CS744 Final Project Report

Hongyi Wang *and* Yuzhe Ma *and* Xin Jin

December 18, 2017

1 Motivation and Background

[1] In distributed machine learning [2], synchronous settings and asynchronous settings are most commonly seen in literature. In synchronous distributed machine learning, models are always hold by a certain node known as parameter server. Synchronous point will always been created in each training iteration, which leads to large time costs when there are stragglers in the cluster. Compared to synchronous settings, asynchronous distributed machine learning has many inherent advantages e.g. easier to scale with increasing cluster dimension, good speedups, and etc. However, effect of staleness provide adversarial effect in asynchronous learning algorithms. Recently, many robust asynchronous distributed learning algorithms have been proposed.

The very recent successful asynchronous distributed machine learning algorithm is HOGWILD [8]!, an entirely lock free asynchronous parallel stochastic gradient descent (SGD) method in memory-shared multi-core system/machine. Proof had shown in HOGWILD! that sparsity of dataset will lead to tightly bounded staleness of gradients. HOGWILD! Was followed up under many conditions by many other works. Instead of avoiding staleness, some works focused on understanding staleness effect in asynchronous distributed machine learning as well as trying to use them to scale the model performance e.g. convergence rate, stability, and etc.

Backup worker strategy is one of the state-of-art methods to avoid straggler effect in distributed synchronous machine learning. Instead of using n workers to training the model, this method allowing to use k extra worker nodes in the training process and always wait for the faster n workers out of $n + k$ in total. However, in this work, gradients on slow workers were going to be dropped when they're returned to master node.

In our approach, were going to aggregated those stale gradients into mode update process and design algorithm to find optimal n, k fraction and weighted stale gradients. Under this problem setting, each worker return their gradients with a global step token (i.e. $t, t-1, t-2, \dots$), for each iteration, master only wait for k faster workers out of n. When gradients from slow workers (t-1, t-2, ...) are received, i.e. the token returned with the gradient is order than the global step, master cache and use them for next model update (for step t). The whole process is visualized in Fig. 1.

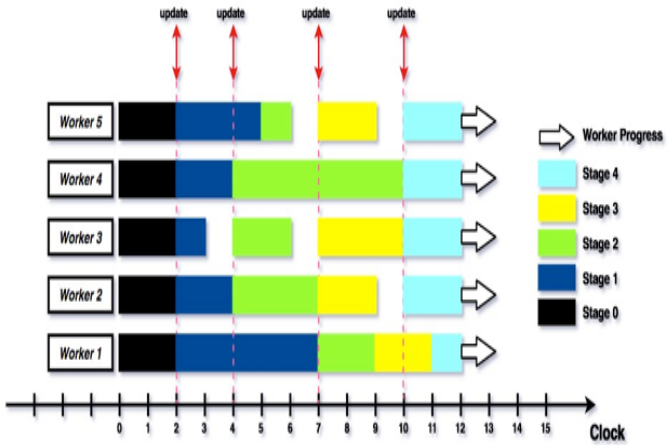


Figure 1: Waiting for the first k workers.

[1] code of this project is powered by: `https://github.com/hwang595/ps_pytorch`

2 Theoretical Analysis

2.1 Setting

We model our setting as the following gradient descent:

$$w_{t+1} = w_t - \sum_{i=t-k+1}^{t} \alpha_i \nabla f(w_i) \tag{1}$$

Different from traditional gradient descent, (1) include all the gradients in the past k steps to compute w_{t+1}. The α_i are the weights associated with $f'(w_i)$. In case $k = 1$, (1) breaks down to the standard gradient descent.

2.2 One-dimensional Case

We assume $f(w)$ is strictly convex so that there exists a unique global minimizer w^*. We assume the generalized curvature of $f(w)$ is bounded both from below and above. To be precise, we first restate the definition of generalized curvature in [9].

Definition 1. *Generalized Curvature. The derivative of $f(x) : \mathbb{R} \to \mathbb{R}$, can be written as*

$$f'(x) = h(x)(x - x^*) \tag{2}$$

for some $h(x) \in \mathbb{R}$, where x^* is the global minimum of $f(x)$. We call $h(x)$ the generalized curvature.

Theorem 1. *Let $f(w)$ be strictly convex, and assume the generalized curvature $h(w) \in [a, b]$, where $0 < a \leq b$. If $c \leq \alpha_t \leq \frac{1}{b}$ for some $c > 0$ and $\sum_{i=t-k+1}^{t-1} \alpha_i \leq \frac{a}{2b}\alpha_t$, then $\lim_{t\to\infty} |w - w^*| = 0$.*

Proof. Note that

$$w_{t+1} = w_t - \sum_{i=t-k+1}^{t} \alpha_i f'(w_i) = w_t - \sum_{i=t-k+1}^{t} \alpha_i h(w_i)(w_i - w^*) \tag{3}$$

Thus we have

$$\begin{aligned} w_{t+1} - w^* &= w_t - w^* - \sum_{i=t-k+1}^{t} \alpha_i h(w_i)(w_i - w^*) \\ &= [1 - \alpha_t h(w_t)](w_t - w^*) - \sum_{i=t-k+1}^{t-1} \alpha_i h(w_i)(w_i - w^*) \end{aligned} \tag{4}$$

Therefore we have

$$\begin{aligned} |w_{t+1} - w^*| &= |[1 - \alpha_t h(w_t)](w_t - w^*) - \sum_{i=t-k+1}^{t-1} \alpha_i h(w_i)(w_i - w^*)| \\ &\leq |1 - \alpha_t h(w_t)||w_t - w^*| + \sum_{i=t-k+1}^{t-1} |\alpha_i h(w_i)||w_i - w^*| \\ &\leq [|1 - \alpha_t h(w_t)| + \sum_{i=t-k+1}^{t-1} |\alpha_i h(w_i)|] \max_{t-k+1\leq i\leq t} |w_i - w^*|. \end{aligned} \tag{5}$$

Since $h(w) \in [a, b]$ regardless of w, $\alpha_t \leq \frac{1}{b}$ and also $\sum_{i=t-k+1}^{t-1} \alpha_i \leq \frac{a}{2b}\alpha_t$, thus

$$\begin{aligned} |1 - \alpha_t h(w_t)| + \sum_{i=t-k+1}^{t-1} |\alpha_i h(w_i)| &= 1 - \alpha_t h(w_t) + \sum_{i=t-k+1}^{t-1} \alpha_i h(w_i) \\ \leq 1 - a\alpha_t + b \sum_{i=t-k+1}^{t-1} \alpha_i &< 1 - a\alpha_t + b\frac{a}{2b}\alpha_t \leq 1 - \frac{a}{2}\alpha_t \leq 1 - \frac{ac}{2} \end{aligned} \tag{6}$$

Plug (6) into (5) we have $|w_{t+1} - w^*| \leq (1 - \frac{ac}{2}) \max_{t-k+1\leq i\leq t} |w_i - w^*|$. Therefore every k steps, the $|w_t - w^*|$ decreases by at least constant factor $1 - \frac{ac}{2}$, which concludes the proof. □

For the first k steps, simply use the gradient descent and then start using (3).

Example 1. *Quadratic Function. Let $f(w) = w^2$. One can check that $h(w) = 2$, and thus $a = b = 2$. If $k = 3$, one can use $\alpha_t = \frac{1}{2}$ and $\alpha_{t-1} + \alpha_{t-2} = \frac{1}{2}\alpha_t = \frac{1}{4}$, then the convergence is guaranteed.*

Example 2. *General Case. In general, one can set $\alpha_t = \frac{1}{b}$, and let $\alpha_i = \frac{1}{2^{t-i+1}}\frac{a}{b^2}$. Note that in this case, $\sum_{i=t-k+1}^{t-1}\alpha_i = \frac{a}{2b^2}\sum_{i=t-k+1}^{t-1}\frac{1}{2^{t-i}} = \frac{a}{2b^2}\sum_{i=1}^{k-1}\frac{1}{2^i} \leq \frac{a}{2b^2} = \frac{a}{2b}\alpha_t$, which satisfies the conditions in Theorem 1.*

2.3 High Dimensional Case

Now assume $w \in \mathbb{R}^d$ and $f(w) : \mathbb{R}^d \to \mathbb{R}$. Similarly, we define the high-dimensional generalized curvature as follows:

Definition 2. *High-dimensional Generalized Curvature. The derivative of a strictly convex function $f(x) : \mathbb{R}^d \to \mathbb{R}$, can be written as*

$$\nabla f(x) = H(x)(x - x^*) \tag{7}$$

for some $\nabla f(x) \in \mathbb{R}^d$, where x^* is the global minimum of $f(x)$. Let $\lambda_i, i \in [d]$ be the eigenvalues of $H(x)$ and also use $v_i, i \in [d]$ to denote the corresponding eigenvectors. We call λ_i the generalized curvature along direction v_i.

Theorem 2. *Let $f(w)$ be strictly convex. Assume the generalized curvature $\lambda_i \in [a, b]$ for all i at any w, where $0 < a \leq b$. If $c \leq \alpha_t \leq \frac{1}{b}$ for some $c > 0$ and $\sum_{i=t-k+1}^{t-1}\alpha_i \leq \frac{a}{2b}\alpha_t$, then $\lim_{t\to\infty}\|w - w^*\| = 0$.*

Proof. Note that

$$w_{t+1} = w_t - \sum_{i=t-k+1}^{t}\alpha_i\nabla f(w_i) = w_t - \sum_{i=t-k+1}^{t}\alpha_i H(w_i)(w_i - w^*) \tag{8}$$

According to the Cauchy-Schwartz inequality, we have

$$\begin{aligned}
\|w_{t+1} - w^*\| &= \|[I - \alpha_t H(w_t)](w_t - w^*) - \sum_{i=t-k+1}^{t-1}\alpha_i H(w_i)(w_i - w^*)\| \\
&\leq \|[I - \alpha_t H(w_t)](w_t - w^*)\| + \sum_{i=t-k+1}^{t-1}\|\alpha_i H(w_i)(w_i - w^*)\| \\
&\leq (1 - \alpha_t a)\|w_t - w^*\| + \sum_{i=t-k+1}^{t-1}\alpha_i b \max_{t-k+1\leq i\leq t-1}\|w_i - w^*\| \\
&\leq (1 - \alpha_t a)\|w_t - w^*\| + \frac{\alpha_t a}{2}\max_{t-k+1\leq i\leq t-1}\|w_i - w^*\| \\
&\leq (1 - \frac{\alpha_t a}{2})\max_{t-k+1\leq i\leq t}\|w_i - w^*\| \leq (1 - \frac{ca}{2})\max_{t-k+1\leq i\leq t}\|w_i - w^*\|.
\end{aligned} \tag{9}$$

Concluding the proof. □

Example 3. *Ridge Regression. Let $f(w) = \|Xw - y\|^2 + \eta\|w\|^2$. The global minimum $w^* = (X^\top X + \eta I)^{-1}X^\top y$. $f(w) = (Xw - y)^\top(Xw - y) + \eta w^\top w$, thus $\nabla f(w) = 2X^\top Xw - 2X^\top y + 2\eta w = 2(X^\top X + \eta I)(w - w^*)$ and $H(w) = 2(X^\top X + \eta I)$, which is a constant with respect to w. Now consider the generalized curvature of $f(w)$, which are the eigenvalues of $H(w) = 2(X^\top X + \eta I)$. Without loss of generality we assume single instances satisfy $\|x\| \leq 1$. Then we have the following claim:*

Theorem 3. *All the eigenvalues of $H(w) = 2(X^\top X + \eta I)$ lies between $[2\eta, 2n + 2\eta]$, where n is the training set size.*

Proof. Let u be a unit vector. The we have

$$\begin{aligned} u^\top H(w)u &= u^\top 2(X^\top X + \eta I)u = 2[u^\top (\sum_{i=1}^{n} x_1^\top x_1)u + \eta u^\top u] \\ &= 2(\sum_{i=1}^{n} u^\top x_1 x_1^\top u + \eta) \in [2\eta, 2n + 2\eta]. \end{aligned} \tag{10}$$

□

Thus, the ridge regression satisfies all the constraints in Theorem 4.

Now let us introduce an indicator function $\mathbb{1}[w_i]$, which is defined as follows: $\mathbb{1}[w_i] = 1$ if $\nabla f(w_i)$ appears in the update and 0 otherwise. We further define a new update rule as:

$$w_{t+1} = w_t - \sum_{i=t-k+1}^{t} \alpha_i \mathbb{1}[w_i] \nabla f(w_i) \tag{11}$$

The above update mimics the real situation where some past gradients could be missing because of the stragglers. One can easily prove that this update rule still guarantees convergence with those α_i set as Theorem 1 and Theorem 4.

3 Extension to SGD

Now we consider the extension to SGD, where the update rule becomes:

Theorem 4. *Let $f(w)$ be strictly convex. Assume the generalized curvature $\lambda_i \in [a, b]$ for all i at any w, where $0 < a \leq b$. If $c \leq \alpha_t \leq \frac{1}{b}$ for some $c > 0$ and $\sum_{i=t-k+1}^{t-1} \alpha_i \leq \frac{a}{2b}\alpha_t$, then $\lim_{t\to\infty} \|\mathbf{E}[w_t - w^*]\| = 0$.*

$$w_{t+1} = w_t - \sum_{i=t-k+1}^{t} \alpha_i X_i, \tag{12}$$

where X_i satisfies $\mathbf{E}[X_i] = \nabla f(w_i)$. Still we want to guarantee the convergence of our iterator.

Proof. Note that we have

$$\begin{aligned} \mathbf{E}_{X_i, t-k+1\leq i\leq t}[w_{t+1} - w^*] &= \mathbf{E}_{X_i, t-k+1\leq i\leq t}\left[w_t - w^* - \sum_{i=t-k+1}^{t} \alpha_i X_i\right] \\ &= \mathbf{E}_{X_i, t-k+1\leq i\leq t}[w_t - w^*] - \sum_{i=t-k+1}^{t} \alpha_i \mathbf{E}_{X_i, t-k+1\leq i\leq t}[X_i] \\ &= \mathbf{E}_{X_i, t-k+1\leq i\leq t}[w_t - w^*] - \sum_{i=t-k+1}^{t} \alpha_i \nabla f(w_i) \\ &= \mathbf{E}_{X_i, t-k+1\leq i\leq t}[w_t - w^*] - \sum_{i=t-k+1}^{t} \alpha_i H(w_i)(w_i - w^*) \end{aligned} \tag{13}$$

Thus we have

$$\begin{aligned} &\mathbf{E}_{X_i, i\leq t-k+1}[\mathbf{E}_{X_i, t-k+1\leq i\leq t}[w_{t+1} - w^*]] = \mathbf{E}_{X_i, i\leq t-k+1}[\mathbf{E}_{X_i, t-k+1\leq i\leq t}[w_t - w^*]] \\ &- \mathbf{E}_{X_i, i\leq t-k+1}\left[\mathbf{E}_{X_i, i\leq t-k+1}\left[\sum_{i=t-k+1}^{t} \alpha_i H(w_i)(w_i - w^*)\right]\right], \end{aligned} \tag{14}$$

which is simply

$$\mathbf{E}[w_{t+1} - w^*] = \mathbf{E}[w_t - w^*] - \mathbf{E}\left[\sum_{i=t-k+1}^{t} \alpha_i H(w_i)(w_i - w^*)\right] \tag{15}$$

Thus we have

$$\begin{aligned}
&\|\mathbf{E}\left[w_{t+1}-w^*\right]\| = \|\mathbf{E}\left[w_t-w^*\right] - \mathbf{E}\left[\sum_{i=t-k+1}^{t}\alpha_i H(w_i)(w_i-w^*)\right]\| \\
&\leq \|\mathbf{E}\left[w_{t+1}-w^*\right] - \mathbf{E}\left[\alpha_i H(w_i)(w_i-w^*)\right]\| + \|\mathbf{E}\left[\sum_{i=t-k+1}^{t-1}\alpha_i H(w_i)(w_i-w^*)\right]\| \\
&\leq \|\mathbf{E}\left[w_{t+1}-w^*\right] - \mathbf{E}\left[\alpha_i H(w_i)(w_i-w^*)\right]\| + \sum_{i=t-k+1}^{t-1}\|\mathbf{E}\left[\alpha_i H(w_i)(w_i-w^*)\right]\| \qquad (16) \\
&\leq \|\mathbf{E}\left[(1-\alpha_t a)(w_{t+1}-w^*)\right]\| + \sum_{i=t-k+1}^{t-1}\|\alpha_i b\mathbf{E}\left[(w_i-w^*)\right]\| \\
&\leq (1-\alpha_t a)\|\mathbf{E}\left[w_{t+1}-w^*\right]\| + \sum_{i=t-k+1}^{t}\alpha_i b\|\mathbf{E}\left[(w_i-w^*)\right]\|.
\end{aligned}$$

Now let $v_t = \mathbf{E}\left[w_t - w^*\right]$, then we have

$$\|v_{t+1}\| = (1-\alpha_t a)\|v_t\| + \sum_{i=t-k+1}^{t}\alpha_i b\|v_i\|. \qquad (17)$$

□

Follow the similar strategy as before, we conclude the proof.

4 Experimental Settings

PyTorch + OpenMPI are used for our system design. We used AutoGrad data structure and execution engine provided in PyTorch 0.3.0 to implement matrix calculation e.g. forward propagation and backward propagation for Deep Neural Network while communications among nodes in the cluster is implemented in OpenMPI 3.0.0.

4.1 System Design

In our distributed ML system, there are three kind of nodes, parameter server, worker, and evaluator, (showed in Fig.2) we will discussed them in detail.

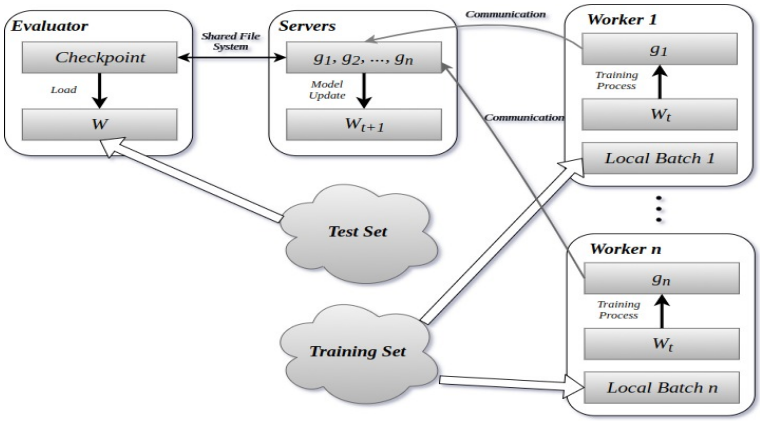


Figure 2: System Design

- **parameter server**: The parameter server we implemented is basically follow Mu Li's parameter server [7] setting. This node serves both as master and parameter server in our system, i.e. it synchronize all workers to enter next iteration by broadcast global step to workers and also store the global model, which are keeping fetched by worker nodes at beginning of one

iteration. For a user defined frequency, parameter server node will save the current model as checkpoint to shared file system (NFS in our system) for model evaluation.

- **worker**: this node is mainly aims at sample data points (or mini-batch) in *i.i.d* manner from their local dataset, computing gradients, and send them back to parameter server.

- **evaluator**: Read the checkpoints from the shared directory, and do model evaluation. Note that: there is only testset data saved on evaluator nodes. In general, there is only one parameter server node and one evaluator node in a distributed cluster. But can have bunch of (e.g. n) worker nodes. Global batch size B is split among workers, which is $\frac{B}{n}$.
  Since all worker nodes are trying to communicate with one parameter server node, the communication overhead is heavy. We also implement gradient compression in our system. This part is done with high-performance compression tool *Snappy*.

4.2 Experimental Details

Experiments are running on m4.2xlarge instances of AWS EC2, which has 16 2.4 GHz Intel Xeon E5-2676 v3 processors with 32 GB memory. We set up our cluster with 1 parameter ser node, 16 worker nodes and 1 evaluator node.
The training set are spread among worker nodes, i.e. every worker node has a replica of training set while only evaluator node holds the test set. Parameter server node saved the trained model as checkpoint file for each 10 iterations to shared file system while evaluator keep checking the shared directory for latest checkpoint. When new checkpoint was detected by the evaluator, it evaluate the model and log performance out to local file system.
We choose MNIST (the hand written digits images) as our dataset, which contains 50K images in the training set and 10K images in the testset. We tried 2 model for experiments, LeNet [6] and fully connected nerual networks, which has 2 hidden layers with 800 and 500 hidden units respectively. In both experiments, we used global batch size at 256 and splitted it among 16 workers nodes and we only allow one-step staleness to be aggregated into our model with the weight exponentially shrinking by design in section 3. Detailed results and discussions will be given in section 5.

5 Experiment Results

5.1 MNIST on LeNet

In Fig. 5, the left figure indicate time cost saved by the backup worker setting where we do not add any gradient staleness. In that case, we can tell that backup worker setting speedup the synchronous SGD with respect to time costs but the model convergence remains at the same level. In the right figure of Fig. 5, we showed that, we tried to change the number of worker we want to wait i.e. we control the number of stale workers for each iteration. Also, we only allow 1-step staleness gradient to be aggregated into the global model. From the experiment results, we can tell that, as we add 25%, the model converge faster (with respect to both time costs and convergence rate) than the baseline (in which we wait for all workers in a certain iteration and use all gradient to update the model). But if we allow more staleness to be aggregated to the model (e.g. 50% in our experiments), we observe worse model convergence performance. Based on this results observed, we think there should be an optimal fraction of staleness that we are allowed to aggregated to the model.
In Fig.6a, we observe the same trend in training set as the test set convergence performance, which indicate the model generalize well during the training process [1].

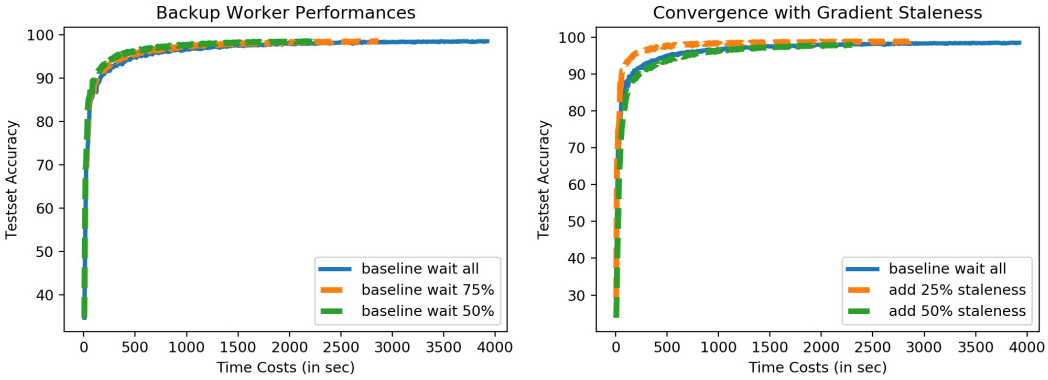
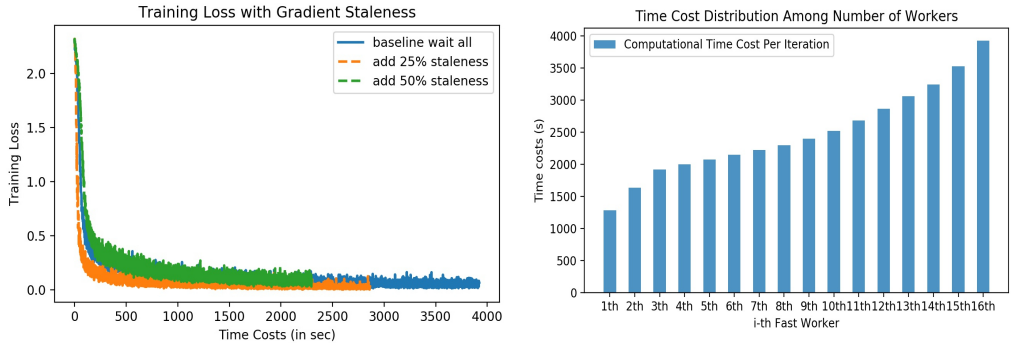


Figure 3: Testset Performance of MNIST running on LeNet with global batch size at 256



(a) Variantion of Training Loss of MNIST on LeNet (b) Total Time Cost Distribution among workers

Figure 4: Convergence Performance on Training Set with Time Cost Distribution

5.2 MNIST on FC Neural Net

For this experiment, we repeat exactly the same experimental setups as experiments for LeNet on MNIST and tried to observe the affect included from machine learning model and number of parameters. Based on the experiment results we can tell that the overall results is similar to the results we observed in LeNet, but adding staleness can help with model convergence with different level.

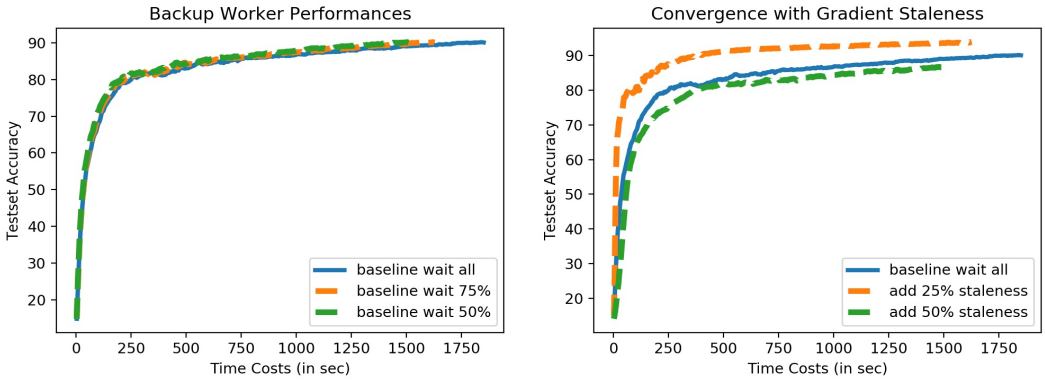
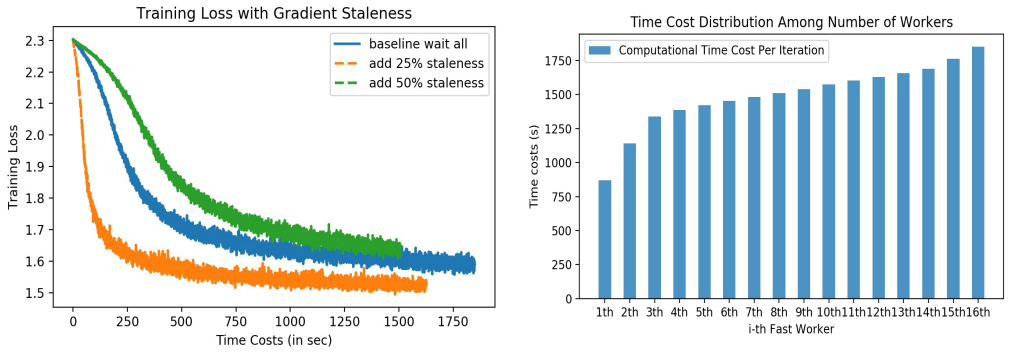


Figure 5: Testset Performance of FC Net running on LeNet with global batch size at 256



(a) Variantion of Training Loss of MNIST on FC Net (b) Total Time Cost Distribution among workers

Figure 6: Convergence Performance on Training Set with Time Cost Distribution

6 Conclusion and Future Workers

In this work, we firstly come up with our assumption that adding gradient staleness to some extent to backup workers distributed setting can help with the model convergence with respect both to time cost and model convergence rate. We then theoretically proved that by design of the parameter that we used to aggregate the stale gradients the model is still guaranteed to converge in both vanilla and stochastic gradient descent cases. And we provide experimental results to verify our proposed claims.
To make this work complete, here is what we need to do in the future:

- provide proof about the new convergence rate with gradient staleness aggregated
- run experiment on larger cluster e.g. 100+ nodes with larger deep neural nets e.g. ResNet, AlexNet [3][5]
- compare our results with other approach e.g. stale synchronous PPS [4]

References

[1] O. Bousquet and A. Elisseeff. Stability and generalization. *J. Mach. Learn. Res.*, 2:499–526, Mar. 2002.

[2] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1223–1231, USA, 2012. Curran Associates Inc.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[4] Q. Ho, J. Cipar, H. Cui, J. K. Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, pages 1223–1231, USA, 2013. Curran Associates Inc.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[7] M. Li, D. G. Anderson, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *Operating Systems Design and Implementation (OSDI)*, pages 583–598, 2014.

[8] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc., 2011.

[9] J. Zhang, I. Mitliagkas, and C. Ré. YellowFin and the Art of Momentum Tuning. *ArXiv e-prints*, June 2017.