Horizontal or Vertical? A Hybrid Approach to Large-Scale Distributed Machine Learning

Short Position Paper

Jinkun Geng, Dan Li and Shuai Wang*
Department of Computer Science and Technology, Tsinghua University
steam1994@163.com,tolidan@tsinghua.edu.cn,s-wang17@tsinghua.edu.cn

ABSTRACT

Data parallelism and model parallelism are two typical parallel modes for distributed machine learning (DML). Traditionally, DML mainly leverages data parallelism, which maintains one model instance for each node and synchronizes the model parameters at the end of every iteration. However, as the model grows larger, communication cost and GPU memory consumption become significant. Data parallelism thus fails to work efficiently in large scale, and model-parallel solutions are proposed in recent years. In this paper, we comprehensively discuss the benefits and drawbacks on both sides. Based on the comparative analysis, we propose Hove, a hybrid approach incorporating data parallelism and model parallelism to balance the overheads and achieve high performance for large-scale DML.

CCS CONCEPTS

• **Networks** → **Network architectures**; • **Computer systems organization** → **Cloud computing**;

KEYWORDS

Data parallelism, model parallelism, hybrid approach, GPU utilization, communication overhead

ACM Reference Format:
Jinkun Geng, Dan Li and Shuai Wang. 2019. Horizontal or Vertical? A Hybrid Approach to Large-Scale Distributed Machine Learning: *Short Position Paper*. In *10th Workshop on Scientific Cloud Computing (ScienceCloud'19), June 25, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3322795.3331461

1 INTRODUCTION

The recent years have witnessed the revolutionary development of artificial intelligence and machine learning (AI/ML) [9], and the AI/ML techniques have been successfully applied in many fields, such as image classification, machine translation, speech recognition, recommender system, and so on. As the training model grows more complex, it is no longer feasible to execute the training with one single machine, and distributed machine learning (DML) becomes the common paradigm in AI/ML field. Reviewing the current practice to execute DML, we divide them into two main categories, namely, data-parallel (vertical) solutions and model-parallel (horizontal) solutions.

As for data-parallel solutions, each node (i.e. worker) maintains one complete model instance, and iteratively refines the model parameters, mostly with stochastic gradient descent (SGD). At the end of every iteration, the nodes need to synchronize their parameters with the others to obtain the global ones. There can be a couple of synchronization algorithms to choose [5], including PS-based synchronization, Mesh-based synchronization, Ring-based synchronization, etc.

As for model-parallel solutions, the training model is partitioned and distributed to different nodes. Each node only maintains one part of the whole model. During every iteration, the model parameters are also refined in a similar way with SGD. However, the nodes do not need to synchronize all the parameters with others, and they can work through a pipe-based [1, 3, 8] or ring-based logic [6].

Traditional DML, especially DNN training, leverages data parallelism due to its simplicity and effectiveness. However, performance bottlenecks and resource under-utilization become more distinct as the model grows large. Figure 1 shows the growth of model size in the past few years, from which we can find that the model size has been increased by an order of magnitude. In order to train such large models efficiently, strong computation power and high communication capability are both necessary. Thanks to the recent development in GPUs and TPUs, the computation power has been improved to a great extent, and there has been 35× growth during the last 5 years [10]. By contrast, the communication capability, though also has made some progress, fails to match the development speed of computation power, thus making the bottleneck more distinct for parameter synchronization.

Considering the drawbacks of data parallelism, the recent pioneering works proposed model-parallel solutions for DML [1, 3, 6, 8], and pipe-based model parallelism is one typical category among them. However, model-parallel solutions also have its inborn deficiencies. Indeed, by partitioning one complete model among multiple nodes, the communication cost can be significantly reduced. However, it increases the load imbalance in DML clusters and causes more serious straggler effect. Besides, the model-parallel solutions can also accumulate the I/O bottleneck on fewer machines, and lead to performance degradation.

*The work was supported by the National Key Basic Research Program of China (973 program) under Grant 2014CB347800, National Key Research and Development Program of China under Grant 2016YFB1000200 and the National Natural Science Foundation of China under Grant No. 61522205, No. 61772305, No. 61432002. Dan Li is the corresponding author of this paper.

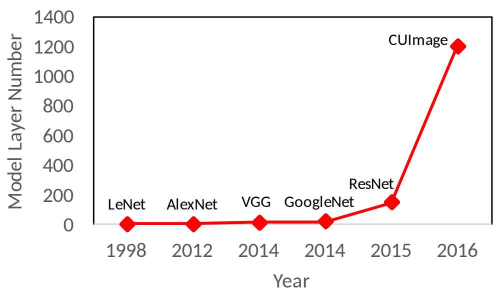

Figure 1: Growth of Training Model Size

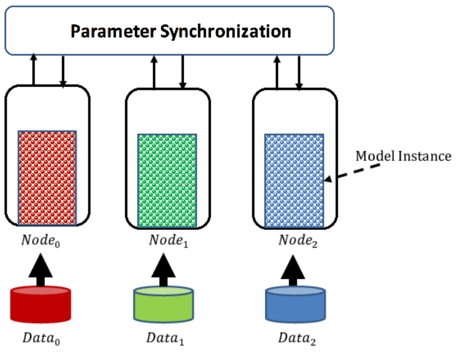

Figure 2: Data-Parallel (Vertical) Solution

We believe that data parallelism and model parallelism are not mutually exclusive. Based on the comparative analysis between data parallelism and model parallelism, we propose Hove in this paper, which adopts hybrid parallelism for large-scale DML. In summary, we make the following contributions.

(1) We compare the benefits and drawbacks for data-parallel solutions (i.e. vertical solutions) and model-parallel solutions (i.e. horizontal solutions) in this paper. Based on the comparison, we argue that neither of them is the ideal option for large-scale DML.

(2) We propose a hybrid approach to large-scale DML named Hove, which incorporates data-parallel and model-parallel methods to achieve higher performance and better scalability.

(3) Hove is still one of our ongoing projects. In this paper, we summarize the potential challenges of Hove implementation, and point out the future directions of our work.

2 VERTICAL SOLUTION VS. HORIZONTAL SOLUTION

We use DNN training as an illustrative example to compare the two categories of solutions. As depicted in Figure 2 and Figure 3, we can see that data parallelism and model parallelism follow different workflows during the iterative training process. Based on the characteristics of their workflows, we simply name them as *vertical solution* and *horizontal solution* respectively.

2.1 Workflow Patterns

As for the vertical solution depicted in Figure 2, each node holds one complete model instance, and is assigned one shard of training

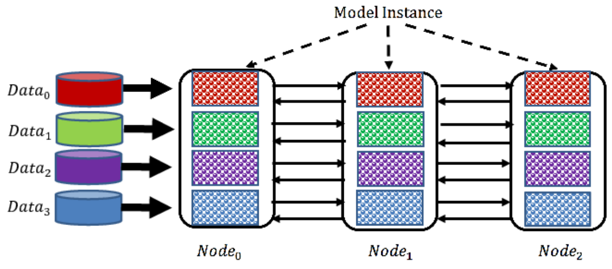

Figure 3: Model-Parallel (Horizontal) Solution

data at the start-up. During each iteration, the node reads the training samples and feeds them to the neural network, then it runs SGD to refine the model parameters. Afterwards, each of them needs to push the model parameters (or the gradients) to the parameter server [1] for synchronization. After the synchronization, they pull the parameters back for new iteration.

As for the horizontal solution depicted in Figure 3, the training process works like a pipe. First, the training model is partitioned and distributed to different nodes, and each node holds part of the model instance (i.e. sub-model). Then, the head node (i.e. the first node) reads the training data and feeds the samples to its sub-model. After the forward pass of the sub-model, the node transmits the boundary parameters to the successor node, so that it can start its forward pass, and so on. Afterwards, the tail node (i.e the last node) finishes the forward pass with its sub-model, and continues its backward propagation. Then, it transmits the boundary parameters to the predecessor node, so that it can also start its backward propagation. Later, the head node also finishes its backward propagation, and one iteration completes. Then, the head node will read another batch of training samples and repeat the workflow. Note that since the model instance has been partitioned among nodes, one sub-model does not cost so much GPU resource. Therefore, there can be multiple sub-models loaded to each node, so that multiple pipe-based workflow can be launched in parallel for better resource utilization.

2.2 Comparison Between Vertical and Horizontal Solution

Comparing the vertical and horizontal solutions, we find that there are advantages and disadvantages for both of them, which we summarize as follows.

Synchronization Cost: As for the vertical solution, every iteration needs to synchronize the model parameters from multiple nodes, thus the communication cost can be significant in large-scale DML. Although some highly efficient synchronization algorithm, such as Ring All-Reduce [11], can be used, the overheads for communication and parameter aggregation still remain large and can badly hurt the overall training performance. By contrast, the horizontal solution suffers from much less overhead for parameter synchronization and aggregation. During every iteration, only one layer of parameters in the boundary needs to be transmitted

[1]Parameter servers can be dedicated for synchronization or can be co-located with other nodes in charge of training.

between two neighbor nodes, which incurs only a small fraction of overhead compared with the vertical solution. For instance, to train a VGG19 model in a 4-node cluster, the vertical solution requires each node to transmit more than 2GB size of parameters during every iteration, whereas the horizontal solution only needs each node to transmit several MBs.

GPU Utilization: Although the GPU performance has been greatly improved, it is non-trivial to fully exploit the computation power of them. As for the vertical solution, the low utilization of GPUs can be attributed to two main aspects. First, the computation in some DML models, such as LSTMs and RNNs, can be only parallelized to a very limited degree because of their sequential structures. Meanwhile, the GPU memory constraint makes it difficult to load many model instances simultaneously. As shown in Figure 2, we suppose each GPU has 4GB memory and each model instance occupies 3GB memory. Then, there is at most 1 model instance held by each GPU, leaving 1GB fragments of GPU memory on each node. Second, the non-computation overheads are significant, thus leading to much idle time for GPUs. As mentioned above, the communication and aggregation cost can be significant for the vertical solution. When the node has completed the parameter (gradient) calculation, it needs more time to wait for the synchronized parameters to continue its next iteration. Although some overlaps can be integrated throughout the training process, the synchronization time is so large that it cannot be completely hidden behind the computation. Therefore, GPUs cannot be well exploited. Compared with the vertical solution, the horizontal solution possesses distinct advantages in the two aspects. First, the training model can be partitioned across multiple GPUs, thus the horizontal solution can avoid the fragments of GPU memory and simultaneously train more instances. As shown in Figure 3, each model is partitioned into 3 parts and we assume they have equal sizes. Then each sub-model only occupies 1GB memory, so one node can hold 4 sub-models and there can be 4 model instances jointly trained in the 3-node cluster. But for the vertical solution, there can only be 3 model instances trained simultaneously. Second, the synchronization cost for the horizontal solution is much less than that of the vertical horizontal solution. Meanwhile, the pipe-based workflow provides better overlap between iterations, and can further accelerate the training process for horizontal solutions [1]. Given the equal amount of time, the horizontal solution suffers from less idle time for GPU, so that it obtains a higher utilization of GPU power.

Load Balance: The horizontal solution needs to partition the model among different nodes. However, each part of the model incurs different amount of computation workload and communication workload. Previous works [1, 8] have proved that different model partitions can lead to a distinct variance of training performance, and it remains as a challenging issue to keep load balance among the nodes for the horizontal solutions. As for the vertical solution, however, each node holds one complete model instance with the same size. The workload for each node, though cannot be exactly the same, is approximately equal. Given the same resource (e.g. computation power and communication bandwidth) configuration for each node, the load balance can be better guaranteed.

Straggler Mitigation: Large-scale DML is usually executed in cloud computing environment, where heterogeneity is unavoidable and the performance of each node shows time-varying characteristics, especially for preemptive tasks. Because of this, the straggler effect can be very distinct in the dynamic cloud environment. As for the vertical solution, since each worker undertakes nearly equal workload, those with lower performance can fall far behind others, thus drag down the overall performance. Although workload stealing or reassignment have been proposed [2, 4, 7], such solutions can cause significant migration cost of training data. Comparing with the vertical solution, the horizontal solution enjoys more flexibility for straggler mitigation, because it can rebalance the workload at much lower cost. Instead of migrating the **training data** (usually TBs), the horizontal solution migrates the **training model** among nodes, thus leading to economic straggler mitigation.

I/O Scalability: As shown in Figure 3, the horizontal solution concentrates the I/O operation on the head node. As the DML cluster grows to a large size, the head node needs to read many batches of training data, so that it can feed these batches to different pipes in parallel. Considering that the I/O operations are mainly scheduled by the CPU cores on the head node, the limit number of CPU cores (usually 16~64) will constrain the I/O speed and further prolong the makespan for each iteration. On the contrary, the vertical solution suffers from little I/O bottleneck, because each node only feeds the training data to its own model instance, and the CPU cores on each node undertakes less burden for I/O operations.

Based on the above analysis, we summarize the comparison results as Table 1.

Table 1: Comparison Summary

	Vertical Solution	Horizontal Solution
Synchronization Cost	N	Y
GPU Utilization	N	Y
Load Balance	Y	N
Straggler Mitigation	N	Y
I/O Scalability	Y	N

2.3 Our Motivation

As shown in Table 1, both vertical solution and horizontal solution have their advantages, as well as drawbacks in different dimensions. Towards large-scale DML, it is not desirable to solely depend on either of them. Therefore, we propose Hove, a hybrid parallel training method which combines horizontal and vertical solutions, and trade off the costs on aforementioned dimensions. Generally speaking, Hove enjoys the following advantages.

(1) Hove divides the cluster nodes into small groups. Within each group, horizontal training is adopted to maximize the number of model instances and fully utilize the GPU power. Across different groups, vertical training is adopted to maintain good I/O scalability.

(2) Hove executes intra-group aggregation and inter-group synchronization. The intra-group aggregation can largely reduce the data amount to synchronize, whereas the inter-group synchronization can reduce the scale and improve the synchronization efficiency.

(3) Hove involves auto-tuning mechanism during runtime, which can dynamically re-distribute the workload on each node, so that the straggler effect can be effectively mitigated.

3 ARCHITECTURE AND WORKFLOW

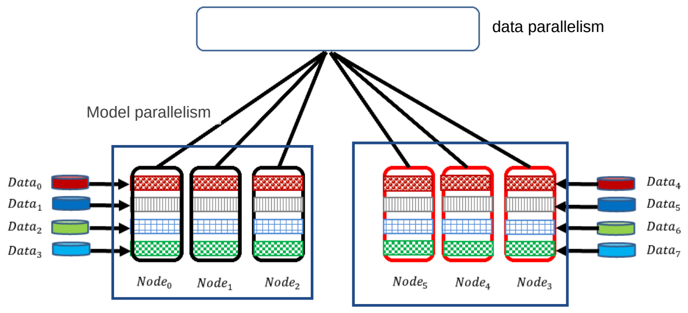

Figure 4: Architecture of Single Node in Hove

The architecture of Hove can be illustrated as Figure 4. We use the example to describe the workflow of Hove.

Group Partition: With a large number of nodes, we first need to find a proper partition and organize these nodes into several groups. As shown in Figure 4, considering there are 6 nodes in the cluster, we first partition them into 2 groups, with each group containing 3 nodes. The group partition aims to maximize the GPU utilization but avoid the I/O bottleneck.

Group-Based Horizontal Training: Within each group, the 3 nodes leverages horizontal training to load more model instances and fully leverage the GPU power. Meanwhile, the I/O bottleneck should be avoided. In Figure 4, there are 4 sub-models loaded on each node, so that there are 4 model instances training simultaneously in each group.

Intra-Group Aggregation: After the horizontal training, the fresh model parameters are calculated and need to be synchronized with other nodes across groups.However, before the synchronization, the parameters within the group are aggregated on each node. For example, the parameters from the four sub-models on $Node_0$ can be aggregated as one replica to reduce the communication cost across groups.

Inter-Group Synchronization: After the aggregation, each node synchronizes the model parameters across groups. Note that the synchronization scale is much smaller than the total number of nodes, because each node only synchronizes the parameters with the nodes which hold the same sub-models in other groups. For instance, $Node_0$ only needs to synchronize parameters with $Node_3$, and $Node_1$ with $Node_4$, and $Node_2$ with $Node_5$. In this way, the inter-group synchronization can be better executed with more parallelism.

Dynamic Workload Tuning: Under the dynamic computing environment, when certain nodes suffer low performance and fall behind others, Hove is able to tune the workload by migrating the sub-model along the horizontal pipe to its neighbor nodes, so that a better load balance can be achieved. For example, when $Node_1$ shows distinct straggler effect and the performance is much lower than other nodes, the sub-models can be further partitioned and migrated to $Node_0$ and $Node_2$ for re-balance. Besides, inter-group migration is also possible, and $Node_1$ can also migrate its sub-models to $Node_4$ and even further cascade to $Node_3$ and $Node_5$, if the intra-group tuning fails to suffice.

In summary, Hove adopts the group-based workflow. Within each group, the nodes leverages the horizontal training to better utilize the GPU resource and reduce inter-node communication. On the other hand, by considering each group as one blackbox, these groups execute the training vertically, with good load balance and I/O scalability. Moreover, both intra-group tuning and inter-group tuning are supported by Hove, so that it can flexibly re-distribute the workload among nodes, and better mitigate the straggler effect in time-varying cloud environment.

4 FUTURE CHALLENGES

We believe Hove will be a promising solution to large-scale DML. Currently it is still an ongoing project, and there are some major challenges which can be summarized as follows.

Model Partition. Model partition can seriously affect the training performance for the horizontal solution, and we can hardly achieve equal partition among different nodes, because the neural network model varies layer by layer. Therefore, proper performance modeling is required to approximately estimate the potential overheads (either computation or communication) for each sub-model and keep load balance.

Hyper-Parameters of Grouping. With a total number of nodes, the grouping needs to take into consideration a couple of hyper-parameters, such as the number of groups, the number of sub-models on each node, the frequency for parameter aggregation and synchronization. The choice of the hyper-parameters is closely related to the scale of the cluster, as well as the characteristics of the training model.

Heterogeneous Horizontal Training. Different sub-models incur computation and communication overheads to different degrees. Therefore, it is a promising option to assign different number of sub-models for each node, considering the heterogeneity of sub-models loaded on each node.

REFERENCES

[1] Harlap Aaron, Narayanan Deepak, Amar Phanishayee, and et al. 2018. PipeDream: Pipeline Parallelism for DNN Training. In *Proceedings of SysML'18*.
[2] Umut A. Acar, Arthur Chargueraud, and Mike Rainey. 2013. Scheduling Parallel Programs by Work Stealing with Private Deques. In *Proceedings of PPoPP'13*.
[3] Chi-Chung Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. 2018. Efficient and Robust Parallel DNN Training through Model Parallelism on Multi-GPU Platform. *arXiv:1809.02839* (2018).
[4] J. Dinan, D. B. Larkins, P. Sadayappan, et al. 2009. Scalable work stealing. In *Proceedings of SC'09*.
[5] Jinkun Geng, Dan Li, Yang Cheng, et al. 2018. HiPS: Hierarchical Parameter Synchronization in Large-Scale Distributed Machine Learning. In *Proceedings of NetAI'18*.
[6] Jinkun Geng, Dan Li, and Shuai Wang. 2019. Rima: An RDMA-Accelerated Model-Parallerized solution to Large-Scale Matrix Factorization. In *Proceedings of ICDE'19*.
[7] Aaron Harlap, Henggang Cui, Wei Dai, et al. 2016. Addressing the Straggler Problem for Iterative Convergent Parallel ML. In *Proceedings of the SoCC'16*.
[8] Yanping Huang, Yonglong Cheng, Dehao Chen, et al. 2018. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. *arXiv preprint arXiv:1811.06965* (2018).
[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.
[10] Liang Luo, Jacob Nelson, Luis Ceze, et al. 2018. Parameter Hub: A Rack-Scale Parameter Server for Distributed Deep Neural Network Training. In *Proceedings of SoCC'18*.
[11] P Pitch and Y Xin. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel and Distrib. Comput.* 69, 2 (2009), 117 – 124.