Orchestra: Adaptively Accelerating Distributed Deep Learning in Heterogeneous Environments

Haizhou Du*
Shanghai University of Electric Power
Shanghai, China
duhaizhou@shiep.edu.cn

Sheng Huang
Shanghai University of Electric Power
Shanghai, China
huangsheng@mail.shiep.edu.cn

Qiao Xiang
Xiamen University
Xiamen, China
qiaoxiang@xmu.edu.cn

ABSTRACT

The synchronized Local-SGD(Stochastic gradient descent) strategy becomes a more popular in distributed deep learning (DML) since it can effectively reduce the frequency of model communication and ensure global model convergence. However, it works not well and leads to excessive training time in heterogeneous environments due to the difference in workers' performance. Especially, in some data unbalanced scenarios, these differences between workers may aggravate low utilization of resources and eventually lead to stragglers, which seriously hurt the whole training procedure. Existing solutions either suffer from a heterogeneity of computing resources or do not fully address the environment dynamics.

In this paper, we eliminate the negative impacts of dynamic resource constraints issues in heterogeneous DML environments with a novel, adaptive load-balancing framework called Orchestra. The main idea of Orchestra is to improve resource utilization by load balance between worker performance and the unbalance of data volume. Additionally, one of Orchestra's strongest features is the number of local updates adaptation at each epoch per worker. To achieve this improvement, we propose a distributed deep reinforcement learning-driven algorithm for per-worker to dynamically determine the number of local updates adaptation and training data volume, subject to mini-batch cost time and resource constraints at each epoch. Our design significantly improves the convergence speed of the model in DML compared with other state-of-the-art.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Distributed computing methodologies**.

KEYWORDS

Distributed Deep Learning, Local Update Adaptation, Load-Balance, Heterogeneous Environments.

ACM Reference Format:
Haizhou Du, Sheng Huang, and Qiao Xiang. 2022. Orchestra: Adaptively Accelerating Distributed Deep Learning in Heterogeneous Environments. In *19th ACM International Conference on Computing Frontiers (CF'22), May 17–19, 2022, Torino, Italy.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3528416.3530246

1 INTRODUCTION

Distributed deep learning has attained profound advances in recent years. Researchers and engineers have applied distributed deep learning technologies to solve their problems in various models including ImageNet, Bert, etc. Distributed deep learning becomes very common to reduce the overall training time by exploiting multiple computing heterogeneous workers (*e.g.*, CPUs/GPUs/TPUs) with the large size of the training datasets in data centers or non-dedicated cloud clusters. However, most clusters in the real world include GPUs and CPUs of different eras and types with different computing capabilities. Unfortunately, slow workers can easily become stragglers, lead to low utilization of computing resources and reduce the model efficiency in heterogeneous environments. Extremely, it will cause the failure of the whole training process.

To cope with the above difficulties, some works have been extensively studied in the literature in recent years. For example, [2, 8] proposed to drop redundant nodes to reduce unexpected waiting time at the end of per iteration. However, these methods lead to more waste of resources and require more iterations to satisfy model convergence. Some works [5, 11] try to eliminate stragglers from the perspective of gradient coding.

However, these methods require many redundancy factors to recover the correct gradient from potential stragglers. Therefore, they lead to much additional overhead in real data centers with resource constraints.

In this paper, we propose a novel distributed deep learning framework based on the idea of load balance. Our goal is to mitigate the impact of stragglers by relaxing the strong synchronization requirements of the traditional Bulk Synchronous Parallel(BSP) scheme and balancing the workload of heterogeneous workers. The basic idea is to make slower workers do fewer local updates before gobble synchronization, and faster workers do more local updates. Using the deep reinforcement learning technique, we reduce the waiting time to minimal per iteration.

This paper makes the following **main contributions**:

(1) We propose a novel load-balancing-based framework named Orchestra, that improves the resource utilization and reduces the whole training time of distributed deep learning by adaptively determining the number of local updates and data partition according to the different performance of works. To the best of our knowledge, we are the first to propose such a learning-driven distributed deep learning framework.
(2) We explore the problem of low utilization of computing resources in heterogeneous clusters and propose a new synchronization mechanism named Load-aware adaptive synchronous parallel. In distributed training, balancing the load

of heterogeneous nodes can improve resource utilization and speed up distributed training.

(3) We implemented this framework through three components, which are the Load Balance Controller, Data Partition Controller, and Barrier Controller. Furthermore, we use the Asynchronous Advantage Actor-Critic (A3C) learning algorithm to adaptively optimize the synchronization barrier to reduce communication. Our experiments show that Orchestra can reduce the end-to-end training time by 63.63% compared to the state-of-the-art.

2 RELATED WORKS

Recent schemes for scaling training to a large number of workers rely on standard mini-batch SGD with very large overall batch sizes [7, 18] , i.e. increasing the global batch size linearly with the number of workers K. [19] has shown that remarkably, with the exponentially growing mini-batch size it is possible to achieve linear speed up (i.e., error of $O(1/KT)$) with only $\log T$ iterations of the algorithm, and thereby, when implemented in a distributed setting, this corresponds to $\log T$ rounds of communication. The result of [19] implies that SGD with exponentially increasing batch sizes has a similar convergence behavior as the full-fledged (non-stochastic) gradient descent. While the algorithm of [19] provides a way of reducing communication in a distributed setting, for a large number of iterations, their algorithm will require large minibatches, and washes away the computational benefits of the stochastic gradient descent algorithm over its deterministic counterpart. Furthermore, it has been found that increasing the mini-batch size often leads to increasing generalization errors, which limits their distributivity [14].

Motivated to better balance the available system resources (computation vs. communication), local SGD (a.k.a. local-update SGD, parallel SGD, or federated averaging) has recently attracted increased research interest [15, 16, 20, 21]. In local SGD, each worker evolves a local model by performing H sequential SGD updates with mini-batch size B, before communication (synchronization by averaging) among the workers.

Although existing works provide convergence guarantees on local-update SGD, there is still no effort to focus on optimally tuning local-update SGD to heterogeneous settings. Different from these works, we try to mitigate stragglers and improve the resource utilization of the cluster.

3 THE DESIGN OF ORCHESTRA FRAMEWORK

3.1 Orchestra Design

We have developed a novel load-balancing-based synchronous training strategy, aiming to improve resource utilization and balance communication costs. As Figure 1 shows, the architecture of Orchestra consists of control nodes and training nodes. There are three core components on the Control Server, namely the Load Balancing Controller, the Data Partition Controller, and the Barrier Controller.

3.1.1 Load Balance Controller. To reasonably allocate batch-load in the distributed training procedure, the batch number of each worker should be adjusted dynamically according to its current

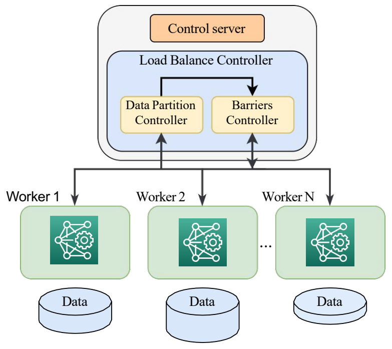


Figure 1: The Architecture Overview of Orchestra

performance. The adjusted batch number was then applied to the training of the next epoch to ensure that workers with different performances completed their tasks as closely as possible. Let each worker keep the computation state as much as possible instead of the idle state. To minimize the waiting time and improve cluster utilization, we propose an efficient algorithm based on the principle of least common multiple. The algorithm describes as follows.

(1) Find the max value in t_b^n and assign it to T.
(2) Get the max idle time t_W in $mod(T, t_i^b)$.
(3) let T increase and repeat the above two steps until the distance of the local step does not satisfy the constraint M.

where $t_b^{n,i}$ denotes the time cost by worker n finish a batch-size and T denotes the barrier time. Finally, among t_W, the T used by the minimum value is the suboptimal T^*. Next, it needs to be input into Barrier Controller for final tuning. The complexity of the algorithm is $O(MN)$ which would not bring additional overhead to the original training system.

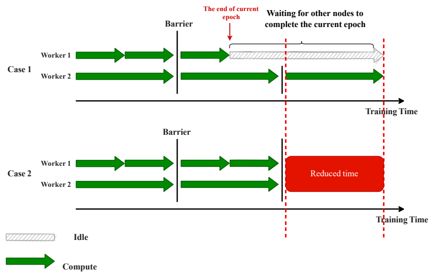


Figure 2: Two cases of dataset partition.

3.1.2 Data Partition Controller. As shown in case 1 in Figure 2, in the first iteration, each worker can complete batches almost simultaneously. Nevertheless, at the last local iteration, the fastest worker completed the training very early, while the slowest worker needed to iterate a few batches. Therefore, we need to balance the overall training dataset. It improves the local and global resource utilization and shortens the training time of each epoch. Moreover, it also can reduce the training time.

The dynamic partition of the datasets is approximate to the ratio of each worker's batch number which represents the number of local updates per iteration. Firstly, we collect the computing speed of each worker by performance monitor, that the speed is represented by $batches/second$. And then we normalize it to get the percentage of speed that is the final data partition.

3.1.3 Barrier Controller. Although the waiting time can be minimized by the load balancing controller, if the number of local updates of each worker is too small, then each epoch would take a lot of time to communicate. Therefore, the performance of Orchestra will degrade to Synchronous Distributed SGD. To address this problem, we use a learning-driven technique to further optimize the barrier and reduce the communication frequency.

From the above analysis, it is obvious that the barrier control problem is a mixed-integer programming problem, which is in general NP-hard and it is difficult to find the optimal value manually or heuristic algorithms. As such, we utilize the recent advances in machine learning to design a barrier controller consisting of a DRL-based algorithm to adaptively determines the synchronization barrier. Consider that Distributed A3C [6] has the properties of multiple agents, which can better interact with heterogeneous nodes and agents can explore the different environments of different workers. Therefore we choose Distributed A3C as the key technology to solve the barrier control problem.

Here are three key features to implement the Distributed A3C model.

(1) **State Space.** The state S is a triple composed of three variables in Orchestra. (1) current training progress, including the global synchronous index and the local iteration index of each worker, (2) current training accuracy or training loss, (3) cost time of waiting for other workers.
(2) **Action Space.** This action determines the synchronization barrier for each local iteration, To be specific, we adjust the value of T. The agent will try to adjust this value at the beginning of each epoch. There are three kinds of actions we can take: increase, decrease, and no change. The action in each step is to adjust the value of T by +1 or -1 or 0.
(3) **Reward.** The reward can be obtained after the completion of an epoch. We use the ratio of incremental accuracy as the metric. The incremental gradients ∇acc can be obtained by, and the increment rate p can be expressed as $p = \frac{\nabla acc}{time} * \sqrt{train_{step}}$, Note $\sqrt{train_{step}}$, the function of this factor is to balance the non-linear change of accuracy. Finally, the $ratio$ can be obtained by comparing it to the best value p^* in the historical record, $ratio = \frac{p}{p^*}$. Therefore the reward can be expressed as:

$$Reward = \begin{cases} -1, & (ratio < 1). \\ 1, & (ratio > 1). \\ 0, & (ratio = 1). \end{cases} \tag{1}$$

We obtain the preliminary synchronization barrier through the Load Balancing Controller, and finally, input it to the Barrier Controller for tuning. The final output synchronization barrier is the optimal synchronization barrier. After obtain the optimal T^*, we can calculate the number of local iterations of each worker before the synchronization parameters by $\tau_n = T^*/t_b^n$. The number of local update of each worker is expressed as: $\tau_1, \tau_2 ... \tau_N$. So Finally, the global model update rule is:

$$x_{t+1}^i = \begin{cases} \frac{1}{N}\sum_{k=1}^{N}(x_t^k - \eta g(x_t^k)), & t \ mod \ \tau_i = 0. \\ x_t^i - \eta g(x_t^i), & \text{otherwise}, \end{cases} \tag{2}$$

where x_t^i denotes the model parameters in the i-th worker.

4 EXPERIMENTAL EVALUATION

4.1 Experimental Environment

We conduct our experiments on a heterogeneous cluster as Table 1. Each server owns the 3.9 GHz Intel Xeon Gold 6250 CPU with multiple cores, 256GB RAM, and the Intel gigabit network interface controller. As shown in Table 1, each cluster is configured with a different number and performance of GPUs. We use the PyTorch framework to build Orchestra.

Table 1: Clusters With Different Worker Configurations

GPU Clusters	Cluster A	Cluster B	Cluster C
Server 1	1*2080Ti	1*2080Ti	4*2080Ti
Server 2	1*T4	3*T4	2*T4
Server 3	1*T4	2*T4	8*T4
Server 4	1*T4	2*T4	2*T4

4.2 Datasets and DML Models

We used CIFAR10 [12] dataset for image classification tasks and CRMC2018 [3] dataset for Natural Language Process task. CIFAR10 datatset has 50,000 training images and 10,000 test images. CMRC2018 is a dataset for Chinese Machine Reading Comprehension. We choose ResNet101 [9] and DenseNet121 [10] as image classification task and ALBERT [13] as NLP task to evaluate Orchestra framework.

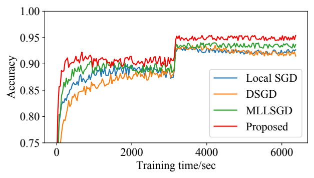
4.3 Baselines

To illustrate the effectiveness of Orchestra, we compare it with the following baselines.

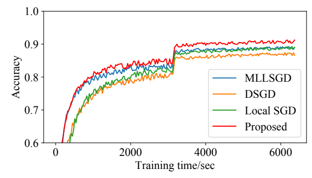
(1) Synchronous Distributed SGD (DSGD) [4]: The naive distributed SGD.
(2) Local SGD [17]: Allow each worker to be synchronized after local updates several times.
(3) Multi-Level Local SGD (MLLSGD) [1]: A distributed gradient method for the heterogeneous multi-level network.

4.4 Results Analysis

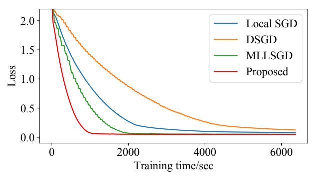
4.4.1 Convergence Speed. As shown in Figure 3, we use cluster C to compare the convergence speed of these three methods using the Cifar10 dataset to train ResNet101 and use cluster B to train DenseNet121. We fix the training time to 1 hour, batch size to 512. It is obvious that Orchestra can perform well in the heterogeneous cluster. In contrast, DSGD cannot perform well in the heterogeneous cluster. The reason is most of the training time is wasted due to idle. Although DSGD has the highest convergence accuracy, its convergence speed is very plodding. For the NLP task, we set epoch to 3, batch size to 128, and learning rate to 2e-5. As shown in Table 2, it is observed that the Local SGD-based approach is significantly faster than the traditional SGD. Orchestra is 31% faster than DSGD,



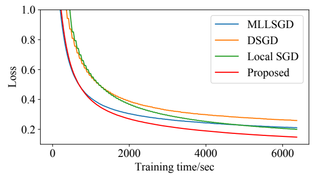
(a) ResNet101 Model



(b) DenseNet121 Model



(c) ResNet101 Model



(d) DenseNet121 Model

Figure 3: Temporal evolution of the accuracy and loss on CIFAR10 dataset with ResNet101 and DenseNet121 model.

Table 2: The comparison of training time on ALBERT model

Approach	Cluster	Accuracy	Loss	Training time(sec)
SGD	1*2080Ti	80.7%	0.93	81123 (22.5H)
DSGD	Cluster A	80.2%	0.97	34514 (9.5H)
Local SGD	Cluster A	78.2%	1.12	32760 (9H)
MLLSGD	Cluster A	78.9%	1.07	27471 (7.6Hour)
Proposed	Cluster A	79.2%	1.02	**23441 (6.6H)**

while the accuracy is slightly reduced due to the inherent loss of information in Local SGD mode. But it can be ignored. Orchestra can effectively improve the utilization rate of computing resources so that more batches can be iterated in unit time. Therefore, higher accuracy can be achieved in the same wall-clock training time.

5 CONCLUSION

To eliminate the negative impacts of computing resource heterogeneity and data unbalance issues in DML, we propose a novel, adaptive load-balancing framework called Orchestra. We propose a distributed learning-driven algorithm for per-worker to dynamically determine the number of local updates adaptation and training data volume, subject to mini-batch cost time and resource constraints at each epoch. The evaluation of various benchmarks and multiple indicators confirms that Orchestra can effectively accelerate by balancing the training load of each heterogeneous worker. Specifically, Orchestra reduces the end-to-end training time by 63.63% compared to the DSGD.

REFERENCES

[1] Timothy Castiglia, Anirban Das, and Stacy Patterson. 2020. Multi-Level Local SGD: Distributed SGD for Heterogeneous Hierarchical Networks. In *International Conference on Learning Representations*.
[2] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981* (2016).
[3] Yiming Cui, Ting Liu, Wanxiang Che, Li Xiao, Zhipeng Chen, Wentao Ma, Shijin Wang, and Guoping Hu. 2019. A Span-Extraction Dataset for Chinese Machine Reading Comprehension. In *International Joint Conference on Natural Language*. Hong Kong, China, 5883–5889.
[4] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. 2012. Large scale distributed deep networks. *Advances in neural information processing systems* 25 (2012), 1223–1231.
[5] Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover. 2016. Short-dot: Computing large linear transforms distributedly using coded short dot products. *Advances In Neural Information Processing Systems* 29 (2016).
[6] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 1406–1415.
[7] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).
[8] S. Gupta, W. Zhang, and F. Wang. 2016. Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE Computer Society, Los Alamitos, CA, USA, 171–180.
[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[10] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. 2014. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869* (2014).
[11] Can Karakus, Yifan Sun, and Suhas Diggavi. 2017. Encoded distributed optimization. In *2017 IEEE international symposium on information theory (ISIT)*. IEEE, 2890–2894.
[12] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. *Learning multiple layers of features from tiny images*. Master's thesis. University of Toronto.
[13] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations*.
[14] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. 2014. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 661–670.
[15] Ryan McDonald, Keith Hall, and Gideon Mann. 2010. Distributed training strategies for the structured perceptron. In *Human language technologies: The 2010 annual conference of the North American chapter of the association for computational linguistics*. 456–464.
[16] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*. PMLR, 1273–1282.
[17] Sebastian U Stich. 2018. Local SGD converges fast and communicates little. *arXiv preprint arXiv:1805.09767* (2018).
[18] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. 2018. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*. 1–10.
[19] Hao Yu and Rong Jin. 2019. On the computation and communication complexity of parallel sgd with dynamic batch sizes for stochastic non-convex optimization. In *International Conference on Machine Learning*. PMLR, 7174–7183.
[20] Xiaohui Zhang, Jan Trmal, Daniel Povey, and Sanjeev Khudanpur. 2014. Improving deep neural network acoustic models using generalized maxout networks. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 215–219.
[21] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. 2010. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*. 2595–2603.