



**HAL**  
open science

# Dynamic Backup Workers for Parallel Machine Learning

Chuan Xu, Giovanni Neglia, Nicola Sebastianelli

► **To cite this version:**

Chuan Xu, Giovanni Neglia, Nicola Sebastianelli. Dynamic Backup Workers for Parallel Machine Learning. 2020. hal-03044199

**HAL Id: hal-03044199**

**<https://inria.hal.science/hal-03044199>**

Preprint submitted on 7 Dec 2020

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Dynamic Backup Workers for Parallel Machine Learning

Chuan Xu, Giovanni Neglia, Nicola Sebastianelli

*Inria, Université Côte d'Azur, Sophia Antipolis, France*

---

## Abstract

The most popular framework for distributed training of machine learning models is the (synchronous) parameter server (PS). This paradigm consists of  $n$  workers, which iteratively compute updates of the model parameters, and a stateful PS, which waits and aggregates all updates to generate a new estimate of model parameters and sends it back to the workers for a new 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 correct choice of the number  $b$  of backup workers depends on the cluster configuration and workload, but also (as we show in this paper) on the hyper-parameters of the learning algorithm and 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.

*Keywords:* Machine learning, parameter server, gradient methods, distributed systems, stragglers.

---

## 1. Introduction

Already in 2014, state-of-the-art machine learning models counted hundreds of billions of parameters and required processing hundreds of terabytes through thousands of cores [1]. As models and datasets keep becoming larger, the need for efficient distributed solutions becomes even more urgent. These distributed systems are different from those used for traditional applications like transaction processing or data analytics, because of statistical and algorithmic characteristics unique to ML programs, like error tolerance, structural dependencies, and non-uniform convergence of parameters [2]. Currently, their operation 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 [3, 4, 5, 6, 7, 8, 9]) is devoted to design adaptive algorithms for a more effective use of computing resources for ML training.

For distributed ML training, there are two popular frameworks, the parameter server (PS) [10] and AllReduce (AR) [11, 12]. In PS, a stateful parameter server maintains the current version of the model parameters and broadcasts them to the workers (computing units e.g., GPUs). Every worker then computes “delta” updates of the parameters, e.g., through a gradient descent step. These updates are then aggregated by the PS in a synchronized way and combined with its current state to produce a new estimate of the optimal parameter vector. As the server may become a communication bottleneck, aggregation can be implemented in a distributed way through an AllReduce collective operation [13]. For example, in Ring-AllReduce [14] with  $n$  workers,  $2(n - 1)$  synchronized communications are required with  $\mathcal{O}(1)$  data transmitted per worker. However, both the PS and AR are sensitive to *stragglers* [15, 16, 17, 18, 19], i.e., “workers that are **randomly**

---

*Email addresses:* [chuan.xu@inria.fr](mailto:chuan.xu@inria.fr) (Chuan Xu), [giovanni.neglia@inria.fr](mailto:giovanni.neglia@inria.fr) (Giovanni Neglia), [nicola.sebastianelli@inria.fr](mailto:nicola.sebastianelli@inria.fr) (Nicola Sebastianelli)

21 *slowed down due to resource contention, background OS activities, garbage collection, and (for ML tasks)*  
22 *stopping criteria calculations” [3].*

23 To mitigate the stragglers problem, coding techniques have been proposed both for PS [20, 21, 22, 23, 24,  
24 25, 18] and AR [26, 19] frameworks. The main idea behind is that each worker performs some additional  
25 computation and codes its update in an opportune way, so that only a subset of the tasks is needed to  
26 recover the full information and to proceed to the next iteration. Hence, the system does not need to  
27 wait for the stragglers. Coding techniques are particularly helpful when data distribution across workers  
28 is heterogeneous [27] as it happens in federated learning [28]. In a cluster, all workers have access to the  
29 whole dataset or to a random sample of it, hence the advantage of coding is significantly reduced, and when  
30 computation time is larger than communication time, coding is even less beneficial [20]. In these settings, the  
31 additional overhead introduced by coding techniques may not be justified.

32 Alternative approaches to deal with stragglers are based on load-aware and interference-aware resource  
33 scheduling to monitor and avoid stragglers [29, 6]. These techniques are effective only if stragglers are  
34 *persistent*, i.e., the same workers are slow over a relatively long time period, but straggler effects often occur  
35 over short timescale.

36 Another possibility is to relax the full synchronization requirement avoiding to collect information from all  
37 workers before computing the new model parameters. One solution is to let the PS operate asynchronously,  
38 updating the parameter vector as soon as it receives the result of a single worker [30, 31]. While this approach  
39 increases system throughput (parameter updates per time unit), workers operate in general on stale versions  
40 of the parameter vector slowing and, in some cases, even preventing convergence to the optimal model [32].  
41 Another solution is to apply decentralized learning methods, where there is no central server, but workers  
42 communicate only with their neighbours on an opportune communication graph [33, 34, 35, 36]. When the  
43 graph is sparse and the stragglers behave in a non-persistent way, such methods work well enjoying high  
44 system throughput and guaranteed convergence [37, 38, 39]. However, persistent stragglers can still slow  
45 down dramatically the throughput performance.

46 In the PS architecture, a simple solution to mitigate the effect of stragglers without jeopardizing convergence,  
47 is to rely on backup workers [40, 27]: instead of waiting for the updates from all workers (say it  $n$ ), the PS  
48 waits for the fastest  $k$  out of  $n$  updates to proceed to the next iteration. The remaining  $b \triangleq n - k$  workers  
49 are called backup workers.<sup>1</sup> Experiments on Google cluster with  $n = 100$  workers show that a few backup  
50 workers (4–6) can reduce the training time by 30% in comparison to the synchronous PS and by 20% in  
51 comparison to the asynchronous PS [40].

52 The number of backup workers  $b$  has a double effect on the convergence speed. The larger  $b$  is, the faster  
53 each iteration is, because the PS needs to wait less inputs from the workers. At the same time, the PS  
54 aggregates less information, so the model update is noisier and more iterations are required to converge.  
55 Currently, the number of backup workers is configured manually through some experiments, before the actual  
56 training process starts. However, the optimal static setting is highly sensitive to the cluster configuration  
57 (e.g., GPU performances and their connectivity) as well as to its instantaneous workload. Both cluster  
58 configuration and workload may be unknown to the users (specially in a virtualized cloud setting) and may  
59 change as new jobs arrive/depart from the cluster. Moreover, in this paper we show that the choice of the  
60 number of backup workers 1) should depend also on hyper-parameters<sup>2</sup> like the batch size, and 2) should  
61 change during the training itself (!) as the loss function approaches a (local) minimum. Therefore, the  
62 static configuration of backup workers does not only require time-consuming experiments, but is particularly  
63 inefficient and fragile.

64 In this paper we propose the algorithm DBW (for Dynamic Backup Workers) that dynamically adapts  
65 the number of backup workers during the training process without prior knowledge about the cluster or

---

<sup>1</sup>We stick to the name used in the original paper [40], even if it is somewhat misleading, because backup workers do not replace other workers when needed. In fact all workers operate identically, and who are the backup workers change from one iteration to the other depending on their execution times at that specific iteration.

<sup>2</sup>An hyper-parameter is a parameter of the learning algorithm (and not of the model), but it can still influence the final model learned.

66 the optimization problem. Our algorithm identifies the sweet spot between the two contrasting effects of  $b$   
 67 (reducing the duration of an iteration and increasing the number of iterations for convergence), by maximizing  
 68 at each iteration the decrease of the loss function *per time unit*.

69 This paper extends our conference submission [41] and is organized as follows. Sect. 2 provides relevant  
 70 background and introduces the notation. Sect. 3 illustrates the different components of our algorithm DBW  
 71 with their respective preliminary assessments. DBW is then evaluated on ML problems in Sect. 4. The results  
 72 show that DBW is robust to different cluster environments and different hyper-parameters' settings. DBW  
 73 does not only remove the necessity to configure an additional parameter ( $b$ ) through costly experiments, but  
 74 also reduce the training time by a factor as large as 3 in comparison to the best static configuration. Sect. 5  
 75 concludes the paper and discusses future research directions. The code of our implementation is available  
 76 online [42].

## 77 2. Background and notation

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

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

80 where  $f(x_l, \mathbf{w})$  is the loss of the model  $\mathbf{w}$  on the datapoint  $x_l$ . For example, in supervised learning, each  
 81 point of the dataset is a pair  $x_l = (\chi_l, y_l)$ , consisting of an input object  $\chi_l$  and a desired output value  $y_l$ . In  
 82 the standard linear regression method  $\chi_l \in \mathbb{R}^d$ ,  $y_l \in \mathbb{R}$ , the input-output function is a linear one ( $\hat{y}_l = \chi_l^\top \mathbf{w}$ )  
 83 and the loss function is the mean squared error  $(\chi_l^\top \mathbf{w} - y_l)^2$ . More complex models like neural networks look  
 84 for an input-output mapping in a much larger and more flexible family of functions, but they are trained  
 85 solving an optimization problem like (1).

86 The standard way to solve Problem 1 is to use an iterative gradient method. Let  $n$  be the number of  
 87 workers (e.g., GPUs) available. In a synchronous setting without backup workers, at each iteration  $t$  the PS  
 88 sends the current estimate of the parameter vector  $\mathbf{w}_t$  to all workers. Each worker computes then a stochastic  
 89 gradient on a random mini-batch of size  $B$  ( $\leq S$ ) drawn from its local dataset. We assume each worker has  
 90 access to the complete dataset  $\mathbb{X}$  as it is reasonable in the cluster setting that we consider. Each worker sends  
 91 the stochastic gradient back to the PS. We denote by  $\mathbf{g}_{i,t}$  the  $i$ -th worker gradient received by the PS at  
 92 iteration  $t$ , i.e.,

$$\mathbf{g}_{i,t} = \frac{1}{B} \sum_{x \in \mathbb{B}_i} \nabla f(x, \mathbf{w}_t), \quad (2)$$

93 and  $\mathbb{B}_i \subseteq \mathbb{X}$  is the random minibatch of size  $B$  on which the gradient has been computed. Once  $n$  gradients  
 94 are received, the PS computes the average gradient

$$\mathbf{g}_t = \frac{1}{n} \sum_{i=1}^n \mathbf{g}_{i,t},$$

95 and updates the parameter vector as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{g}_t, \quad (3)$$

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

97 When  $b$  backup workers are used [40], the PS only waits for the first  $k = n - b$  gradients and then evaluates  
 98 the average gradient as

$$\mathbf{g}_t = \frac{1}{k} \sum_{i=1}^k \mathbf{g}_{i,t}. \quad (4)$$

99 In our dynamic algorithm (Sect. 3), the value of  $k$  is no longer static but changes in an adaptive manner  
100 from one iteration to the other, ensuring faster convergence speed. We denote by  $k_t$  the number of gradients  
101 of  $\mathbf{w}_t$  the PS needs to wait for at iteration  $t$ , and by  $T_{i,t}$  the time interval between the update of the parameter  
102 vector  $\mathbf{w}_t$  at the PS and the reception of the  $i$ -th gradient  $\mathbf{g}_{i,t}$ .

103 The general backup-workers scheme can be implemented in different ways with quite different performance.  
104 When implementing the backup workers scheme, there are two general ways to synchronize the PS and the  
105 workers: either the PS *pushes* the updated parameter vector to workers or the workers *pull* the most updated  
106 parameter vector from the PS.

107 *Pull (PI)*. Whenever available to perform a new computation, a worker pulls the most updated parameter  
108 vector from the PS. Google’s framework for distributed ML—TensorFlow 1.x [43]—implements PI through a  
109 shared blocking FIFO queue of size  $n$  where the PS enqueues  $n$  copies of tokens indicating the corresponding  
110 iteration number. Whenever a worker becomes idle, it dequeues the token from the queue and retrieves the  
111 parameter vector directly from the PS.<sup>3</sup>

112 *Push & Interrupt (PsI)*. After the PS updates the new parameter vector  $\mathbf{w}$ , it pushes  $\mathbf{w}$  to all workers, which  
113 interrupt any ongoing computation to start computing a new gradient at  $\mathbf{w}$ . Interrupts can be implemented  
114 in different ways. For example, in [44, Algo. 2], the main thread at each worker creates a specific thread for  
115 each gradient computation and keeps listening for a new parameter vector. Once the worker receives the new  
116 one from PS, the computing thread is killed. However, the overhead of online creating/destroying threads is  
117 not negligible since it requires run-time memory allocation and de-allocation, which may even slow down the  
118 system [45]. In [46], the same thread performs the computation but periodically checks for new parameter  
119 vectors from the PS. When the worker receives a new parameter vector, it stops its ongoing computation.  
120 The performance of this interrupt mechanism depends on how often workers listen for messages from PS.

121 *Push & Wait (PsW)*. The PS pushes the new parameter vector to each worker as in PsI, but the worker  
122 completes its current computation before dequeuing the most recent parameter vector from a local queue.  
123 PsW can be easily implemented using MPI non-blocking communication package [18] or the FIFO queue  
124 provided in TensorFlow [47].

125 Our algorithm works with any of the variants listed above, with minor adaptations. We have implemented  
126 and tested it both with PsI and PsW in the PyTorch framework [48]. Results are similar, therefore, in what  
127 follows, we refer only to PsW.

128 To the best of our knowledge, there are two other proposals to dynamically adapt the number of backup  
129 workers [44, 27]. Both consider a PsI approach. In [44] the PS uses a deep neural network to predict  
130 the time  $T_{k,t}$  needed to collect  $k = 1, 2, \dots, n$  new gradients. It then greedily chooses  $k_t$  as the value that  
131 maximizes  $k/T_{k,t}$ . This neural network for time series forecasting needs itself to be trained in advance for  
132 each cluster and each ML model to be learned. No result is provided in [44] about the duration of this  
133 additional training phase or its sensitivity to changes in the cluster and/or ML models. Our algorithm DBW  
134 also selects  $k_t$  to maximize a similar ratio, but 1) replaces the numerator by the expected decrease of the loss  
135 function, 2) uses a simple estimator for  $T_{k,t}$ , that does not require any preliminary training. Moreover, results  
136 in [44] do not show a clear advantage of the proposed mechanism in comparison to the static setting suggested  
137 in [40] (see [44, Fig. 4]). Our experiments in Sect. 4 confirm that indeed considering a gain proportional to  $k$   
138 as in [44] is too simplistic (and leads to worse results than DBW). The recent paper [27] proposes ADASYNC  
139 that selects  $k_t$  to minimize the average expected squared norm of the gradients over a time horizon. ADASYNC  
140 relies on an upper bound for the expected squared norm of the gradients and analytical formulas for  $T_{k,t}$  for  
141 specific distributions of the computation times—they only develop the case for shifted exponential random  
142 variables. Finding the optimal  $k_t$  would require to know or estimate at run-time some quantities like the

---

<sup>3</sup>We describe what appears to be an inefficient implementation. The parameter vector retrieved by the worker may correspond to a more recent iteration than what indicated in the token. Nevertheless, the corresponding gradient is still associated to the old iteration and then will be discarded at the PS. The worker may start then a computation that is already known to be useless!

143 Lipschitz constant or noise variance. ADASync instead determines  $k_t$  by solving an approximate quadratic  
 144 equation that only depends on the current loss. On the contrary, DBW estimates the different quantities  
 145 online without prior information about the distribution of the computation times, and it is then able to adapt  
 146 to changes in the cluster, e.g., due to dynamic resource allocation (Sect. 4.3). When computation times are  
 147 distributed according to a shifted exponential distribution, our experiments show that DBW trains faster  
 148 than ADASync when computation variability is small (Sect. 4.4).

149 Our approach to estimate the loss decrease as a function of  $k$  is inspired by the work [49] which evaluates  
 150 the loss decrease as a function of the batch size. In fact, aggregating  $k$  gradients, each computed on a  
 151 mini-batch of  $B$  samples, is almost equivalent to compute a single gradient on a mini-batch of  $kB$  samples.

152 While our algorithm adapts the number of backup workers  $b$  given an available pool of  $n$  workers, the  
 153 authors of [4] proposes a reinforcement learning algorithm to adapt  $n$  in order to minimize the training time  
 154 under a budget constraint. This algorithm and DBW are then complementary: once selected  $n$  with the  
 155 approach in [4], DBW can be applied to tune the number of backup workers.

### 156 3. Dynamic backup workers

157 The rationale behind our algorithm DBW is to adaptively select  $k_t$  in order to maximize  $\frac{F(\mathbf{w}_t) - F(\mathbf{w}_{t+1})}{T_{k,t}}$ ,  
 158 i.e., to greedily maximize the decrease of the empirical loss per time unit. We decide  $k_t$  just after the update  
 159 of  $\mathbf{w}_t$ .<sup>4</sup> In the following subsections, we detail how both numerator and denominator can be estimated, and  
 160 how they depend on  $k$ . The notation is listed in Table 1.

$t$	iteration number
$n$	number of workers
$\mathbf{w}_t$	parameter vector at iteration $t$
$F$	(global) loss function to minimize
$B$	batch size
$\eta$	learning rate
$L$	Lipschitz smoothness constant of $F$
$\mathbf{g}_{i,t}$	$i^{\text{th}}$ stochastic gradient PS receives at iteration $t$
$\mathbb{V}(\mathbf{g}_{i,t})$	variance of $\mathbf{g}_{i,t}$
$k_t$	number of stochastic gradients PS waits for at iteration $t$
$\mathbf{g}_t$	average gradient at iteration $t$
$\mathcal{G}_{k,t}$	gain (expected loss decrease) if PS receives $k$ gradients
$T_{k,t}$	time between $\mathbf{w}_t$ update and $\mathbf{g}_{k,t}$ reception at PS
$\mathbf{t}_{h,i,t}$	time between $\mathbf{w}_t$ update and $\mathbf{g}_{i,t}$ reception at PS when PS has waited for $h$ gradients at iteration $t - 1$
$\mathcal{T}_{h,k}$	random variable from which $\mathbf{t}_{h,k,t}$ values are assumed to be sampled
$\mathbb{T}_{h,k,t}$	set of $\mathbf{t}_{h,k,t'}$ samples available up to iteration $t$

Table 1: Notation

#### 161 3.1. Empirical Loss Decrease

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

$$\|\nabla F(\mathbf{w}') - \nabla F(\mathbf{w}'')\| \leq L\|\mathbf{w}' - \mathbf{w}''\|, \forall \mathbf{w}', \mathbf{w}'' . \quad (5)$$

163 Smoothness is a standard assumption in convergence results of gradient methods (see for example [50, 51]).  
 164 In our experiments we show DBW reduces the convergence time also when the loss is not a smooth function.

<sup>4</sup>It is possible in principle to refine the choice of  $k_t$  upon the arrival of the first gradients of  $\mathbf{w}_t$ .

165 From (5) and (3) it follows (see [51, Sect. 4.1] for a proof):

$$\Delta F_t \triangleq F(\mathbf{w}_t) - F(\mathbf{w}_{t+1}) \geq \eta \nabla F(\mathbf{w}_t)^\top \mathbf{g}_t - \frac{L\eta^2}{2} \|\mathbf{g}_t\|^2. \quad (6)$$

166 In order to select  $k_t$ , DBW uses this lower bound as a proxy for the loss decrease. We note, however, that  $\mathbf{g}_t$   
 167 depends on the value of  $k_t$  (see (4)) and the random mini-batches drawn at the workers. So at the moment  
 168 to decide for  $k_t$ ,  $\mathbf{g}_t$  is a random variable. We consider then the expected value (over the possible choices for  
 169 the mini-batches) of the right-hand side of (6). We call it the *gain* and denote by  $\mathcal{G}_{k,t}$ , i.e.,:

$$\mathcal{G}_{k,t} \triangleq \mathbb{E} \left[ \eta \nabla F(\mathbf{w}_t)^\top \mathbf{g}_t - \frac{L\eta^2}{2} \|\mathbf{g}_t\|^2 \right]. \quad (7)$$

170 Each stochastic gradient is an unbiased estimator of the full gradient, then  $\mathbb{E}[\mathbf{g}_t] = \nabla F(\mathbf{w}_t)$ . Moreover, for  
 171 any random variable  $X$ , it holds  $\mathbb{E}[X^2] = \mathbb{E}[X]^2 + \text{Var}(X)$ . Applying this relation to each of the component  
 172 of the vector  $\mathbf{g}_t$ , and then summing up, we obtain:

$$\mathbb{E}[\|\mathbf{g}_t\|^2] = \|\nabla F(\mathbf{w}_t)\|^2 + \mathbb{V}(\mathbf{g}_{i,t})/k, \quad (8)$$

173 where  $\mathbb{V}(\mathbf{g}_{i,t})$  denotes the sum of the variances of the different components of  $\mathbf{g}_{i,t}$ , i.e.,  $\mathbb{V}(\mathbf{g}_{i,t}) \triangleq \sum_{l=1}^d \text{Var}([\mathbf{g}_{i,t}]_l)$ .  
 174 Notice that  $\mathbb{V}(\mathbf{g}_{i,t})$  does not depend on  $i$ , because each worker has access to the complete dataset. Then,  
 175 combining (7) and (8),  $\mathcal{G}_{k,t}$  can be rewritten as

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

176 Equation (9) shows that the gain increases as  $k$  increases. This corresponds to the fact that the more  
 177 gradients are aggregated at the PS, the closer the stochastic gradient  $-\mathbf{g}_t$  is to its expected value  $-\nabla F(\mathbf{w}_t)$ ,  
 178 i.e., to the steepest descent direction for the loss function. We also remark that the gain sensitivity to  $k$   
 179 depends on the relative ratio of  $\mathbb{V}(\mathbf{g}_{i,t})$  and  $\|\nabla F(\mathbf{w}_t)\|^2$ , that keeps changing during the training (see for  
 180 example Fig. 1). Correspondingly, we can expect that the optimal value of  $k$  will vary during the training  
 181 process, even when computation and communication times do not change in the cluster. Experiments  
 182 in Sect. 4 confirm this point.

183 Computing the exact value of  $\mathcal{G}_{k,t}$  would require the workers to process the whole dataset, leading to  
 184 much longer iterations. We want rather to evaluate  $\mathcal{G}_{k,t}$  with limited overhead for the workers. In what  
 185 follows, we discuss how to estimate  $\|\nabla F(\mathbf{w}_t)\|^2$ ,  $\mathbb{V}(\mathbf{g}_{i,t})$ , and  $L$  to approximate  $\mathcal{G}_{k,t}$  in (9). We first provide  
 186 estimators that use information available *at the end* of iteration  $t$ , i.e., after  $k_t$  has been selected and the  $k_t$   
 187 fastest gradients have been received. Then, we build from these estimators new ones, that can be computed  
 188 *at the beginning* of the iteration  $t$  and then can be used to select  $k_t$ . Given a quantity  $\theta_t$  to be estimated at  
 189 iteration  $t$ , we denote the first estimator as  $\widehat{\theta}_t^+$  and the second one as  $\widehat{\theta}_t^-$ .

190 We start by estimating  $\mathbb{V}(\mathbf{g}_{i,t})$  through the usual unbiased estimator for the variance:

$$\widehat{\mathbb{V}(\mathbf{g}_{i,t})}^+ = \sum_{l=1}^d \frac{1}{k_t - 1} \sum_{j=1}^{k_t} ([\mathbf{g}_{j,t} - \mathbf{g}_t]_l)^2. \quad (10)$$

191 It is possible to have more precise estimates (even when  $k_t = 1$ ), if each worker can estimate  $\mathbb{V}(\nabla f(x, \mathbf{w}_t))$   
 192 from its mini-batch. As GPUs' low-level APIs do not provide access to such information, we do not further  
 193 develop the corresponding formulas here.

194 Next, we study the estimator of  $\|\nabla F(\mathbf{w}_t)\|^2$ . First, we can trivially use  $\|\mathbf{g}_t\|^2$  to estimate  $\mathbb{E}[\|\mathbf{g}_t\|^2]$ ,  
 195 i.e.,  $\widehat{\mathbb{E}[\|\mathbf{g}_t\|^2]}^+ = \|\mathbf{g}_t\|^2$ . Since  $\|\nabla F(\mathbf{w}_t)\|^2 = \mathbb{E}[\|\mathbf{g}_t\|^2] - \mathbb{V}(\mathbf{g}_{i,t})/k_t$  (from (8)), we can estimate  $\|\nabla F(\mathbf{w}_t)\|^2$  as  
 196 follows

$$\|\nabla F(\mathbf{w}_t)\|^2 = \max \left( \widehat{\mathbb{E}[\|\mathbf{g}_t\|^2]}^+ - \frac{\widehat{\mathbb{V}(\mathbf{g}_{i,t})}^+}{k_t}, 0 \right), \quad (11)$$

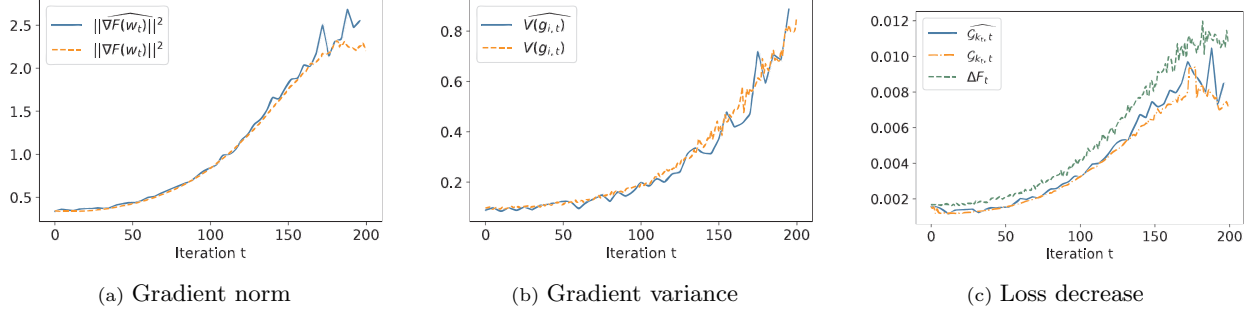


Figure 1: Estimation of the loss decrease. MNIST,  $n = 16$  workers, batch size  $B = 500$ , learning rate  $\eta = 0.01$ , estimates computed over the last  $D = 5$  iterations.

197 where the max operation guarantees non-negativity of the estimate.

198 To estimate  $L$ , we need also to estimate  $\mathcal{G}_{k_{t-1}, t-1}$ . In most of the existing implementations of distributed  
 199 gradient methods for ML (including PyTorch’s one), each worker  $i$  can send to the PS the local average loss  
 200 computed on its mini-batch. The PS can thus estimate the loss as

$$\widehat{F}_t = \frac{1}{k_t} \sum_{i=1}^{k_t} \frac{1}{B} \sum_{x \in \mathbb{B}_i} h(x, \mathbf{w}_t).$$

Thus, we have

$$\widehat{\mathcal{G}}_{k_{t-1}, t-1}^+ = \widehat{F}_{t-1} - \widehat{F}_t,$$

and substituting it to the left of (9), we get:

$$\widehat{L}_t^+ = \frac{2 \left( \eta \|\nabla \widehat{F}(\mathbf{w}_{t-1})\|^2 - \widehat{\mathcal{G}}_{k_{t-1}, t-1}^+ \right)}{\eta^2 \left( \|\nabla \widehat{F}(\mathbf{w}_{t-1})\|^2 + \mathbb{V}(\widehat{\mathbf{g}}_{i, t-1}) / k_{t-1} \right)} \quad (12)$$

201 Estimates in (10), (11) and (12) cannot be computed at the beginning of iteration  $t$ , but it is possible to  
 202 compute them for earlier iterations, and use these past estimates to predict the future value. DBW simply  
 203 averages the past  $D$  estimates (or the first  $t - 1$  if  $t \leq D$ ), i.e.,

$$\mathbb{V}(\widehat{\mathbf{g}}_{i, t}) = \frac{1}{D} \sum_{v=1}^D \mathbb{V}(\widehat{\mathbf{g}}_{i, t-v})^+, \quad (13)$$

$$\|\nabla \widehat{F}(\mathbf{w}_t)\|^2 = \frac{1}{D} \sum_{v=1}^D \|\nabla \widehat{F}(\mathbf{w}_{t-v})\|^2, \quad (14)$$

$$\widehat{L}_t = \frac{1}{D} \sum_{v=1}^D \widehat{L}_{t-v}^+. \quad (15)$$

204 Combining (9), (13), (14) and (15), the estimate of the gain is

$$\widehat{\mathcal{G}}_{k, t} = \left( \eta - \frac{\widehat{L}_t \eta^2}{2} \right) \|\nabla \widehat{F}(\mathbf{w}_t)\|^2 - \frac{\widehat{L}_t \eta^2}{2} \frac{\mathbb{V}(\widehat{\mathbf{g}}_{i, t})}{k}. \quad (16)$$

205 In Fig. 1 and Fig. 2, we show our estimates during one training process on the MNIST and CIFAR10  
 206 dataset respectively (details in Sect. 4), where our algorithm (described in Sect. 3.3) is applied to dynamically



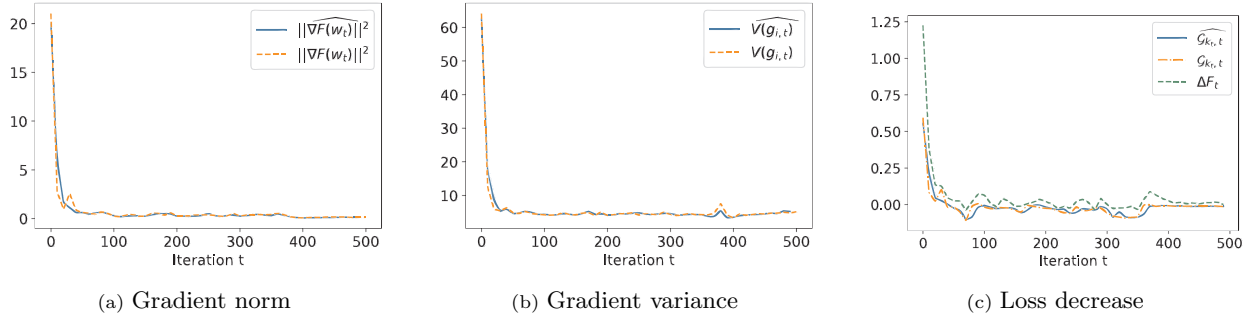


Figure 2: Estimation of the loss decrease. CIFAR10,  $n = 16$  workers, batch size  $B = 256$ , learning rate  $\eta = 0.05$ , estimates computed over the last  $D = 5$  iterations.

207 choose  $k$ . The solid lines are the estimates given by (13), (14), and (16). The dashed lines present the exact  
 208 values (we have instrumented our code to compute them). We can see from Figures 1(a), 2(a), 1(b) and 2(b)  
 209 that the proposed estimates  $\|\widehat{\nabla F(\mathbf{w}_t)}\|^2$  and  $\widehat{V(\mathbf{g}_{i,t})}$  are close to the true ones. Figures 1(c) and 2(c) compare  
 210 the loss decrease  $\Delta F_t$  (observed a posteriori) and  $\widehat{\mathcal{G}}_{k,t}$ . As expected  $\widehat{\mathcal{G}}_{k,t}$  is a lower bound for  $\Delta F_t$ , but the  
 211 two quantities are almost proportional. This is promising, because, if the lower bound  $\widehat{\mathcal{G}}_{k,t}/T_{k,t}$  and the  
 212 function  $\Delta F_t/T_{k,t}$  were exactly proportional, their maximizers would coincide. Then, working on the lower  
 213 bound, as we do, would not be an approximation. Note that, for CIFAR10 dataset, the stochastic gradients  
 214 are so noisy that the gradient variance is much larger than the gradient norm (as observed also in [52]). Thus,  
 215 the expected gain (9), which is the lower bound for the loss decrease, may become negative. In this case,  
 216 DBW cautiously selects  $k_t = n$  (see Sect. 3.3).

### 217 3.2. Iteration Duration

218 In this subsection, we discuss how to estimate the time  $T_{k,t}$  the PS needs to receive  $k$  gradients of  $\mathbf{w}_t$  after  
 219 the update  $\mathbf{w}_t$  at iteration  $t$ . As in [53], we call *round trip time* the total (random) time an idle worker needs  
 220 to 1) retrieve the new parameter vector, 2) compute the corresponding gradient, and 3) send it back to the PS.  
 221 Our estimators implicitly assume the cluster is stationary and homogeneous, in the sense that the distribution  
 222 of round trip times does not change over time and from worker to worker. But in the experimental section,  
 223 we show that they work also in dynamic and heterogeneous scenarios.

224 When the PS starts a new iteration  $t$  ( $t > 0$ ), there are  $k_{t-1}$  workers ready to compute the new gradient  
 225 while the other  $n - k_{t-1}$  workers are still computing stale gradients, i.e., relative to past parameter vectors  $\mathbf{w}_{t-\tau}$   
 226 with  $\tau > 0$ .  $T_{k,t}$  depends not only on the value of  $k$  but also on the value of  $k_{t-1}$  and the  $n - k_{t-1}$  residual  
 227 round trip times (i.e., the remaining times for the  $n - k_{t-1}$  busy workers to complete their tasks). We assume  
 228 that most of such dependence is captured by the number  $k_{t-1}$ . This would be correct if round trip times were  
 229 exponential random variables due to their memoryless properties. Let  $\mathbf{t}_{h,i,t}$  denote the time the PS spends  
 230 for receiving the  $i$ -th gradient of  $\mathbf{w}_t$ , provided that it has waited  $k_{t-1} = h$  gradients at iteration  $t - 1$ . Under  
 231 our assumptions, for given values of  $h$  and  $i$ , the values  $\{\mathbf{t}_{h,i,t}\}$  can be seen as samples of the same random  
 232 variable that we denote by  $\mathcal{T}_{h,i}$ . For estimating  $T_{k,t}$ , we consider  $\widehat{T}_{k,t} = \mathbb{E}[\widehat{\mathcal{T}}_{k,k}]$ .<sup>5</sup>

233 Consider  $k_{t-1} = h$  and  $k_t = k$ . The PS can collect the samples  $\mathbf{t}_{h,i,t}$  for  $i \leq k$  (it needs to wait  $k$  gradients  
 234 before moving to the next iteration), but also for  $i > k$  because late workers still complete the ongoing  
 235 calculations. In fact, late workers may terminate the computation and send their (by now stale) gradients to  
 236 the PS, before they receive the new parameter vector. Even if a new parameter vector is available at the

<sup>5</sup>It could seem more appropriate to consider  $\widehat{T}_{k,t} = \mathbb{E}[\widehat{\mathcal{T}}_{k_{t-1},k}]$ , but we want to select a value of  $k$  that leads to good performance on the long term, i.e., if constantly used. For this reason, we use  $\mathbb{E}[\widehat{\mathcal{T}}_{k,k}]$ , that corresponds to select  $k$  at each iteration.

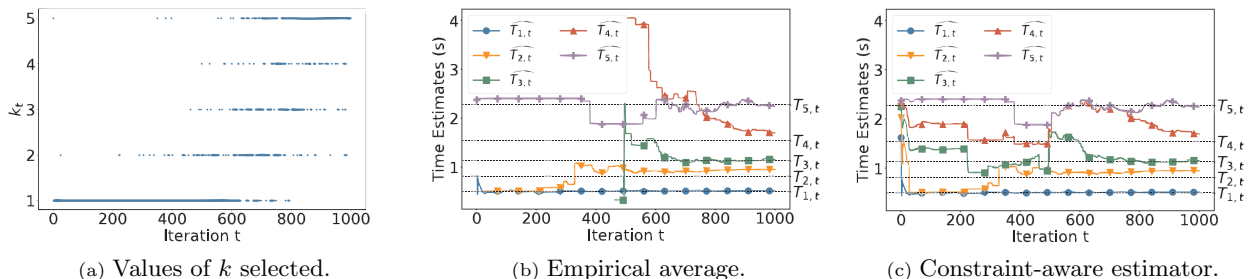


Figure 3: Estimation of  $T_{k,t}$ .  $n = 5$  workers.

237 local queue (and then they know their gradient is not needed), in DBW workers still notify the completion  
 238 to the PS, providing useful information to estimate  $T_{k,t}$  with limited communication overhead.

239 A first naive approach to estimate  $\mathbb{E}[\mathcal{T}_{k,k}]$  is to average the samples obtained over the past history.  
 240 But, actually, there is much more information that can be exploited to improve estimations if we jointly  
 241 estimate the complete set of values  $\mathbb{E}[\mathcal{T}_{h,k}]$ , for  $h, k = 1, \dots, n$ . In fact, the following pathwise relation holds  
 242 for each  $h$  and  $i$ :  $\mathbf{t}_{h,i,t} \leq \mathbf{t}_{h,i+1,t}$ , because the index  $i$  denotes the order of arrivals of the gradients. As a  
 243 consequence,  $\mathbb{E}[\mathcal{T}_{h,i}] \leq \mathbb{E}[\mathcal{T}_{h,i+1}]$ . Moreover, coupling arguments lead to conclude that  $\mathbb{E}[\mathcal{T}_{h+1,i}] \leq \mathbb{E}[\mathcal{T}_{h,i}]$  and  
 244  $\mathbb{E}[\mathcal{T}_{i,i}] \leq \mathbb{E}[\mathcal{T}_{i+1,i+1}]$ . These two inequalities express the following intuitive facts: 1) if an iteration starts  
 245 with more workers available to compute, the PS will collect  $i$  gradients faster (on average), 2) constantly  
 246 waiting a smaller number of gradients leads to faster iterations. As  $\mathbb{E}[\mathcal{T}_{i,i}] \leq \mathbb{E}[\mathcal{T}_{i+1,i+1}]$  may be less evident,  
 247 we provide a proof in Appendix A. These inequalities allow us to couple the estimations of  $\mathbb{E}[\mathcal{T}_{h,k}]$ , for  
 248  $h, k = 1, \dots, n$ . Samples for a given pair  $(h, k)$  can thus contribute not only to the estimation of  $\mathbb{E}[\mathcal{T}_{h,k}]$  but  
 249 also to the estimations of other pairs. This is useful because the number of samples for  $(h, k)$  is proportional  
 250 to the number of times  $k_t$  has been selected equal to  $h$ . There can be many samples for a given pair and  
 251 much less (even none) for another one.

252 Let  $\mathbb{T}_{h,k,t}$  be the set of samples available up to iteration  $t$  for  $(h, k)$ , i.e.,  $\mathbb{T}_{h,k,t} = \{\mathbf{t}_{h,k,t'}, \forall t' \leq t\}$ . We  
 253 propose to estimate  $\{\mathbb{E}[\mathcal{T}_{h,k}], h, k = 1, \dots, n\}$  by solving the following optimization problem:

$$\begin{aligned}
 & \underset{x_{h,k}}{\text{minimize}} && \sum_{h,k=1}^n \sum_{y \in \mathbb{T}_{h,k,t}} (y - x_{h,k})^2 && (17) \\
 & \text{subject to} && x_{h,k} \leq x_{h,k+1}, && \text{for } k = 1, \dots, n-1 \\
 & && x_{h+1,k} \leq x_{h,k}, && \text{for } h = 1, \dots, n-1 \\
 & && x_{k,k} \leq x_{k+1,k+1}, && \text{for } k = 1, \dots, n-1
 \end{aligned}$$

254 Let  $x_{h,k}^*$  be the solution of problem (17). Then,  $\mathbb{E}[\widehat{\mathcal{T}}_{h,k}] = x_{h,k}^*$ ,  $\forall h, k = 1, \dots, n$  and we have  $\widehat{T}_{k,t} = x_{k,k}^*$ . We  
 255 observe that, without the constraints, the optimal value  $x_{h,k}^*$  at iteration  $t$  is the empirical average of the  
 256 corresponding set  $\mathbb{T}_{h,k,t}$ . Hence, Problem (17) is a natural way to extend the empirical average estimators,  
 257 while accounting for the constraints. For our application, the quadratic optimization problem (17) can be  
 258 solved fast through solvers like CVX [54, 55] for the typical values of  $n$  (10 – 1000).

259 In Fig. 3, we compare our estimator with the naive one (the empirical average). We observe that the naive  
 260 method 1) cannot provide estimates for a given value  $h$  before it selects  $k_t = h$ , 2) leads often to estimates  
 261 that are in the wrong relative order. By enforcing the inequality constraints, our estimator (17) is able to  
 262 obtain more precise estimates, in particular for the values  $k = 3$  and  $k = 4$  that are tested less frequently in  
 263 this experiment. Experiments similar to those in Sect. 4 (but not shown in this paper) confirm that naive  
 264 estimators lead to longer training time.

265 *3.3. Dynamic Choice of  $k_t$*

266 DBW rationale is to select the parameter  $k_t$  that maximizes the expected decrease of the loss function  
 267 per time unit, i.e.,:

$$k_t = \arg \max_{1 \leq k \leq n} \frac{\widehat{\mathcal{G}}_{k,t}}{T_{k,t}}. \quad (18)$$

268 Note that (18) does not select values of  $k$  for which  $\widehat{\mathcal{G}}_{k,t} < 0$ , unless  $\widehat{\mathcal{G}}_{k,t} < 0$  for all values  $k$ , in which case  
 269  $k_t = n$ .

270 This behaviour is correct. In fact,  $\widehat{\mathcal{G}}_{k,t} < 0$  indicates the aggregate batch size  $kB$  may be too low to  
 271 guarantee that the stochastic gradient  $\mathbf{g}_t$  corresponds to a descent direction and then it is opportune to  
 272 increase  $k$  (if possible). Our approach then recovers some behaviour of dynamic sample size methods (see [51,  
 273 Sect. 5.2], [56]). At the same time,  $\mathcal{G}_{k,t}$  is a lower bound for the loss decrease  $\mathbb{E}[\Delta F_t]$  (see Eq. (6)). It may  
 274 happen then that  $\widehat{\mathcal{G}}_{k,t} < 0$ , even if  $\mathbb{E}[\Delta F_t] > 0$ . In this situation, DBW’s choice of  $k_t$  may not be optimal,  
 275 as we observe in some settings in Sect. 4.3, but still DBW errs on the side of caution to prevent the loss  
 276 function from increasing.

277 In addition, DBW exploits the local average loss  $\widehat{F}_t$  to avoid decreasing  $k_t$  from one iteration to the other,  
 278 when the loss appears to be increasing (and then we need more accurate gradient estimates, rather than  
 279 noisier ones). We modify (18) to

$$k_t = \max \left( \arg \max_{1 \leq k \leq n} \frac{\widehat{\mathcal{G}}_{k,t}}{T_{k,t}}, (k_{t-1} + 1) \cdot \mathbb{1}_{\{\widehat{F}_{t-1} > \beta \widehat{F}_{t-2}\} \wedge \{k_{t-1} < n\}} \right), \quad (19)$$

280 where  $\beta \geq 1$  (we select  $\beta = 1.01$  in our experiments) and  $\mathbb{1}_A$  denotes the indicator function (equal to 1 iff  $A$   
 281 is true). If the loss has become  $\beta$  times larger since the previous iteration, then (19) forces  $k_t \geq k_{t-1} + 1$ .

282 **4. Experiments**

283 We have implemented DBW in PyTorch [48], using the MPI backend for distributed communications. The  
 284 experiments have been run on a real CPU/GPU cluster platform, with different GPUs available (e.g., GeForce  
 285 GTX 1080 Ti, GeForce GTX Titan X, and Nvidia Tesla V100). In order to have a fine control over the round  
 286 trip times, our code can generate computation and communication times according to different distributions  
 287 (uniform, exponential, Pareto, etc.) or read them from a trace provided as input file. The system operates at  
 288 the maximum speed guaranteed by the underlying cluster, but it maintains a virtual clock to keep track of  
 289 when events would have happened. Note that the virtual time is not a simple relabeling of the time axis: for  
 290 example virtual time instants at which gradients are received by the PS determine which of them are actually  
 291 used to update the parameter vector. So the virtual time has an effect on the optimization dynamics. Our  
 292 code is available online [42].

293 In what follows, we show that the number of backup workers should vary, not only with the round trip  
 294 time distribution, but also with the hyper-parameters of the optimization algorithm like the batch size  $B$ .  
 295 Moreover, the optimal setting depends as well on the stage of the training process, and then changes over  
 296 time, even when the cluster is stationary (round trip times do not change during the training period).

297 In all experiments, DBW achieves nearly optimal performance in terms of convergence time, and sometimes  
 298 it even outperforms the optimal static setting, that is found through an exhaustive offline search over all  
 299 values  $k \in \{1, \dots, n\}$ . We also compare DBW with a variant where the gain  $\mathcal{G}_{k,t}$  is not estimated as in (16),  
 300 but it equals the number of aggregated gradients  $k$ , as proposed in [44]. We call this variant blind DBW  
 301 (B-DBW), because it is oblivious to the current state of the training. We find that this approach is too  
 302 simplistic: ignoring the current stage of the optimization problem leads to worse performance than DBW.

303 We evaluated DBW, B-DBW, and different static settings for  $k$  on two classification problems 1) MNIST [57],  
 304 a dataset with 70000  $28 \times 28$  images portraying handwritten digits from 0 to 9 and 2) CIFAR10 [58], a  
 305 dataset with 60000  $32 \times 32$  colour images in 10 classes.<sup>6</sup> We trained a neural network with two convolutional

---

<sup>6</sup>Both dataset include 10000 test images.

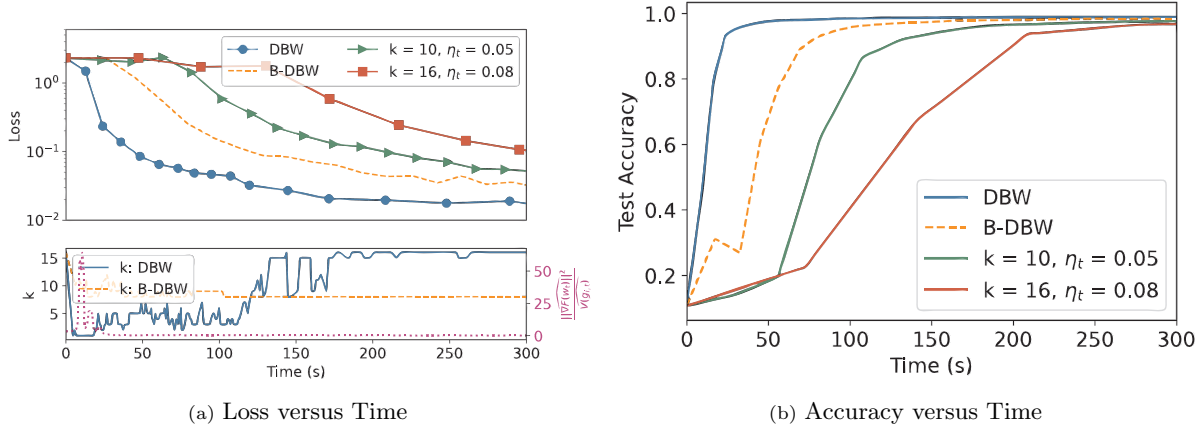


Figure 4: Training on MNIST, batch size  $B = 500$ ,  $n = 16$  workers, estimates computed over the last  $D = 5$  iterations, proportional rule with  $\eta(k) = 0.005k$ , round trip times follow shifted exponential distribution  $0.3 + 0.7\text{Exp}(1)$ .

layers with  $5 \times 5$  filters and two fully connected layers for MNIST and we trained a ResNet18 [59] network for CIFAR10. The loss function was the cross-entropy one. For MNIST, every worker had access to the entire dataset. For CIFAR10, the data set was split uniformly at random among workers.

The learning rate is probably the most critical hyper-parameter in ML optimization problems. Ideally, it should be set to that largest value that still guarantees convergence. It is important to note that different static settings for the number of backup workers require different values for the learning rate. In fact, the smaller is  $k$ , the noisier is the aggregate gradient  $\mathbf{g}_t$ , so that the smaller should be the learning rate. The rule of thumb proposed in the seminal paper [40] 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 [60, 61]. In static settings, aggregating  $k$  gradients is equivalent to use a batch size equal to  $kB$ , so that the learning rate should scale accordingly. An alternative approach is to tune the learning rate independently for each static value of  $k$  according to the empirical rule in [62], that requires to run a number of experiments and determine the inflection points of a specific curve. This rule leads as well to learning rates increasing with  $k$ . We call the two settings respectively the *proportional* and the *knee* rule. The maximum learning rate for the proportional rule is set equal to the value determined for  $k_t = n$  by the knee rule. The same value is also used as learning rate for DBW and B-DBW, independently from the specific value they select for  $k_t$ . In fact, DBW and B-DBW can safely operate with a large learning rate because they dynamically increase  $k_t$  up to  $n$ , when they detect that the loss is increasing.

Figures 4(a) and 5(a) show, 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. For static settings, the learning rate follows the proportional rule and the optimal static settings are  $k^* = 10$  for MNIST and  $k^* = 8$  for CIFAR10. We can see that DBW achieves the fastest convergence across all other tested configurations of  $k$ , 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. For MNIST, up to iteration 38,  $\mathbb{V}(\mathbf{g}_{i,t})$  is negligible in comparison to  $\|\nabla F(\mathbf{w}_t)\|^2$ . DBW then selects small values for  $k_t$  losing 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,  $\|\nabla F(\mathbf{w}_t)\|^2$  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 to the randomness of the estimates  $\widehat{T}_{k,t}$ . For CIFAR10, as the stochastic gradients are more noisy, the ratio values  $\|\nabla F(\mathbf{w}_t)\|^2 / \mathbb{V}(\mathbf{g}_{i,t})$  are smaller than in MNIST, DBW selects higher values for  $k_t$  (around 10) in the beginning of the training. After iteration 130, the gain becomes more

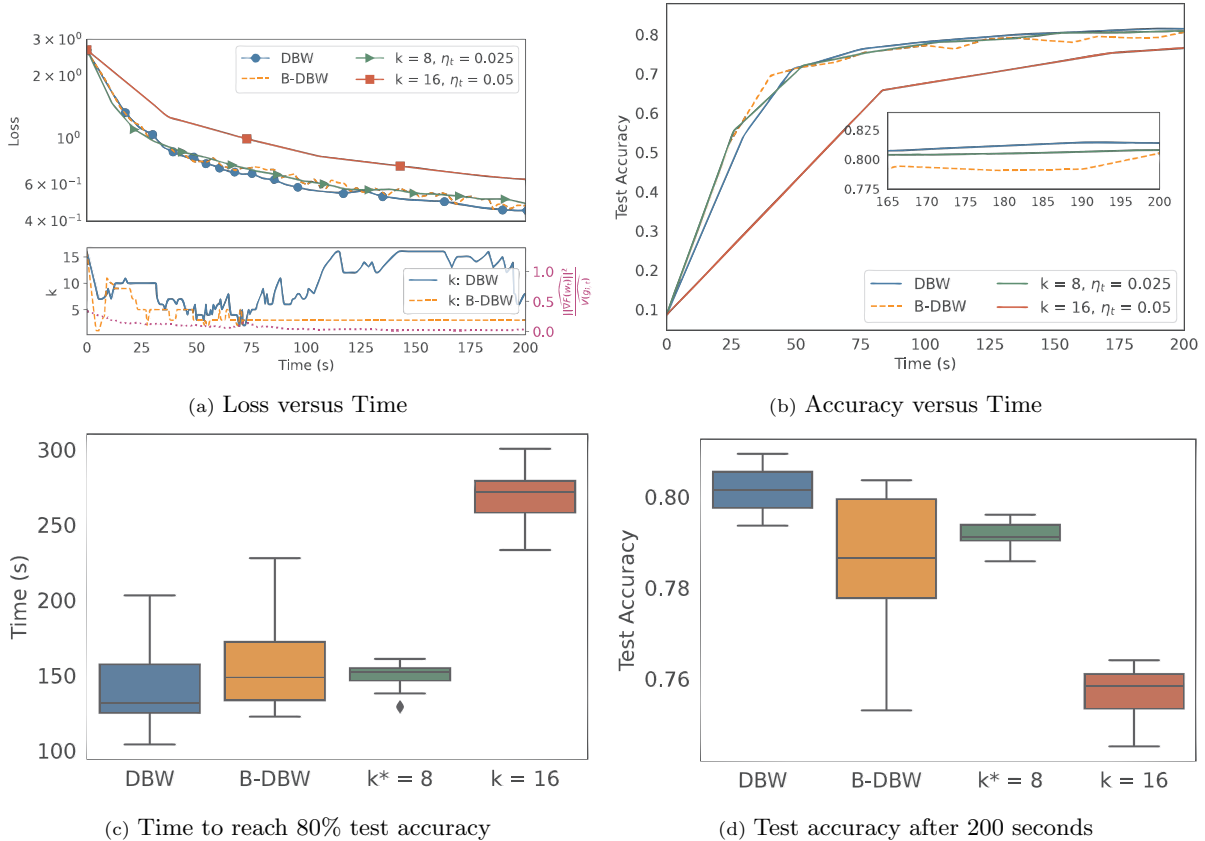


Figure 5: Training on CIFAR10, batch size  $B = 256$ ,  $n = 16$  workers, estimates computed over the last  $D = 5$  iterations, proportional rule with  $\eta(k) = \frac{0.05k}{16}$ , round trip times follow exponential distribution  $\text{Exp}(1)$ . Box plots are based on 20 independent runs.

339 sensitive to  $k$  and thus DBW progressively increases  $k_t$  as observed in MNIST dataset. Note that DBW  
 340 performs less advantageous in CIFAR10, although it is still the best one. As discussed in Sect. 3.1, the  
 341 gain (9) can be negative when the stochastic gradients are very noisy, which is the case for CIFAR10 dataset.  
 342 This results in DBW cautiously selecting  $k_t = n$  according to (18), while the optimal  $k_t$  at the iteration  $t$   
 343 may be smaller. Note that working with significantly larger batch sizes would reduce the variability of the  
 344 stochastic gradients.

345 Figures 4(b) and 5(b) show, for a single run of the training process, the evolution of the test accuracy  
 346 over time. We can see that DBW converges to a better model faster than the other methods for MNIST.  
 347 The advantages of DBW on CIFAR10 are less evident on this specific run, but Figs. 5(c) and 5(d) show  
 348 the distribution of the time to reach 80% test accuracy and the distribution of the test accuracy after 200  
 349 seconds using box plots.<sup>7</sup> On average DBW performs better than B-DBW or the optimal static setting.

#### 350 4.1. Round trip time effect

351 In this subsection we consider round trip times (see Sect. 3.2) are i.i.d. according to a shifted exponential  
 352 random variable  $1 - \alpha + \alpha \times \text{Exp}(1)$ , where  $0 \leq \alpha \leq 1$ . We consider later realistic time distributions. This  
 353 choice, common to [53, 63], allows us to easily tune the variability of the round trip times by changing  $\alpha$ .

<sup>7</sup>The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution. The middle bar gives the median value.

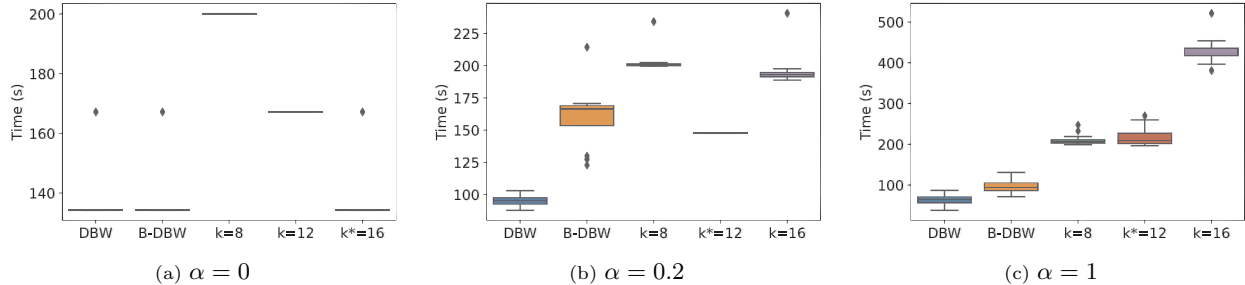


Figure 6: Effect of round trip time distribution. MNIST,  $n = 16$  workers, batch size  $B = 500$ , estimates computed over the last  $D = 5$  iterations, proportional rule for  $\eta(k)$  in static settings where  $\eta(k) = 0.005k$ .

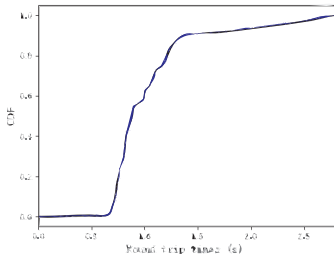


Figure 7: Empirical distribution of round trip times on a Spark cluster

354 When  $\alpha = 0$ , all gradients arrive at the same time at the PS, so that the PS should always aggregate all  
 355 of them. As  $\alpha$  changes from 0 to 1, the variance of the round trip times increases, and waiting for  $k < n$   
 356 gradients becomes advantageous.

357 Figure 6 compares the time needed to reach a training loss smaller than 0.2 for the two dynamic algorithms  
 358 and the static settings  $k = 16$ ,  $k = 12$ , and  $k = 8$ , that are optimal respectively for  $\alpha = 0$ ,  $\alpha = 0.2$ , and  
 359  $\alpha = 1$ . For each of them, we carried out 20 independent runs with different seeds. We find that our dynamic  
 360 algorithm achieves the fastest convergence in all three scenarios, it is even 1.2x faster and 3x faster than the  
 361 optimal static settings for  $\alpha = 0.2$  and  $\alpha = 1$ . There are two factors that determine this observation. First, as  
 362 discussed for Fig. 4, there is no unique optimal value of  $k$  to be used across the whole training process, and  
 363 DBW manages to select the most indicated value in different stages of the training process. Second, DBW  
 364 takes advantage of a larger learning rate. Both factors play a role. For example if we focus on Fig. 6(c), the  
 365 learning rate for DBW is twice faster than that for  $k = 8$ , but DBW is on average 3x faster. Then, adapting  
 366  $k$  achieves an additional 1.5x improvement. The importance of capturing the dynamics of the optimization  
 367 process is again also evident by comparing DBW with B-DBW. While B-DBW takes advantage of a higher  
 368 learning rate as well, it performs worse than our solution DBW.

#### 369 4.2. Batch size effect

370 The batch size  $B$  is another important hyper-parameter. It is often limited by the memory available  
 371 at each worker, but can also be determined by generalization performance of the final model [64]. In this  
 372 subsection we highlight how  $B$  also affects the optimal setting for  $k$ . These findings confirm that configuring  
 373 the number of backup workers is indeed a difficult task, and knowing the characteristics of the underlying  
 374 cluster is not sufficient.

375 The experiments differ in two additional aspects from those in Fig. 6. First, the distribution of the round  
 376 trip times (shown in Fig. 7) is taken from a training a ML model through stochastic gradient descent on a  
 377 production Spark cluster with sixteen servers, each with two 8-core Intel E5-2630 CPUs running at 2.40GHz.  
 378 The cluster was managed using Zoe Analytics [65]. Second, learning rates are configured according to the  
 379 knee rule. We observe that the knee rule leads to a weaker variability of the learning rate in comparison to

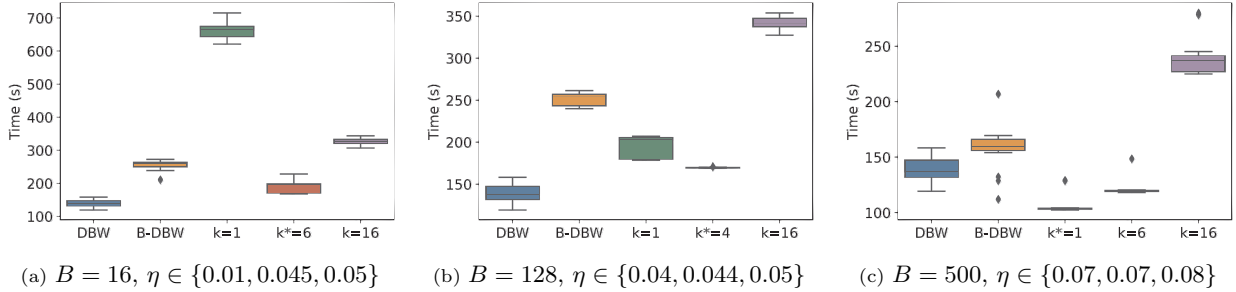


Figure 8: Effect of batch size  $B$ . MNIST,  $n = 16$  workers, estimates computed over the last  $D = 5$  iterations, knee rule for  $\eta$  in static settings with values shown above for each  $k$ .

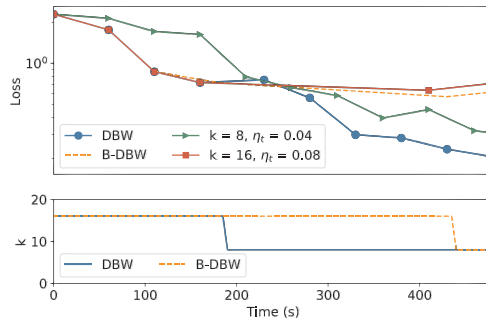


Figure 9: Robustness to slowdowns of the system. MNIST,  $n = 16$  workers, batch size  $B = 500$ , estimates computed over the last  $D = 5$  iterations, proportional rule for  $\eta(k)$  in static settings where  $\eta(k) = 0.005k$ .

380 the proportional rule: for example, for  $B = 16$ ,  $\eta$  increases by less than a factor 5 when  $k$  changes from  $k = 1$   
 381 to  $k = 16$ , and it increases much less for larger  $B$ .

382 Figure 8 shows the results for  $B = 16, 128, 500$ , comparing the dynamic methods with a few static settings,  
 383 including the optimal static one that decreases from  $k^* = 6$  for  $B = 16$  to  $k^* = 1$  for  $B = 500$ . Again,  
 384 Equation (9) helps to understand this change of the optimal static setting with different batch size: as the  
 385 batch size increases, the variability of gradients decreases, so that the numerator depends less on  $k$ . The  
 386 advantage of reducing  $T_{k,t}$  by selecting a small  $k$  can compensate the corresponding decrease of the gain  $\mathcal{G}_{k,t}$ .  
 387

388 Since learning rates chosen by the knee rule for the static settings are now close to dynamic ones, DBW  
 389 does not outperform the optimal static setting, but its performance are quite close, and significantly better  
 390 than B-DBW for  $B = 128, 500$ . It is worthy to stress that, when running a given ML problem on a specific  
 391 cluster environment, the user cannot predict the optimal static setting  $k^*$  without running preliminary short  
 training experiments for every  $k$ . DBW does not need them.

#### 392 4.3. Robustness to slowdowns

393 Until now, we have considered a stationary setting where the distribution of round trip times does not  
 394 change during the training. Figure 9 shows an experiment in which half of the workers experience a sudden  
 395 slowdown during the training process. Initially, round trip times are all equal and deterministic, so that the  
 396 optimal setting is  $k_t = n = 16$ . Suddenly, at time  $t = 160$ s, half of the workers in the clusters slow down by a  
 397 factor 5 and the optimal static configuration is now to select  $k_t = n/2 = 8$ . We can see that DBW detects  
 398 the slowdowns in the system and then correctly selects  $k_t = 8$ .

#### 399 4.4. Comparison with ADASync

400 ADASync [27] is a dynamic backup scheme designed for the Push and Interrupt (PsI) case, under the  
 401 assumption that the round trip times follow shifted exponential distribution. For the comparison, we consider

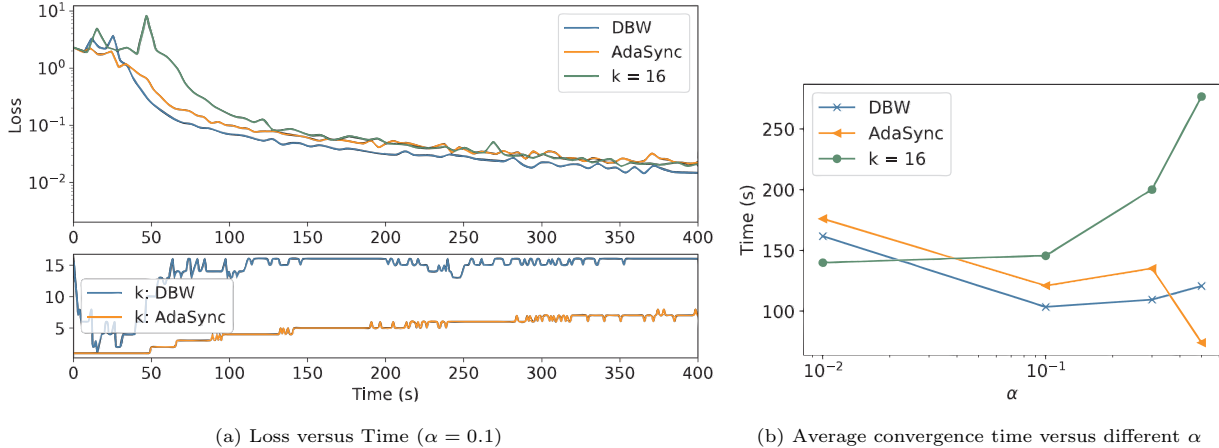


Figure 10: Training on MNIST, batch size  $B = 500$ ,  $n = 16$  workers, estimates computed over the last  $D = 5$  iterations.  $\eta = 0.08$ . Round trip times follow shifted exponential distribution  $1 - \alpha + \alpha \text{Exp}(1)$

402 then this setting. For ADASync, the quadratic formulation in [27, Appendix D.1] is used to derive the  
 403 number of backup workers. ADASync updates  $k$  at the end of a time-window. We consider this time-window  
 404 small enough for ADASync evaluating the possibility to update  $k_t$  at each iteration, as DBW does.

405 Figure 10(a) shows, for a single run of the training process, the evolution of the loss over time and the  
 406 corresponding choices of  $k_t$  for DBW and ADASync, when  $\alpha = 0.1$ , i.e., round trip times follow distribution  
 407  $0.9 + 0.1\text{Exp}(1)$ . DBW quickly reaches a large value of  $k_t$  close to  $n$ . For small  $\alpha$  the variance of round trip  
 408 time is small, so choosing large  $k_t$  does not lead large iteration times  $\mathbb{E}[T_{k,t}]$  but benefits the gain in (9).  
 409 The approximated formula used by ADASync, even if derived under the assumption of shifted exponential  
 410 distributions, does not depend on  $\alpha$ , and ADASync fails to increase fast the value of  $k_t$ .

411 Fig. 10(b), shows the average convergence time<sup>8</sup> computed over 10 independent runs under different  $\alpha$ .  
 412 The larger  $\alpha$ , the larger the variance of round trip times. We can see that when  $\alpha$  is smaller than 0.3, DBW  
 413 performs better than ADASync. While, ADASync works better for larger  $\alpha$ , which suggests DBW may be  
 414 too conservative on the number of backup workers in the late phase of the training.

415 Remember that the estimated gain  $\widehat{\mathcal{G}}_{k,t}$  used in (18) for choosing  $k_t$ , is a lower bound for the true loss  
 416 decrease. In the late training phase, when the gradient norm becomes smaller, small values of  $k$  may lead to  
 417 estimate a negative (see (16)). In this case, DBW conservatively chooses a larger  $k$  for which the gain is  
 418 estimated to be positive. On the other hand, ADASync requires prior knowledge on the round trip time  
 419 distribution. This distribution may be hard to estimate and may change during the training period, that is  
 420 often very long for state-of-the-art machine learning models (e.g., weeks). Notice that DBW does not require  
 421 any prior knowledge on the system.

## 422 5. Conclusions

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

428 As a future research direction, we want to extend the scope of DBW to dynamic resource allocation,  
 429 e.g., by automatically releasing computing resources if  $k_t < n$  and the fastest  $k_t$  gradients are always coming

<sup>8</sup>The convergence time noted here is the time when the training loss reaches 0.07.



430 from the same set of workers. In general, we believe that distributed systems for ML are in need of adaptive  
 431 algorithms in the same spirit of the utility-based congestion control schemes developed in our community  
 432 starting from the seminal paper [66]. As our work points out, it is important to define new utility functions  
 433 that take into account the learning process. Adaptive algorithms are even more needed in the federated  
 434 learning scenario [67], where ML training is no more relegated to the cloud, but it occurs in the wild over the  
 435 whole internet. Our paper shows that even simple algorithms can provide significant improvements.

## 436 6. Acknowledgements

437 This work has been carried out in the framework of a common lab agreement between Inria and Nokia  
 438 Bell Labs (ADR 'Rethinking the Network'). We thank Alain Jean-Marie for having suggested the estimation  
 439 technique in Sect. 3.2 and Pietro Michiardi for many helpful discussions.

## 440 Appendix A. Proof of $\mathbb{E}[\mathcal{T}_{i,i}] \leq \mathbb{E}[\mathcal{T}_{i+1,i+1}]$

441 Remember that we assume that  $T_{k,t}$  depends on the past only through the number of workers  $k_{t-1}$  selected  
 442 at the previous iteration. This approximation is correct when round trip times are exponentially distributed.  
 443 We start proving the inequality under the assumption that round trip times are exponentially distributed.  
 444 We move then to the general case.

445 Consider the beginning of a new iteration  $t$  when the PS systematically waits for  $i+1$  nodes. Without loss  
 446 of generality, let us assume that the workers who finished the computation are labeled  $1, 2, \dots, i+1$ . Worker  
 447  $j \leq i+1$  needs an exponentially distributed round trip time  $\omega_j$  to complete the new computation. Worker  
 448  $j > i+1$  needs to complete iteration  $t-1$ , with residual time  $\omega'_j$ , and possibly start a new one with the  
 449 updated parameter vector, with corresponding residual time  $\omega_j$ ; both  $\omega_j$  and  $\omega'_j$  are exponentially distributed.

450 Let  $\mu(l, A)$  denote the  $l$ -th smallest element of the multiset  $A$ . The duration of the new iteration is then  
 451  $T_{i+1,t} = \mu(i+1, \{\omega_1, \dots, \omega_i, \omega_{i+1}, \omega'_{i+2} + \omega_{i+2}, \dots, \omega'_n + \omega_n\})$ .

452 Now consider the case when the PS only waits for the  $i$  workers. Again we assume the the first workers who  
 453 finished the iteration are labeled  $1, 2, \dots, i$ . We also couple all the round trip times so that  $\omega_j$  for  $j = 1, \dots, n$   
 454 and  $\omega'_j$  for  $j = i+2, \dots, n$  denote the same quantities and have the same values. In this case also worker  $i+1$   
 455 needs to terminate the previous computation; this will require a time  $\omega'_{i+1}$ , but its specific value is irrelevant.  
 456 The duration of the new iteration is  $T_{i,t} = \mu(i, \{\omega_1, \dots, \omega_i, \omega'_{i+1} + \omega_{i+1}, \omega'_{i+2} + \omega_{i+2}, \dots, \omega'_n + \omega_n\})$ .

$$\begin{aligned}
 T_{i+1,t} &= \mu(i+1, \{\omega_1, \dots, \omega_i, \omega_{i+1}, \omega'_{i+2} + \omega_{i+2}, \dots, \omega'_n + \omega_n\}) \\
 &\geq \mu(i+1, \{\omega_1, \dots, \omega_i, 0, \omega'_{i+2} + \omega_{i+2}, \dots, \omega'_n + \omega_n\}) \\
 &= \mu(i, \{\omega_1, \dots, \omega_i, \omega'_{i+2} + \omega_{i+2}, \dots, \omega'_n + \omega_n\}) \\
 &\geq \mu(i, \{\omega_1, \dots, \omega_i, \omega'_{i+1} + \omega_{i+1}, \omega'_{i+2} + \omega_{i+2}, \dots, \omega'_n + \omega_n\}) \\
 &= T_{i,t},
 \end{aligned}$$

457 where the first inequality follows from the fact that replacing an element in the set with a smaller one can  
 458 only decrease the  $(i+1)$ -th smallest element of the multiset, the second equality from the fact that 0 is  
 459 necessarily the smallest value in the multiset, and the last inequality from the fact that enlarging a multiset  
 460 cannot increase its  $i$ -th smallest element.

461 In the general case, we show that the time at which the  $t$ -th iteration will start is not larger when the PS  
 462 waits for  $i$  workers than when it waits for  $i+1$  workers. We will couple the round trip times so that in both  
 463 cases the duration of the  $m$ -th round trip time for worker  $j$  is the same in both systems.

464 Let  $\chi_{i,t}$  denote the time at which the  $t$ -th system iteration starts when then PS waits for  $i$  workers. We  
 465 also consider a *lazy* system, where the PS does not need to start the new iteration as soon as  $i$  new updates  
 466 are available, but it can start after an arbitrary delay. We say that a sequence  $(\chi_{i,t}^{(l)})_{t \in \mathbb{N}}$  is feasible for the  
 467 lazy system, if it corresponds to a valid sequence of starting times. We observe that for any feasible sequence

468  $\chi_{i,t}^{(l)} \geq \chi_{i,t}$  for each  $t$  as the lazy system can only introduce slack times. Finally, we note that  $(\chi_{i+1,t})_{t \in \mathbb{N}}$  is a  
469 feasible sequence for the lazy system, as at each time  $\chi_{i+1,t}$ , the system has available  $i$  new updates (it has  
470  $i + 1$ ) and can then start a new iteration. It follows that  $\chi_{i+1,t} \geq \chi_{i,t}$ .

## 471 References

- 472 [1] K. Canini, T. Chandra, E. Ie, J. McFadden, K. Goldman, M. Gunter, J. Harmsen, K. LeFevre, D. Lepikhin,  
473 T. Lloret Llinares, I. Mukherjee, F. Pereira, J. Redstone, T. Shaked, Y. Singer, Sibyl: A system for large  
474 scale supervised machine learning, technical talk (2014).
- 475 [2] E. P. Xing, Q. Ho, P. Xie, D. Wei, Strategies and principles of distributed machine learning on big data,  
476 *Engineering* 2 (2) (2016) 179 – 195.
- 477 [3] A. Harlap, H. Cui, W. Dai, J. Wei, G. R. Ganger, P. B. Gibbons, G. A. Gibson, E. P. Xing, Addressing the  
478 straggler problem for iterative convergent parallel ML, in: *Proceedings of the Seventh ACM Symposium*  
479 *on Cloud Computing, SoCC '16*, ACM, New York, NY, USA, 2016, pp. 98–111.
- 480 [4] H. Wang, D. Niu, B. Li, Distributed machine learning with a serverless architecture, in: *IEEE INFOCOM*  
481 *2019 - IEEE Conference on Computer Communications*, 2019, pp. 1288–1296.
- 482 [5] Z. Shi, A. Eryilmaz, A flexible distributed optimization framework for service of concurrent tasks in  
483 processing networks, in: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019,  
484 pp. 1072–1080.
- 485 [6] Y. Bao, Y. Peng, C. Wu, Deep learning-based job placement in distributed machine learning clusters, in:  
486 *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 505–513.
- 487 [7] C. Chen, W. Wang, B. Li, Round-robin synchronization: Mitigating communication bottlenecks in  
488 parameter servers, in: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019,  
489 pp. 532–540.
- 490 [8] S. Shi, X. Chu, B. Li, Mg-wfbp: Efficient data communication for distributed synchronous SGD  
491 algorithms, in: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp.  
492 172–180.
- 493 [9] X. Zhang, J. Wang, G. Joshi, C. Joe-Wong, Machine learning on volatile instances, in: *IEEE INFOCOM*  
494 *2020 - IEEE Conference on Computer Communications*, 2020, pp. 139–148. doi:10.1109/INFOCOM41043.  
495 2020.9155448.
- 496 [10] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, B.-Y.  
497 Su, Scaling distributed machine learning with the parameter server, in: *Proceedings of the 11th USENIX*  
498 *Conference on Operating Systems Design and Implementation*, 2014, pp. 583–598.
- 499 [11] P. Patarasuk, X. Yuan, Bandwidth optimal all-reduce algorithms for clusters of workstations, *J. Parallel*  
500 *Distributed Comput.* 69 (2) (2009) 117–124. doi:10.1016/j.jpdc.2008.09.002.  
501 URL <https://doi.org/10.1016/j.jpdc.2008.09.002>
- 502 [12] S. Jeaugey, Massively scale your deep learning training with nccl 2.4. (2019).  
503 URL <https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/>
- 504 [13] MPI ([https://www.mpich.org/static/docs/v3.1/www3/MPI\\_Allreduce.html](https://www.mpich.org/static/docs/v3.1/www3/MPI_Allreduce.html)).
- 505 [14] A. Gibiansky, Bringing hpc techniques to deep learning (2017).
- 506 [15] J. Dean, L. A. Barroso, The tail at scale, *Communications of the ACM* 56 (2) (2013) 74–80.

- 507 [16] G. Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica, Effective straggler mitigation: Attack of the  
508 clones, in: Proc. of the 10th USENIX Conf. NSDI, 2013, pp. 185–198.
- 509 [17] C. Karakus, Y. Sun, S. Diggavi, W. Yin, Straggler mitigation in distributed optimization through data  
510 encoding, in: Proc. of NIPS, 2017, pp. 5434–5442.
- 511 [18] S. Li, S. M. M. Kalan, A. S. Avestimehr, M. Soltanolkotabi, Near-optimal straggler mitigation for  
512 distributed gradient methods, in: IEEE International Parallel and Distributed Processing Symposium  
513 Workshops, 2018, pp. 857–866.
- 514 [19] A. Reisizadeh, S. Prakash, R. Pedarsani, A. S. Avestimehr, Codedreduce: A fast and robust framework  
515 for gradient aggregation in distributed learning, CoRR abs/1902.01981 (2019). [arXiv:1902.01981](https://arxiv.org/abs/1902.01981).  
516 URL <http://arxiv.org/abs/1902.01981>
- 517 [20] R. Tandon, Q. Lei, A. G. Dimakis, N. Karampatziakis, Gradient coding: Avoiding stragglers in distributed  
518 learning, Vol. 70 of Proceedings of Machine Learning Research, PMLR, International Convention Centre,  
519 Sydney, Australia, 2017, pp. 3368–3376.  
520 URL <http://proceedings.mlr.press/v70/tdandon17a.html>
- 521 [21] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, K. Ramchandran, Speeding up distributed machine  
522 learning using codes, IEEE Transactions on Information Theory 64 (3) (2017) 1514–1529.
- 523 [22] W. Halbawi, N. A. Ruhi, F. Salehi, B. Hassibi, Improving distributed gradient descent using reed-solomon  
524 codes, in: 2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June  
525 17-22, 2018, IEEE, 2018, pp. 2027–2031. doi:10.1109/ISIT.2018.8437467.  
526 URL <https://doi.org/10.1109/ISIT.2018.8437467>
- 527 [23] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, S. A. Avestimehr, Lagrange coded computing:  
528 Optimal design for resiliency, security, and privacy, Vol. 89 of Proceedings of Machine Learning Research,  
529 PMLR, 2019, pp. 1215–1225.  
530 URL <http://proceedings.mlr.press/v89/you19b.html>
- 531 [24] A. Reisizadeh, S. Prakash, R. Pedarsani, A. S. Avestimehr, Coded computation over heterogeneous  
532 clusters, IEEE Transactions on Information Theory 65 (7) (2019) 4227–4242.
- 533 [25] E. Ozfatura, S. Ulukus, D. Gündüz, Straggler-aware distributed learning: Communication–computation  
534 latency trade-off, Entropy 22 (5) (2020) 544.
- 535 [26] A. Reisizadeh, S. Prakash, R. Pedarsani, A. S. Avestimehr, Tree gradient coding, in: 2019 IEEE  
536 International Symposium on Information Theory (ISIT), IEEE, 2019, pp. 2808–2812.
- 537 [27] S. Dutta, J. Wang, G. Joshi, Slow and stale gradients can win the race (2020). [arXiv:arXiv:2003.10579](https://arxiv.org/abs/2003.10579).
- 538 [28] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning  
539 of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp.  
540 1273–1282.
- 541 [29] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Guo, Optimus: an efficient dynamic resource scheduler for deep  
542 learning clusters, in: Proceedings of the Thirteenth EuroSys Conference, 2018, pp. 1–14.
- 543 [30] J. Cipar, Q. Ho, J. K. Kim, S. Lee, G. R. Ganger, G. Gibson, K. Keeton, E. Xing, Solving the straggler  
544 problem with bounded staleness, in: Presented as part of the 14th Workshop on Hot Topics in Operating  
545 Systems, 2013.
- 546 [31] S. Dutta, G. Joshi, S. Ghosh, P. Dube, P. Nagpurkar, Slow and stale gradients can win the race:  
547 Error-runtime trade-offs in distributed SGD, AISTATS (2018).

- 548 [32] W. Dai, Y. Zhou, N. Dong, H. Zhang, E. P. Xing, Toward understanding the impact of staleness in  
549 distributed machine learning, in: 7th International Conference on Learning Representations, ICLR 2019,  
550 2019.
- 551 [33] A. Nedic, A. Ozdaglar, Distributed subgradient methods for multi-agent optimization, *IEEE Transactions*  
552 *on Automatic Control* 54 (1) (2009) 48–61.
- 553 [34] J. C. Duchi, A. Agarwal, M. J. Wainwright, Dual averaging for distributed optimization: Convergence  
554 analysis and network scaling, *IEEE Trans. on Automatic Control* 57 (3) (2012) 592–606.
- 555 [35] X. Lian, C. Zhang, H. Zhang, C. Hsieh, W. Zhang, J. Liu, Can decentralized algorithms outperform  
556 centralized algorithms? A case study for decentralized parallel stochastic gradient descent, in: *Advances*  
557 *in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing*  
558 *Systems 2017*, 2017, pp. 5330–5340.
- 559 [36] A. Elgabli, J. Park, A. S. Bedi, M. Bennis, V. Aggarwal, Gdmm: Fast and communication efficient  
560 framework for distributed machine learning., *Journal of Machine Learning Research* 21 (76) (2020) 1–39.
- 561 [37] G. Neglia, G. Calbi, D. Towsley, G. Vardoyan, The role of network topology for distributed machine  
562 learning, in: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp.  
563 2350–2358.
- 564 [38] G. Neglia, C. Xu, D. Towsley, G. Calbi, Decentralized gradient methods: does topology matter?, Vol.  
565 108 of *Proceedings of Machine Learning Research*, PMLR, 2020, pp. 2348–2358.
- 566 [39] O. Marfoq, C. Xu, G. Neglia, R. Vidal, Throughput-optimal topology design for cross-silo federated  
567 learning, in: *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- 568 [40] J. Chen, R. Monga, S. Bengio, R. Jozefowicz, Revisiting distributed synchronous SGD, in: *International*  
569 *Conference on Learning Representations Workshop Track*, 2016.
- 570 [41] C. Xu, G. Neglia, N. Sebastianelli, Dynamic backup workers for parallel machine learning, in: *2020 IFIP*  
571 *Networking Conference, Networking 2020, Paris, France, June 22-26, 2020*, IEEE, 2020, pp. 574–578.  
572 URL <https://ieeexplore.ieee.org/document/9142724>
- 573 [42] DBW, <https://gitlab.inria.fr/chxu/dbw>.
- 574 [43] TensorFlow (<https://www.tensorflow.org/>).
- 575 [44] M. Teng, F. Wood, Bayesian distributed stochastic gradient descent, in: *Advances in Neural Information*  
576 *Processing Systems* 31, 2018, pp. 6378–6388.
- 577 [45] Y. Ling, T. Mullen, X. Lin, Analysis of optimal thread pool size, *SIGOPS Oper. Syst. Rev.* 34 (2) (2000)  
578 42–55.
- 579 [46] M. M. Amiri, D. Gündüz, Computation scheduling for distributed machine learning with straggling  
580 workers, in: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019*,  
581 2019, pp. 8177–8181.
- 582 [47] Q. Luo, J. Lin, Y. Zhuo, X. Qian, Hop: Heterogeneity-aware decentralized training, in: *Proceedings of*  
583 *the Twenty-Fourth International Conference on Architectural Support for Programming Languages and*  
584 *Operating Systems*, 2019, pp. 893–907.
- 585 [48] PyTorch (<https://pytorch.org/>).
- 586 [49] L. Balles, J. Romero, P. Hennig, Coupling adaptive batch sizes with learning rates, in: *Proc. of the*  
587 *Thirty-Third Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

- 588 [50] S. Bubeck, Convex optimization: Algorithms and complexity, *Found. Trends Mach. Learn.* 8 (3-4) (2015)  
589 231–357.
- 590 [51] L. Bottou, F. E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, *Siam Review*  
591 60 (2) (2018) 223–311.
- 592 [52] G. Neglia, C. Xu, D. Towsley, G. Calbi, Decentralized gradient methods: does topology matter?,  
593 in: S. Chiappa, R. Calandra (Eds.), *The 23rd International Conference on Artificial Intelligence and*  
594 *Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy], Vol. 108 of Proceedings of*  
595 *Machine Learning Research, PMLR, 2020, pp. 2348–2358.*  
596 URL <http://proceedings.mlr.press/v108/neglia20a.html>
- 597 [53] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, K. Ramchandran, Speeding up distributed machine  
598 learning using codes, *IEEE Transactions on Information Theory* 64 (3) (2018) 1514–1529.
- 599 [54] M. Grant, S. Boyd, CVX: Matlab software for disciplined convex programming, version 2.1 (Mar.  
600 <http://cvxr.com/cvx>, 2014).
- 601 [55] M. Grant, S. Boyd, Graph implementations for nonsmooth convex programs, in: *Recent Advances in*  
602 *Learning and Control, Lecture Notes in Control and Information Sciences, Springer-Verlag Limited,*  
603 2008, pp. 95–110.
- 604 [56] S. De, A. Yadav, D. Jacobs, T. Goldstein, Automated inference with adaptive batches, in: *Proceedings*  
605 *of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), 2017, pp.*  
606 1504–1513.
- 607 [57] MNIST database (<http://yann.lecun.com/exdb/mnist/>).
- 608 [58] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images (2009).
- 609 [59] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: *Proceedings of the*  
610 *IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.*
- 611 [60] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia,  
612 K. He, Accurate, large minibatch SGD: training imagenet in 1 hour, *CoRR abs/1706.02677* (2017).  
613 [arXiv:1706.02677](https://arxiv.org/abs/1706.02677).
- 614 [61] S. L. Smith, P.-J. Kindermans, Q. V. Le, Don’t decay the learning rate, increase the batch size, in:  
615 *International Conference on Learning Representations, 2018.*
- 616 [62] L. N. Smith, Cyclical learning rates for training neural networks, in: *Applications of Computer Vision*  
617 *(WACV), 2017 IEEE Winter Conference on, IEEE, 2017, pp. 464–472.*
- 618 [63] S. Dutta, G. Joshi, S. Ghosh, P. Dube, P. Nagpurkar, Slow and stale gradients can win the race:  
619 Error-runtime trade-offs in distributed SGD, in: *International Conference on Artificial Intelligence and*  
620 *Statistics, AISTATS 2018, 2018, pp. 803–812.*
- 621 [64] E. Hoffer, I. Hubara, D. Soudry, Train longer, generalize better: closing the generalization gap in large  
622 batch training of neural networks, in: *Advances in Neural Information Processing Systems 30, Curran*  
623 *Associates, Inc., 2017, pp. 1731–1741.*
- 624 [65] F. Pace, D. Venzano, D. Carra, P. Michiardi, Flexible scheduling of distributed analytic applications,  
625 *CCGrid '17, IEEE Press, 2017, pp. 100–109.*
- 626 [66] F. P. Kelly, A. K. Maulloo, D. K. Tan, Rate control for communication networks: shadow prices,  
627 proportional fairness and stability, *Journal of the Operational Research society* 49 (3) (1998) 237–252.
- 628 [67] J. Konečný, B. McMahan, D. Ramage, Federated optimization: Distributed optimization beyond the  
629 datacenter, in: *Neural Information Processing Systems (workshop), 2015.*