Dynamically Adjusting the Stale Synchronous Parallel Model for Edge Computing

Sangsu Lee¹, Taeho Lee², Yili Wang² and Hee Yong Youn²

¹Dept. of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas, USA
²Dept. of Electronic, Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea

Abstract - The advent and proliferation of the Internet of Things calls for a paradigm shift toward edge computing, which facilitates collective intelligence of edge nodes. Among several parallel machine learning models, Stale Synchronous Parallel (SSP) model is believed to be well-suited to conduct distributed machine learning with edge nodes. This paper seeks for a versatile solution that warrants robustness in the cost of computation of edge nodes in hostile conditions, and introduces a novel scheme dynamically assigning optimal staleness level to the nodes. Computer simulation reveals that the proposed approach substantially outperforms the conventional SSP model of fixed staleness level with respect to computation cost and robustness in various operational conditions. Two new notions of loss, structural loss and update computation cost of edge nodes in hostile conditions, and seeks for a versatile solution that warrants robustness in the distributed machine learning with edge nodes. This paper introduces a novel scheme dynamically assigning optimal staleness level to the nodes. Computer simulation reveals that the proposed approach substantially outperforms the conventional SSP model of fixed staleness level with respect to computation cost and robustness in various operational conditions. Two new notions of loss, structural loss and update loss, are developed to evaluate the robustness.

Keywords: Edge computing, Machine learning, Consistency, Internet of Things, Stale synchronous parallel

1 Introduction

The Internet of Things (IoT) environment is often envisioned as a network consisting of a massive number of nodes. Since the devices swarming in the network inundate the servers with big data, efficient interconnection and management of a large number of devices are critical. This is scalability issue, and effective solution for it is one of the key features of IoT network [2]. The conventional client-server model cannot warrant small response time in handling large quantity data produced from numerous nodes since all the data and requests are processed in the centralized server [3].

Edge computing is a powerful paradigm of taking away the burden from the center of the network, realizing computational offloading [4] and therefore becoming a reasonable solution for IoT environment. In Edge computing, the functionalities of cloud computing are brought to the edge devices as much as possible. The interconnected nodes collaboratively process data instead of consuming the network bandwidths for transmitting and receiving data, reducing the volume of traffic between the edge device and the server by over 90% [1]. In edge computing computation is allowed at the edge of the network, taking the burden from the data centers. In order to conduct collaborated operation at the edge of the network, especially machine learning (ML) operation, a fault-tolerant and scalable distributed solution is needed. One of the primary issues is synchronization, which involves handling communication consistency throughout the edge devices.

Bulk Synchronous Parallel (BSP) is a fully synchronous parallel model with which the nodes are updated in each iteration, being synchronized through barriers. Even though the barriers make the BSP system robust to computational failure, it is a costly scheme since each node has to wait for all other nodes in each step. This model has been implemented in various Parallel ML frameworks. Shotgun [9] implements this concept and proposes a scalable algorithm for loss minimization model. REEF [10] and Mlbase [11] are distributed data analysis system that makes use of BSP. On the other hand, asynchronous parallel models avoid synchronization, which means a node does not wait for other nodes to keep abreast of recent updates. For instance, Hogwild [6] showed that the Stochastic Gradient Descent (SGD) algorithm can be implemented without any locking, in other words, asynchronous operation. Cyclades [17] takes this notion with conflict-free parallel execution. NOMAD [7] also provides a solution for matrix problems handled in distributed manner, with fully asynchronous variable updates for each processor.

Recently, the Stale Synchronous Parallel (SSP) scheme has been the newcomer of distributed ML and big data processing, recognized by its performance, correctness, and versatility. While the conventional BSP is handicapped from long execution time, SSP provides a more flexible solution while preserving data consistency. It lets the computing nodes proceed to the next iteration without waiting for other nodes as long as they are within certain range of iteration. This range of barriers is addressed as staleness level. In order to properly decide the staleness level, this paper attempts to predict the convergence time according to various features of distributed ML such as the number of nodes, cost of computation, and most importantly, the staleness level. By approximating the computational cost of the nodes into Gaussian distribution, this paper comes up with a solution that efficiently decides a staleness level minimizing the cost of computation. Through scrupulous analysis and simulation, it is shown that the proposed scheme significantly reduces the computation cost compared to the existing SSP scheme of fixed staleness level.
in various operational conditions. It is also identified that the proposed scheme is quite robust against hostile environment of edge computing of the nodes of abruptly changing performance. Two new notions of loss, structural loss and update loss, are developed to evaluate the robustness.

The rest of the paper is organized as follows. Section II discusses the related work. In Section III the derivation of cost function of an SSP system which takes staleness as an argument is introduced. The cost function is then analyzed to reveal the properties of SSP and decide optimal staleness level. Based on this, the dynamic allocation of staleness (DAS) scheme is proposed. The validity of the proposed approach and its performance are evaluated under various conditions in Section IV. Finally, Section V concludes the paper with some remarks.

2 Related Work

2.1 Bridging Nodes with Parameter Server

One of the problems of implementing the conventional cloud computing model in IoT network is that delivery of massive data generated by numerous nodes to the cloud server causes impractically high communication cost. Also, the desire for the reduction of computation latency motivated the IoT nodes to embrace distributed computing rather than processing data in the big, centralized server. Because high versatility is essential in edge computing environment, the bridging model [19] capturing the resources of distributed system is needed. The parameter server framework depicted in Fig. 1, which was first introduced as sampling architecture for inferring latent topic models [5], is the backbone of the proposed scheme.

![Fig. 1. The framework of parameter server.](image)

Here a server node maintains and synchronizes the parameters shared between the worker nodes in a worker group. In other words, the server node gathers and processes the information of the workers and communicates with other servers. The concept of parameter server was originated from blackboard system, where multiple knowledge sources provide information to contribute to solving the problem, which is analogical to the nodes updating data in reference to a server. Reference [5] established a method of parallelizing Latent Dirichlet Allocation (LDA) sharing global state variables with memcached. However, the cost for synchronization and latency of memcached is substantially high.

Later on, parameter server was employed in the application specific implementations. Along with introducing an efficient communication protocol, YahooLDA [21] developed its own key-value storage which ensures synchronization. Petuum [18] is one of the second-generation parameter servers, which improved YahooLDA by adopting the SSP model. These approaches target specific applications instead of addressing the universal method of parallelizing the algorithms applicable to a wide variety of algorithms including ML. Parameter server framework [15] is considered to be an efficient model for conducting the ML system. It endeavours to be a top notch framework for distributed ML, which is scalable, fault-tolerant, easy to use, and having minimum network overhead. The worker nodes keep only a designated portion of training data and perform computation with them [20]. This model is shown to be applicable to a number of models such as risk minimization, Gibbs sampling, deep learning, and so on. Furthermore, large-scale ML systems such as MXNet [23] and TensorFlow [22] adopt the concept of parameter server. Especially, TensorFlow improved the conventional parameter server framework by implementing user-defined management of shared data.

2.2 Stale Synchronous Parallel

One of the crucial issues with designing parameter server is how to warrant data consistency while retaining cost-effectiveness. In the parameter server model the parameter server synchronizes the workers in each worker group, and the basic consistency model is BSP Model [19]. It is also adopted in REEF [10] and MIbase [11]. As aforementioned, the distributed systems like GraphLab [16] are asynchronous. While the goal of an ML model is to find a reasonable local minimum, giving a parallel model a leeway in consistency is usually a good choice, if considerable reduction of cost is guaranteed.

Recently, an SSP model, which is also known as a partially synchronous model or a bounded model is introduced. It utilizes the concept of bounded staleness [13], where the workers receive stale version of updates from the parameter server. Refer to Fig. 2. This is a depiction of iterations of a running SSP system from the perspective of worker-1, which is the fastest worker. The tick marks denote the finishing point of each iteration. For example, the graph shows worker-3 has finished $(i-1)$th iteration and is proceeding to $i$th iteration. With staleness $s$, a worker at $i$th iteration obtains the updates older than $(i-s)$th iteration, which means that it sees from iteration $0$th to $(i-s-1)$th iteration as depicted as a shaded region in Fig. 2. This is because the server distributes only the delayed version of parameters when a worker requests it, although the workers commit the result of their work immediately after each
The model for the cost of BSP under an assumption that the edge computing environment of numerous attributes affecting the cost of computation of the nodes will yield the condition for Gaussian distribution. Also, the cost of computation of SSP model is analyzed under various operational conditions. From this, the process for obtaining optimal staleness level for the target environment is decided. Lastly, the scheme of dynamic allocation of staleness is introduced, which allows the system of parallel ML to effectively adapt to the dynamically changing environment.

### 3.1 Cost of BSP

The cost for an individual superstep [12] of BSP system is

\[
\text{Cost}_{\text{BSP}} = w_{\text{max}} + h + l
\]

where \(w_{\text{max}}\) is the maximum cost of computation and \(h\) is the maximum communication for all processors in an individual superstep. \(l\) is the latency, which is the time needed for synchronizing the nodes. In the proposed scheme only \(w_{\text{max}}\) is taken into account since it is a dominating factor.

Obtaining the expected value of \(w_{\text{max}}\) with \(n\) nodes following Gaussian distribution is equivalent to the problem of getting the expected value of maximum computation cost for any node in one superstep. Therefore, the expected value of \(\max(X)\) for \(n\) independent random variables, \(X_1, X_2, \ldots, X_n\), which all follow Gaussian distribution, \(N(\mu, \sigma^2)\), will be the estimated value of \(w_{\text{max}}\). If \(\Phi\) is the cumulative distribution function of Gaussian distribution,

\[
P\left(\max_i X_i < \mu + t\sigma\right) = \prod_{i=0}^{n} P(X_i < \mu + t\sigma) = \Phi(t)^n
\]

\[
w = E\left[\max_i X_i\right] = \mu + \sigma \int_{-\infty}^{\infty} t \frac{d}{dt} \Phi(t)^n dt
\]

Then, if the BSP requires for \(i\) supersteps for convergence, the total cost of computation becomes

\[
\text{Cost}_{\text{BSP}} = w \cdot i
\]

### 3.2 Cost of SSP

Similar to the model for BSP, the worker nodes assigned to a single parameter server is approximated into a Gaussian distribution, \(N(\mu, \sigma^2)\). The evaluation is done by taking one node worker into consideration. The cost of computation for \(i\)th iteration is

\[
\text{Cost}_i(s) = P_b \cdot E_b(X) + (1 - P_b) \cdot E(X)
\]

where \(P_b\) and \(E_b\) are the probability and expected cost in bounded condition, respectively, and \(E(X)\) is the expected cost of \(X\) of Gaussian distribution. In other words, the cost for conducting an iteration for a node is the sum of expected cost in bounded and unbounded condition. The cost for all iterations is then

\[
\text{Cost}_{\text{SSP}} = \sum_{i=1}^{s} E_i(X) + \sum_{i=s+1}^{I} \text{Cost}_i(l)
\]
Recall that the cost of BSP was estimated in supersteps, while that of SSP is done with each iteration separately. Here, \( I \) is the total number of iterations expected to be needed for convergence. \( \text{Cost}_{\text{SSP}}(s) \) is summed up from \((s+1)\)th to \(i\)th iteration because the staleness bound cannot take effect before \((s+1)\)th iteration. The cost of computation before \((s+1)\)th iteration is aggregated mean of random variables representing the cost for individual iteration.

Assume that the cost of computation of \( n \) worker nodes for one iteration is determined by random variables \( X_1, X_2, \ldots, X_n \). The given status is that worker-1 has just finished \( i \)th iteration and needs to decide whether it has to proceed to the next iteration or not. The probability that worker-1 will be bounded by worker-2 at any iteration is

\[
P_b = P\left( \frac{X_1}{X_2} < \frac{1}{s+1} \right)
\]

(7)

This is acquired by the ratio distribution [14] between \( X_1 \) and \( X_2 \). The cumulative distributed function of the ratio distribution between the two random variables which have the same mean and deviation is,

\[
F(w) = \frac{b(w) \cdot d(w)}{\sqrt{2\pi\sigma^2}a^3(w)} \left[ \Phi \left( \frac{b(w)}{a(w)} \right) - \Phi \left( \frac{h(w)}{a(w)} \right) \right] + \frac{1}{\pi\sigma^2 a^2(w)} \exp \left( -\frac{c}{2} \right)
\]

(9)

where

\[
a(w) = \sqrt{\frac{w^2 + 1}{\sigma}}
\]

(10)

\[
b(w) = \frac{\mu}{\sigma^2}(w+1)
\]

(11)

\[
c = \frac{2\mu^2}{\sigma^2}
\]

(12)

\[
d(w) = \exp \left( \frac{b^2(w) - c^2a^2(w)}{2a^2(w)} \right)
\]

(13)

This is applicable to \( X_1, X_2, \ldots, X_n \) since they follow the same distribution and independent from each other. By making use of \( f(w) \), \( P_b \) is obtained as follows.

\[
P_b = P\left( \frac{X_1}{X_2} < \frac{1}{s+1} \right) = f \left( w < \frac{1}{s+1} \right)
\]

(14)

\( E_b \) is the expected cost of computation when a worker is bounded by staleness barrier. Before it was bounded, the expected cost would be \( \mu \). The time it has to wait will be the maximum difference between to random variables. Since no matter how many slow workers cause a worker to be bounded, the delay time would be the difference between the slowest worker and the worker of consideration. Therefore, a random variable, \( Z = (X_1 - X_2) \), is defined which follows Gaussian distribution \( N(0, \sigma^2) \). Then,

\[
E_b = \mu + E\left[ \max(Z) \right]
\]

(15)

The second term of Eq. (15) is derived with a method similar to Eq. (3).

\[
E\left[ \max(Z) \right] = \sigma \int_{-\infty}^{\infty} t \frac{d}{dt} \Phi(t)^n dt
\]

(16)

To sum up, a function for the total cost for SSP system with staleness as an input is

\[
\text{Cost}_{\text{SSP}}(s) = \sum_{i=s+1}^{I} f \left( w < \frac{1}{s+1} \right) \cdot E_b + \left( 1 - f \left( w < \frac{1}{s+1} \right) \right) \cdot \mu + s \cdot u
\]

(17)

### 3.3 Optimal Staleness

As mentioned earlier, the SSP model is identical to BSP model when the staleness level is 0. As the staleness level increases, the cost of computation decreases whereas the algorithm tends to diverge due to the inconsistency of data between the nodes. Finally, when the staleness level reaches to the point where no synchronization occurs, the system is considered to be asynchronous. The optimal staleness level would be where the cost of computation is reduced the most, while the nodes do not suffer from huge inconsistency. In the proposed scheme, the ideal staleness level is obtained as follows. When \( \mu, \sigma, I, \) and \( n \) are provided, the set of optimal staleness is attained from

\[
S = \left\{ s | s \in N \bigcap \arg \left[ \frac{d^2\text{Cost}(s)}{ds^2} \equiv 0 \right] \right\}
\]

(18)

The ideal staleness level is the one making the value of second derivative of the cost function become 0. This point is also known as an ‘elbow’ of the function. It is considered as the point where the cost benefit hits maximum while maintaining data consistency as much as possible.

### 3.4 Dynamic Allocation of Staleness (DAS)

Once the ideal level of staleness under the given condition is decided, the parameter server allocates it to the nodes. The proposed scheme of dynamic allocation of staleness (DAS) predicts the best staleness level for the current process, and allocates it regularly. Note that the optimal level of staleness is decided based on the gathered cost data from worker nodes. Because the nodes notify the parameter server their cost data whenever an iteration is over, the node acting as parameter server does not need to keep
track of every worker node. This ensures the scalability of the parallel model. Furthermore, the servers for different worker groups can share the parameter values, allowing the edge network to be aware of the progress of ongoing tasks.

For the first step of DAS, the parameter server estimates mean and deviation of Gaussian distribution using the dataset of the costs. This dataset is gathered by the parameter server during some number of iterations when the workers process data under the BSP condition. When a sufficient amount of data is collected, it allocates the staleness level with CostSSP(s), allowing the nodes to run at their optimum level of staleness. After the initial determination of staleness level, the parameter server continuously accumulates data and periodically allocates the best level of staleness to the worker nodes it manages. For example, if the adjustment occurs for every $V$ intervals of iteration, the set of updating iterations would be

$$R = \{i | i \in N \land (i + i_d) \mod V = 0\}$$

where $N$ is the set of natural numbers.

4 Performance Evaluation

The performance of the proposed scheme is evaluated by running distributed ML algorithm of JAVA with a worker group in edge computing environment. The performance of the proposed DAS scheme is investigated in various operational conditions and compared with the conventional SSP of fixed staleness level.

Making the distributed computation model such as distributed ML in edge computing environment scalable is important since the nodes can frequently join and leave the system. The model gets more robust with DAS because it allocates the best staleness level considering the current condition. The cost of computation is evaluated with the worker nodes as shown in Fig. 4. The allocated staleness levels with DAS are marked at dot. Whereas the cost of the fixed SSP continues to increase as the number of nodes gets larger, that of DAS is restrained by allocating different staleness levels. When the number of nodes is 200, 350, and 500, the cost decreases as the staleness increases. DAS avoids drastic increases in the cost by distributing new staleness level to the worker nodes.

By dynamically allocating the staleness level to the nodes, DAS noticeably reduces the cost of computation of distributed ML. Fig. 5 illustrates total estimated cost for convergence using three different values of fixed staleness and DAS implementation. In Fig. 5(a) 100 nodes are used with a mean of 100 seconds for 100 iterations, while the three values are increased to 200 in Fig. 5(b). Fig. 5(a) shows that DAS is a lot less costly than the fixed cases of same or close staleness level as DAS. Similar results are also obtained with different conditions of Fig. 5(b).

Even though the SSP model is cost-effective compared to BSP, it has its own downside. The algorithm may diverge since the nodes perform ML with a delayed version of parameters. With large staleness, the update the workers obtain from the server is a greatly delayed one, and the difference between the progresses of the nodes gets bigger. The “loss of consistency” refers to the amount of different information between the nodes. Specifically, the “loss” is categorized into two types; structural loss and update loss. The structural loss is the one the SSP model suffers unlike the BSP model, while update loss is the aggregation of the gap of
individual worker with respect to the server data. These two new notions of losses are utilized to evaluate the performance of DAS.

Fig. 6. The loss with different staleness levels and DAS.

Fig. 6 compares structural and update loss with fixed staleness levels and DAS. Observe from the figure that the proposed DAS scheme consistently displays smaller loss for both the types. The robustness of the proposed scheme is investigated by running the JAVA application of distributed ML in unstable edge computing environment with increasing deviation in the cost of the nodes. Refer to Fig. 7 which depicts the loss in Fig. 6(a) obtained as the iteration increases. Here 200 nodes are used, and individual cost of the nodes was generated by Gaussian distribution with the mean value of 100. The initial sampling interval of DAS is 30 iterations. Notice that the proposed scheme displays notably smaller loss than with fixed staleness. In Fig. 6(b), smaller sampling interval is used and the loss is much less than the fixed cases. Despite the large mean of allocated staleness with 8.66, its loss is smaller than the model with fixed staleness of 5. The change in computation cost with DAS in both the cases is less than 1.8%. As the deviation grows larger, the gap between the updates of the nodes widens. However, DAS dynamically allocates optimal value of staleness, and thus minimizes the structural loss. Unless the costs of the workers are too much divergent, DAS will show noticeably better performance than the models with fixed staleness.

Fig. 7. The cost of computation with different staleness levels and DAS implementation.

Fig. 7 shows the total cost of computation of fixed staleness of 3, 4, and 5, and DAS. Observe from the figure that DAS shows stable performance compared to the fixed cases. For example, DAS of the allocated staleness level of the mean of 4.02 takes almost same time as fixed staleness of 4. Also, DAS with the mean of 3.52 has the cost between the fixed staleness of 3 and 4. While DAS shows balanced cost of computation, it shows its strength in keeping the loss of information as low as possible even in hostile environment.

Fig. 8. The loss of computation during the ML in a hostile environment.

Fig. 8 shows the structural loss during the ML computation where 10 out of the 100 nodes confront problem during the operation and get drastically slow starting from 800th iteration. Deviation of 35 was assumed for this evaluation. Notice that the loss with fixed staleness of 5 becomes larger than that of DAS starting from around 870th iteration due to the event at 800th iteration. This example reveals the robustness of DAS against abrupt changes in the nodes of edge computing environment.

5 Conclusion

In this paper a novel scheme predicting and allocating optimal staleness level for the stale synchronous parallel model has been introduced. It adjusts the staleness level according to the current condition of the nodes, and thus making itself quite robust against dynamically changing node and network condition. Computer simulation revealed that the distributed machine learning in edge computing environment takes much less time with DAS than the SSP model of fixed staleness level. Furthermore, the proposed scheme is robust...
against the change in the operation condition of the target system. Two new notions of loss, structural loss and update loss, have been developed to evaluate the robustness. The proposed scheme is also applicable to any target system displaying different distributions in the cost of computation, with small adjustment of the distribution in the cost function.

In the future more research will be conducted on the behavior of a large worker group on practical devices, especially regarding the cost of computation. The scheme guaranteeing data consistency for various ML algorithms while changing staleness level in the middle of distributed learning will also be sought.

6 Acknowledgment

This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. 2016-0-00133, Research on Edge computing via collective intelligence of hyperconnection IoT nodes), Korea, under the National Program for Excellence in SW supervised by the IITP (Institute for Information & communications Technology Promotion) (2015-0-00914), Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology(2016R1A6A3A11931385, Research of key technologies based on software defined wireless sensor network for real-time public safety service, 2017R1A2B2009095, Research on SDN-based WSN Supporting Real-time Stream Data Processing and Multi-connectivity), the second Brain Korea 21 PLUS project.

7 References