Master's Plan B Project Report

Topic: Exploring Distributed Deep Learning in LAN and WAN environments

Rankyung Hong

hongx293@umn.edu

Contents

1. Introduction

The advent of *Deep Neural Networks (DNNs)* has renewed the inference or prediction records conducted by machine learning techniques in many areas and applications such as image or video classification, speech or text recognition, image-to-text generation. The key to the success is a large amount of data and hidden non-linear multilayers. In the previous machine learning, algorithms called perceptron using single-layer linear classifiers were the main stream to make a classification decision based on the value of a linear combination of the characteristics of objects. In DNNs, however, the paradigm has been changed. Algorithms themselves figure out the characteristics of objects by using the multiple non-linear layers while being fed with the huge volume of data. Lower layers of DNNs closed to the input detect primitive characteristics such as lines of various slopes, colors, whereas higher layers closed to the output perceive high-level characteristics such as shape of objects.

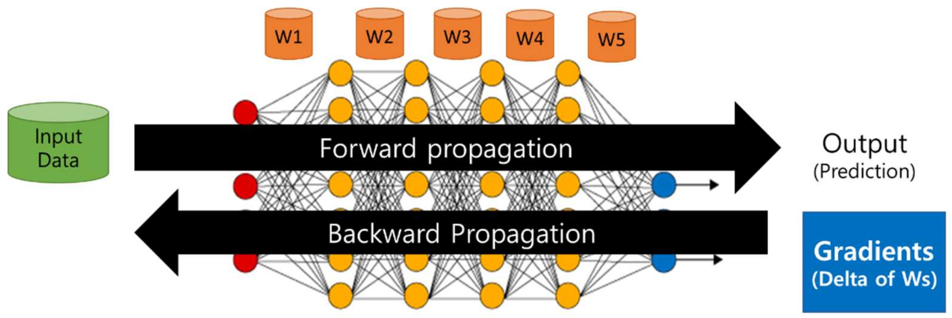


Figure 1. Deep Learning (DL)

Deep Learning (DL) is a learning process to detect the low- and high- level features of object using a DNN and data so as to achieve applications' goal. In figure 1, a DNN model is composed of input layer (vertices in red), hidden layers (vertices in yellow), output layer (vertices in blue), and weights (edges connecting the vertices). The *weights (W)* are the only trainable variables. Their initial values are small random numbers and final values affect the performance (accuracy) of the model. Input can be any data such as pixels of images, words of documents, sound wave of voice. Output can differ from the purpose of application such as prediction of an unseen images, generation of description given an image. Through forward propagation, output is calculated by a series of multiplications of input and the weights. *Gradients* are modification to the values of the weights, the partial derivatives of an objective function given the difference between the output of the model and the ground-truth given by application users. Through backward propagation, gradients are applied to weights by a series of backward multiplication with the weights trained so far to update the previous weights.

DL has two phases – training and inference phases. *Training phase* is to find the best values of weights in all layers to obtain the highest accuracy in object classification through forward and backward propagations. *Inference phase* is to get predictions to unseen data in training based on the weights trained. The predictions are the output through forward propagation.

There are two common statements in deep learning area. First, the larger the amount of training data is, the higher the prediction accuracy of the trained model can have. Seconds, the larger DL model is required to handle the larger data. For example, a DL model can learn *invariant* characteristics of cars when given a large variety of images of cars such as various colors, types of cars, different backgrounds, angles of pictures, etc. Along with it, the number of weights of the model should become larger in order to learn numerous features of the object. As a result, when any unseen images of cars are fed to the trained model, it can result in highly accurate prediction to the images whether they are cars or not. However, it raises a pain point in training phase when a single machine is used to process the large volume of data. Moreover, it may be impossible to store the data in a machine as the data size is growing. Consequently, distributed deep learning is inevitable to handle a large size of data and/or the DL model for better accuracy.

1.1. Distributed Deep Learning (DL)

Multiple machines are used to train a DL model given a large size of data in distributed deep learning. There are three kinds of parallelisms – Data, Model, and Hybrid parallelism. For *data parallelism*, the input data is partitioned and distributed to multiple machines which each machine has an identical whole DL model. For *model parallelism*, the model is partitioned and distributed to multiple machines while each machine processes the same whole data. For *hybrid parallelism*, both the model and data are partitioned and distributed to machines. Since the size of data is relatively much larger than the one of DL models, and it is the main cause of the lack of storage resource, the data parallelism is most frequently used in current deep learning.

1.2. Centralized and Decentralized DL

There are two different ways to update weights of DL models in training phase – Centralized and Decentralized deep learning. Figure 2 illustrates the centralized DL and figure 3 demonstrates the decentralized DL.

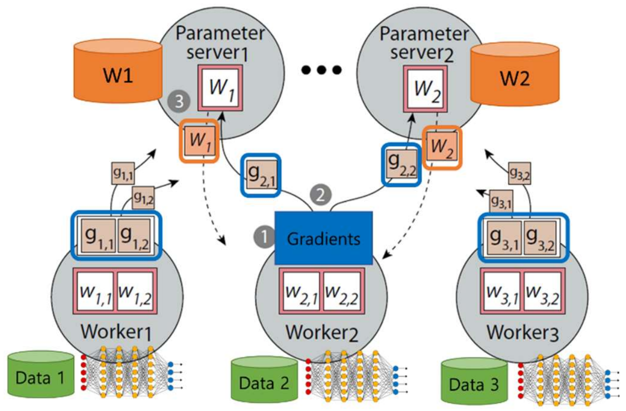


Figure 2. Centralized Deep Learning: Parameter Servers and Workers

In the *centralized DL*, there are central components called parameter servers (PS) to store and update weights. The number of parameter servers can be one to many, which depends on the size of weights of a DL model or policies of the application. Each worker pulls the latest values of the weights from the parameter servers, calculating gradients with the weight values and their data, and then pushing the gradients to the parameter servers. The parameter servers update the weights by applying the gradients collected from all the workers through back propagation.

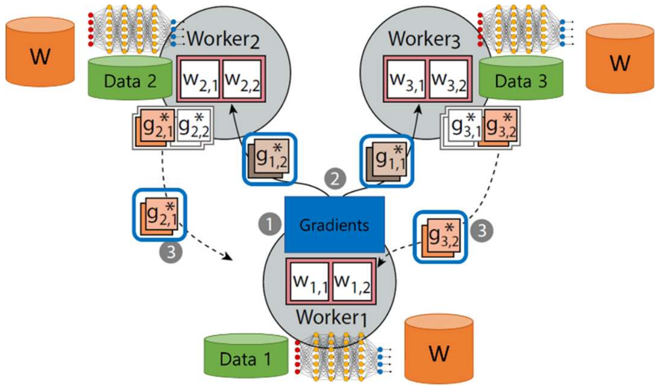


Figure 3. Decentralized Deep Learning: Workers without Parameter Servers

In the *decentralized DL*, there are no central components, parameter servers. Every worker maintains the latest values of the weights by themselves. They do not exchange any weight values from others, yet they update their own weights through gradients of others. The final weights of the workers can differ from each other as training goes since they do not synchronize the weights in any phases in the decentralized DL. Therefore, their accuracies are more susceptible to different initial values of weights and different training speed of individual workers than the ones in the centralized DL. In the other hand, workers sharing parameter servers can have relatively similar weights at the end since they start gradient calculation every step with the same weight values.

1.3. Synchronous and Asynchronous DL

Total training dataset is divided into multiple mini-batches, which are a bundle of data samples shown in figure 4. A *mini-batch* is the data processing unit to calculate gradients of the weights in a single *iteration*. A single round of whole mini-batches processing is called an *epoch*. The most popular optimization method used in recent deep learning is *stochastic gradient descent (SGD)*, also known as incremental gradient descent. The goal in the training phase is to minimize the loss value of object function. During such an optimization process, DL keeps modifying the weights by a series of gradients derived from a mini-batch and the weights in previous iteration.

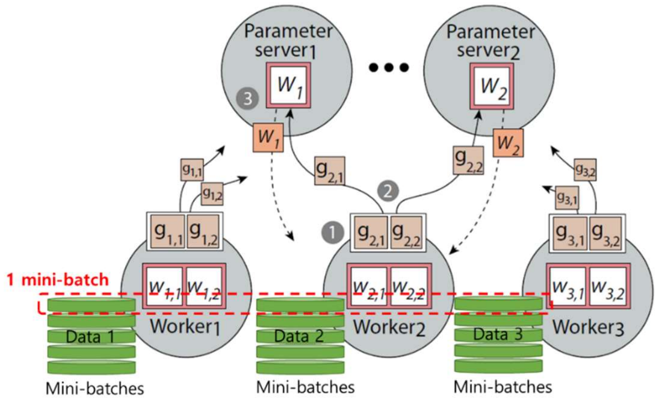


Figure 4. Mini-batch: a data unit for a single gradient calculation

Workers can proceed the weight update iterations either synchronously or asynchronously. In *synchronous DL*, the workers should wait until everybody finishes previous iteration consisting of a cycle of forward and backward propagations. Discrepancies in data processing speed among workers or the presence of stragglers cause longer training time. On the other hand, the *asynchronous DL* are free from the issue since each worker proceed their own iterations no matter of others' progress. However, the larger the progress difference is among workers, the lower the final accuracy of DL model has. Because workers cannot get all gradients from others before starting the next iteration, weight modification information contained in gradients sent by the slower processing workers are unable to apply to the iteration. In worst case, DL models can even be diverged. That is, the loss value of the object function is getting larger and lager as iteration goes. Therefore, recent researches employ *bounded synchronous DL*, allowing workers to advance their iterations within a limit of iteration difference between the slowest worker so as to gain advantages from both sides – higher accuracy from pure synchronous DL and fast training time from pure asynchronous DL.

2. Motivation

Distributed deep learning enables to effectively train DL models with a large dataset. However, their performance is bounded by the network capacity of machines as the number of workers increase.

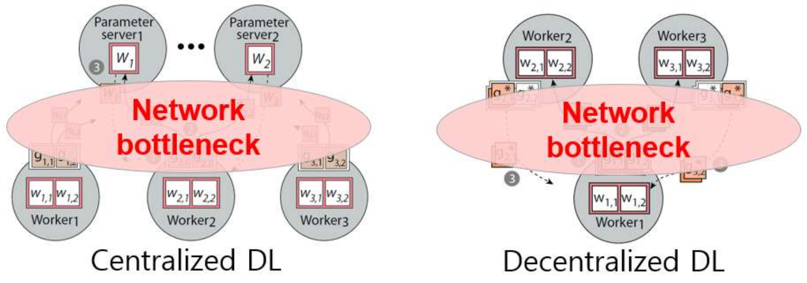


Figure 5: Network bottleneck issues both in distributed centralized and decentralized DL

For example, moderate machines have 1Gbps (125MBps) network capacity these days. In the circumstance, most DL models tend to become larger for higher accuracy. The winners of the most famous image recognition challenge (ILSVRC) used large size DL modes such as AlexNet (244MB), GoogLeNet (28MB), VGG16 (536MB). That size of gradients and weight values should be exchanged with either parameter servers in centralized DL or workers in decentralized DL shown in figure 5. It directly affects the performance of the distributed DL requiring hundreds of millions of iterations until the convergence of the model given large dataset in training phase.

3. Problem Definition

The goal of the most recent researches in distributed deep learning area to train a DL model with large-size training data using multiple machine much faster while maintaining the similar level of accuracy with the one of single-machine DL or synchronous DL model. For such a goal, the key problem to address is to reduce data size exchanging among machines every iteration without harming the model convergence.

4. Related Work

4.1. TensorFlow: A Distributed DL Framework

TensorFlow is an open source software library for efficient machine learning algorithms especially for deep learning. It was originally developed by researchers and engineers from the Google Brain team. It has flexible architecture allowing easy deployment of various machine learning and deep learning models across a variety of platform such as CPUs, GPUs, or TPUs from desktops to cluster of servers to mobile devices. It also provides different levels of APIs and various client languages so that application developers can easily make their own model and train it upon TensorFlow. It uses a *dataflow graph* to represent a DL model in terms of the dependencies between individual operations, and a *session* to run all or parts of the graph across a set of local and remote machines.

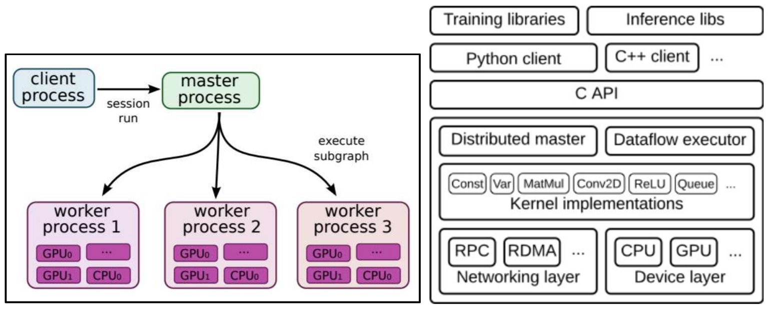


Figure 6. TensorFlow architecture and major components in a distributed setting

Distributed TensorFlow cluster comprises a one or more jobs. For centralized DL, there are parameter server job and worker job. Each job has one or more tasks serving the same job. All TensorFlow servers have both master process and worker process shown in figure 6.

Master process takes the role of an RPC service providing remote access to a set of distributed machines, acting as a session target, and coordinating work across one or more worker services. Worker process provides an RPC service executing parts of Tensorflow graph using its local machine. TensorFlow provides various APIs for centralized DL. However, it currently does not have appropriate APIs for decentralized DL.

4.2. Gaia: Centralized DL with Important Weights Sharing

Gaia is a paper (Geo-distributed machine learning approaching LAN speeds) published to NSDI 2017 by researchers from Carnegie Mellon University and ETH Zurich. The authors claims the necessity of geo-distributed machine learning spanning multiple data centers because it is impossible to move large-scale data generated rapidly all over the world into one location for training. They propose a new synchronization model called ASP (Approximate Synchronous Parallel) used among parameter server across multiple data centers. In data centers, typical centralized DL is used as shown in the figure 7.

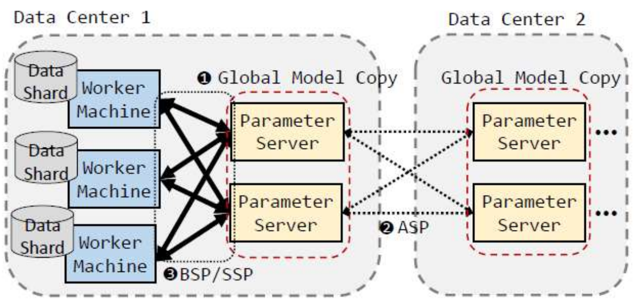


Figure 7. Gaia system overview

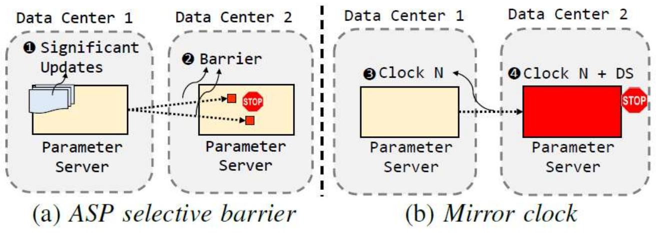


Figure 8. Gaia ASP (approximate synchronous parallel) techniques

The key idea of Gaia's ASP is to share significant updates out of all weights when exchanging the values across data centers in order to reduce the network traffic in WANs. In addition, they also suggested additional techniques illustrated in figure 8. When the WAN B/W is not enough to send the significant weights to other regions, the system sets ASP selective barrier by sending only the indexes of the weights to other servers in different regions so that they can pose next iteration until the actual values of the significant weights are sent over the WAN. It also provides a concept of mirror clock to avoid model divergence or severe accuracy degradation like the bounded synchronous DL. They show that Gaia provides 1.8 – 53.5 x speedup when running models across 11 amazon global regions.

4.2. Ako: Decentralized DL with Partial Gradient Exchange

Ako a paper (Decentralized deep learning with partial gradient exchange) published to SOCC 2016 by researchers from Imperial College London and MIT. It addresses the same network bottleneck issue in distributed deep learning covered in section 2 and 3 by using the decentralized DL shown in figure 3 with a new technique, partial gradient exchange.

The key idea of Ako's partial gradient exchange is to partition gradients and sent out a partition of the gradients every iteration. Since every workers eventually can receive all gradients of other workers within as many iterations as the number of partitions, the technique does not affect the model convergence. The rest of gradients not sent yet are accumulated with newly generated gradients in local machine during those iterations until being sent. More specifically, each worker sends *a partition of accumulated gradients* to other workers every iteration.

$$p = \frac{m(n-1)}{B}$$

Equation 1. An Ako parameter P: the number of partitions of gradients

They introduce an equation shown in equation 1 to find the right number of partitions of gradients where P is the number of partition, m is the size of a DL model, n is the number of workers, and B is the outgoing network bandwidth. They show that the decentralized Ako system can make a DL model converged faster than deployments with parameter servers using 64-node cluster.

5. System Design for Ako Implementation in TensorFlow

I design a decentralized DL system and implement Ako idea in TensorFlow in this plan B project. Figure 9 demonstrates the architecture of Ako worker and figure 10 illustrate my system design for the decentralized DL in TensorFlow.

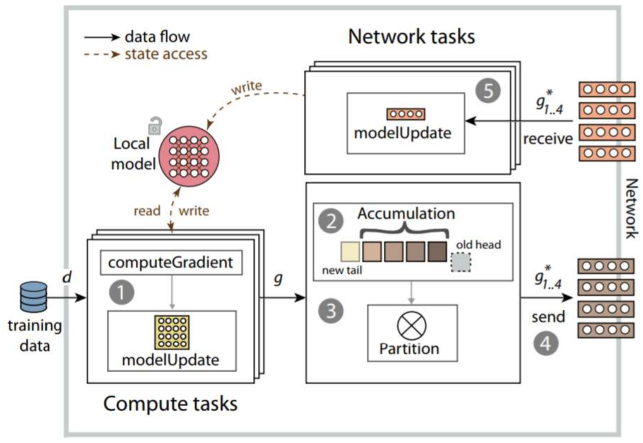


Figure 9. Architecture of Ako worker

Each worker computes gradients with a mini-batch training data and the weight values of the local model maintained in each machine in step 1 of the figure 9. The gradients just generated apply to the local model immediately. In step 2, the gradients are accumulated to the previously generated gradients for a certain iteration period in order to make sure every update information can be shared to other worker eventually. In step 3, the accumulated gradients are divided into multiple partitions. In step 4, one of the partition is sent to other workers. In step 5, other workers' gradient partitions are received asynchronously. Since it is a non-blocking operation, multiple threads receive the partitions from others, and apply them to the local model immediately. These steps repeat every iteration until the model is converged.

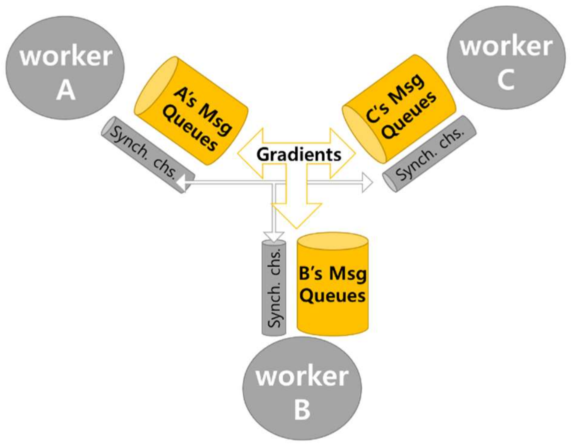


Figure 10. My system design for the decentralized DL in TensorFlow

I have designed the decentralized DL as shown in the figure 10. Every workers have their own message queues in order to share gradients each other for the steps 4 and 5 of figure 9. They also have a synchronization channel to ensure everybody go together every iteration for the synchronous DL. I have utilized *Redis*, an open source library for in-memory data structure store. Redis has several features – key-value storage, publish/subscribe messaging system as well as various memory optimization, caching policies, mass insertion of data, partitioning of data, and distributed locks. The key-value storage is used for the synchronize channel and pub/sub messaging system is used for the message queues for workers.

Current version of TensorFlow does not provide appropriate APIs for deployments of the decentralized DL like Ako. I initially implemented the message queues and synchronization channel by utilizing a TensorFlow API, ft.FIFOQueue. However, when training a DL model with the API, the size of Tensorflow graph was growing as iteration goes. TensorFlow keeps appending new nodes to the graph whenever the ft.FIFOQueue operation is called since the operation is used when a session is running, which is supposed to be called only in graph building. Therefore, I replaced it with Redis.

6. Beyond Ako in LAN and WAN environments

Ako has two assumptions. First, DL models are running in a cluster in a data center unlike Gaia. That is, AKO considers only LAN-speed network bandwidth between workers. Second, the authors also assume uniform network bandwidth among workers having similar level of bandwidth. Even though they mentioned a factor of B described in equation 1, the actual value of B becomes identical under the second assumption, and the other factors m and n are same in all workers. As a result, every worker has the same value of parameter P.

I wondered if the decentralized DL can have the same performance and accuracy when being applied in WAN environments. Furthermore, it was curious that how the performance and accuracy would change under the heterogenous LAN and WAN settings. Therefore, I planned to conduct the following experiments for comparison between a decentralized DL (my AKO implementation in TensorFlow) and a centralized DL (a deployment with parameter servers utilizing only TensorFlow operators) in various LAN and WAN environments.

- In uniform LAN
  - ✓ Synchronous vs Asynchronous Centralized DL
  - ✓ Synchronous vs Asynchronous Decentralized DL
  - ✓ Centralized DL vs Decentralized DL with fixed iterations
  - ✓ Centralized DL vs Decentralized DL with fixed learning time
- In heterogenous LAN, uniform WAN, heterogenous WAN
  - ✓ Centralized DL vs Decentralized DL with fixed learning time

7. Experiment settings

7.1. Testbed

Experiments are carried out on 5 local servers with 6 CPU cores (Intel Xeon CPU E5-2620 v3 @ 2.4GHz), approximately 45G available memory, and 1Gbps ethernet capacity per server. Cluster size varies as per experiment settings.

7.2. Dataset

The dataset is CIFAR10 consisting of 60,000 32 by 32 color images, with 6000 images per class. There are 50,000 training images and 10,000 test images. There are 10 classes in the dataset such as airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

7.3. My DL model

As shown in the figure 11, there are an input layer, an output layer, and 3 hidden layers, which includes 2 convolutional layers and 2 fully-connected layers. Weights are initialized with normal distribution with zero mean and 0.1 standard deviation, and bias are initialized to 0.0. Relu (rectified linear unit) as an activation function is applied to every layer except for the last output layer. Drop-out is applied to the first fully-connected layer for regularization. For the second fully-connected layer, softmax is applied for multiclass classification. Loss function is cross-entropy function and optimization is stochastic gradient decent (SGD). The model has 4M number of weights and its size is 17 MB. A mini-batch contains 104 samples.

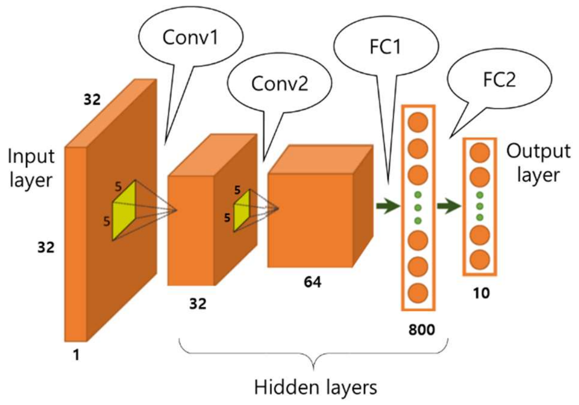


Figure 11. My DL model used for all experiments

7.3. Metrics

The following two metrics are used in all experiments

- ✓ Training time (= execution time): the elapsed time in seconds from training start to training stop
- ✓ Accuracy: the percentage of the number of test images correctly predicted over 10,000 test images

7.4. LAN and WAN emulation

For uniform LAN experiments, every link among workers has 800 Mbps network bandwidth. For heterogeneous LAN experiments, 800/400/200 Mbps network bandwidth are assigned to links. For uniform WAN experiments, 40 Mbps is used. For heterogenous WAN environments, 40/20/10 Mbps are set up.

TC commands are used to manipulate network traffic in the Linux kernel. Whenever the kernel needs to send a packet to an interface such as eth0, it is enqueued to a qdisc (queueing discipline). By adding a class option with different network bandwith condition to a qdisc, the qdisc may control the traffic as registered. A filter option is used to determine in which class a packet will be enqueued.

8. Evaluation and Analysis

All the numbers in the evaluation are calculated based on the average of three trails. The error bars indicate 95% confidence interval. N workers are used in both centralized and decentralized DL for N-worker experiments. One additional parameter server is used for the centralized DL settings. P = X indicates that the number of partition of gradients is X in all workers of decentralized DL. In the centralized DL, workers always share whole gradients to the parameter server, so it implies P = 1. When workers have their own number of partition, I mark it as P = various.

8.1. Centralized DL (TensorFlow)

The figure 12 shows that the difference in execution time and accuracy between synchronous and asynchronous centralized DL. The models are trained with 2400 iterations in uniform LAN environment. As the number of workers increases, the execution time decreases in both synchronous and asynchronous DLs since the amount of data processed in a worker reduced. At the same time, the accuracy also decreases since every iteration each worker calculates their gradients in the absence of larger portion of modification information handled by other workers. The biggest difference in training time reduction of 42% appears in 4-worker setting because 4-worker synchronous DL should wait all four workers are done to start next iteration. The accuracies of the two DLs remain same in 2-worker setting, whereas 8% discrepancy happens in 4-worker setting because there is no restriction to workers being ahead of the slowest worker in asynchronous DL.

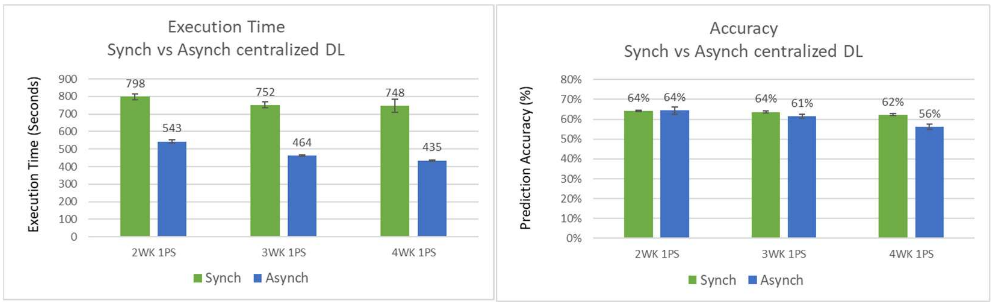


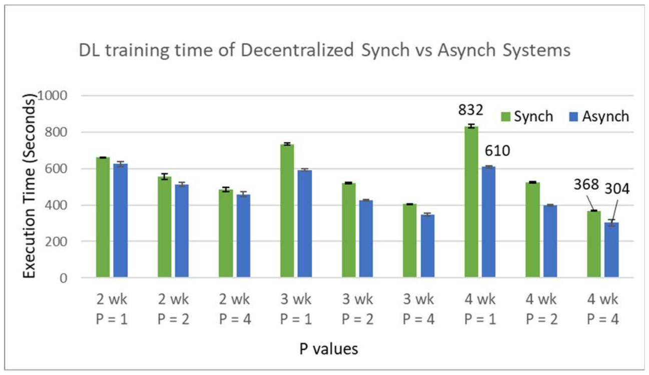
Figure 12. Execution time and accuracy difference

between Synchronous and Asynchronous Centralized DL in uniform LAN

8.2. Decentralized DL (Ako)

The figure 13 shows that the difference in execution time and accuracy between synchronous and asynchronous decentralized DL. The models are trained with 2400 iterations in uniform LAN environment. Like the case of centralized DL in the figure 12, it shows the similar patterns. As the number workers increase, training time decreases in all partition cases. We can directly compare P = 1 cases of N-worker settings in decentralized DL with the one of centralized DL. The 4-worker P = 1 setting has the largest difference of 27% in the training time between synchronous and asynchronous decentralized DLs. The difference diminishes to 17% as the number of gradient partitions increases to P = 4 because the amount of data exchanged every iteration in P =4 case is smaller than the one of P = 1 case and the processing time is reduced accordingly.

The decreasing accuracy as the number of workers increase follows the same pattern of centralized DL due to the same reasons with the centralized DL. However, decentralized DL has no difference in accuracy between synchronous and asynchronous DLs unlike the centralized DL having 8% of discrepancy between them. The network resource is equally dominated by the same amount of gradients data in all workers since the P values are the same and it is trained in uniform LAN network. It may prevent some workers from being far ahead of others, which results in similar accuracy between the two DLs.



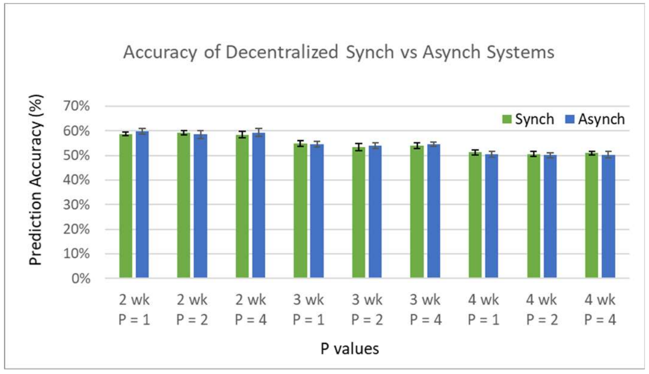


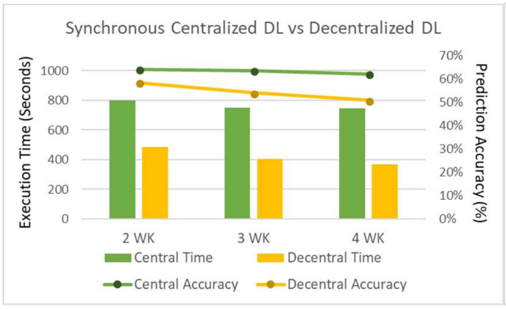
Figure 13. Execution time and accuracy difference

between Synchronous and Asynchronous Decentralized DL in uniform LAN

8.3. Centralized vs Decentralized DLs with fixed learning iterations

Figure 14 shows the difference between best cases of centralized and decentralized DLs. The experiments are conducted in uniform LAN environment with 2400 iterations. As the number of workers increase, the training time difference between the two DLs increase in both synchronous and asynchronous settings. On the other hand, the accuracy difference increases only in synchronous setting from 4% to 10%, whereas the difference in asynchronous setting is consistent as 4%. The reason of the increasing difference in execution time is obviously because of the effect of gradient partitioning of the decentralized DL, which can reduce the date exchanged among workers.

The 4% accuracy difference between the two DLs is caused by the absence of parameter servers in decentralized DL. This is because each worker in decentralized DL has no change to synchronize their weights each other, and solely depends on the gradient sharing. This tendency is more pronounced in synchronous setting because all workers get synchronized every iteration through parameter servers. Therefore, the accuracy difference in 4-worker setting becomes larger from 4% to 10%.



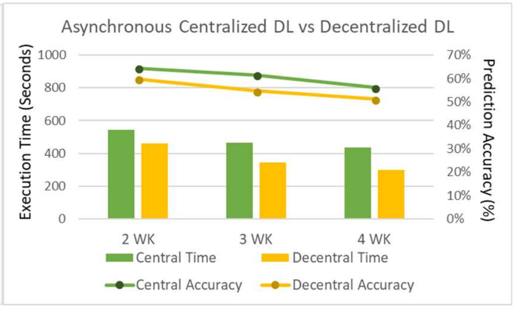


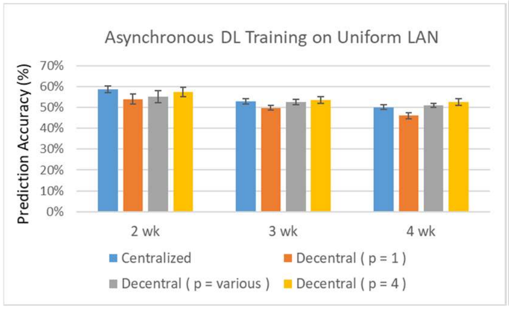
Figure 14. Centralized vs Decentralized DLs trained 2400 iterations in Uniform LAN

8.4. Centralized vs Decentralized DLs with fixed learning time

In this section, all cases are trained for 300 seconds. Figure 15 shows the results trained in uniform and heterogenous LANs and figure 16 shows the results trained in uniform and heterogenous WANs. Decentral (P = various) case indicates the accuracy obtained using different P values for each worker.

The key findings across from all LAN and WAN settings shown in the figure 15 and 16 are the followings:

- ✓ Decentralized DL outperforms centralized DL when network resource is scares.
- ✓ More partitions (larger P values) leads fast learning given fixed execution time.
- ✓ Decentralized DL (P = various) can get the best or similar accuracy in most cases.



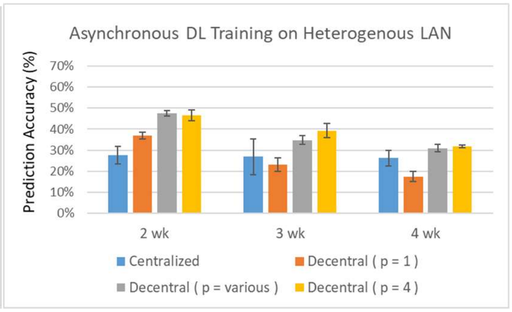


Figure 15. Centralized vs Decentralized DLs trained for 300 seconds

in Uniform and Heterogenous LANs

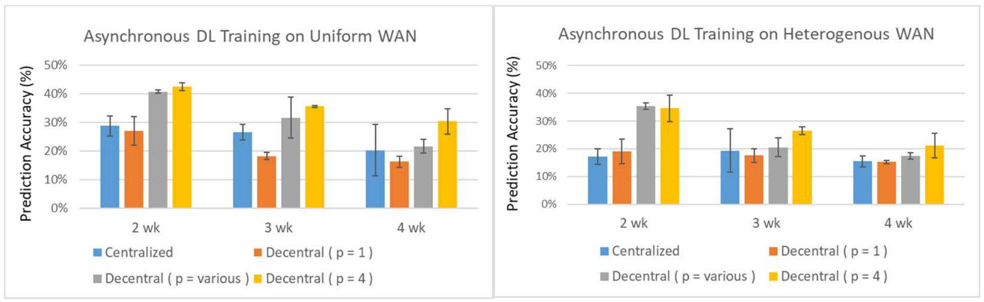


Figure 16. Centralized vs Decentralized DLs trained for 300 seconds

in Uniform and Heterogenous WANs

8.5. LAN vs WAN

Figure 17 shows the required iteration for given accuracy goal (40%) and the accuracy change of a worker as time goes. Decentralized DL requires extra iterations for the same accuracy in uniform LAN setting shown in the upper left graph in figure 17 because gradients are partitioned, and addition iterations are needed until the gradients are fully delivered to all workers. However, in the lower left graphs shows the opposite in which the centralized DL requires more iterations than the decentralized DL. With too few parameter servers or scarce network bandwidth, the efficiency of the parameter servers approach declines.

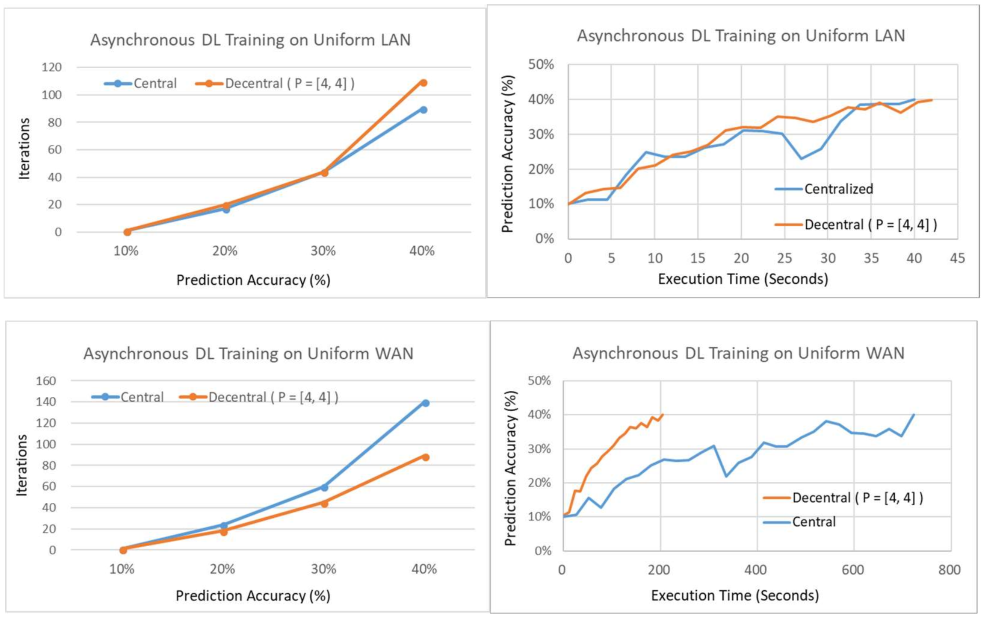


Figure 17. Centralized vs Decentralized DLs trained 300 seconds in Uniform LAN and WAN

The accuracy fluctuation over time of the centralized DL in both LAN and WAN is greater than the one of the decentralized DL shown in the two graphs in the right of figure 17. It is because of side effect of using parameter servers. When applying new weights pulled from the parameter server at the beginning of every iteration, it temporarily harms on the accuracy of the worker. However, it eventually helps the model converged faster when network bandwidth is enough because it widens the opportunity to explore gradient space.

It is obvious that the execution time of decentralized DL is much less than the one of centralized DL when the network bandwidth is scarce because the small size of data exchanged every iteration as shown in the right bottom graph in figure 17. The behavior of the centralized DL becomes like decentralized DL with P = 1 in WAN setting.

9. Conclusion

I have explored a variety of characteristics of distributed deep learning in both LANs and WANs. In uniform LAN environments, centralized DL and decentralized DL has similar performance in asynchronous setting in training time and prediction accuracy, whereas decentralized DL has much better performance in training time, but lower accuracy in synchronous settings. In other network environments – heterogenous LAN, uniform WAN, and heterogenous WAN, decentralized DL outperforms centralized DL when it is trained with more partitions of gradients. The reduced amount of data exchanged every iteration influence both training time and accuracy in a positive way. In addition, the decentralized DL with different P values can get the best or similar accuracies in most cases. The experiments also show that scares network bandwidth in WANs deteriorates the efficiency of centralized DL.

10. References

Abadi, Martín, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

Abadi, Martín, et al. "TensorFlow: A System for Large-Scale Machine Learning." OSDI. Vol. 16. 2016.

Hsieh, Kevin, et al. "Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds." NSDI. 2017.

Watcharapichat, Pijika, et al. "Ako: Decentralised deep learning with partial gradient exchange." Proceedings of the Seventh ACM Symposium on Cloud Computing. ACM, 2016.

Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009).

https://www.tensorflow.org