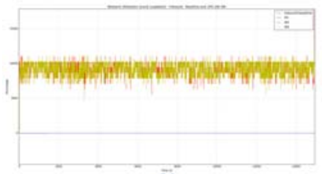

30

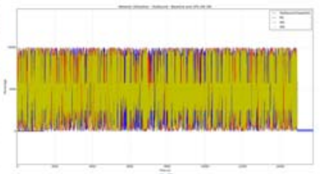

31

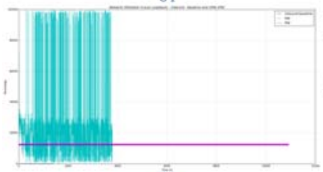

32

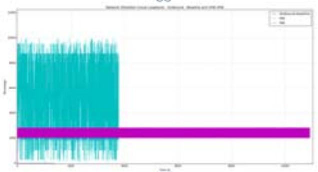

33

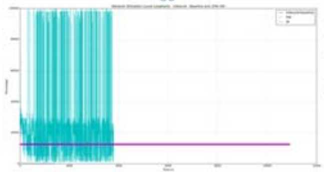
These graphs demonstrate the inbound and outbound network utilization of different patterns of parameter server and worker servers. Figure 28 and 29 layout the inbound and outbound network utilization of mode PS-W-W. In a similar way, picture 30 and 31, picture 32 and 33 refer to the mode PW-PW and the mode PW-W respectively. Graph 28, 30, 32 in the left side are the ones of inbound network utilization, and the graph 29, 31,33 in the right side refer to the outbound network utilization. Thus, the network load can be compared in horizontal and longitudinal, among which the pattern PS-W-W has the heaviest network load.

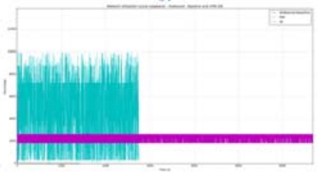

34


35


36

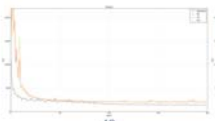

37

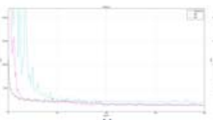

38


39

Originally the division of two machines looks the same. Because of the parameter server does not need to send data to itself, it’s 0 in graph 34. The condition of the local loopback also should be the same. However, there is also a chief in TensorFlow. (The official instruction for chief is: training with replicas you deploy the same program in a Cluster. One of the tasks must be identified as the chief: the task that handles initialization, checkpoints, summaries, and recovery. The other tasks depend on the chief for these services.)

Therefore, the task will be a little heavier than the others. When task_index=0 defined as chief, the condition of the two machines are different.


40


41

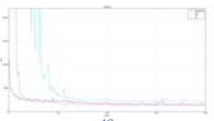
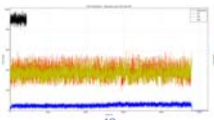
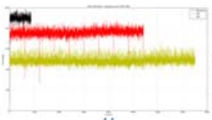
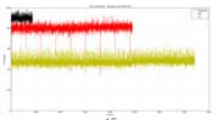

42

Figure 40, 41, and 42 display the output loss of the mode PS-W-W, PW-PW, and PW-W. Their loss of pattern PS-W-W converging faster than others.

3.3.2 Model: Transformer

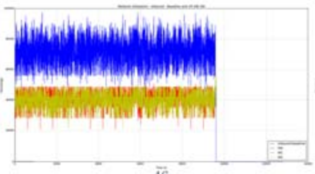

43

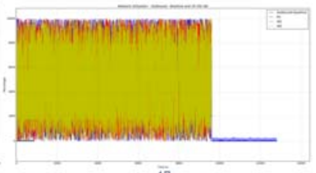

44

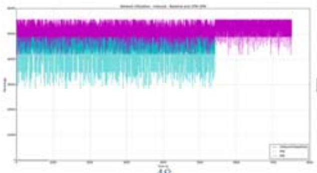

45

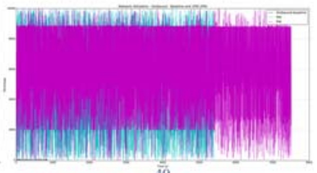
Same as the situation of model AutoEncoder, making a observe of the mode P-W-W in figure 43, the similar CPU utilization of two workers is lower, for the reason that servers need to communicate for parameters.

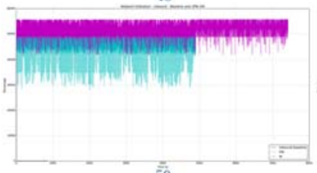
As to graph 44 and 45, the CPU utilization of the pattern PW-PW and PW-W is almost the same.

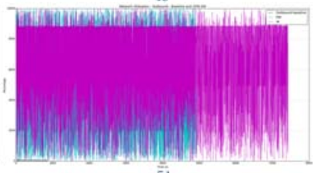

46


47

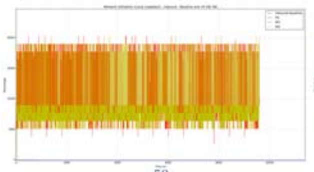

48

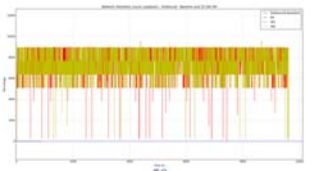

49

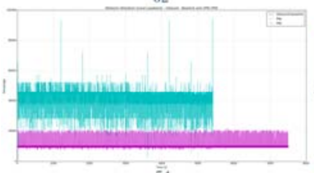

50

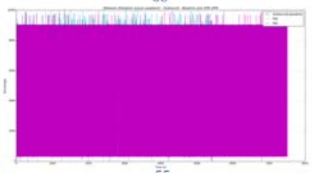

51

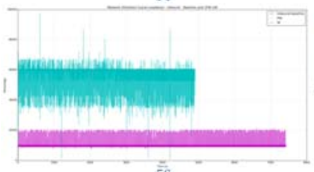
Comparing figure 46-51 in horizontal and longitudinal, it is obviously that the mode PS-W-W has the heaviest network load, which is the same as the model AutoEncoder.

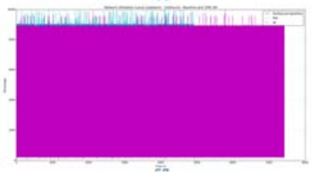

52


53

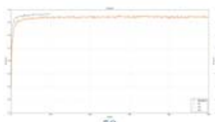

54

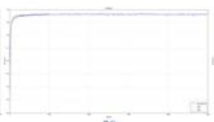

55

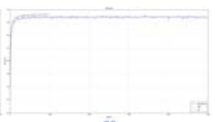

56


57

In figure 52 and 53, the pressure of network between workers are balanced in the mode PS-W-W. And obviously, there is no significant difference between the network utilization of pattern PW-PW and PW-W according to picture 54-57. Also, the task of chief in TensorFlow will be a little heavier than the others.

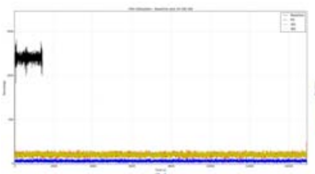

58

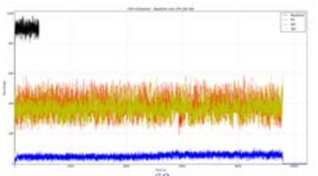

59

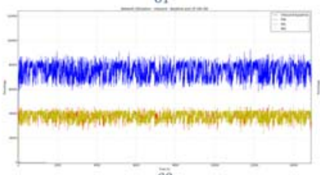

60

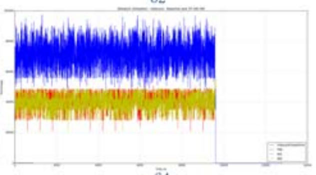
The loss of output among modes PS-W-W, PW-PW, PW-W seems similar.

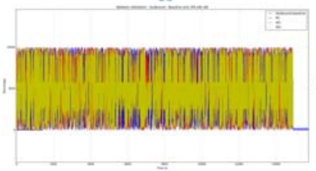
3.4 Comparing the performance of model AutoEncoder and Transformer

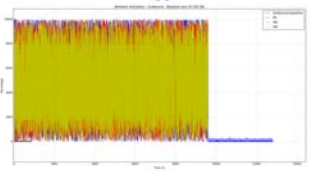

61


62


63


64


65


66

In our experiment, changing model size of AutoEncoder algorithm is realized by changing the numbers of hidden layer neurons. According to graph 61, the CPU utilization is not high, it may be blocked. The network utilization in figure 63 of PS is high with just a little change.

In the meantime, the change of model size of Transformer is implemented by changing the middle layer.

Compared with AutoEncoder, in picture 62, the CPU utilization is higher, but still not as good as standalone version. Because of the need to network and cross server scheduling, it’s understandable that the CPU utilization can’t compared to standalone version. The situation for network is similar, but it seems that the changing frequency is higher than AutoEncoder.

Therefore, the situation of Transformer algorithm for network blocked is better than AutoEncoder. So, the CPU utilization of Transformer is higher.

4. CONCLUSION AND FUTURE WORK

4.1 Conclusion

Through above analysis, it's obvious to have the preliminary conclusion. First of all, when changing the size of the data amount, the CPU utilization will not increase. As for some specific algorithm, it may prolong its running time and change the network utilization. Hence, the major consideration is focusing on different algorithms. When the model size changes from time to time, it may generate higher pressure to CPU and network. When make a division to the cluster, network transmission will become the bottleneck of the performance. Consequently, it's necessary to reduce the cluster network transmission among each node as far as possible. In the case of the CPU resources are rich, it's better to distribute the PS and Worker together as much as possible. In this way, it can enhance the cluster computation efficiency and CPU using efficiency.

4.2 Future work

We don't have the analysis of system memory and hard disk IO. Moreover, we don’t have the opportunity to observe the performance of more servers. Since we only have two to three servers and merely modify two algorithms to do the basic test and preliminary analysis. There might be some data error, because we only have the overall performance of the machine without doing single calculation step. It’s better to do more experiments and then it’s easier to make analysis. For example, we can modify more algorithms, make an observation on more data, and obtain test data step by step through calculation process.

5. REFERENCES

[1] Martín, A. Ashish, A. Paul, B. (eds.). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv.org, Distributed, Parallel, and Cluster Computing, Learning (cs.LG).

[2] About TensorFlow. https://www.tensorflow.org/

[3] Autoencoders and Sparsity. http://deeplearning.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity

[4] Distributed TensorFlow Example. https://ischlag.github.io/2016/06/12/async-distributed-tensorflow/