A Comparative Analysis of Load Balancing Techniques for Efficient Distributed Machine Learning

Guillermo Marr
Florida State University
Tallahassee, USA
gem21e@fsu.edu

ABSTRACT

Modern machine learning (ML) models require intensive computational power for training, often ruling out single-node training impractical due to prolonged duration's. Nowadays, it is very common to use distributed computing to train ML models. However, most of these distributed computing environments are non-dedicated, heterogeneous hardware environments, which leads to performance discrepancies amongst nodes. The faster nodes have to wait at synchronization points for slower nodes, leading to the well-documented "straggler" problem. To mitigate this, load balancing solutions have been proposed to optimize resource utilization and reduce idle times without introducing significant overhead. In this paper we compare three state-of-the-art load balancing solutions, analyzing their effectiveness of enhancing the overall efficiency of computational resources and mitigating straggler effects in distributed ML training. The comparative analysis of these solutions highlights their efficiencies in speeding up training times of ML models, by balancing load across the cluster, reducing idle times and maximizing throughput.

KEYWORDS

Distributed Deep Learning, Load Balancing

ACM Reference Format:
Guillermo Marr. 2025. A Comparative Analysis of Load Balancing Techniques for Efficient Distributed Machine Learning. In *Proceedings of* . ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

It is common to train machine learning (ML) models on distributed computing clusters. These clusters can either be dedicated or non-dedicated clusters. Non-dedicated is more common in which the cluster is heterogeneous and differs in hardware. This differentiation is what causes the very common "straggler" problem. The straggler problem occurs when the faster nodes in the cluster finish computing their tasks and idle at the synchronization point due to slower nodes tailing behind. The stragglers delay the training speed of models and waste computational resources. ML models utilize distributed computing by typically partitioning the dataset amongst nodes, known as data parallelism. There is also model parallelism in which the whole dataset is trained on different models, but in this paper we focus on data parallelism. Having the data partitioned amongst nodes evenly means nodes with better hardware will wait on the slower capable nodes. Load balancing solutions have been proposed in literature to solve this issue of delays and waste of computational resources. Such solutions involve relaxing the synchronization barriers, work stealing, and other researched topics. These solutions have issues such as a not being able to converge and too much overhead induced in machine learning systems. Ultimately the necessary step in speeding up training time and balancing node clusters, involves addressing the mismatch in workers load and its capability. The question when to load balance is an ongoing research question still to this day. However, solutions have been proposed analyzing effective methods of when to load balance to justify using it, in other words, making sure that the load balancing algorithm itself doesn't cause more overhead than time it saves. In this paper we survey three solutions that mitigate the straggler problem and enhance training times of ML models. These solutions propose semi dynamically load balancing by right-sizing the sample batches at the synchronization barriers, dynamically load balancing the batch-size and data partition after each epoch, dynamically adjusting batch-size off real time performance modeling of GPUs. Each of these solutions effectively speeds up training time and provides a unique solution to the straggler problem. This paper aims to evaluate three state-of-the-art load balancing techniques to determine their effectiveness in reducing idle times and enhancing throughput in distributed ML training

2 TARGETED PROBLEM AND MOTIVATION

The targeted problem that these solutions tackle is the "straggler" problem. There are two types of stragglers, static and dynamic. Static stragglers are caused by difference in hardware capabilities, while dynamic is caused by contention of resources due to tasks sharing a cluster. It has been thought that dynamic stragglers causes more issues, however recent experiments show that static stragglers are the prominent issue in non-dedicated clusters. Solving this issue of stragglers in distributed computing clusters especially HPC environments is crucial to maximizing computer resources and throughput. Mitigating the straggler and evenly distributing load amongst nodes allows for minimal idle times and faster training times. By solving the straggler issue in HPC environments, research and ML models in companies can be deployed faster. Ensuring data is balanced amongst nodes is crucial as AI grows more computationally expensive as well as data for these models.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.1 Problems in Solving

The reason the straggler problem is hard to solve is due to the many factors involved. Determining when to load balance is an ongoing research problem that has not been solved. The solutions surveyed in this paper all have their issues. One issue includes overhead due to the rapid monitoring of performance in the cluster which may introduce latency. Others include not responding fast enough when rapid changes occur during iterations or during an epoch. Monitoring the performance of a node during iteration time may make the load balancing itself useless if trumped by the overhead of the monitoring. When calculating the new partition of batch size and data during iterations, one cannot do this every iteration because the cost is greater than the performance benefit. Finding the optimal time to load balance can be very challenging due to various factors of different hardware, cost vs performance benefit, and complexities.

2.2 Related Works

We will briefly cover algorithms used for synchronous distributed training. A common algorithm SSGD is used to effectively train the model in parallel by evenly distributing the data in the dataset amongst the nodes. Each node calculates their own local gradient on the data partition they are given and use this to calculate the global gradient. However at each iteration since the global gradient needs to synchronize with all local gradients, the fast workers that calculate their gradients are left idling waiting on the slower nodes. This also delays in calculating global weights. We can demonstrate the consumed training time of the whole DNN with the following equation.

$$Ta = Tgpu + Tw + Ts \tag{1}$$

Where Tw is the waiting time due to idling, Ts is the synchronization time because communication amongst workers, and Tgpu the consumed time of GPU. Obviously speeding up GPU cards with more capable GPU cards is not practicle, but decrease of the Tw and Ts by solving the straggler problem catches much attention. A solution proposed to reduce Tw is Local SGD, this method is utilized as collecting the weights of different workers to calculate the global weights after several iteration, instead of every iteration. This leads to reduction in synchronization overhead by less frequently synchronizing weights. However this leads to bias of accuracy and is suggested that more frequent averaging should be used to improve performance. This method as well introduces another hyperparameter to be optimized (synchronous interval step). Little understanding of averaging frequency affecting the performance of parallel SGD is quite limited in current research and literature.

3 SOLUTION TO THE PROBLEM

Load-Balanced Bulk Synchronous Parallel (LB-BSP), an integrated scheme atop BSP, for efficient distributed learning in non-dedicated clusters. We achieve this by semi dynamically load balancing the batch-size and data during synchronization points based analytically or numerically depending on CPU or GPU cluster. This is done by having the batch sizes be static and alleviating all load balancing strategies during iteration time but during the iteration boundaries dynamically adjusting the batch size as a hyperparameter. The problem formulation for this solution is a optimization problem of finding the worker batch sizes that can minimize the longest batch processing time amongst all workers. In shared CPU clusters problems arise with resource contention and the sample processing speed may vary dynamically. To fix this solution the batch sizes use the NARX model (an extended recurrent neural network), that takes past values of sample processing speed and current and past values of the CPU and memory usage. From this the NARX model is able to predict processing speed for CPUs and converges fast due to the simplicity of the model. This processing speed is used to set the batch size for the worker during the next iteration. In GPU clusters this becomes more difficult. Due to the strong parallel processing capability, GPUs are hard to get real time performance measurements because they incur a lot of overhead. Instead of monitoring the usage memory and previous processing speeds, we identify a leader and a straggler GPU. If the leader has consistency preceded the straggler during a observed window, the leader takes on more load and the straggler alleviates part of its load. If the stragglers load is to go below 0 the training process should be reset without this worker. When aggregating the gradients of the cluster, the solution keeps in mind that different workers have greater or smaller batch-sizes. The batch-size is used as a weight for the local gradient in determining how much it contributes to the global gradient. The next solution analyzed is DLB (Dynamic Load Balancing) in distributed deep training of DNNs. The proposed DLB is a novel approach to tackling the straggler problem. This solution calculates the batch-size and data partition after each epoch. The first epoch the data is evenly partitioned amongst the cluster and evaluated at the next. After the first epoch, the next batch-size for each worker i is calculated based off its previous epochs performance. The formula for calculating the performance for worker i during epoch j is given as

$$pi^j = di^j / ti^j \tag{2}$$

this performance determines our batch-size for the next iteration by having

$$bi^{j+1} = pi^j / \sum_{1}^{n} pi^j \tag{3}$$

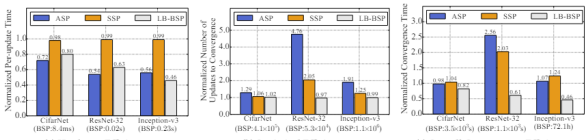
where bi j+1 is the next batch-size for worker i during the next epoch. These batch-sizes sum up to B, the global batch-size amongst all the workers. These batch-sizes need to adjust with the data as well to ensure that all batch-size and data partition sum to 1. To ensure that proper data partition is met, the batch-sizes of the next iteration are used as input for the DynamicDatasetAdjust. During this part of the solution the batch sizes for the next epoch are rounded down and then we subtract this to the total batch size, giving us k = B - b1j+1, b2j+1,... bnj+1, where K is the most values we can round up. We then sort these original values based off their decimal value in descending order, and round the top k most values. This result gives us the data in which the worker i will be portioned. We use the interval [previous worker batch-size percent, batch-size percent + previous worker] to partition our data. Essentially if we had [13.7,16.5,19.6,14.2] we would round this down yielding [13, 16, 19, 14] assuming our batch-size is 64 we get 64 - 13+16+19+14 = 2 now we have [0,2] for which we can round up. Sorting in descending

order off decimals we get .7 >= .6 >= .5 >= .2, we round up the k most values resulting in [14, 16, 20, 14]. We then divide our B by the bi j+1 batch size to get the percent of the batch size the worker is handling. We use this to calculate the data portion intervals [[0,.22], [.22, .47], [.47, .78], [.78,1]], this evenly distributes our batch-size and dataset now for the next epoch. Effectively this calculates and reduces the load of slower nodes and allows for a more distributed load based off their hardware capabilities. The last solution go over is the LBB (Load-Balanced Batching). This solution proposes a two fold strategy, dynamic batch size adjustment, performance based coordination. The LBB analyzes the cluster and sets a starting batch size for each GPU based off its computational capabilities. During training the LBB dynamically changes the batch sizes in real times of these GPUs based on their performance assessments. This solution is effective in addressing static and dynamic stragglers by re adjusting workload based on each GPUs current performance and making sure no GPUs are left idling. The LBB oversees the performance of each GPU and identifies potential stragglers or leaders. Batch sizes are adjusted to the most up to date performance data, guaranteeing even distribution amongst the GPUs is met, minimizing bottlenecks and maximizing throughput. The process involves, pre-training analysis, real-time adjustment, performance monitoring, and batch size coordination. These methods effectively tackle the straggler problem very dynamically during each iteration rather than during iterations or after epochs.

4 MAIN CONTRIBUTIONS

These solutions bring forth many state-of-the-art algorithms that handle the straggler problem. Each of these solutions uniquely contributes new findings to further explore and expand. The semi-dynamic load balancing solution brings forth adaptability to batch-sizes at iteration boundaries rather than during computations themselves. This solution combines static planning with dynamic adjustment, as well as reduction in overhead due to adjustment points being limited and less frequent. This solution is effective in non-dedicated clusters and managing computational resources effectively. The DLB solution for deep learning adjusts batch sizes after epochs and effectively uses past performance data to adjust batch-sizes. This solution provides stable adjustments in the load of workers by getting thorough analysis in previous performance rather than more frequent less thorough analysis. This solution has been shown to perform well even when disturbed by tasks sharing the clusters resources. The last solution LBB is a GPU specific optimization and tailored for non-dedicated GPU clusters. This solution provides optimizations by analyzing current performance and adjusting batch-sizes in real time. This solution is both effective in static and dynamic straggler problems, by adjusting to ongoing performance differences and resource contentions amongst the GPUs.

5 EVALUATION RESULTS



(a) Hardware Efficiency (b) Statistical Efficiency (c) Overall Convergence Efficiency

Figure 6: Training efficiency under different worker coordination schemes in Cluster-A.

The evaluation results from the semi-dynamic load balancing solution show that it outperforms other strategies such as BSP, SSP in updating the gradient. Even compared to ASP, which is an asynchronous approach that lets faster nodes go further without waiting, the LB-BSP outperforms in one instance. The statistical efficiency, measured as the number of updates required to reach the target accuracy. We see that LB-BSP outperforms other methods and its overall convergence efficiency is up to 54% better than the second best in some instances.

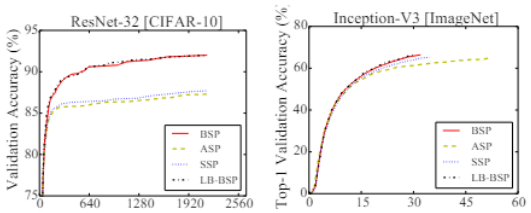
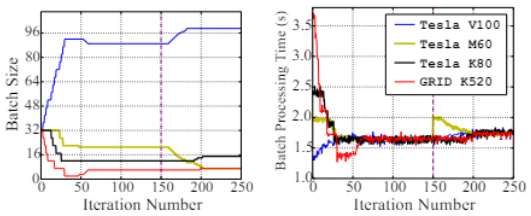
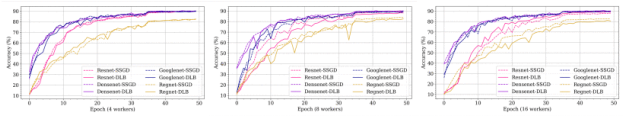


Figure 7: Test accuracy against training epochs.

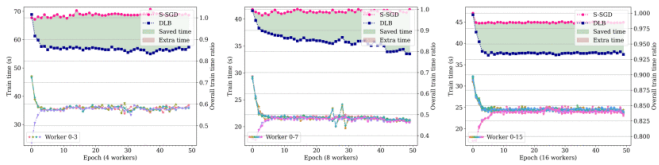


The test accuracy against training epochs as well demonstrates that LB-BSP is not only superior in convergence efficiency, it also maintains high accuracy. The other graph shows LB-BSP effectively adapts batch sizes based on their capabilities showing the Tesla V100 handling significantly more than the other workers, as well the batch processing time enters equilibrium after the first couple of iterations. These experiments were done on Amazon EC2 and clearly show evidence that this solution can effectively eliminate stragglers in non-dedicated clusters, and can speed up convergence by other 50%.

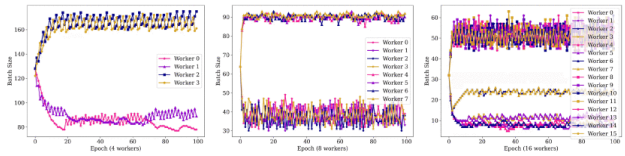


The next solution DLB for distributed training of DNNs evaluation results show that its convergence aligns with SSGD, a state-of-the-art model. This shows that the DLB proposed solution does indeed converge and is sometimes more accurate due to the strongest

worker handling more load while the load in SSGD is static with each worker receiving the same amount.The experimental analysis of testing four DNNs on two datasets shows the same pattern.

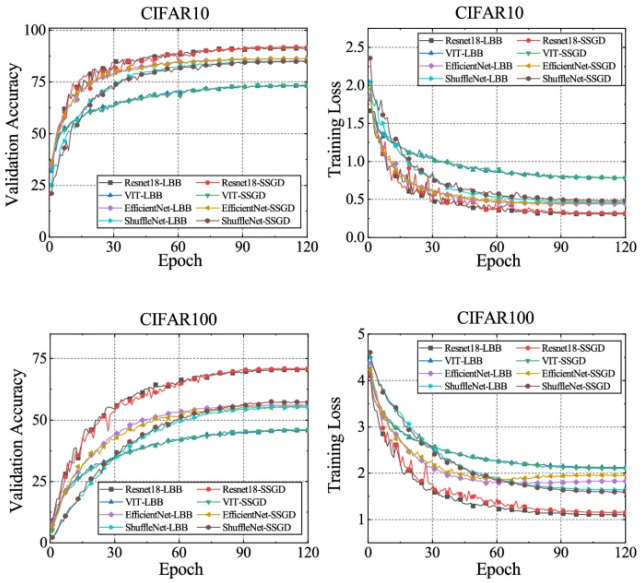


The evaluation above shows the time saved in training due to the DLB solution compared to SSGD. The comparison is between the total time consumed by each epoch and the green between the two represents the time saved. As we can see the DLB is effective in mitigating the straggler issue by dynamically adjusting data and batch-sizes speeding up overall training times. One thing to note is as the cluster grows the saved time is reduced due to cluster maintenance overhead increasing, as well expansion of the cluster reduces the proportion of GPU time in the total training time.



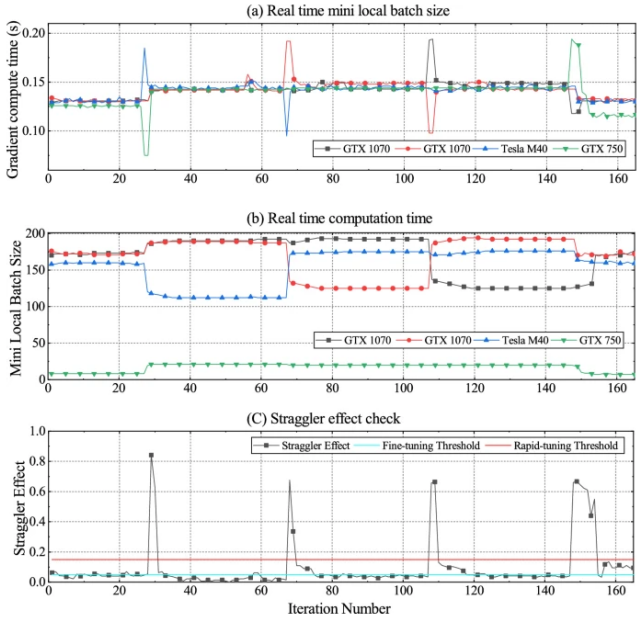
Evaluating the results of the batch sizes during epochs of training, we can clearly see the batch sizes level out accordingly. At first all nodes are given the same batch-size and then they steer off respectively to their computational handling and remain relatively consistent. This shows the solution properly distributes the load and maintains the equilibrium of load handling amongst the cluster.

Fig. 5



The last solution we evaluate the LBB load-balanced batching for efficient distributed learning on heterogeneous GPU clusters. The figure above demonstrates the convergence of the solution in comparison to the SSGD scheme. This evaluation demonstrates that the solution doesn't compromise the excellent convergence of SSGD, and shows a statistical efficiency very similar to SSGD in all eight training tasks.

Fig. 7



This graphic demonstrates how LBB mitigates the straggler effect in distributed deep learning systems. The SE (straggler effect) determines when to balance the load after reaching a certain threshold. The computation of the SE is done on the CPU and doesn't affect the GPU computation leading to little overhead on the training. The

LBB fine tunes the batch-size until the SE falls below the fine-tuning threshold. We can see this in iteration 140 where the SE rises drastically and exceeds rapid tuning threshold due to a disturbing for GTX750. As we can see the LBB continuously monitors SE across the cluster and uses rapid-tuning to coordinate the batch-size of workers. The figure above verifies the mitigation of both static and dynamic stragglers when imposed on the GPU cluster.

6 FUTURE WORKS

These solutions comprise of state-of-the-art methods used to mitigate the straggler problem. However these solutions are not all perfect. Semi-dynamic load balancing could improve its methods for GPU clusters by devising mechanisms that utilize CPU resources to effectively predict load imbalances and adjust the workload distribution similar to LBB. DLB for distributed training could focus on enhancing synchronization mechanisms to reduce synchronization overhead which limits the current solution as the cluster grows larger. The LBB real time processing solution could be optimize the communication between GPU workers as the size of the cluster grows larger as well the communication between CPU and GPUs to calculate the performance and SE.

7 CONCLUSION

In this paper we compare and survey three state-of-the-art load balancing algorithms in non-dedicated clusters that are common in HPC. These solutions focus on eliminating the straggler problem of delayed training time due to slower node and inefficient distribution of computational resources. These solutions effectively mitigate this problem and help speed up training time and converge effectively compared to other proven methods. These solutions pave the way for distributed systems that are heterogeneous and HPC systems to fully utilize their computational resources and effectively train models more efficiently.

8 REFERENCES

Yao, F., Zhang, Z., Ji, Z. et al. LBB: load-balanced batching for efficient distributed learning on heterogeneous GPU cluster. J Supercomput (2024). https://doi.org/10.1007/s11227-023-05886-w

Q. Ye, Y. Zhou, M. Shi, Y. Sun and J. Lv, "DLB: A Dynamic Load Balance Strategy for Distributed Training of Deep Neural Networks," in IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 7, no. 4, pp. 1217-1227, Aug. 2023, doi: 10.1109/TETCI.2022.3220224. keywords: Training;Synchronization;Parallel processing;Load modeling;Heuristic algorithms;Computational modeling;Clustering algorithms;Local SGD;Load balance;Straggler problem;Distributed DNN training,

Chen Chen, Qizhen Weng, Wei Wang, Baochun Li, and Bo Li. 2020. Semi-dynamic load balancing: efficient distributed learning in non-dedicated environments. In Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20). Association for Computing Machinery, New York, NY, USA, 431–446. https://doi.org/10.1145/3419111.3421281

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009