How Much Should We Invest for Network Facility: Quantitative Analysis on Network 'Fatness' and Machine Learning Performance

Duo ZHANG, Mingxi LI
University of Tsukuba
Tsukuba, 305–8577, Japan
chou.2016@aist.go.jp

Yusuke TANIMURA, Hidemoto NAKADA
National Institute of
Advanced Industrial Science and Technology
Tsukuba, 305–8560, Japan
{yusuke.tanimura,hide-nakada}@aist.go.jp,

Abstract

Multi-node execution is becoming more and more popular for machine learning because of it's huge amount of computation. The question we are trying to answer here is that, how should we design computer systems for deep learning, especially in terms of investment for the network. Traditional cluster based 'super-computers' require huge amount of investment on network switches since the network 'fatness' is quite important for the typical applications of super-computers. Do the machine learning workloads share the characteristics with such kinds of applications? To answer this questions, we quantitatively analyze the impact of network fatness on several type of machine learning application types with several network configurations. The results we obtained strongly implies that the network fatness is not important for machine learning applications, and thus we could safely reduce investment on network facilities.

1 Introduction

Cluster computers for machine learning are getting popular, along with recent rapid development of deep learning frameworks that support distributed learning. Typical configuration of these cluster with more than 100 nodes requires tree like hierarchical network structure since single switch cannot handle that much nodes. However, simple hierarchical tree structure is notorious for performance degradation for specific kinds of applications. This problem is widely recognized in the HPC community.

Bisection bandwidth, which is defined as the bandwidth available between two partitions, is one of the measure for network structure in the HPC community. Recently, a network structure called Clos is widely employed as the large scale cluster network. Clos is designed to preserve the bisection bandwidth with small number of network switches, however, it still cost much to keep high bisection bandwidth. The question here is; how much bisection bandwidth do we really need for large scale clusters for machine learning? Do we have to invest network switches as in the HPC community?

To answer this question, we conducted comprehensive simulation study. We tested 2 layered and 3 layered Clos network with several bisection bandwidth setting. For the target application, we assumed data parallel machine learning, which is getting popular these days. In the data parallel machine learning application, each machine learning module exchange gradients to proceed the computation. We assumed 2 exchange methods; centralized server based method and direct exchange method. [1]

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

[1] Part of the result shown in this paper is already published as [1]. This paper is a throughly rewritten version of [2], which is non-peer reviewed article in Japanese domestic workshop.

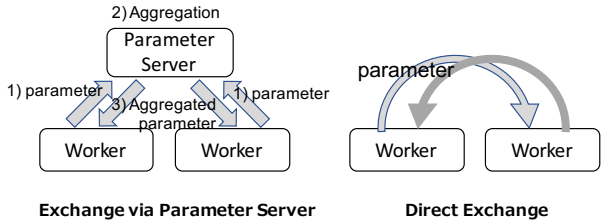

Figure 1: Gradient Exchange Methods.

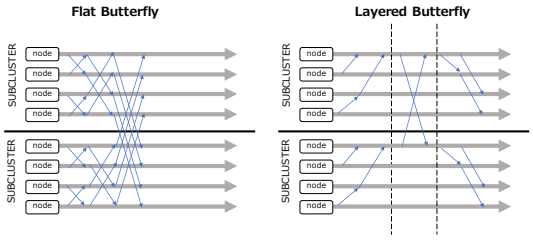

Figure 2: Butterfly Communication Diagram.

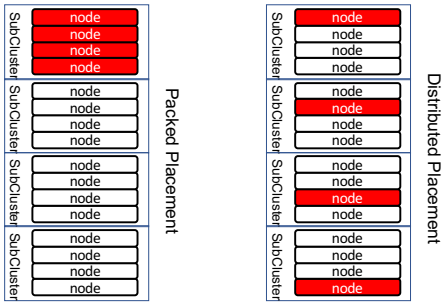

Figure 3: Parameter Server Placements.

In the next section we give the background of the research; data parallel machine learning systems, the simulator we used, and the Clos network. Section 3 describes experimental setup and the result of the experiments. Section 4 gives summary of the paper and the future work.

2 Background

2.1 Parameter Exchange Methods for Large Scale Machine Learning Systems

To parallelize machine learning systems, there are two methods; **Data Parallel** and **Model Parallel**. While data parallel method simultaneously trains multiple machine learning models synchronizing each other, model parallel parallelize inside a single machine learning model. While these two methods are not exclusive each other and often used complementarily, we focus on data parallel in this paper. Data parallel machine learning methods could be categorized into two types; synchronous methods and asynchronous methods; synchronous means all the machine learning models are strictly becomes same periodically, while asynchronous methods allow slight difference among the models. This paper deals with synchronous methods only. To implement data parallel machine learning systems, the are two methods; parameter server based method and direct communication method.

Parameter Server based Method Central server to exchange parameters are often called parameter server [3][4][5]. The left diagram in Figure 1 shows the parameter server based parameter exchange. The workers (machine learning modules) send parameters (or gradients) to the parameter server, the parameter server aggregates the parameter, and send back them to the workers. Often, multiple parameter servers are used to shard the parameters and balance the load; each parameter server is responsible for a subset of parameters.

There are two options for parameter server placement as shown in Figure 3. The left diagram shows the 'packed' placement where parameter servers are concentrated to one or few sub-clusters. The right diagram shows the 'distributed' placement where parameter servers are evenly distributed to all the sub-clusters. Note that the parameter server node in each sub-cluster is selected in round-robin fashion to avoid unnecessary network contention in the upper layer switches.

Direct Exchange Method It is possible to synchronize the models without using central server. by repeating peer-to-peer exchange of parameters [6]. The left diagram in Figure 2 shows the com-

munication with 8 workers. Communication pattern likes this is known as butterfly communication, which is widely used, for example, by the allreduce in MPI[7]. It can exchange information with all the nodes within Log_2N steps of communication where N is the number of workers.

Cluster Aware Direct Exchange Method It is possible to further optimize the butterfly method, given the hierarchical structure. To reduce the inter sub-cluster communication, this method once gather the information inside the sub-clusters to the head nodes of sub-clusters, then perform butterfly among the head nodes of the sub-clusers, and then distribute the exchanged information in each cluster. We call this method **layered butterfly**. The right diagram in Figure 2 shows the layered butterfly method.

This communication pattern requires $log_2n + log_2m + log_2n$ steps where n is the number of nodes per sub-cluster and m is number of sub-clusters. Note that the flat butterfly shown above takes $log_2N = log_2nm = log_2n + log_2m$ steps; therefore the layered method requires log_2n more steps.

2.2 SimGrid: a Distributed Environment Simulator

SimGrid[8][9] is a simulation framework for distributed parallel applications. SimGrid is based on a discrete event simulation; it does not perform any real computation / communication. It just estimates times to perform computation / communication based on given parameters and records events like ' start / end of computation / communication '. The advantage of this type of simulator is that the simulation cost is relatively small. Even with single node computer, SimGrid can handle several thousands of communicating nodes.

To simulate a distributed system in SimGrid, users have to describe platform description and deployment description in XML, and the simulation code in C or C++.

2.3 Cluster Networks Topologies

Bisection bandwidth and 'full'-bisection One of the widely used metrics to evaluate a network is the **Bisection bandwidth**, which is defined as the following; if the network is bisected into two partitions, the bisection bandwidth is the bandwidth available between the two partitions[10]. If the bisection bandwidth of a network equals to the total bandwidth of one half of the nodes, we call the network with '**full-bisection**' bandwidth. We introduce the term **bisection ratio** which is defined as follows.

$$bisection\ ratio = bisection\ bandwidth / total\ bandwidth\ of\ one\ half \quad (1)$$

The bisection ratio of 'full-bisection' network is 1.0.

Clos Network Clos network is a class of network which is originally proposed by Charles Clos in 1953, as a non-blocking network for telephone switching[11]. The core idea is to build a large network using multi-stages of small cross-bar switches.

The term is now used to refer a class of fat-tree network[12], which could be considered as a folded version of the original Clos network. We tested 2 types of Clos networks; 2-layered and 3-layered one.

2-layered Clos is relatively simple[13], as shown in Figure 4. Multiple sub-clusters connected with local-switches are connected by multiple upper layer (right in the figure) root switches to mitigate congestion and improve the bisection bandwidth.

In this configuration, number of ports of the switches determines the maximum number of sub-clusters we can have. With n-node switch, n sub-clusters are the maximum, since all the higher layer switches have to be connected with the all the lower layer switches. Figure 4 shows the configuration with 8-ports switches. The left diagram shows the 'full-bisection' configuration with 8 port switches and 32 nodes in total. We can configure networks with less bisection bandwidth by reducing the number of upper layer switches.

3-layered Clos is rather complicated[12], as shown in Figure 5. The network is composed of multiple 'pods' which has local 2-layered network structure in them. The pods are connected by multiple root nodes just like 2-layered case. Number of switch ports determines the network structure. Figure 5 shows 3-layered Clos networks with 4 ports switches.

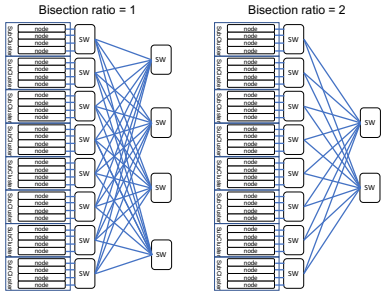

Figure 4: 2 Layered-Clos Network with 8 Port Switches.

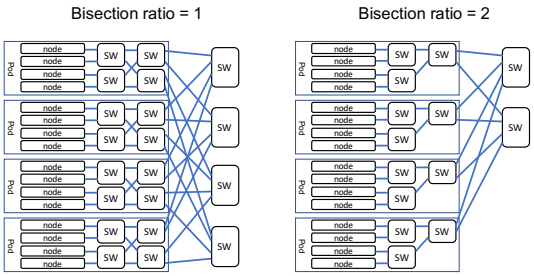

Figure 5: 3 Layered-Clos Network with 4 Port Switches.

Table 1: Comparison of the 2-layered and 3layered Clos networks

Name	#Nodes	#Switches	#Links
2layered-Clos	$k^2/2$	$k/2(2+r)$	$k^2/2(1+r)$
3layered-Clos	$k^3/4$	$k^2/4(2+3r)$	$k^3/4(1+2r)$

With k ports switches, we can have k pods. There are k pods, each pod consists of $(k/2)^2$ servers and 2 layers of $k/2$ k-port switches. Each edge switch connects to $k/2$ servers and $k/2$ aggregation switches. Each aggregation switch connects to $k/2$ edge and $k/2$ root switches. There are $(k/2)^2$ k-port root switches, each root switch has one port connected to each of k pods. The i th port of any root switch is connected to pod i such that consecutive ports in the aggregation layer of each pod switch are connected to root switches on $(k/2)$ strides. In general, a 3-layered Clos network built with k-port switches supports $k^3/4$ hosts.

Comparison of the networks Table 1 shows the comparison of the two networks. r denotes the bisection ratio. Figure 6 shows the required number of switches to construct a network with number of nodes specified in x-axis for bisection ratio 1, $1/2$, $1/4$, and $1/8$. Note that number of switches are normalized with the 2×2 switch equivalent number; assuming k port switches could be implemented by $(k/2)^2$ 2×2 cross bar switches

The network resource requirement for 3-layered Clos network is smaller than 2-layered one. 3-layered Clos is considered to be better topology in terms of resource requirement.

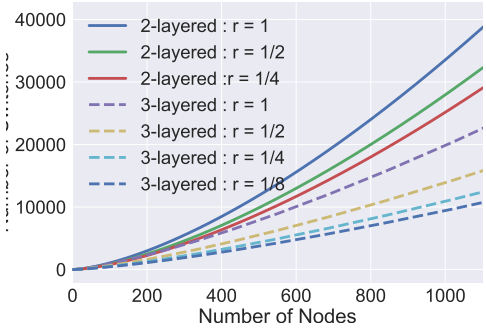

Figure 6: Number of Switches for Number of Nodes.

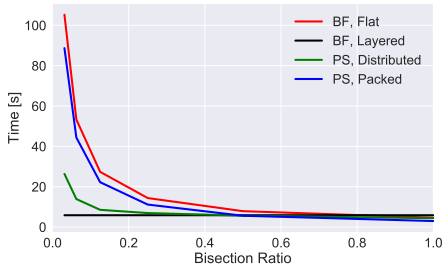

Figure 7: 2-Layered, 2048 nodes, 4GB/s.

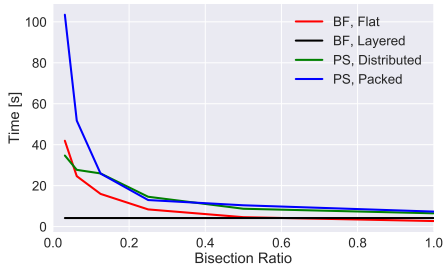

Figure 8: 3-Layered, 1024 nodes, 4GB/s..

3 Experiments

We performed experiments using simulation with the network structure and the parameter exchange methods described below, with SimGrid. We measured the time to perform one gradients exchange, no computation are took into account so that we can focus on the communication cost only. Time for aggregating the gradient are also omitted in the simulation, since the time are relatively small and could be ignored.

3.1 Network Setup

We have setup clusters with the two network topologies; 128, 512, and 2048 nodes for 2-layered Clos network, and 256 and 1024 nodes for 3-layered Clos network. We set the bandwidth of links as 4GBytes/s (assuming 40G Infiniband with TCP overhead), and 1GBytes/s (assuming 10G Ethernet with TCP overhead), and the switch latency as 0.2 μs and 1 μs .

3.2 Parameter Exchange Methods

We test one parameter server based method and two butterfly based methods. For the parameter server based method, we assume $1/8$ of whole nodes in the cluster are used for parameter servers while the others are used for workers. We tested two placement strategy for parameter server based method. One is 'packed' and the other is 'distributed', shown in Figure 3.

For the butterfly based methods, all the nodes are used for workers. [2] We test the simple flat-butterfly method with the layered-butterfly method.

In summary, we test four settings; namely, parameter server with packed placement (**PS, packed**) and distributed placement (**PS, distributed**) , flat butterfly (**BF, flat**) , and layered butterfly (**BF, layered**.

3.3 Results and Discussion

Due to space limitation we show some results only here. Note that the results are quite consistent regardless of the size of cluster. Figure 7 shows the result with 2-layered Clos network with 2048 nodes Figure 8 shows the result with 3-layered Clos network with 1024 nodes. The results are for 4GBytes/s bandwidth and 0.2 μs switch latency. The x-axis shows the bisection ratio. The y-axis shows the execution time to perform one gradient exchange.

From this result, it can be seen that the method using the parameter server is inferior to the butterfly network based method in basic performance. This is because the connections to the parameter servers becomes the bottleneck.

[2] Actually, the number of worker nodes is different between butterfly network based method and parameter server based method. Number of worker nodes of parameter server based method is always $1/8$ nodes fewer. However, even if the butterfly network based method reduces n nodes, the execution time is expected to be the same.

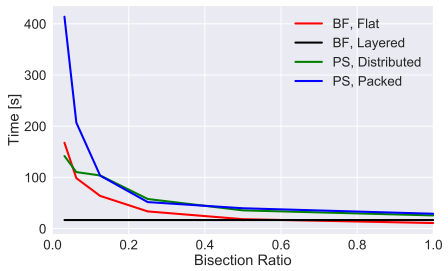

Figure 9: 3-Layered, 1024 nodes, 1GB/s.

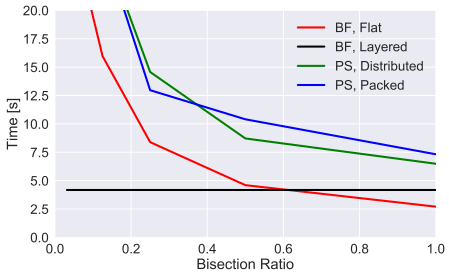

Figure 10: 3-Layered, 1024 nodes, enlarged.

Parameter server method with packed placement setting exhibit significant performance drops as the bisection ratio decreases. On the other hand, with distributed placement setting, the parameter server method is hardly affected by the reduction of the bisection bandwidth. This is because of the network traffic is smoothed throughout the cluster by distributing the parameter server nodes.

Butterfly network based method is faster than parameter server based method, in general. The flat butterfly method tends to be affected by the reduced bisection bandwidth, since it performs inter cluster communication heavily. In contrast, the layered butterfly method is not affected by the bisection bandwidth at all. This result implies that with this method, we do not have to invest in the bisection bandwidth.

While it is difficult to directly compare the Clos and Fattree networks, since it is very difficult to setup them with same nodes, they share same trend in results, as shown in Figure 8 and Figure 7. Given that the Clos requires less network resources, we could conclude that Clos network is more suitable for this particular application.

Figure 9 shows the result with 1GBytes/s network, instead of 4GBytes/s network in Figure 8. Comparing Figure 9 and Figure 8, we could see that network bandwidth linearly affect the gradient exchange speed. This implies that investing faster network technology will be fruitful.

Figure 10 shows a close up of Figure 8. As shown in the figure, the flat butterfly is slightly faster than layered butterfly with full-bisection bandwidth. This is because the flat butterfly requires fewer steps than the layered one, as discussed in 2.1.

4 Conclusion

We have quantitatively evaluated the performance of several parameter exchange method for two network topologies; namely, 2-layered and 3-layered Clos networks, with several bisection-ratio to investigate the proper investment on network for distributed machine learning applications. We have revealed that, 1) Bisection ratio affects some of the parameter exchange methods, but cluster aware direct exchange method does not get affected, 2) Network speed linearly affect the parameter exchange speed, 3) Parameter server based methods are substantially slower than the direct exchange methods, 4) Cluster aware direct exchange method (layered butterfly) outperforms naive exchange method (flat butterfly), except for the case with full-bisection bandwidth.

We conclude that if we employ proper parameter exchange method, we could substantially reduce the investment on the network.

Our future work include the followings:

- Investigate asynchronous gradient exchange setting which will require less network resources.
- Confirm the simulation result with real settings.

Acknowledgement

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO). This work was supported by JSPS KAKENHI Grant Number JP16K00116.

References

[1] Mingxi LI and Yusuke TANIMURA and Hidemoto NAKADA . A quantitative analysis on required network bandwidth for large-scale parallel machine learning. In *International Conference on Machine Learning, Optimization and Big Data*, September 2017.

[2] Duo Zhang, Mingxi Li, Yusuke Tanimura, and Hidemoto Nakada. A study on network structure and parameter exchange method in large-scale cluster for machine learning. In *IEICE technical report, vol. 117, no. 153, CPSY2017-29*, pages 140–150, 2017.

[3] Parameter server: `http://parameterserver.org/`. Accessed: 2015-06-20.

[4] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO, October 2014. USENIX Association.

[5] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc' Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *NIPS 2012: Neural Information Processing Systems*, 2012.

[6] Huasha Zhao and John Canny. Butterfly mixing: Accelerating incremental-update algorithms on clusters. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, 2013.

[7] Rajeev Thakur and W. D. Gropp. Improving the performance of mpi collective communication on switched networks. Technical Report ANL/MCS-P1007-1102, Argonne National Laboratory, 11/2002 2002.

[8] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.

[9] Simgrid: Versatile simulation of distributed systems: `http://simgrid.gforge.inria.fr/index.php`. Accessed: 2016-07-11.

[10] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach (Third ed.).* Morgan Kaufmann Publishers, Inc., 2003.

[11] Charles Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(2):406—-424, 1953.

[12] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[13] Charles E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, Oct. 1985.