Exploration of Various Distributed Machine Learning Schemes

Yash Adhikari
College of Information and Computer Sciences
University of Massachusetts Amherst

Abstract

In this modern age of big data, it is becoming increasingly evident that we have to develop and explore parallel computing. Even though parallelism for computation has existed for decades, there is still much work to be done for improvements geared towards machine learning algorithms. We explore the performance of various distributed schemes such as fully connected graphs, expander graphs, and chain graphs through a simulation.

1 Introduction

1.1 Task Description

Large scale data is more frequently being analyzed with the assistance of machine learning algorithms. This application of parallelism to machine learning is possible due to the stochastic nature of the underlying algorithms. Machine learning algorithms typically are based on learning parameters that update with each iteration by calculating the gradient descent.

$$w = w - \alpha \nabla L(w) \qquad [1]$$

In equation 1 above, we see that the parameter w updates after each iteration by α or the learning rate times the gradient of loss (L) with respect to w. We can see here how machine learning algorithms are stochastic since an update made with loss calculated from a few examples may not be optimal. This phenomenon is epcially true when we perform stochastic gradient descent (SGD) by calculating loss from one training example. What is important is that in the long run, the gradient descent will guide the parameters to a globally optimal solution. Thus, we can take advantage of this feature and introduce parallelism for a faster time of convergence.

Processing such massive data in parallel does introduce new challenges in computing. These include designing and effectively utilizing parallel computing schemes for reasonable training and testing time. In general two different modes exist in order to achieve parallelism. One is data parallelism which partitions the data to allow a number of machines to compute updates separately. The other mode is model parallelism which separates the parameters to be trained at each worker. We focused our work on the first mode of data parallelism and distributed system schemes associated with it. In particular, we

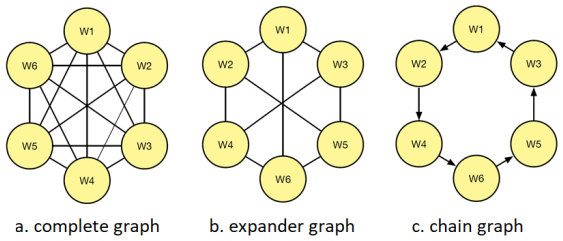

Figure 1: Distributed system schemes visualized with nodes as workers. Please note that the following figure has been re-purposed and altered from the previously proposed ASAP scheme (Kadav and Kruus, 2016)

explore distributed schemes seen above in figure 1. In a complete graph, every worker is connected to every other worker and thus once it computes an update, these updates will be forwarded to every other worker. In an expander graph with an out degree greater 1, we see that a worker communicates it updates to a specified number of nodes between 1 and the total number of nodes, not inclusive. Finally, in a chain graph, one worker only provides updates to one other worker. There are also other schemes widely used such as the parameter server where a master node collects updates from all other workers. Unfortunately, due

to time constraints this scheme was not explored in this paper.

1.2 Technical Challenges

There were a number of technical challenges with this project. Foremost among them was deciding the cost for communication among different workers. Even though processing cost could be estimated as discussed in section 3.1, estimating communication cost relative to processing cost is not trivial. For the sake of simplicity communication cost was chosen to be a factor of the fastest found processing cost prior to simulation run. The communication cost was anchored to the processing cost as the communication cost from each worker would increase as the number of parameters increases. The assumption here is that the processing cost would also see an increase as the number of parameters increases. Since there was no prior work found that would allow some basis for deciding communication cost, a static number of 75% of the fastest processing cost was set. This communication cost was then added each time a worker communicated its updates to other workers.

Unfortunately, real world distributed systems are unlikely to work in such a manner. Communication is often limited to the bandwidth of the bus. If all communication can be handled by this bandwidth, addition of the communication cost is not necessary. The expectation is that the approximation for communication cost can be improved in the future for a more accurate representation of distributed systems.

1.3 Contributions

The main contribution of this project is the implemented simulation of an expander graph with variable out degree at each node as well as variable number of nodes. This includes the ability to create a single node with no connections to replicate a non-distributed system. At this time, we are not aware of any work that attempts to evaluate various distributed systems through a simulation. Through this simulation we are able to track key performance metrics such as time to convergence, number of epochs, and epoch loss. This framework was built to be exploited by future works as sub components may be improved for better accuracy. The implementation has been uploaded to a github repository and it is publicly available at /yashadhikari/DistributedML.

We also found through this simulation that all the experimental distributed schemes outperform a single worker based on time to convergence. Even chain graphs see an improvement of a single worker as data is partitioned and SGD is calculated at multiple workers at the same time.

2 Related Work

There have been a number of research work that have added important information towards machine learning for distributed systems. However, in order to contribute something significant to the field, they often propose a new distributed system model or bring improvements to existing ones. The result is often that the research focuses on showcasing their proposed model and how it is an improvement on existing schemes. One such paper includes the proposal of ASAP or Asynchronous Approximate Data-Parallel Computation (Kadav and Kruus, 2016). ASAP quite cleverly brings significant improvements to asynchronous distributed systems by enforcing a worker to wait for all updates to be received. However, their results focused on how they brought improvements over a single worker as well as a scheme based on a complete graph. The authors chose to use an out degree at every node of 2 and never varied their number of nodes or number of connections. Even though the provided results are well justified for purposes of their research, we still do not have a good grasp on how a machine learning algorithm would perform under varied circumstances.

Similarly, we see the same situation occur in the proposal of the Batched Coupon's Collector where the model reaches near optimality to within a logarithmic factor (Li et al., 2017). The authors propose an improvement to the parameter server scheme and only compare their performance to other parameter server schemes such as uncoded or cyclic repetition schemes. Although such a comparison does well to justify the improvements brought by the BCC model, we do not know how these models would perform in comparison to distributed schemes such as those presented by ASAP.

3 Methodology

The simulation of distributed schemes was designed in multiple stages. The first step was to find a probability distribution for processing time cost. Even though we use total time cost as a performance metric where total time cost is the sum of processing time and communication cost, we have only clearly been able to approximate a distribution for processing time. For communication cost, we use a a simple constant for each time a worker has to communicate with another worker. After the distribution for processing time was defined, we proceeded design a connected graph with variable number of nodes and variable out-degree at each node. This allowed any worker to either communicate with just one other worker or all other workers. Finally, in order to get an approximation for the average time cost, we first found the appropriate number of runs to be more confident with our results.

3.1 Probability distribution for SGD calculation at each worker

In order to approximate the time cost for processing one training example, a SVM model as trained on two clusters of data 500 different times for 100 epochs. Based on the number of data points, processing time for 20 million gradient updates were found. This timing information can be seen in figure 1 as a histogram of number of occurrences for each section of processing time. In order to use this

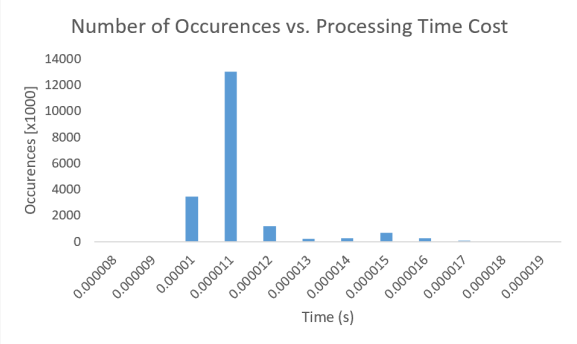

Figure 2: Distribution of processing time cost for calculation of SGD at each worker

information as a probability distribution, we created another data set which contains the cumulative sum of number of occurrences. Then, each time a worker calculates updates for SGD, we can create a random number between 0 and 20 million. Based on the placement of the number we can associate it with the processing time cost using the newly created cumulative sum data.

3.2 Design of the connected graph

The implementation of the connected graph as a distributed system contained several key components. At initialization, all data received was divided among the different workers and indices were found for which nodes each worker was going to be sending updates. During each iteration, a worker would first update its parameters, calculate the SGD for a given training example, and then send updates to appropriate workers. This process would repeat until all the data points were covered completing an epoch. Once multiple epochs were processed, model convergence was estimated and fed back to the simulator.

Two key decisions are made during initialization of the connected graph - data partitioning and deciding where worker updates are sent. The partitioning of data is mostly trivial as we simply divided the data among the nodes. If number of nodes was set to 1, all of the data would be given to that particular worker. An issue arises if the data cannot be evenly partitioned where the number of training examples divided by the number of nodes does not result in n integer value. For simplicity, the remaining data was added to the final worker. The downside to this approach is that a completion of an entire epoch in the simulation may not mean that the model has trained over all the training examples. All of our experiments used evenly divisible data for accuracy and simplicity. The other key component for sending worker updates is similarly simple in its implementation. Each worker at index i gives updates to worker i+1 : i+(c+1) where c represents the number of connections or out degree of the node at index i. If any of the forward indices do not exist, the indices are simply looped back starting at 0.

Once the data has been partitioned, the distributed system can start training on a machine learning model. For our work, an SVM model was passed to the connected graph. However, the implemented graph is quite flexible and will accept any other machine learning models. During training, each iteration comprises of three steps. First, the parameters are updated at each worker with the calculated gradient times a

defined learning rate. Then, each worker calculates the SGD for the next training example. Due to potential uneven spread of the data, the next data to be trained is also tracked. Finally, each worker sends its updates to appropriate nodes based on indices found during initialization. In our system, loss is tracked from each worker and an average is taken once all the training examples have been processed to calculate the epoch loss.

From the simulator, during each simulation run, we continue training the model until convergence criteria is achieved. We chose the convergence criterion to be 0.00001 where the subsequent epoch loss must be less than the defined criterion. When the absolute value of the change in epoch loss is found to be less than the criterion, we can move to the next repetition in the simulation. This criterion was chosen based on previous runs of the created data set and may differ for different machine learning models.

3.3 Selecting number of runs for simulation

We obtained the total cost in time for each run through summing processing cost and communication cost. Since this time defers for each run, we get the average and the variance of the total cost as an estimate by running the simulation 100 times. The average and the variance can then be used to estimate the total number of runs required for determining cost to within a certain specified accuracy. Equation [2] below was used to get the estimate on the total number of runs for each scheme used.

$$n = (\hat{s}^2 z_\delta^2)/(\epsilon^2 \hat{\mu}^2) \qquad [2]$$

We use the above equation to estimate the total number of runs required for estimating time cost to within $\pm 100\epsilon\%$ with $100(1-\delta)$ probability. In equation [2], n is the estimate of the number of runs, $\hat{s}^2$ is the variance of total cost, and $\hat{\mu}$ is the average estimate of the total cost. Equation 2 can simply be derived from the central limit theorem where $(\hat{s} z_\delta)/\sqrt{n}$ is the half width of the confidence interval. If we set the half width equal to ϵ we can obtain equation 2. Please note that $\hat{\mu}$ is inserted into the denominator after the fact because we are interested in relative precision intervals.

4 Experiments

4.1 Dataset

The data for the SVM model was created using samples_generator in the Scikit-learn library. The make_blobs method created two centers with two features and 400 samples. In order to compare the time to convergence criteria, a new data set was created for each run in the simulation. However, this is not feasible for comparing epoch loss as the data set differs during each run. Thus, for comparing epoch loss, only a single data set was created with which the simulation was run.

4.2 Increasing number of nodes

During the first phase of the experiments, the number of nodes was gradually increased while the total number of connections per node was set to a constant 2. A single worker with no connections was also run alongside for understanding improvements a distributed system may bring to machine learning algorithms. The results for this experiment are tabulated in table 1.

Nodes	Out degree	Total Cost	Processing	Communication
1	0	0.0527	0.0527	0
3	2	0.0101	0.0073	0.0029
5	2	0.0151	0.0092	0.0059
10	2	0.0255	0.0118	0.0137

Table 1: Cost in seconds with increasing number of nodes and the same number of connection per node

We see that as the number of nodes increase, the processing cost increases as well (excluding the single worker system). The scheme with fastest average processing time as well as average total time was found to be $n = 3$. Furthermore, we see that the communication cost continues to increase as the number of nodes increase. As the graph size increases, the amount of communication among different nodes also increases.

Nodes	Out degree	Avg. Final Epoch Loss
1	0	0.00013
3	2	0.00005
5	2	0.00010
10	2	0.00052

Table 2: Average final epoch loss with increasing number of nodes and the same number of connection per node

In table 2 we observe that the final epoch loss tends to increase as the number of nodes increases. The epoch loss was comparable only because we chose to make the artificially created data remain the same for the duration of these particular runs. We

did not want stochasticity from the data set changing the performance of the different distributed systems.

4.3 Increasing connectivity

The second phase of the the experiments included increasing the number of connections for each worker while maintaining the total number of workers. A single node was again included in the analysis for comparison against a non-distributed system. We observe in table 3 that as the number of connections per node increases, the communication cost increases as well. This is expected since each worker is sending updates to increasing number of other workers. The processing cost also follows a similar trend with increasing number of connections usually resulting in a decrease in cost.

Nodes	Out degree	Total Cost	Processing	Communication
1	0	0.0527	0.0527	0
10	1	0.0166	0.0105	0.0061
10	2	0.0255	0.0118	0.0137
10	5	0.0237	0.0061	0.0176
10	9	0.0341	0.0055	0.0286

Table 3: Cost in seconds with increasing number of connections and the same number of workers

In table 4 we see that initially an increase in the number of connections tends to increase the final epoch loss, but there is a turn around in the trend with a complete graph giving us the best results for final epoch loss.

Nodes	Out degree	Avg. Final Epoch Loss
1	0	0.00013
10	1	0.00029
10	2	0.00052
10	5	0.00012
10	9	0.00007

Table 4: Average final epoch loss with increasing number of connections and the same number of workers

4.4 Analysis

From table 1 and table 3 we saw that the average processing cost tends to decrease as our distributed system increases in size regardless of whether this means more workers or more connections among the workers. This phenomenon occurs because of both the parallel nature of our system and a faster time to convergence. Computation in parallel allows the system to calculate multiple SGDs at the same time. Furthermore updates from one worker gets passed along to its neighbours. As all the workers are updating at a much quicker pace, our models converge at a much faster pace. This is why we are bound to outperform a single worker and why a larger distributed system will likely outperform a smaller one with respect to total average processing time.

The same situation does not occur for communication cost. It is evident that as a distributed system gets larger either in connectivity or number of workers, we see an increase in communication cost. This occurs because workers have to pass along the same information to more number of workers or more workers are passing along updates. In table 1 and 3 we see that for average total cost a chain graph actually performs the best. However, this may not necessarily be the case given that the communication cost implemented was a gross approximation to a real world distributed system. However, we can clearly see the benefits each of the distributed systems provide over a single worker as well as how processing time and communication cost change with varying number of nodes and connections.

We also looked at the average final epoch loss for each of the distributed system and there was a clear trend of decreasing final epoch loss with increasing number of connections. In fact, we saw the lowest average final epoch loss for a complete graph where each node is connected to every other node both in table 2 and table 4. An explanation for such a phenomenon may be that during training parameters at every node are much more robust from noise as every worker gets updates from every other worker. This may result in parameters being updated to be much closer to the global minimum.

5 Conclusion and Future Work

We have successfully implemented a distributed system with varied number of workers and connections geared towards tackling machine learning algorithms. We measured key performance metrics such as time to convergence which included average processing cost and average communication cost and average final epoch loss. The data illustrates how communication cost increases while processing cost decreases with increasing size of the distributed system. We also observed that a complete graph usually results in a much lower average final epoch loss. These results provide much insight into how potential systems

can best benefit machine learning algorithms.

There can be a number of improvements on the framework that has been designed in this project. Primary among them is the calculation of communication cost. It plays a significant role in deciding the outcome of time to convergence. A constant communication time cost serves well as a framework, but a dynamic one would be better suited for this simulation. We would have also liked to implement more models for comparison including parameter servers and start including asynchronous systems such as ASAP. Due to time constraints, these models were not implemented, but they would provide valuable insight into inner workings of distributed systems when it comes to machine learning.

Acknowledgements

This work was part of a seminar provided by the University of Massachusetts Amherst proctored by Peter Haas, PhD and Don Towsley, PhD.

References

Asim Kadav and Erik Kruus. 2016. Asap: Asynchronous approximate data-parallel computation.

Songze Li, Seyed Mohammadreza Mousavi Kalan, A. Salman Avestimehr, and Mahdi Soltanolkotabi. 2017. Near-optimal straggler mitigation for distributed gradient methods.