Dynamic Backup Workers for Parallel Machine Learning

Chuan Xu Giovanni Neglia Nicola Sebastianelli
Inria, Université Côte d'Azur, Sophia Antipolis, France
firstname.familyname@inria.fr

***Abstract*—The most popular framework for parallel training of machine learning models is the (synchronous) parameter server (PS). This paradigm consists of n workers and a stateful PS, which waits for the responses of every worker's computation to proceed to the next iteration. Transient computation slowdowns or transmission delays can intolerably lengthen the time of each iteration. An efficient way to mitigate this problem is to let the PS wait only for the fastest $n - b$ updates, before generating the new parameters. The slowest b workers are called *backup workers*. The optimal number b of backup workers depends on the cluster configuration and workload, but also (as we show in this paper) on the current stage of the training. We propose DBW, an algorithm that dynamically decides the number of backup workers during the training process to maximize the convergence speed at each iteration. Our experiments show that DBW 1) removes the necessity to tune b by preliminary time-consuming experiments, and 2) makes the training up to a factor 3 faster than the optimal static configuration.**

I. Introduction

In 2014, Google's Sybil machine learning (ML) platform was already processing hundreds of terabytes through thousands of cores to train models with hundreds of billions of parameters [1]. At this scale, no single machine can solve these problems in a timely manner, and, as time goes on, the need for efficient parallel solutions becomes even more urgent. Currently, the operation of ML parallel systems requires a number of ad-hoc choices and time-consuming tuning through trial and error, e.g., to decide how to distribute ML programs over a cluster or how to bridge ML computation with inter-machine communication. For this reason, significant research effort (also from the networking community [2], [3], [4], [5]) is devoted to design adaptive algorithms for a more effective use of computing resources for ML training.

Currently, the most popular template for parallel ML training is the parameter server (PS) framework [6]. This paradigm consists of workers, that perform the bulk of the computation, and a stateful parameter server that maintains the current version of the model parameters. Workers use locally available versions of the model to compute gradients which are then aggregated by the PS and combined with its current state to produce a new estimate of the optimal parameter vector. However, if the PS waits for all workers before updating the parameter vector (synchronous operation), *stragglers*, i.e., slow tasks, can significantly reduce computation speed in a multi-machine setting [7], [8]. A simple solution that mitigates the effect of stragglers is to rely on backup workers [9]: instead of waiting for the updates from all workers (say it n), the PS waits for the fastest k out of n updates to proceed to the next iteration. The remaining $b \triangleq n - k$ workers are called backup workers. Experiments on Google cluster with $n = 100$ workers show that a few backup workers (4–6) can reduce the training time by 30% [9].

The number of backup workers b has a double effect on the convergence speed. The larger b is, the faster each iteration is, because the PS needs to wait less inputs from the workers. At the same time, the PS aggregates less information, so the model update is noisier and more iterations are required to converge. Currently, the number of backup workers is configured manually through some experiments, before the actual training process starts. However, the optimal static setting is highly sensitive to the cluster configuration (e.g, GPU performances and their connectivity) as well as to its instantaneous workload. Both cluster configuration and workload may be unknown to the users (specially in a virtualized cloud setting) and may change as new jobs arrive/depart from the cluster. Moreover, in this paper we show that the optimal number of backup workers changes during the training itself(!) as the loss function approaches a minimum. Therefore, the static configuration of backup workers does not only require time-consuming experiments, but is particularly inefficient and fragile.

In this paper we propose the algorithm DBW (for

ISBN 978-3-903176-28-7© 2020 IFIP

Dynamic Backup Workers) that dynamically adapts the number of backup workers during the training process without prior knowledge about the cluster or the optimization problem. Our algorithm identifies the sweet spot between the two contrasting effects of b (reducing the duration of an iteration and increasing the number of iterations for convergence), by maximizing at each iteration the decrease of the loss function *per time unit*.

The paper is organized as follows. Sect. II provides relevant background and introduces the notation. Sect. III illustrates the different components of DBW with their respective preliminary assessments. DBW is then evaluated on ML problems in Sect. IV. Sect. V concludes the paper and discusses future research directions. Our code is available online [10].

II. Background and notation

Given a dataset $\mathbb{X} = \{x_l, l = 1, \dots S\}$, the training of ML models usually requires to find a parameter vector $\boldsymbol{w} \in \mathbb{R}^d$ minimizing a loss function:

$$\underset{\boldsymbol{w} \in \mathbb{R}^d}{\text{minimize}} \quad F(\boldsymbol{w}) \triangleq \frac{1}{S} \sum_{l=1}^{S} f(x_l, \boldsymbol{w}), \tag{1}$$

where $f(x_l, \boldsymbol{w})$ is the loss of the model $\boldsymbol{w}$ on the datapoint x_l.

The standard way to solve Problem (1) is to use an iterative gradient method. Let n be the number of workers (e.g., GPUs) available. In a synchronous setting without backup workers, at each iteration t the PS sends the current estimate of the parameter vector $\boldsymbol{w}_t$ to all the workers. Each worker computes then a stochastic gradient on a random mini-batch drawn from its local dataset, and sends it back to the PS. We assume each worker has access to the complete dataset $\mathbb{X}$ as it is reasonable in the cluster setting that we consider. Once n gradients are received, the PS computes the average gradient

$$\boldsymbol{g}_t = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{g}_{i,t} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{B} \sum_{x \in \mathbb{B}_i} \nabla f(x, \boldsymbol{w}_t),$$

where $\boldsymbol{g}_{i,t}$ denotes the i-th gradient received by the PS, computed on the random minibatch $\mathbb{B}_i \subseteq \mathbb{X}$. Then the PS updates the parameter vector as follows:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}_t, \tag{2}$$

where $\eta > 0$ is called the learning rate.

When b backup workers are used [9], the PS only waits for the first $k = n - b$ gradients and then evaluates the average gradient as

$$\boldsymbol{g}_t = \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{g}_{i,t}. \tag{3}$$

In our dynamic algorithm (Sect. III), the value of k is no longer static but changes in an adaptive manner from one iteration to the other, ensuring faster convergence speed. We denote by k_t the number of gradients the PS needs to wait for at iteration t, and by $T_{i,t}$ the time interval between the update of the parameter vector $\boldsymbol{w}_t$ at the PS and the reception of the i-th gradient $\boldsymbol{g}_{i,t}$.

To the best of our knowledge, the only other work proposing to dynamically adapt the number of backup workers is [11]. The PS uses a deep neural network to predict the time $T_{k,t}$ needed to collect new k gradients and greedily chooses k_t as the value that maximizes $k/T_{k,t}$. This neural network needs itself to be trained in advance for each cluster and each ML model to be learned. No result is provided in [11] about the duration of this additional training phase or its sensitivity to changes in the cluster and/or ML models. Moreover, results in [11] do not show a clear advantage of the proposed mechanism in comparison to the static setting suggested in [9] (see [11, Fig. 4]). Our experiments in Sect. IV confirm that indeed considering a gain proportional to k as in [11] is too simplistic (and leads to worse results than DBW).

III. Dynamic backup workers

The rationale behind our algorithm DBW is to adaptively select k_t in order to maximize $\frac{F(\boldsymbol{w}_t) - F(\boldsymbol{w}_{t+1})}{T_{k,t}}$, i.e., to greedily maximize the decrease of the empirical loss per time unit. We decide k_t just after computing $\boldsymbol{w}_t$. In the following subsections, we detail how both numerator and denominator can be estimated, and how they depend on k.

A. Empirical Loss Decrease

We assume that the empirical loss function $F(\boldsymbol{w})$ is L-smooth, i.e., it exists a constant L such that

$$\|\nabla F(\boldsymbol{w}') - \nabla F(\boldsymbol{w}'')\| \leq L \|\boldsymbol{w}' - \boldsymbol{w}''\|, \forall \boldsymbol{w}', \boldsymbol{w}''. \tag{4}$$

From (4) and (2) it follows (see [12, Sect. 4.1]):

$$\begin{aligned} \Delta F_t &\triangleq F(\boldsymbol{w}_t) - F(\boldsymbol{w}_{t+1}) \\ &\geq \eta \nabla F(\boldsymbol{w}_t)^\intercal \boldsymbol{g}_t - \frac{L\eta^2}{2} \|\boldsymbol{g}_t\|^2. \end{aligned} \tag{5}$$

In order to select k_t, DBW uses this lower bound as a proxy for the loss decrease. We consider then the expected value (over the possible choices for the mini-batches) of the right-hand side of (5). We call it the *gain* and denote by $\mathcal{G}_{k,t}$, i.e.,:

$$\mathcal{G}_{k,t} \triangleq \mathbb{E}\left[\eta \nabla F(\boldsymbol{w}_t)^\intercal \boldsymbol{g}_t - \frac{L\eta^2}{2} \|\boldsymbol{g}_t\|^2\right]. \tag{6}$$

Each stochastic gradient is an unbiased estimator of the full gradient, then $\mathbb{E}[\boldsymbol{g}_t] = \nabla F(\boldsymbol{w}_t)$ and

$$\mathbb{E}[\|\boldsymbol{g}_t\|^2] = \|\nabla F(\boldsymbol{w}_t)\|^2 + \mathbb{V}(\boldsymbol{g}_{i,t})/k, \tag{7}$$

where $\mathbb{V}(\boldsymbol{g}_{i,t})$ denotes the sum of the variances of the different components of $\boldsymbol{g}_{i,t}$, i.e., $\mathbb{V}(\boldsymbol{g}_{i,t}) \triangleq \sum_{l=1}^{d} \mathrm{Var}([\boldsymbol{g}_{i,t}]_l)$. Then, combining (6) and (7), $\mathcal{G}_{k,t}$ can be rewritten as

$$\mathcal{G}_{k,t} = \left(\eta - \frac{L\eta^2}{2}\right) \|\nabla F(\boldsymbol{w}_t)\|^2 - \frac{L\eta^2}{2} \frac{\mathbb{V}(\boldsymbol{g}_{i,t})}{k}. \tag{8}$$

When full batch gradient descent is used, the optimal learning rate is $\eta = 1/L$, because it maximizes the expected gain. With this choice of the learning rate, Eq. (8) becomes:

$$\mathcal{G}_{k,t} = \frac{\eta}{2} \left(\|\nabla F(\boldsymbol{w}_t)\|^2 - \frac{\mathbb{V}(\boldsymbol{g}_{i,t})}{k}\right). \tag{9}$$

Equation (9) shows that the gain increases as k increases. This corresponds to the fact that the more gradients are aggregated at the PS, the closer $-\boldsymbol{g}_t$ is to its expected value $-\nabla F(\boldsymbol{w}_t)$, i.e., to the steepest descent direction for the loss function. We also remark that the gain sensitivity to k depends on the relative ratio of $\mathbb{V}(\boldsymbol{g}_{i,t})$ and $\|\nabla F(\boldsymbol{w}_t)\|^2$, that keeps changing during the training (see for example Fig. 1). Correspondingly, we can expect that the optimal value of k will vary during the training process, even when computation and communication times do not change in the cluster. Experiments in Sect. IV confirm this is the case.

Computing the exact value of $\mathcal{G}_{k,t}$ would require the workers to process the whole dataset, leading to much longer iterations. We want rather to evaluate $\mathcal{G}_{k,t}$ with limited overhead for the workers. In what follows, we give our estimates for $\|\nabla F(\boldsymbol{w}_t)\|^2$ and $\mathbb{V}(\boldsymbol{g}_{i,t})$ to approximate $\mathcal{G}_{k,t}$ in (9). The derivation can be found in [13]. We have

$$\widehat{\mathbb{V}(\boldsymbol{g}_{i,t})} = \frac{1}{D} \sum_{v=1}^{D} \mathbb{V}\widehat{(\boldsymbol{g}_{i,t-v})}^{+}, \tag{10}$$

$$\|\nabla \widehat{F(\boldsymbol{w}_t)}\|^2 = \frac{1}{D} \sum_{v=1}^{D} \max\left(\|\boldsymbol{g}_t\|^2 - \frac{\widehat{\mathbb{V}(\boldsymbol{g}_{i,t})}^{+}}{k_t}, 0\right), \tag{11}$$

where $\widehat{\mathbb{V}(\boldsymbol{g}_{i,t})}^{+} = \sum_{l=1}^{d} \frac{1}{k_t - 1} \sum_{j=1}^{k_t} ([\boldsymbol{g}_{j,t} - \boldsymbol{g}_t]_l)^2$ and D is the number of past estimates considered. Combining (9), (10), and (11), the gain estimate is

$$\widehat{\mathcal{G}_{k,t}} = \frac{\eta}{2} \left(\|\nabla \widehat{F(\boldsymbol{w}_t)}\|^2 - \frac{\widehat{\mathbb{V}(\boldsymbol{g}_{i,t})}}{k}\right). \tag{12}$$

In Fig. 1, we show our estimates during one training process on the MNIST dataset (details in Sect. IV), where our algorithm (described below in Sect. III-C) is applied to dynamically choose k. The solid lines are the estimates given by (10), (11), and (12). The dashed lines present the exact values (we have instrumented our code to compute them). We can see from Figures 1(a) and 1(b) that the proposed estimates $\|\nabla \widehat{F(\boldsymbol{w}_t)}\|^2$ and $\widehat{\mathbb{V}(\boldsymbol{g}_{i,t})}$ are very accurate. Figure 1(c) compares the loss decrease ΔF_t (observed a posteriori) and $\widehat{\mathcal{G}_{k_t,t}}$. As expected $\widehat{\mathcal{G}_{k_t,t}}$ is a lower bound for ΔF_t, but the two quantities are almost proportional. This is promising, because if the lower bound $\widehat{\mathcal{G}_{k,t}}/T_{k,t}$ and the function $\Delta F_t/T_{k,t}$ were exactly proportional, their maximizers would coincide. Then, working on the lower bound, as we do, would not be an approximation.

B. Iteration Duration

In order to estimate $T_{k,t}$, the PS keeps collecting online time samples $\{t_{h,k}\}$ for $h, k = 1, ..., n$ that record the time the PS spends for receiving the k-th gradient, provided that it has waited h gradients at the previous iteration. Our estimators are described in [13].

C. Dynamic Choice of k_t

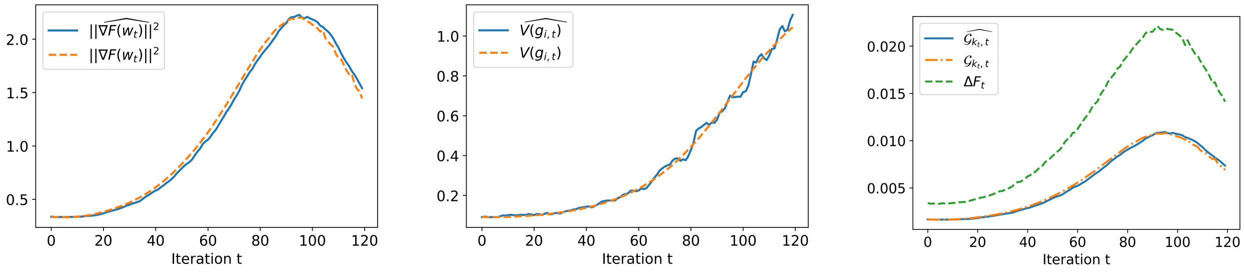
DBW rationale is to select the parameter k_t that maximizes the expected decrease of the loss function per time unit, i.e., $k_t = \arg\max_{1 \le k \le n} \frac{\widehat{\mathcal{G}_{k,t}}}{T_{k,t}}$. Moreover, we exploit additional information about local average loss at each worker [13].

IV. Experiments

We have implemented DBW in PyTorch [14] using the MPI backend. The experiments have been run on a CPU/GPU cluster. In order to have a fine control over the round trip times, our code can generate computation and communication times according to different distributions (uniform, exponential, Pareto, etc.) or read them from a trace provided as input file.

In all experiments DBW achieves nearly optimal performance in terms of convergence time, and sometimes it even outperforms the optimal static setting, that is found through an exhaustive offline search over all values $k \in \{1, ..., n\}$. We also compare DBW with a variant where the gain $\mathcal{G}_{k,t}$ is not estimated as in (12), but it equals the number of aggregated gradients k, as proposed in [11]. We call this variant blind DBW (B-DBW), because it is oblivious to the current state of the training.

We evaluated DBW, B-DBW, and different static settings for k on MNIST, a dataset with 60000 images portraying handwritten digits. For MNIST, we trained a



(a) Gradient norm

(b) Gradient variance

(c) Loss decrease

Fig. 1: Estimation of the loss decrease. MNIST, $n = 16$ workers, batch size $B = 500$, learning rate $\eta = 0.01$, estimates computed over the last $D = 5$ iterations.

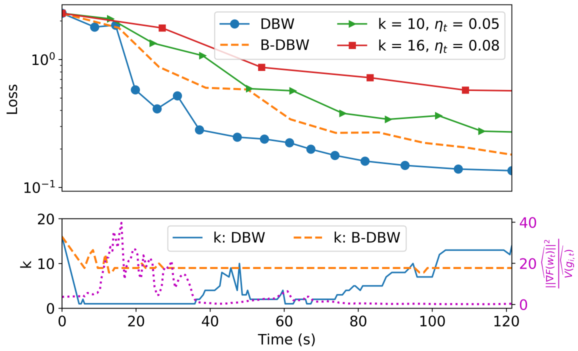


Fig. 2: Loss versus time. MNIST, batch size $B = 500$, $n = 16$ workers, estimates computed over the last $D = 5$ iterations, proportional rule with $\eta(k) = 0.005k$, round trip times follow shifted exponential distribution $0.3 + 0.7\text{Exp}(1)$.

neural network with two convolutional layers with 5×5 filters and two fully connected layers. The loss function was the cross-entropy one.

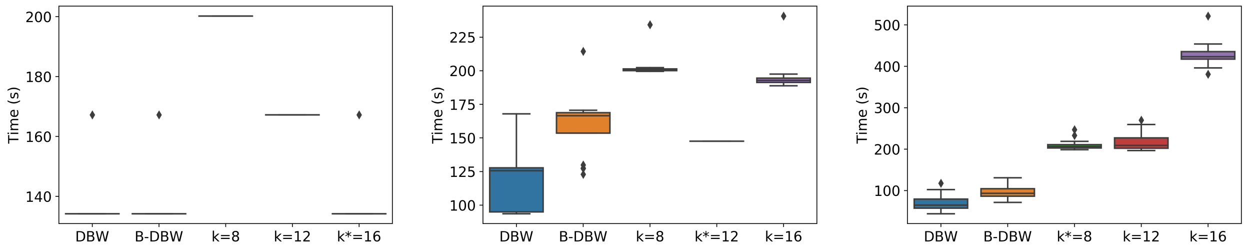
The learning rate is probably the most critical hyper-parameter in ML optimization problems. The rule of thumb proposed in the seminal paper [9] is to set the learning rate proportional to k, i.e., $\eta(k) \propto k$. This corresponds to the standard recommendation to have the learning rate proportional to the (aggregate) batch size [15], [16].

Figure 2 shows, for a single run of the training process, the evolution of the loss over time and the corresponding choices of k_t for the two dynamic algorithms. The optimal static setting is $k^* = 10$. We can see that DBW achieves the fastest convergence by using a different value of k in different stages of the training process. In fact, as we have discussed after introducing (9), the effect of k on the gain depends on the module of the gradient and on the variability of the local gradients. In the bottom subplot, the dotted line shows how their ratio varies during the training process. Up to iteration 38, $\mathbb{V}(\boldsymbol{g}_{i,t})$ is negligible in comparison to $\|\nabla F(\boldsymbol{w}_t)\|^2$. DBW then selects small values for k_t loosing a bit in terms of the gain, but significantly speeding up the duration of each iteration by only waiting for the fastest workers. As the parameter vector approaches a local minimum, $\|\nabla F(\boldsymbol{w}_t)\|^2$ approaches zero, and the gain becomes more and more sensitive to k, so that DBW progressively increases k_t up to reach $k_t = n = 16$ as shown by the solid line. On the contrary B-DBW (the dashed line) selects most of the time $k_t = 9$ with some variability due to the randomness of the estimates $\widehat{T_{k,t}}$.

A. Round trip time effect

In this subsection we study the effect of round trip times variability. A round trip time includes the time to transmit the parameter vector from the PS to the worker, the time to compute the gradient, and the time to transfer the gradient to the PS. We assume that round trip times are i.i.d. according to a shifted exponential random variable $1 - \alpha + \alpha \times \text{Exp}(1)$, where $0 \leq \alpha \leq 1$. This choice allows us to easily tune the variability of the round trip times by changing α. When $\alpha = 0$, all gradients arrive at the same time at the PS, so that the PS should always aggregate all of them. As α changes from 0 to 1, the variance of the round trip times increases, and waiting for $k < n$ gradients becomes advantageous.

Figure 3 compares the time needed to reach a training loss smaller than 0.2 for the two dynamic algorithms and the static settings $k = 16$, $k = 12$, and $k = 8$, that are optimal respectively for $\alpha = 0$, $\alpha = 0.2$, $\alpha = 1$. For each of them, we carried out 20 independent runs with different seeds. We find that our dynamic algorithm achieves the fastest convergence in all three scenarios, it is even 1.2x faster and 3x faster than the optimal static settings for



(a) $\alpha = 0$

(b) $\alpha = 0.2$

(c) $\alpha = 1$

Fig. 3: Effect of round trip time distribution. MNIST, $n = 16$ workers, batch size $B = 500$, estimates computed over the last $D = 5$ iterations, proportional rule for $\eta(k)$ in static settings where $\eta(k) = 0.005k$.

$\alpha = 0.2$ and $\alpha = 1$. There are two factors that determine this observation. First, as discussed for Fig. 2, there is no unique optimal value of k to be used across the whole training process, and DBW manages to select the most indicated value in different stages of the training process. Second, DBW takes advantage of a larger learning rate. Both factors play a role. For example, if we focus on Fig. 3(c), the learning rate for DBW is twice faster than that for $k = 8$, but DBW is on average 3x faster. Then, adapting k achieves an additional 1.5x improvement. The importance of capturing the dynamics of the optimization process is again also evident by comparing DBW with B-DBW. While B-DBW takes advantage of a higher learning rate as well, it does not perform as well as our solution DBW.

V. Conclusions and Acknowledgement

In this paper, we have shown that the number of backup workers needs to be adapted at run-time and the correct choice is inextricably bounded, not only to the cluster's configuration and workload, but also to the stage of the training. We have proposed a simple algorithm DBW that, without priori knowledge about the cluster or the problem, achieves good performance across a variety of scenarios, and even outperforms in some cases the optimal static setting.

As a future research direction, we want to extend the scope of DBW to dynamic resource allocation, e.g., by automatically releasing computing resources if $k_t < n$ and the fastest k_t gradients are always coming from the same set of workers. In general, we believe that distributed systems for ML are in need of adaptive algorithms in the same spirit of the utility-based congestion control schemes developed in our community starting from the seminal paper [17]. As our work points out, it is important to define new utility functions that take into account the learning process. Adaptive algorithms are even more needed in the federated learning scenario [18], where ML training is no more relegated to the cloud, but it occurs in the wild over the whole internet. Our paper shows that even simple algorithms can provide significant performance improvements.

This work has been carried out in the framework of a common lab agreement between Inria and Nokia Bell Labs. We thank Alain Jean-Marie for his suggestions.

References

[1] K. Canini *et al.*, "Sibyl: A system for large scale supervised machine learning," 2014, technical talk.

[2] A. Harlap *et al.*, "Addressing the straggler problem for iterative convergent parallel ML," in *7th ACM SoCC*, 2016, pp. 98–111.

[3] G. Neglia *et al.*, "The role of network topology for distributed machine learning," in *INFOCOM*, 2019, pp. 2350–2358.

[4] Y. Bao *et al.*, "Deep learning-based job placement in distributed machine learning clusters," in *INFOCOM*, 2019, pp. 505–513.

[5] C. Chen *et al.*, "Round-robin synchronization: Mitigating communication bottlenecks in PS," in *INFOCOM*, 2019, pp. 532–540.

[6] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *11th USENIX OSDI*, 2014, pp. 583–598.

[7] C. Karakus *et al.*, "Straggler mitigation in distributed optimization through data encoding," in *Proc. of NIPS*, 2017, pp. 5434–5442.

[8] S. Li *et al.*, "Near-optimal straggler mitigation for distributed gradient methods," in *IEEE IPDPS*, 2018, pp. 857–866.

[9] J. Chen *et al.*, "Revisiting distributed synchronous SGD," in *ICLR Workshop Track*, 2016.

[10] "DBW," https://gitlab.inria.fr/chxu/dbw.

[11] M. Teng *et al.*, "Bayesian distributed stochastic gradient descent," in *Advances in NIPS 31*, 2018, pp. 6378–6388.

[12] L. Bottou *et al.*, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[13] C. Xu *et al.*, "Dynamic backup workers for parallel machine learning," 2020, arXiv:2004.14696.

[14] "PyTorch," https://pytorch.org/.

[15] P. Goyal *et al.*, "Accurate, large minibatch SGD: training imagenet in 1 hour," *CoRR*, vol. abs/1706.02677, 2017.

[16] S. L. Smith *et al.*, "Don't decay the learning rate, increase the batch size," in *ICLR*, 2018.

[17] F. P. Kelly *et al.*, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research society*, vol. 49, no. 3, pp. 237–252, 1998.

[18] J. Konecný *et al.*, "Federated optimization: Distributed optimization beyond the datacenter," in *NIPS (workshop)*, 2015.