Reproducible Parallel Stochastic Gradient Descent for Very High Dimensional Data

Ziyad Benomar
École polytechnique, Institut Polytechnique de Paris,
Applied mathematics department

supervisor
Victor Nicollet
Chief Technical Officer, Lokad

referent
Matthieu Lerasle
Professor at Institut Polytechnique de Paris
Head of the Master's program "Mathematics of Randomness - StatML"

April - August 2022

Abstract

We are interested in the study of distributed stochastic gradient descent algorithms and their implementation on a machine with several processors. We attack the problem from a performance and computational speed point of view, and we will also impose on our solutions to be reproducible, i.e. to give the same results if run on the same data set. We will focus mainly on the PR-SGD algorithm introduced in [1], and we will give some adaptations that improve the execution speed when time needed to load the data on the computer's RAM is very long. For each proposed algorithm, we give a proof of convergence and an estimate of its runtime. Finally, we compare our algorithms using a notion of ε-performance that we define.

INSTITUT POLYTECHNIQUE DE PARIS

LOKAD Quantitative Supply Chain

Acknowledgments

First of all, I would like to thank my supervisor, Victor Nicollet, who trusted for this internship, and gave me the opportunity to work on a very rich research subject. During these last months, he followed my work very closely, reviewed my code, helped me to develop good intuitions about parallel gradient descent algorithms, and taught me a lot of new concepts and techniques related to distributed computing and programming with C#.
I would also like to thank Joannes Vermorel, CEO of the company, who first presented Lokad to me and who also was very available to answer my questions related to the company or to the internship, and who regularly gave me feedback about my work. I also thank Paul Peseux, PhD student at Lokad who reviewed my report, and the other members of the R&D team for helping me when needed and for all the interesting conversations, and of course I thank the Admins team for welcoming me in Lokad and for helping me with the administrative procedures during the whole period of the internship.
Finally, I would like to thank Matthieu Lerasle, my referent at Institut polytechnique de Paris, and the leading jury member for my defense, and I also thank the other members of the jury for taking the time to read this report and evaluate my work.

About Lokad [1]

Lokad is a software editor created in 2008, with the objective of introducing advanced technologies in the supply chain environment. Since its creation, the company has invested a lot on the R&D part. The research aspect has allowed to better understand the business problems and their mathematical modeling, while the development aspect has allowed to develop powerful computational tools to tackle these problems. Notably, Lokad released in 2014 its relational programming language Envision, providing features to make predictions from data given by the clients, and include them in the decision making process, taking into account the accuracy of the estimator.
The company's areas of interest are optimization, machine learning and algorithms. The company has today more than 45 employees, partitioned into several teams including the R&D team responsible for maintaining and improving Envision, and the supply chain scientists team that uses it to analyze data and propose solutions to the clients.

[1] The work presented in this report was carried out during my internship in Lokad, for my Master's degree in Ecole Polytechnique and Institut Polytechnique de Paris

Contents

1 Introduction

Many problems in science or engineering can be reduced to constrained optimization problems. Such problems are increasingly at stake, especially with the exponential rise of deep learning, having applications in countless fields ranging from medicine to automotive production. The objective is typically to minimize a loss function that is difficult to express or study analytically. In the context of machine learning in general we can only observe noisy values of our objective function and its gradient, thus we use stochastic gradient descent algorithms (SGD).

Although these algorithms have proven to be efficient theoretically and also in practice, the handling of massive data and the size of the neural networks used make the optimization process slow and require a very long computation time. Work has therefore been done to exploit the computational power of today's machines in order to accelerate the speed of computation, in particular by distributing the computations on different processors. An immediate difficulty to do this comes from the fact that SGD algorithms are by nature sequential: the result at step $t-1$ is used at step t. Distributing the computation might therefore give a result that would be less good in terms of convergence of the objective function to its minimum after using the same number of observations. However, the distribution of the computation is still efficient because the execution time is faster, and thus we reach small values of our objective function F in a shorter computation time. The distribution of the optimization process is even more relevant in other contexts such as federated learning, where each worker has its own private data set that cannot be shared. We are not interested in this case, and we place ourselves in the setting of a machine with several processors, and the objective is to improve the execution time of the algorithm necessary to reach a certain value $F^\star + \varepsilon$ of the objective function.

We are also interested in the often neglected aspect of the reproducibility: we want two independent runs of the algorithm on the same training set and with the same initial conditions to give the same result (the execution time may vary from a run to another). In the majority of articles in the literature that focus on distributed SGD, reproducibility is not taken into account. However, this has become an important issue in recent years especially after the reproducibility crisis revealed by a statistic published in Nature 2016, which showed that a large part of the results presented in machine learning papers were not reproducible. Having reproducible experiments allows to prove the correctness of the adopted approach, and confirms that the results are not the result of a lucky chance. Otherwise, one could deduce that a certain configuration of parameters or a new method is more efficient when it is not the case, especially when conducting costly experiments that cannot be made enough times to have a good estimate of the average result. Moreover, from a code production point of view, it is also necessary to be able to detect eventual problems in the code and correct them.

We place ourselves in the case of very high dimensional data, that requires an important time to be loaded from the hard disk of the computer to its Random Access Memory (RAM). This will reduce the efficiency of PR-SGD, and our objective will be to design adaptations that overcome this difficulty in that particular case.

2 Related Work

In the context of stochastic gradient descent, we want to minimize a differentiable stochastic objective function F. During T iterations, we observe realizations $\nabla F_1, \ldots, \nabla F_T$ of its gradient ($\mathbb{E}[\nabla F_t] = \nabla F$), and we use them instead of the real gradient of F for updating the parameters vector with a learning rate λ. The convergence and the performance of SGD have been widely studied and tested with different classes of objective functions. There have also been adaptations and improvements of SGD, leading to many more powerful optimization algorithms such as ADAM [2].
A possible way to improve SGD is to try to speed up the algorithm by parallelizing the computation. Indeed, in the context of machine learning, we have a table of observations used to compute the stochastic gradients, we can imagine having several processors reading the simultaneously different lines from the table in order to reduce the computation time. A first idea would be to make a gradient descent by mini-batches as proposed in [3], but a better strategy is to have each processor doing a local gradient descent

and to synchronize the obtained parameters regularly after a few iterations, this gives a better convergence than the mini-batch approach as proved in [4] and [5]. Several papers were interested in the theoretical and experimental study of this algorithm and its variants like [6] and [7]. One concern is the waiting times: when a processor is faster than the others or when the execution times are random, we have idles appearing, that is to say that the processors having finished their tasks first must wait for the last ones to finish too before synchronizing and continuing their calculations. The Hogwild algorithm introduced in [8] gives a lock free solution, and it is one of the most popular distributed gradient descent algorithms. In our case, we are interested as explained in the introduction in the aspect of reproducibility, and Hogwild is not reproducible by construction because the synchronization moment is independent from the number of iterations done by each processor, the randomness in the outcome of the algorithm comes therefore from the randomness of the execution times of the processor: the time needed for a same processor to execute a task is never deterministic. The Parallel Restarted Stochastic Gradient Descent (PR-SGD) [1], on the other hand, is reproducible and it is the algorithm we will focus on. Further explanations on PR-SGD are given in Section 5.

3 Formal Problem

3.1 Setting

The optimization problem In the classical setting of stochastic gradient descent (SGD), we want to optimize a noisy objective function F. If we assume that the noise is caused only by the observed data, then the problem can be written as follows: we have a function $f : \mathcal{X} \times \mathbb{R}^d \mapsto \mathbb{R}$ differentiable with respect to the second variable, where $\mathcal{X}$ is the set of observations. We have N observations $a_1, \dots, a_N \in \mathcal{X}$ that are iid and follow a probability law ζ. At each step $1 \leq n \leq N$ we observe $F_n := f(a_n; \cdot)$, and we want to find the vector $\theta^\star$ minimizing

$$F : \theta \in \mathbb{R}^d \to \mathbb{E}_{a\sim\zeta}[f(a;\theta)] = \mathbb{E}[F_n(\theta)].$$

The functions $F, F_1, \dots, F_N$ are differentiable with respect to θ (for any $a_1, \dots, a_N$), and we have $\nabla F(\theta) = \mathbb{E}[\nabla F_n(\theta)]$ for any $\theta \in \mathbb{R}^d$. We also assume that we dispose of k workers (processors) $w_1, \dots, w_k$, and we wish to perform a parallel stochastic gradient descent (PSGD). We impose however one constraint on our PSGD algorithm: that is to be reproducible (see Section 3.2).
We can assume having infinite observations by defining $a_n := a_{(n \bmod N)}$ for $n \geq N$.

Assumptions on the Objective Function The functions encountered in the optimization of neural network parameters are not convex in general functions and they present very few regularity properties. However, in order to give theoretical guarantees of the optimization algorithms, it is necessary to make some assumptions on the functions f and F. In addition to differentiability, we will assume that

- F is L-smooth:
$$\|\nabla F(\theta) - \nabla F(\theta')\| \leq L\|\theta - \theta'\| \quad \forall \theta, \theta' \in \mathbb{R}^d, \tag{A1}$$
- there exists a constants $\sigma, G > 0$ such that for any $\theta \in \mathbb{R}^d$
$$\mathbb{E}_{a\sim\zeta}[\|\nabla_\theta f(a,\theta) - \nabla F(\theta)\|^2] \leq \sigma^2, \tag{A2}$$
$$\mathbb{E}_{a\sim\zeta}[\|\nabla_\theta f(a,\theta)\|^2] \leq G^2. \tag{A3}$$

These assumptions are the same as in [1]. The first one ensures that the gradient cannot change too quickly, thus the value of the gradient is in a sense informative of the region where it is and not only the point θ, and with a sufficiently small step size we are sure to move in a good direction.
The second assumption is essential for the parallel setting. Without going much into details, the idea later will be to have a local parameter vector θ^i for each worker w_i, and a central vector θ that is their average. Having a bounded variance guarantees that the averaged parameter vector has a smaller variance compared to a normal gradient descent: it would be divided by the number of workers k, and this enhances the convergence of the parallel algorithm.

The data We place ourselves in the case of very high dimensional data, and contrary to the federated learning setting, all the workers have access to the whole data set. This might seem to simplify the problem, but in fact it makes the access to the data more time consuming. In fact, due to the very high dimension of the observations space $\mathcal{X}$, we cannot read all the dataset at once: it is too large to be loaded on the Random Access Memory (RAM) of the computer. Moreover, the dataset can only be accessed via the first observation, and it can only be read linearly, i.e reading an observation a_n requires passing over all the observations $a_1, \dots, a_{n-1}$.
To respect the limied memory constraint, each worker is only allowed to load P lines into the memory at once: when a worker reaches the observation a_P, it frees its dedicated memory and loads the next P observations $a_{P+1}, \dots, a_{2P}$. In all the remainder, these blocks of P observations are called "Pages".

The workers Each worker w_i has a local parameters vector θ^i and a head h^i positioned on some observation a_{h^i}. Initially, the heads of all the workers are pointing to the first observation: $h^i = 1$.

The workers are controlled by a core script, from which we can command them to execute the following elementary requests:

- NextObs(w_i): moves the head of the worker to the next observation ($h^i \leftarrow h^i + 1$). If the worker reaches the end of a page, it loads the next one in order to do this action.
- SgdStep(w_i, λ): performs one step of the stochastic gradient descent using the parameters vector θ^i and the observation a_{h^i}: $\theta^i \leftarrow \theta^i - \lambda \nabla F_{h^i}(\theta^i)$.
- SetParameters($w_i; \theta$): sets the worker's parameters vector to θ: $\theta^i \leftarrow \text{Copy}(\theta)$
- GetParameters(w_i): returns the parameter's vector of the worker θ^i.

We also assume that the workers have the same processing speed. However, the time needed for executing a task is never deterministic. having identical workers means that the time needed by each of them to execute a same task follows the same probability distribution.

3.2 Reproducible Algorithms

We explained in the introduction the importance of having reproducible algorithms. We give here a formal definition of reproducibility.

Definition 1. *Consider an algorithm $\mathcal{A}$ that takes an input X, executes set of actions* $\text{action}_1, \text{action}_2, \dots, \text{action}_m$ *that can be random, then returns an output Y. This algorithm is said to be reproducible if Y depends only on X.*

The pseudo-randomness due to generating random values during the algorithm can easily be controlled, but the randomness we cannot control is the one due to the execution time. This can be encountered in many parallel algorithms as Hoglwild [8].

Example 2. *A simpler example would be having several processors, each executing a different task returning some numerical value, and let us say that the output of the algorithm is the value returned by the first processor to finish its calculation. The output of this algorithm is not reproducible because of the randomness of the execution times of the processors.*

3.3 Benchmarks and Performance Indicators

The speed on an algorithm is usually measured by an asymptotic big-O estimate of the number of its elementary operations, but we will aim in the following for precise non-asymptotic estimates. In order to evaluate the performance of a distributed SGD algorithm, we will compare its runtime and the loss it obtains to those of the non-distributed SGD to measure how much we gain from using several workers.

Convergence Criterion Since we are interested in non-convex functions, it is very difficult to have convergence rates for the value of the objective function. Following the convention used in [1], we will measure the convergence by estimating the quantity $\|\nabla F(\theta)\|^2$. Having small gradients means that we are close to a local minimum or maximum of F, and with an algorithm that is not very bad we would hopefully be close to a local minimum.
Note that in the case of κ-strongly convex functions, this gives a control on the value of the function or the distance to the optimum $\theta^\star$ of F thanks to the inequality

$$\frac{\kappa}{2}\|\theta - \theta^\star\|^2 \leq F(\theta) - F(\theta^\star) \leq \frac{1}{2\kappa}\|\nabla F(\theta)\|^2 \qquad \forall \theta \in \mathbb{R}^d.$$

Performance Evaluation To compare the execution time of two algorithms, they must have comparable stopping criteria. In our case we compare the time they need to process a same number of observations N. The ideal with k workers would be to have an execution time divided by k compared to SGD, but of course this cannot be reached because there are communication and synchronization costs. As for the loss, stopping after processing N observations means that with more workers, each one will do less local SGD steps leading to a slower convergence. On the other hand, averaging their results might play a corrective role and improve the convergence. It is therefore hard to predict the effect of using multiple workers on the final loss value.

In the remainder, for any algorithm $\mathcal{A}$ reading the observations table and aiming at minimizing F, we will note $\mathcal{T}(\mathcal{A}, N)$ the time it needs for processing N observations.

Definition 3 (Execution Time Ratio)**.** *Consider a distributed SGD algorithm $\mathcal{A}$ with k workers. We then define the execution time ratio of $\mathcal{A}$ by*

$$\mathrm{TR}(\mathcal{A}) := \lim_{N\to+\infty}\left(\frac{\mathcal{T}(\mathrm{SGD}; N)}{k \cdot \mathcal{T}(\mathcal{A}; N)}\right) \leq 1.$$

The value 1 cannot be reached because $\mathcal{A}$ cannot be k times faster than SGD due to the additional communication time when using multiple workers instead of just one.

We take the limit when $N \to \infty$ because we do not want to take into account the eventual constant time at the beginning of any algorithm for initializing the workers or do other tasks that are not described in the pseudo-code.
This indicator only concerns the execution time, but we must not neglect the loss. We need to find a good compromise between the time we gain and the quality of the final result. The real indicator to maximize is therefore the "ε-performance" in the definition below.

Definition 4 (ε-performance)**.** *Consider a distributed algorithm $\mathcal{A}$ minimizing F using k workers, that computes a sequence of parameter vectors $\theta_1, \theta_2, \ldots$, processing B observations to compute each new θ_t. We define the number of iterations necessary for $\mathcal{A}$ to reach a certain level ε of $\|\nabla F\|^2$:*

$$T_\varepsilon(\mathcal{A}) := \min\{t \in \mathbb{N} :\ \|\nabla F(\theta_t)\|^2 \leq \varepsilon\},$$

and we define the execution time necessary for that

$$\mathcal{T}_\varepsilon(\mathcal{A}) := \mathcal{T}(\mathcal{A}; BT_\varepsilon(\mathcal{A})).$$

Note that $BT_\varepsilon(\mathcal{A})$ is simply the number of observations processed after $T_\varepsilon(\mathcal{A})$ observations. $\mathcal{T}_\varepsilon$ is eventually $+\infty$ if the aimed loss level ε is never reached. We then define the "ε-performance" of $\mathcal{A}$ by

$$\mathcal{P}_\varepsilon(\mathcal{A}) := \frac{\mathcal{T}_\varepsilon(\mathrm{SGD})}{k \cdot \mathcal{T}_\varepsilon(\mathcal{A})}.$$

The ideal would be to have a performance value close to 1.

The definitions above hold when considering that the execution time is deterministic, but this is never the case. We should thus write them as ratios of expectations:

$$\mathrm{TR}(\mathcal{A}) := \lim_{N\to+\infty}\left(\frac{\mathbb{E}[\mathcal{T}(\mathrm{SGD}; N)]}{k \cdot \mathbb{E}[\mathcal{T}(\mathcal{A}; T_N)]}\right), \qquad \mathcal{P}_\varepsilon(\mathcal{A}) := \frac{\mathbb{E}[\mathcal{T}_\varepsilon(\mathrm{SGD})]}{k \cdot \mathbb{E}[\mathcal{T}_\varepsilon(\mathcal{A})]}.$$

Estimating the Runtime To give estimations of the execution times of our Algorithms, we need to make some assumptions: let us fix $I \in \mathbb{N}^*$, we assume that

- the time needed by a worker to achieve I times SgdStep is an independent uniform random variable in $[\alpha_I, \beta_I]$ where $\beta_I > \alpha_I > 0$,
- the synchronization time between the core and the workers is of the form γk with $\gamma > 0$,
- no time is needed for the executing the operation NextObs when the worker stays on the same page (it is negligible compared to the other time constants),
- loading a new page of observations into the memory requires a time $\mu > 0$.

In all the following, we will write simply α, β instead of α_I, β_I as long as there is no confusion on the value of I.

Runtime and Convergence Rate of SGD

Proposition 5. *With the notations above, the expected runtime of* SGD *implemented with the actions described in Section 3.1 is*

$$\mathbb{E}[\mathcal{T}(\text{SGD}; N)] = \left(\frac{\alpha+\beta}{2I} + \frac{\mu}{P}\right) N.$$

Proof. The SGD can be implemented by repeating the actions SgdStep(w_1, λ) then NextObs(w_1) using the same worker N times. All the runs of NextObs(w_1) during the N steps will take a time $\mu N/P$, because N/P pages are loaded.
For any $t = 1, \ldots, TN/I$, let τ_t the time needed to do I stochastic gradient descent steps using the observations $a_{tI+1}, \ldots, a_{(t+1)I}$. We have

$$\mathbb{E}[\mathcal{T}(\text{SGD}; N)] = \mathbb{E}\left[\sum_{t=1}^{N/I} \tau_t + \frac{\mu N}{P}\right] = \sum_{t=1}^{N/I} \mathbb{E}[\tau_t] + \frac{\mu N}{P}\mu = \frac{\alpha+\beta}{2I} N + \frac{\mu N}{P}.$$

□

We give also the convergence rate of SGD. We will not prove it here because later in Theorem 9 we demonstrate a more general result.

Theorem 6. *If F satisfies the assumptions A1 and A2, then for any $T \geq k$, with $\lambda = \frac{1}{L\sqrt{T}}$, we have that the sequence of $\mathbb{R}^d$ defined by $\theta_0 \in \mathbb{R}^d$ and $\theta_t = \theta_{t-1} - \lambda \nabla F_t(\theta_{t-1})$ for $t \geq 1$ verifies*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] \leq (2L(F(\theta_0) - F(\theta^\star)) + \sigma^2)\frac{1}{\sqrt{T}}.$$

4 Skip-Take Requests

As mentioned in the previous section, the workers' heads are initially positioned on the first observation, and they can move through the observation table only linearly: When the head is placed on an observation, the worker can choose to process it and thus perform a gradient descent or just ignore it and move on to the next one. This must be done in coordination with the other workers so that each observation is processed exactly once.
To avoid communication after each observation, we will send to workers requests in the form of integer pairs (Skip,Take). The worker will therefore ignore the first "Skip" observations (counting from the one on which its head is placed) and then perform stochastic gradient descents using the following "Take" observations. This functioning is explained in Algorithm 1.

Algorithm 1: stRequest$(w_i, \theta, S, I, \gamma)$ Skip-Take Request

Input : Worker w_i, a parameters vector θ, non-negative integers (S, I) and a learning rate λ.
Output: The local parameters' vector θ^i of w_i is set to the result of gradient descent using the initial parameters θ and the observations assigned by (S, I)

```
θ^i ← copy of θ;
for ℓ = 1, ..., S do
    NextObs(w_i);
end for
for ℓ = 1, ..., I do
    SgdStep(w_i, λ);
    NextObs(w_i);
end for
```

We now need to choose the values of (S^i, I^i) for each w_i so as to scan the entire table of observations without overlaps between workers. A naive way to do this is to take for example

$$\forall 1 \leq i \leq k : \qquad S^i = (i-1) \times T/k, \qquad I^i = T/k.$$

this choice is far from being from efficient because the workers who must skip until the last observations will have to load all the pages of data they come through even though they will not process them.
In the general case, I^i are much smaller than T/k and may be different from each other or even change from a request to another. The values $(S^i)_{1 \leq i \leq k}$ should be modified accordingly to cover all the observations.

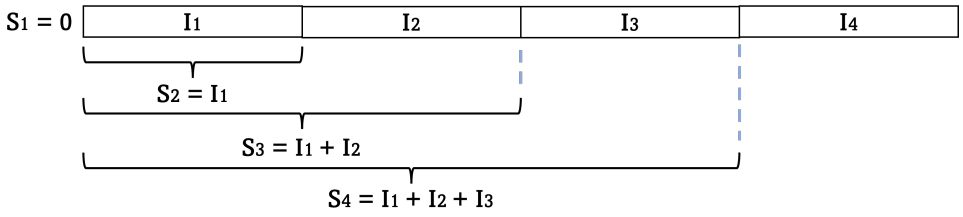

Figure 1: Initial skips for fixed $I_1, \ldots, I_k$ in the case of $k = 4$

Proposition 7. *If we denote (S_t^i, I_t^i) the request submitted to the worker w_i at step t, then all the observations are processed exactly one time if the skip values verify*

- *for all $1 \leq i \leq k$,*
$$S_t^1 = \sum_{j<i} I_t^1,$$
- *for all $1 \leq i \leq k$ and for all $t \geq 2$*
$$S_t^i = \sum_{j<i} I_t^j + \sum_{j>i} I_{t-1}^j.$$

Proof. For each $n \geq 1$, let $\Sigma_t := \sum_{u=1}^{t-1} \sum_{i=1}^{k} I_u^i$. At step t, we process all the observations a_n such that $\Sigma_t < n \leq \Sigma_{t+1}$. More precisely, each worker w_i will process the observations such that

$$\Sigma_t + \sum_{j=1}^{i-1} I_t^j < n \leq \Sigma_t + \sum_{j=1}^{i} I_t^j. \tag{1}$$

and its head h^i will be positioned by the end of the step on the observation $\Sigma_t + \sum_{j=1}^{i} I_t^j + 1$. For $t = 1$, this is clearly true as $\Sigma_1 = 0$. For $t \geq 2$ and for a worker w_i, assuming that the result is true up to $t-1$, we have at the beginning of the step $h^i = \Sigma_{t-1} + \sum_{j=1}^{i-1} I_{t-1}^j + 1$. After skipping S_n^i observations we have

$$\begin{aligned} h^i &= (\Sigma_{t-1} + \sum_{j=1}^{i} I_{t-1}^j + 1) + S_t^i = \Sigma_{t-1} + \sum_{j=1}^{i} I_{t-1}^j + \sum_{j<i} I_t^j + \sum_{j>i} I_{t-1}^j + 1 \\ &= \Sigma_{t-1} + \sum_{j=1}^{k} I_{t-1}^j + \sum_{j<i} I_t^j + 1 \\ &= \Sigma_t + \sum_{j<i} I_t^j + 1. \end{aligned}$$

w_i will after that process I_t^i observations, i.e the ones having an index n satisfying Inequality 1, and we will have by the end of the step $h_t^i = \Sigma_t + \sum_{j=1}^{i} I_t^j + 1$. □

To update all the variables $(S^i)_{1 \leq i \leq k}$ as defined in Proposition 7 we need $O(k^2)$ time, but we can do it in just a $O(k)$ time by observing that for any $n \geq 2$ we have the following induction formula

$$\begin{aligned} S_t^1 &= \sum_{i=2}^{k} I_{t-1}^i \\ S_t^i &= S_t^{i-1} + I_t^{i-1} - I_{t-1}^i \text{ for all } 2 \leq i \leq k. \end{aligned}$$

Remark 8. *When we will present the pseudo-code of distributed SGD algorithms that use the the function stRequest(), we will not explicitly give the "skip" size S, we will always choose it as in Proposition 7.*

5 First Solutions

The most immediate distributed SGD algorithm we can think of is a batch SGD [3] with the batch being of size k. As explained in Section 2, adopting gradient descents with synchronization [5] is more efficient. We present therefore the Parallel Restarted SGD algorithm (PR-SGD) studied in [1], described in Algorithm 2.

Algorithm 2: PR-SGD(θ_0, T, I, λ): Parallel Restarted SGD

Input : Initial parameters vector θ_0, a positive integer T, a learning rate λ and a synchronization interval I
Output: a parameters vector θ_T

```
for t=1, ..., T do
    for [Parallel] i = 1, ..., k do
        stRequest(w_i, θ_{t-1}, I, λ);          // see Algorithm 1
    end for
    Wait for all the workers to finish executing their requests;
    θ_t ← (1/k) Σ_{i=1}^{k} θ_t^i;
end for
return θ_T
```

The synchronization interval I, or what we will also call the "take size" is the number of iterations of local SGD each worker does before averaging their parameters. The choice of I will be discussed later in Section 8.2, but let us note that it must be chosen very carefully because a small take size will slow down the execution, and choosing a big take size will give a bad convergence because the different workers might converge to different local minima, and averaging the obtained parameters then makes no sense as we illustrate in Figure 2.

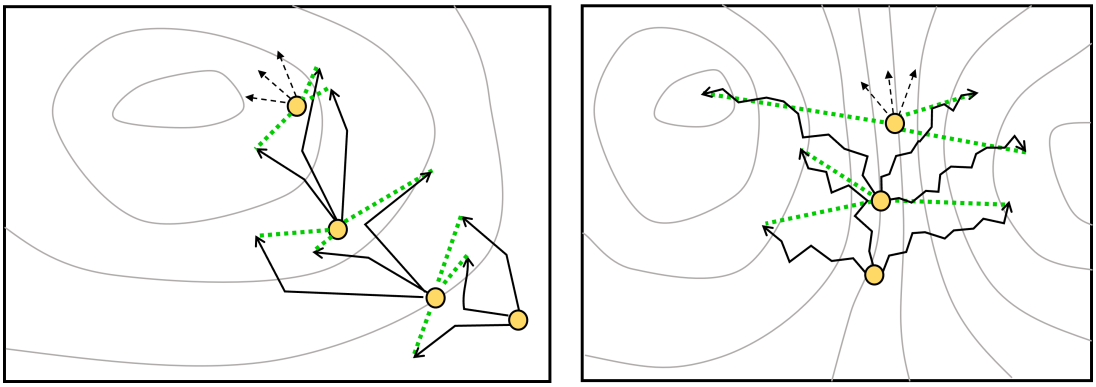

Figure 2: PR-SGD with $k = 3$, run examples in the cases of small and big I

It is proved in [1] that if F satisfies Assumptions A1, A2 and A3, then for learning rate and a synchronization interval I well chosen we have

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] \leq O\left(\frac{1}{\sqrt{kIT}}\right).$$

This bound indicates that we gain a linear speed up when using k threads. In fact, over T steps, Algorithm 2 processes $N := kIT$ observations. The bound obtained is therefore proportional to $1/\sqrt{N}$, which is the same theoretical bound we have for the non-distributed SGD. This means that using the same number of observations we get the same convergence rate for SGD and PR-SGD. Of course PR-SGD has the advantage of being much faster than SGD.
The main issue with Algorithm 2 is that before every synchronization, the workers must wait for each other to finish their requests, a part of their computational capacity will be therefore wasted.
If the workers have different speeds, then this can be solved by assigning to them tasks with different sizes as explained in [1]. In fact, if each worker w_i has a computation speed v_i, then we simply need to assign tasks of I_i such that all the values $v_i \times I_i$ are equal.

In the case of the workers being processors of a same machine, they will always have the same computation capabilities, but as each processors will run some operations other than the ones in the algorithm (related to the operating system, memory cleaning, ...) we will have random execution times, and the idles cannot be eliminated by changing the tasks' sizes.

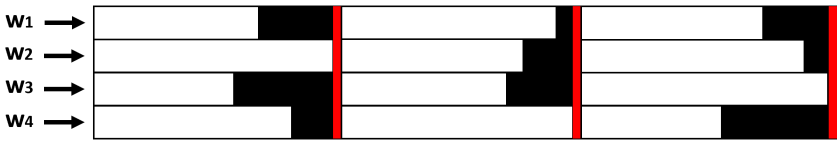

Figure 3: Idles in the case of 4 workers

We show in Figure 3 an example of such a situation: each row presents the activity of a worker over the time: white rectangles indicate that the worker is running a task, black ones that it is waiting, and the red ones indicate the time spent for synchronization.

5.1 Convergence Analysis in the case $I = 1$

We assume in this section that $I = 1$. We will redo here the convergence proof established in [1], but with lighter calculations because we are in a simpler setting. In fact, the authors of [1] treat the case of any positive integer I, and they suppose moreover that each worker w_i observes data according to a

certain law ξ_i that can be different, and tries to minimize a function g_i, and they study the convergence of Algorithm 2 towards a minimum of $g := \frac{1}{k}\sum_{i=1}^{k} g_i$.

The reason we reproduce this proof is that we need to use the same proof scheme later for other derivated algorithms, and we will want to compare the differences between the proofs and the bounds we obtain.

We denote $F_{i,t} := F_{(t-1)k+i}$ the noisy realization of F observed by worker i at step t for each $t \geq 1$ and $i \in \{1, \dots, k\}$. With $I = 1$, we have for any $t \geq 1$ that $\theta_t^i = \theta_{t-1} - \lambda \nabla F_{i,t}(\theta_{t-1})$ for all $1 \leq i \leq k$ and $\theta_t = \frac{1}{k}\sum_{1 \leq i \leq k} \theta_t^i$ and thus

$$\theta_t = \theta_{t-1} - \frac{\lambda}{k} \sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-1}). \tag{2}$$

Theorem 9. *If F is a differentiable function satisfying the assumptions A1 and A2, then for any $T \geq k$, with $\lambda = \frac{\sqrt{k}}{L\sqrt{T}}$ the sequence of $\mathbb{R}^d$ defined by $\theta_0 \in \mathbb{R}^d$ and Equation 2 verifies*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] \leq (2L(F(\theta_0) - F(\theta^\star)) + \sigma^2)\frac{1}{\sqrt{kT}}.$$

Proof. let $t \in \{1, \dots, T\}$, using the L-smoothness of F and the linearity of the expectation we have

$$\mathbb{E}[F(\theta_t)] - \mathbb{E}[F(\theta_{t-1})] \leq \mathbb{E}[\langle \nabla F(\theta_{t-1}), \theta_t - \theta_{t-1}\rangle] + \frac{L}{2}\mathbb{E}[\|\theta_t - \theta_{t-1}\|^2], \tag{3}$$

we will now prove upper bounds on the two terms in the right, that will make $\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2]$ appear. By Equation 2 we have

$$\mathbb{E}[\langle \nabla F(\theta_{t-1}), \theta_t - \theta_{t-1}\rangle] = -\frac{\lambda}{k}\sum_{i=1}^{k} \mathbb{E}[\langle \nabla F(\theta_{t-1}), \nabla F_{i,t}(\theta_{t-1})\rangle],$$

but since the observations are independent and θ_{t-1} only depends on the observations a_m with $m \leq (t-1)kI$ ($F_{i,u} = F_{(u-1)kI+i} = f(a_{(u-1)kI+i}; .)$), we have for any $1 \leq i \leq k$

$$\begin{aligned}
\mathbb{E}[\langle \nabla F(\theta_{t-1}), \nabla F_{i,t}(\theta_{t-1})\rangle] &= \mathbb{E}[\mathbb{E}[\langle \nabla F(\theta_{t-1}), \nabla F_{i,t}(\theta_{t-1})\rangle \mid \theta_{t-1}]] \\
&= \mathbb{E}[\langle \nabla F(\theta_{t-1}), \mathbb{E}[\nabla F_{i,t}(\theta_{t-1}) \mid \theta_{t-1}]\rangle] \\
&= \mathbb{E}[\langle \nabla F(\theta_{t-1}), \nabla F(\theta_{t-1})\rangle] \\
&= \mathbb{E}[\|\nabla F(\theta_{t-1})\|^2]
\end{aligned}$$

thus

$$\mathbb{E}[\langle \nabla F(\theta_{t-1}), \theta_t - \theta_{t-1}\rangle] = -\lambda \mathbb{E}[\|\nabla F(\theta_{t-1})\|^2]. \tag{4}$$

On the other hand, using again Equation 2:

$$\begin{aligned}
\mathbb{E}[\|\theta_t - \theta_{t-1}\|^2] &= \lambda^2 \mathbb{E}[\|\frac{1}{k}\sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-1})\|^2] \\
&= \lambda^2 \mathbb{E}[\mathbb{E}[\|\frac{1}{k}\sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-1})\|^2 \mid \theta_{t-1}]] \\
&= \lambda^2 \mathbb{E}[\mathbb{E}[\|\frac{1}{k}\sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-1}) - \nabla F(\theta_{t-1})\|^2 \mid \theta_{t-1}] + \mathbb{E}[\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2 \mid \theta_{t-1}]] \\
&= \lambda^2 \mathbb{E}[\|\frac{1}{k}\sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-1}) - \nabla F(\theta_{t-1})\|^2] + \lambda^2 \mathbb{E}[\|\nabla F(\theta_{t-1})\|^2].
\end{aligned}$$

The third equality is true because of Lemma 30 applied with the conditional expectation $E[\cdot \mid \theta_{t-1}]$. The variables $(\nabla F_{i,t}(\theta_{t-1}))_{1\leq i\leq k}$ are independent conditionally to θ_{t-1} and their respective expectations knowing θ_{t-1} are $\nabla F_{i,t}(\theta_{t-1})$, we can therefore use Lemma 31 and write

$$\mathbb{E}[\|\frac{1}{k}\sum_{i=1}^{k}\nabla F_{i,t}(\theta_{t-1}) - \nabla F(\theta_{t-1})\|^2 \mid \theta_{t-1}] = \frac{1}{k^2}\sum_{i=1}^{k}\mathbb{E}[\|\nabla F_{i,t}(\theta_{t-1}) - \nabla F(\theta_{t-1})\|^2 \mid \theta_{t-1}].$$

Taking the expectation on this inequality and using Assumption A2 we deduce that

$$\mathbb{E}[\|\theta_t - \theta_{t-1}\|^2] \leq \frac{\lambda^2\sigma^2}{k} + \lambda^2\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2]. \tag{5}$$

Finally, with the inequalities 3, 4 and 5 and observing that $\lambda = \frac{\sqrt{k}}{L\sqrt{T}} \leq 1/L$ (because $T \geq k$), we have

$$\begin{aligned}
\mathbb{E}[F(\theta_t)] - \mathbb{E}[F(\theta_{t-1})] &\leq -\lambda\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] + \frac{\lambda^2\sigma^2 L}{2k} + \frac{\lambda^2 L}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] \\
&= -\frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] + \frac{\lambda^2\sigma^2 L}{2k} + \underbrace{(\frac{\lambda^2 L}{2} - \frac{\lambda}{2})}_{=\lambda L/2(\lambda - 1/L)\leq 0}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] \\
&\leq -\frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] + \frac{\lambda^2\sigma^2 L}{2k}.
\end{aligned}$$

Summing over $t = 1, \ldots, T$ and dividing by T gives

$$\frac{1}{T}(F(\theta^\star) - F(\theta_0)) \leq \frac{1}{T}(\mathbb{E}[F(\theta_T)] - F(\theta_0)) \leq -\frac{\lambda}{2T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] + \frac{\lambda^2\sigma^2 L}{2k},$$

rearranging the terms we obtain

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] \leq \frac{2}{\lambda T}(F(\theta_0) - F(\theta^\star)) + \frac{\lambda\sigma^2 L}{k} = (2L(F(\theta_0) - F(\theta^\star)) + \sigma^2)\frac{1}{\sqrt{kT}}.$$

□

5.2 Estimating the Execution Time

Let $T := N/(kI)$ the number of steps necessary for Algorithm 2 to process N observations. Its execution time can be written as

$$\mathcal{T}(\text{PR-SGD}; N) = \sum_{t=1}^{T}\left(\max_{1\leq i\leq k}\{\tau_{i,t}\} + \gamma k\right) + \xi_T, \tag{6}$$

where $\tau_{i,t} \sim \mathcal{U}[\alpha_I, \beta_I]$ is the time spent by w_i at step t for running I times SgdStep. We also have γk in the sum because we need to synchronize the results at each step. Finally, ξ_T is the delay generated because of loading the pages of data.

Lemma 10. *If $2kI < P$ and $\mu > 2(\beta - \alpha)$ then ξ_T is of the form*

$$\xi_T = (1 + \varepsilon)\mu\frac{kIT}{P}.$$

where ε is a random variable taking values almost surely in $[0, 1]$.

Proof. The condition $2kI < P$ only means that the number of observations $2kI$ processed during two steps is smaller than the number of observations per page P. We will show in the following that charging a page by all the workers is done at most on two steps, the previous condition simply guarantees that we

will not have different pages charging during the same step.
Let $m \in \{1, \dots, M\}$ with $M = kIT/P$ is the number of pages visited during T observations, and let $\ell := mP + 1$: a_ℓ is the first observation on the page m. Using the proof of Proposition 7, we have that a_ℓ is processed by the worker w_j at step t such that

$$((t-1)k + (j-1))I < \ell \leq ((t-1)k + j)I,$$

because in the case of all the workers having the same task size I, the quantity Σ_t in Proposition 7 becomes simply $\Sigma_t = ((t-1)k + (j-1))I$. Concretely, $t-1$ and $j-1$ are respectively the quotient and the rest of the division of $[\ell/I]$ by k.
We also have by the proof of Proposition 7 that at step $t-1$, all the workers were processing observations a_n such that $n \leq \Sigma_t < \ell$, therefore all workers w_i such that $j > i$ skipped a_ℓ during step t because they process observations coming after it ($n > \Sigma_t + (i-1)I \geq \Sigma_t + jI \geq \ell$), which means that they loaded the page m on their cache memory during step t. Whereas the workers w_i having $i < j$ are processing during step t observations a_n with $n < \ell$, which means that they are still in the previous page, but at step $t+1$ they will be processing observations a_n with $n > \Sigma_{t+1} = \Sigma_t + kI \geq \ell$, thus they will load the page m during step $t+1$.
If all the workers load the page m during the step t, then a time μ is added to their execution times at step t:

$$\max_{1 \leq i \leq k} \{\mu + \tau_{i,t}\} = \max_{1 \leq i \leq k} \{\tau_{i,t}\} + \mu.$$

Otherwise, it is loaded by the j fist workers at step t and by the others at step $t+1$, the total execution time at step t is thus

$$\begin{aligned}
\tau_t &= \max \left\{ \max_{1 \leq i \leq j} \{\mu + \tau_{i,t}\}, \max_{j < i \leq k} \{\tau_{i,t}\} \right\} \\
&= \max_{1 \leq i \leq j} \{\mu + \tau_{i,t}\} \\
&= \mu + \max_{1 \leq i \leq j} \{\tau_{i,t}\} \\
&= \max_{1 \leq i \leq k} \{\tau_{i,t}\} + \mu + \left(\max_{1 \leq i \leq j} \{\tau_{i,t}\} - \max_{1 \leq i \leq k} \{\tau_{i,t}\} \right),
\end{aligned}$$

the second equality is true because for any i_1, i_2 such that $i_1 \leq j < i_2$ we have

$$\mu + \tau_{i_1,t} \geq 2(\beta - \alpha) + \alpha = \beta + (\beta - \alpha) \geq \beta \geq \tau_{i_2,t},$$

in the same way we find

$$\tau_{t+1} = \max_{1 \leq i \leq k} \{\tau_{i,t+1}\} + \mu + \left(\max_{j < i \leq k} \{\tau_{i,t+1}\} - \max_{1 \leq i \leq k} \{\tau_{i,t+1}\} \right),$$

thus we have

$$\tau_t + \tau_{t+1} = \max_{1 \leq i \leq k} \{\tau_{i,t}\} + \max_{1 \leq i \leq k} \{\tau_{i,t+1}\} + (1 + \delta_m)\mu$$

with

$$\delta_m := \frac{1}{\mu} \Big(\mu + \underbrace{\left(\max_{1 \leq i \leq j} \{\tau_{i,t}\} - \max_{1 \leq i \leq k} \{\tau_{i,t}\} \right)}_{\in [-(\beta-\alpha), 0] \subset [-\mu/2, 0] \ a.s.} + \underbrace{\left(\max_{j < i \leq k} \{\tau_{i,t+1}\} - \max_{1 \leq i \leq k} \{\tau_{i,t+1}\} \right)}_{\in [-(\beta-\alpha), 0] \subset [-\mu/2, 0] \ a.s.} \Big) \in [0, 1],$$

the inclusion $[-(\beta - \alpha), 0] \subset [-\mu/2, 0]$ is because of the condition $\mu > 2(\beta - \alpha)$. Finally, the delay generated when loading the page m is either μ if all the workers load the page at the same step, either $(1 + \delta_m)\mu$ with $\delta_m \in [0, 1]$ otherwise. In both cases, we can write it as $(1 + \varepsilon_m)\mu$ with $\varepsilon_m \in [0, 1]$ and the total delay due to charging the pages is therefore

$$\xi_T = \sum_{m=1}^{M} (1 + \varepsilon_m)\mu = \left(1 + \frac{1}{M} \sum_{m=1}^{M} \varepsilon_m \right) \mu M = (1 + \varepsilon)\mu M,$$

with $\varepsilon := \frac{1}{M} \sum_{m=1}^{M} \varepsilon_m$ a random variable taking values in $[0, 1]$ with probability 1. □

With the previous lemma, we can estimate the expectation of $\mathcal{T}(\text{PR-SGD}; T)$.

Proposition 11. *The expected runtime of Algorithm 2 is given by*

$$\mathbb{E}[\mathcal{T}(\text{PR-SGD}; N)] = \left(\frac{\alpha + k\beta}{k+1} + \gamma k + (1 + \mathbb{E}[\varepsilon])\frac{\mu k I}{P}\right) T,$$

with $\mathbb{E}[\varepsilon] \in [0, 1]$ *and* $T = N/(kI)$.

Proof. Using Equation 6 and Lemma 10 we have

$$\begin{aligned}\mathbb{E}[\mathcal{T}(\text{PR-SGD}; N)] &= \sum_{t=1}^{T}\left(\mathbb{E}\left[\max_{1\le i\le k}\{\tau_{i,t}\}\right] + \gamma k\right) + (1 + \mathbb{E}[\varepsilon])\frac{\mu k I}{P} T \\ &= \left(\mathbb{E}\left[\max_{1\le i\le k}\{\tau_{i,1}\}\right] + \gamma k\right) T + (1 + \mathbb{E}[\varepsilon])\frac{\mu k I}{P} T.\end{aligned}$$

the second equality is true because the variables $\tau_{i,t}$ with $1 \le i \le k$ and $1 \le t \le T$ are i.i.d.
With Lemma 32, since the variables $\left(\frac{\tau_{i,1}-\alpha}{\beta-\alpha}\right)_{1\le i\le k}$ follow a uniform distribution in $[0, 1]$, we obtain

$$\mathbb{E}\left[\max_{1\le i\le k}\{\tau_{i,1}\}\right] \;=\; \alpha + (\beta-\alpha)\mathbb{E}\left[\max_{1\le i\le k}\frac{\tau_{i,1}-\alpha}{\beta-\alpha}\right] \;=\; \alpha + (\beta-\alpha)\frac{k}{k+1} \;=\; \frac{\alpha+k\beta}{k+1},$$

which concludes the proof. □

Remark 12. *The term* $\frac{\mu k I}{P}$ *can be seen as the time needed for loading the fraction of the page that is processed during a step* $t \in \{1, \dots, T\}$*:* μ/P *is the time for loading one observation into the memory and* kI *is the number of observations used during a single iteration of the algorithm.*

Let us now give an estimation of the execution time ratio (Definition 3) of PR-SGD.

Proposition 13. *The execution time ratio of Algorithm 2 is given by*

$$\frac{1}{\text{TR}(\textit{PR-SGD})} = 1 + \frac{1}{\frac{\alpha+\beta}{2} + \frac{\mu I}{P}}\left(\frac{\beta-\alpha}{2}\left(1 - \frac{2}{k+1}\right) + \gamma k + ((1 + \mathbb{E}[\varepsilon]])k - 1)\frac{\mu I}{P}\right).$$

Remark 14. *The execution time ratio is smaller than 1 as claimed in Section 3.3, and Proposition 13 shows why for Algorithm 2:*

- $\frac{\beta-\alpha}{2}\left(1 - \frac{2}{k+1}\right)$*: this term corresponds to the delay generated by the idles, and it is due to reasons: having* $\beta \neq \alpha$ *and also having* $k \ge 2$*, if we only use one worker then we will not have idles anymore, and this term will be null,*
- $((1 + \mathbb{E}[\varepsilon]])k - 1)\mu I/P$*: the additional time cost we have for charging the pages: the operation of charging a page cannot be distributed, this is why it is linear in* k*, thus it is very costly,*
- γk*: the cost of communication between the workers and synchronization. This cost is also proportional to* k *between it is an operation we do not do in the case of non-distributed SGD.*

Proof. From Propositions 5 and 11 we have

$$\mathbb{E}[\mathcal{T}(\text{SGD}; N)] = \left(\frac{\alpha+\beta}{2} + \frac{\mu I}{P}\right)\frac{N}{I},$$

$$\mathbb{E}[\mathcal{T}(\text{PR-SGD}; N)] = \left(\beta - \frac{\beta-\alpha}{k+1} + \gamma k + (1 + \mathbb{E}[\varepsilon])\frac{\mu I}{P}\right)\frac{N}{kI},$$

we deduce that

$$\begin{aligned}\frac{1}{\mathrm{TR(PR\text{-}SGD)}} - 1 &= \lim_{N\to\infty} \frac{k\mathbb{E}[\mathcal{T}(\text{PR-SGD}; N)] - \mathbb{E}[\mathcal{T}(\text{SGD}; N)]}{\mathbb{E}[\mathcal{T}(\text{SGD}; N)]} \\ &= \frac{1}{\frac{\alpha+\beta}{2} + \frac{\mu I}{P}} \left(\beta - \frac{\beta-\alpha}{k+1} + \gamma k + (1+\mathbb{E}[\varepsilon])\frac{\mu I}{P} - \frac{\alpha+\beta}{2} - \frac{\mu I}{P} \right) \\ &= \frac{1}{\frac{\alpha+\beta}{2} + \frac{\mu I}{P}} \left(\frac{\beta-\alpha}{2}\left(1 - \frac{2}{k+1}\right) + \gamma k + ((1+\mathbb{E}[\varepsilon]])k - 1)\frac{\mu I}{P} \right).\end{aligned}$$

□

6 Pipelining: Almost Eliminating the Idles

Algorithm 2 performs well in terms of the value of the objective function found at the end. However, to eliminate the waiting times, we must keep the workers busy after finishing their tasks. We can therefore think of assigning to each worker a pipeline with tasks to be performed one after the other without waiting for the others to finish theirs. The difficulty in implementing this idea is to respect the reproducibility constraint when averaging the parameters.

In this section, we define for any $i \in \{1, \dots, k\}$ and $t \leq 0$ the function $F_{i,t}$ as the zero function on $\mathbb{R}^d \to \mathbb{R}$. This convention will avoid us treating unnecessary particular cases.
We also assume that $I = 1$, i.e at each step the workers do each a single SGD iteration then we average their parameters before moving to the next step. In the end of the section we will explain how the presented approach can be generalized to $I \geq 1$.

6.1 Parallel Descent and Synchronization

It is necessary that the workers who finish their tasks first start working on new tasks that do not impact the synchronization. This can be done by using delayed gradients. As shown in the Figure 4, if at step t we do not use the vectors computed at step $t-1$ but those computed at step $t-2$ or even before, we can allow the workers who finished first to start the next processing, and when the last one finishes we can synchronize the results and so on. This approach might considerably reduce the idles, but it will not completely eliminate them as we will see later in 6.3.

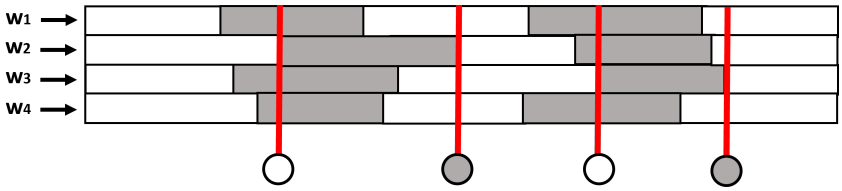

Figure 4: Workers' activity when pipelining. Each line shows the activity of a worker: white and gray rectangles correspond each to the processing of one stRequest(). The rectangles of a same step have the same color. Vertical red lines indicate synchronization moments, the disks below them are of the same color of the step just finished.

To schematize, the operations done in Algorithm 2 are successions of two blocks (in the case $I = 1$), a gradient descent block that we will denote Dsc_t, that takes as argument the parameters vector θ_{t-1} and returns k gradients $\nabla F_{1,t}(\theta_{t-1}), \dots, \nabla F_{k,t}(\theta_{t-1})$, and synchronization block Sync_t that takes as argument these gradient, updates the vector $\theta_t = \theta_{t-1} - \lambda/k \sum_{1\leq i\leq k} \nabla F_{i,t}(\theta_{t-1})$ and then passes it to the next descent block Dsc_{t+1}. This functioning is shown in Figure 5.

What we want is to allow the next Dsc block to start before the synchronization is done, i.e we want Dsc_{t+1} and Sync_t to be independent of each other. That will allow running them simultaneously as in Figure 6: θ_{-1} is given as input for Dsc_1, and the workers who finish computing their assigned gradient can

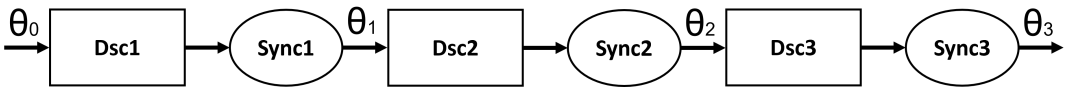

Figure 5: Succession of blocks defining Algorithm 2

take θ_0 as new input and start computing the next gradient using the next observations, corresponding to Dsc_2. When all the workers finish processing the first observations, i.e end of Dsc_1, we compute the average of the gradients and the new vector θ_1 (Sync_1) that is given as input for Dsc_3, etc

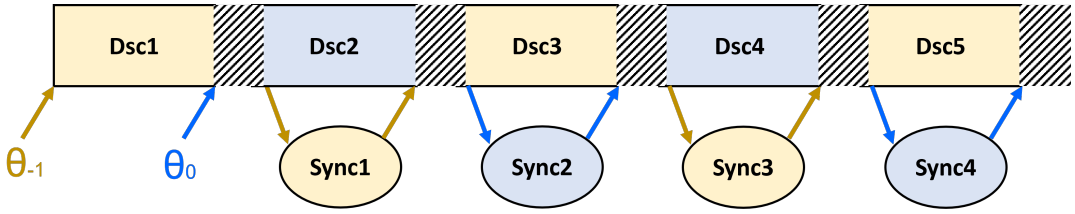

Figure 6: The blocks chaining we want to have. The hatched areas indicate possible overlaps between two successive Dsc blocks (as in Figure 4).

This scheme is mathematically translated by: $\theta_{-1}, \theta_0 \in \mathbb{R}^d$ and for each $t \geq 1$

$$\begin{cases} \theta_t^i & = \theta_{t-2} - \lambda \nabla F_{i,t}(\theta_{t-2}) \qquad \forall 1 \leq i \leq k, \\ \theta_t & = \frac{1}{k} \sum_{i=1}^k \theta_t^i, \end{cases}$$

with the convention $F_{i,t} = 0$ for $t \leq 0$.
The problem is that this implementation gives two independent gradient descents, one for even integers t and one for odd ones, and by the end of the algorithm averaging the two obtained vectors would not make sense. The final result is then no better than doing a stochastic gradient descent using only half the available data.

To force dependency between these two gradient descents, we use at step t the gradients from both steps $t-2$ and $t-3$:

$$\begin{cases} \theta_t^i & = \theta_{t-2} - \lambda \nabla F_{i,t}(\theta_{t-2}) - \lambda \nabla F_{i,t-1}(\theta_{t-3}) \qquad \forall 1 \leq i \leq k, \\ \theta_t & = \frac{1}{k} \sum_{i=1}^k \theta_t^i. \end{cases} \tag{7}$$

The logic of the implementation will therefore be as in Figure 7. We call this algorithm PR-SGD-PIP.

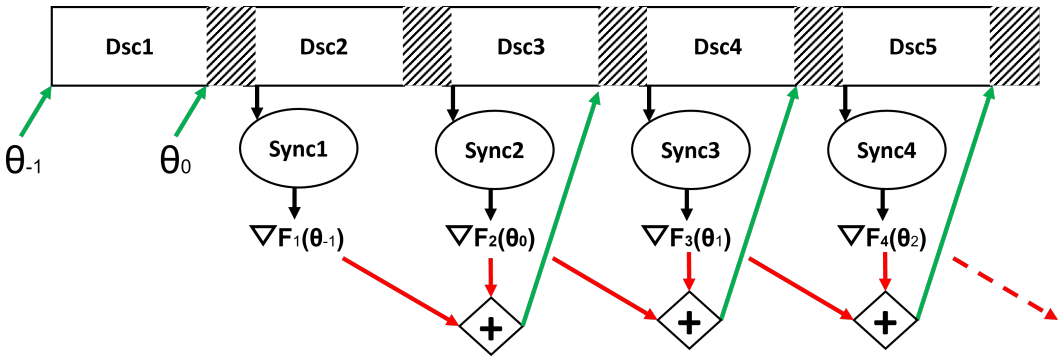

Figure 7: Implementation of Equation 7

6.2 Convergence Analysis

In the following, we will explain why the sequence $(\theta_t)_t$ defined by Equation 7 is a satisfying solution to our problem. We show in Lemma 15 that $(\theta_t)_t$ satisfies a simple induction formula between θ_{t-1} and θ_t, and then we will prove its convergence.

Lemma 15. *If $\theta_{-1} = \theta_0$ and θ_t is defined as in Equation 7 for any $t \geq 1$ then we have*

$$\theta_t = \theta_{t-1} - \frac{\lambda}{k} \sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-2})$$

with the convention $F_{i,t} = 0$ for all $1 \leq i \leq k$ and $t \leq 0$.

Proof. We can prove this by induction for $t \geq 0$ considering that we have a vector $\theta_{-2} \in \mathbb{R}^d$.
For $t = 0$, it is true because $\theta_0 = \theta_{-1}$ and $F_{i,0} = 0$. Let $t \geq 1$, and assume that the Lemma's statement is true for $t - 1$. We have

$$\begin{aligned}
\theta_t &= \frac{1}{k} \sum_{i=1}^{k} (\theta_{t-2} - \lambda \nabla F_{i,t}(\theta_{t-2}) - \lambda \nabla F_{i,t-1}(\theta_{t-3})) \\
&= \left(\theta_{t-2} - \frac{\lambda}{k} \sum_{i=1}^{k} \nabla F_{i,t-1}(\theta_{t-3}) \right) - \frac{\lambda}{k} \sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-2}) \\
&= \theta_{t-1} - \frac{\lambda}{k} \sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-2}).
\end{aligned}$$

The two first inequalities are true by definition of $(\theta_t^i)_i, \theta_t$, the last one is by the induction hypothesis. This concludes the proof. □

This Lemma shows that, from a theoretical point of view, PR-SGD-PIP is equivalent to PR-SGD using delayed gradients. In the case of one processor $k = 1$, the equation in Lemma 15 becomes

$$\theta_t = \theta_{t-1} - \lambda \nabla F_t(\theta_{t-2}).$$

The algorithm defined by this induction is known as SGD with delayed updates, and it has been studied in the general case of a delay $h \geq 1$ in [9], [10] and [11]. The distributed version also have been analyzed in [12] and [13]. Note that for such algorithms the smoothness of the objective function is a crucial assumption because we need to guarantee that the delayed gradient $\nabla F_t(\theta_{t-2})$ is not far from the actual gradient $\nabla F_t(\theta_{t-1})$, which will guarantee that the descent is almost done in the right direction.

Theorem 16. *If F is a differentiable function satisfying the assumptions A1, A3 and A2, then for any $T \geq k^3$, with $\lambda = \frac{\sqrt{k}}{L\sqrt{T}}$, we have that the sequence defined by $\theta_0 \in \mathbb{R}^d$ and Equation 7 verifies*

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] \leq (2L(F(\theta_0) - F(\theta^\star)) + 2\sigma^2 + G^2) \frac{1}{\sqrt{kT}}.$$

Using a delayed gradient slows down the convergence by only an additional term $(\sigma^2 + G^2)\frac{1}{\sqrt{kT}}$, the quality of the convergence is therefore almost the same.
The proof of this Theorem uses the same main arguments as the proof of Theorem 9 (and of Theorem 1 in [1]), but it needs to be adapted because we use delayed gradients.

Proof. let $t \in \{1, \ldots, T\}$, using the L-smoothness of F we have

$$\mathbb{E}[F(\theta_t)] - \mathbb{E}[F(\theta_{t-1})] \leq \mathbb{E}[\langle \nabla F(\theta_{t-1}), \theta_t - \theta_{t-1} \rangle] + \frac{L}{2} \mathbb{E}[\|\theta_t - \theta_{t-1}\|^2]. \tag{8}$$

Using Lemma 15, we upper bound the term $\mathbb{E}[\|\theta_t - \theta_{t-1}\|^2]$ as in Theorem 9: we use the law of total expectations to write $\mathbb{E}[\cdot] = \mathbb{E}[\mathbb{E}[\cdot \mid \theta_{t-2}]]$ and then we use Lemmas 30 then 31

$$\begin{aligned}
\mathbb{E}[\|\theta_t - \theta_{t-1}\|^2] &= \lambda^2 \mathbb{E}[\|\frac{1}{k}\sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-2})\|^2] \\
&= \lambda^2 \mathbb{E}[\mathbb{E}[\|\frac{1}{k}\sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-2})\|^2 \mid \theta_{t-2}]] \\
&= \lambda^2 \mathbb{E}\left[\mathbb{E}[\|\frac{1}{k}\sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-2}) - \nabla F(\theta_{t-2})\|^2 \mid \theta_{t-2}] + \mathbb{E}[\|\nabla F(\theta_{t-2})\|^2 \mid \theta_{t-2}]\right] \\
&= \lambda^2 \mathbb{E}\left[\frac{1}{k^2}\sum_{i=1}^{k} \mathbb{E}[\|\nabla F_{i,t}(\theta_{t-2}) - \nabla F(\theta_{t-2})\|^2 \mid \theta_{t-2}] + \mathbb{E}[\|\nabla F(\theta_{t-2})\|^2 \mid \theta_{t-2}]\right] \\
&= \frac{\lambda^2}{k^2}\sum_{i=1}^{k} \underbrace{\mathbb{E}[\|\nabla F_{i,t}(\theta_{t-2}) - \nabla F(\theta_{t-2})\|^2]}_{\leq \sigma^2 \text{ by A2}} + \lambda^2 \mathbb{E}[\|\nabla F(\theta_{t-2})\|^2],
\end{aligned}$$

thus we obtain

$$\mathbb{E}[\|\theta_t - \theta_{t-1}\|^2] \leq \frac{\lambda^2 \sigma^2}{k} + \lambda^2 \mathbb{E}[\|\nabla F(\theta_{t-2})\|^2]. \tag{9}$$

By independence between $(\theta_{t-1}, \theta_{t-2})$ and the observations a_m for $m > (t-1)kI$ we have

$$\begin{aligned}
\mathbb{E}[\langle \nabla F(\theta_{t-1}), \theta_t - \theta_{t-1}\rangle] &= -\frac{\lambda}{k}\sum_{i=1}^{k} \mathbb{E}[\langle \nabla F(\theta_{t-1}), \nabla F_{i,t}(\theta_{t-2})\rangle] \\
&= -\frac{\lambda}{k}\sum_{i=1}^{k} \mathbb{E}[\langle \nabla F(\theta_{t-1}), \mathbb{E}[\nabla F_{i,t}(\theta_{t-2}) \mid \theta_{t-1}, \theta_{t-2}]\rangle] \\
&= -\frac{\lambda}{k}\sum_{i=1}^{k} \mathbb{E}[\langle \nabla F(\theta_{t-1}), \nabla F(\theta_{t-2})\rangle] \\
&= -\lambda \mathbb{E}[\langle \nabla F(\theta_{t-1}), \nabla F(\theta_{t-2})\rangle] \\
&= -\frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] - \frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-2})\|^2] + \frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1}) - \nabla F(\theta_{t-2})\|^2].
\end{aligned}$$

The last equality yields from the identity $\langle x, y\rangle = \frac{1}{2}(\|x\|^2 + \|y\|^2 - \|x-y\|^2)$. The last term in the equation above can be upper bounded using the L-smoothness and Inequality 9, in fact, if we apply Inequality 9 to $t-1$ instead of t and using Assumption A3 (bounded second moment) we get

$$\begin{aligned}
\mathbb{E}[\|\nabla F(\theta_{t-1}) - \nabla F(\theta_{t-2})\|^2] &\leq L^2 \mathbb{E}[\|\theta_{t-1} - \theta_{t-2}\|^2] \\
&\leq \frac{\lambda^2 \sigma^2 L^2}{k} + \lambda^2 L^2 \mathbb{E}[\|\nabla F(\theta_{t-3})\|^2] \\
&\leq \lambda^2 L^2(\sigma^2/k + G^2) \\
&\leq \lambda^2 L^2(\sigma^2 + G^2),
\end{aligned}$$

Which leads to the upper bound

$$\mathbb{E}[\langle \nabla F(\theta_{t-1}), \theta_t - \theta_{t-1}\rangle] \leq -\frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] - \frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-2})\|^2] + \frac{\lambda^3 L^2}{2}(\sigma^2 + G^2). \tag{10}$$

Finally, substituting 9 and 10 into 8, and with the inequality $\lambda = \frac{\sqrt{k}}{L\sqrt{T}} < \frac{1}{L}$ we have

$$\begin{aligned}
\mathbb{E}[F(\theta_t)] - \mathbb{E}[F(\theta_{t-1})] &\leq -\frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] - \frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-2})\|^2] + \lambda^2 L^2(\sigma^2 + G^2) \\
&\quad + \frac{\lambda^2\sigma^2 L}{2k} + \frac{\lambda^2 L}{2}\mathbb{E}[\|\nabla F(\theta_{t-2})\|^2] \\
&\leq -\frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] + \frac{\lambda^2\sigma^2 L}{2k} + \lambda^2 L^2(\sigma^2 + G^2) \\
&\quad + \frac{\lambda L}{2}\underbrace{(\lambda - 1/L)}_{<0}\mathbb{E}[\|\nabla F(\theta_{t-2})\|^2] \\
&\leq -\frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] + \frac{\lambda^2\sigma^2 L}{2k} + \frac{\lambda^3 L^2}{2}(\sigma^2 + G^2),
\end{aligned}$$

summing over $t = 1, \ldots, T$ and dividing by T gives

$$\frac{1}{T}(F(\theta^\star) - F(\theta_0)) \leq \frac{1}{T}(\mathbb{E}[F(\theta_T)] - F(\theta_0)) \leq -\frac{\lambda}{2T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] + \frac{\lambda^2\sigma^2 L}{2k} + \frac{\lambda^3 L^2}{2}(\sigma^2 + G^2)$$

now we rearrange the terms and replace λ by its value

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] &\leq \frac{2}{\lambda T}(F(\theta_0) - F(\theta^\star)) + \frac{\lambda\sigma^2 L}{k} + \lambda^2 L^2(\sigma^2 + G^2) \\
&= \frac{2L}{\sqrt{kT}}(F(\theta_0) - F(\theta^\star)) + \frac{\sigma^2}{\sqrt{kT}} + (\sigma^2 + G^2)\frac{k}{T} \\
&\leq (2L(F(\theta_0) - F(\theta^\star)) + 2\sigma^2 + G^2)\frac{1}{\sqrt{kT}}.
\end{aligned}$$

the last inequality holds because $k^3 \leq T$ and thus $\frac{k}{T} \leq \frac{1}{\sqrt{kT}}$. □

6.3 Higher Pipeline Depth

Algorithm PR-SGD-PIP defined by Equation 7 reduces the waiting time and we proved in Theorem 16 that it has a good convergence rate. The problem however is that it requires to finish computing θ_{t-2} before step t starts, but with random execution times it possible that a worker becomes at some point two steps behind the others, which will again generate idles as shown in Figure 8

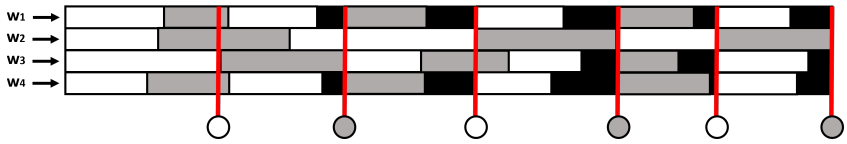

Figure 8: Workers' activity when pipelining. Each line shows the activity of a worker: white and gray rectangles correspond each to the processing of one stRequest(). The rectangles of a same step have the same color. Vertical red lines indicate synchronization moments, the disks below them are of the same color of the step just finished.

To solve this problem, we push our solution even deeper by pipelining at a depth h, i.e we allow h successive steps to overlap. In order to do that, at step t we can use only information available up to step $t - h$. Inspired by the case $h = 2$ seen previously, we propose the following algorithm: $\theta_{-h+1} = \ldots = \theta_0$ and for $t \geq 1$

$$\left\{\begin{array}{ll} \theta_t^i & = \theta_{t-h} - \lambda\sum_{u=0}^{h-1}\nabla F_{i,t-u}(\theta_{t-h-u}), \\ \theta_t & = \frac{1}{k}\sum_{i=1}^{k}\theta_t^i. \end{array}\right. \tag{11}$$

Let us estimate the convergence rate of this algorithm. Fist we show an induction formula equivalent to the one in Lemma 15, showing that Equation 11 is equivalent to having a stochastic gradient descent with a gradient delayed by $h-1$ steps.

Lemma 17. *For any $t \geq 1$ we have*

$$\theta_t = \theta_{t-1} - \frac{\lambda}{k} \sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-h})$$

with the convention $F_{i,t} = 0$ for all $1 \leq i \leq k$ and $t \leq 0$.

Proof. To lighten the notations, we define $g_t := \frac{1}{k} \sum_{i=1}^{k} \nabla F_{i,t}(\theta_{t-h})$ for all $t \geq 0$. We immediately have from Equation 11 that

$$\theta_t = \theta_{t-h} - \lambda \sum_{u=0}^{h-1} g_{t-u},$$

and we want to prove that for any $t \geq 1$ we have $\theta_t = \theta_{t-1} - \lambda g_t$. We will prove the result for $t \geq -h$, assuming that we have vectors $\theta_s = \theta_0$ for example for all $s < 0$.
For any $t \leq 0$ it is trivially true because $\theta_t = \theta_{t-1} = \theta_0$ and $g_t = 0$. Let $t \geq 1$ and assume that the result is true for any $s \in \{-h, \dots, t-1\}$. We have by definition that

$$\theta_t = \theta_{t-h} - \lambda \sum_{u=0}^{h-1} g_{t-u},$$
$$\theta_{t-1} = \theta_{t-h-1} - \lambda \sum_{u=0}^{h-1} g_{t-u-1},$$

the difference of the two equations yields

$$\begin{aligned}
\theta_t - \theta_{t-1} &= \theta_{t-h} - \theta_{t-h-1} - \lambda \sum_{u=0}^{h-1} (g_{t-u} - g_{t-u-1}) \\
&= \theta_{t-h} - \theta_{t-h-1} - \lambda (g_t - g_{t-h}) \\
&= (\theta_{t-h} - \theta_{t-h-1} + \lambda g_{t-h}) - \lambda g_t \\
&= -\lambda g_t,
\end{aligned}$$

in the last line we used the induction hypothesis for $s = t-h$: $\theta_{t-h} = \theta_{t-h-1} - \lambda g_{t-h}$. □

Theorem 18. *If F is a differentiable function satisfying the assumptions A1, A3 and A2, then for $\lambda = \frac{\sqrt{k}}{L\sqrt{T}}$ the sequence of $\mathbb{R}^d$ defined by $\theta_0 \in \mathbb{R}^d$ and Equation 11 verifies for any $T \geq k^3$*

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] \leq \left(2L(F(\theta_0) - F(\theta^\star)) + \sigma^2 + (h-1)^2(\sigma^2 + G^2)\right) \frac{1}{\sqrt{kT}}.$$

Proof. Using the same proof as in Theorem 16 we have the upper bound

$$\mathbb{E}[\|\theta_t - \theta_{t-1}\|^2] \leq \frac{\lambda^2 \sigma^2}{k} + \lambda^2 \mathbb{E}[\|\nabla F(\theta_{t-h})\|^2], \tag{12}$$

from which we have by Assumption A3

$$\mathbb{E}[\|\theta_t - \theta_{t-1}\|^2] \leq \frac{\lambda^2 \sigma^2}{k} + \lambda^2 G \leq \lambda^2(\sigma^2 + G^2), \tag{13}$$

and the identity

$$\mathbb{E}[\langle \nabla F(\theta_{t-1}), \theta_t - \theta_{t-1} \rangle] = -\frac{\lambda}{2} \mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] - \frac{\lambda}{2} \mathbb{E}[\|\nabla F(\theta_{t-h})\|^2] + \frac{\lambda}{2} \mathbb{E}[\|\nabla F(\theta_{t-1}) - \nabla F(\theta_{t-h})\|^2].$$

and we have respectively by L-smoothness of F, the triangle inequality, the Cauchy-Schwarz inequality and Inequality 13 that

$$\begin{aligned}\mathbb{E}[\|\nabla F(\theta_{t-1}) - \nabla F(\theta_{t-h})\|^2] &\leq L^2\mathbb{E}[\|\theta_{t-1} - \theta_{t-h}\|^2] \\ &\leq L^2\mathbb{E}\left[\left(\sum_{u=1}^{h-1}\|\theta_{t-u} - \theta_{t-u-1}\|\right)^2\right] \\ &\leq L^2(h-1)\sum_{u=1}^{h-1}\mathbb{E}[\|\theta_{t-u} - \theta_{t-u-1}\|^2] \\ &\leq \lambda^2L^2(h-1)^2(\sigma^2 + G^2),\end{aligned}$$

which gives

$$\mathbb{E}[\langle\nabla F(\theta_{t-1}), \theta_t - \theta_{t-1}\rangle] = -\frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] - \frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-h})\|^2] + \frac{\lambda^3L^2}{2}(h-1)^2(\sigma^2 + G^2). \quad (14)$$

Finally, given that $\lambda < 1/L$ we deduce as in Theorem 16 that

$$\begin{aligned}\mathbb{E}[F(\theta_t)] - \mathbb{E}[F(\theta_{t-1})] &\leq \mathbb{E}[\langle\nabla F(\theta_{t-1}), \theta_t - \theta_{t-1}\rangle] + \frac{L}{2}\mathbb{E}[\|\theta_t - \theta_{t-1}\|^2] \\ &\leq -\frac{\lambda}{2}\mathbb{E}[\|\nabla F(\theta_{t-1})\|^2] + \frac{\lambda^2\sigma^2L}{2k} + \frac{\lambda^3L^2}{2}(h-1)^2(\sigma^2 + G^2),\end{aligned}$$

after summing, rearranging the terms and replacing λ by its value we obtain the statement of the theorem. □

Remark 19. *This theorem can be seen as a generalization of Theorems 9 ($h = 1$) and 16 ($h = 2$). It shows that having a pipeline of depth h leads to a convergence rate $O((h-1)^2)$ times slower.*

6.4 In Practice

The case $I > 1$ We only treated the case $I = 1$. But this algorithm can be generalized to the general case as follows: at each step t, each worker executes I steps of SgdStep using the observations $a_{((t-1)k+(i-1))I+1}, \ldots, a_{((t-1)k+i)I}$ bringing its parameters vector from an initial value θ to a final value $g_{i,t,I}(\theta)$. If we denote $\Delta_{i,t,I} : \theta \in \mathbb{R}^d \to g_{i,t,I}(\theta) - \theta$, then we can generalize PR-SGD-PIP at depth h to $I \geq 1$ as: $\theta_{-h+1} = \ldots = \theta_0 \in \mathbb{R}^d$ and for $t \geq 1$

$$\begin{cases} \theta_t^i &= \theta_{t-h} - \lambda\sum_{u=0}^{h-1}\nabla\Delta_{i,t-u,I}(\theta_{t-h-u}), \\ \theta_t &= \frac{1}{k}\sum_{i=1}^{k}\theta_t^i. \end{cases} \quad (15)$$

Remark 20. *Note that for $I = 1$ we have $\Delta_{i,t,1} = \nabla F_{i,t}$.*

Although we do not provide any theoretical guarantees on this generalization, we will give an example how it behaves at depth $h = 2$ for $I > 1$ in the experiments section 8.

Limitations Even at a depth h, we may eventually encounter the same problem described in Figure 8 if a worker had some successive slow executions. the bigger h is, the more chances we have not to face waiting times, but this has on the other hand an important impact on the convergence rate.
The algorithm is also costly in terms of memory: to use PR-SGD-PIP at at a depth h, it is necessary to keep in memory the last h parameter vectors at each step.

Execution time It is difficult to estimate the execution time of this algorithm, but we can give a simple lower bound. In order to process N observations, we need to load the N/P pages containing them, hence we have

$$\mathbb{E}[\mathcal{T}(\text{PR-SGD-PIP}; N)] \geq \frac{\mu N}{P},$$

and thus

$$\text{TR(PR-SGD-PIP)} := \lim_{N\to\infty} \frac{\mathbb{E}[\mathcal{T}(\text{SGD}; N)]}{k\mathbb{E}[\mathcal{T}(\text{PR-SGD-PIP}, N)]} \leq \left(\frac{(\alpha+\beta)/(2I) + \mu/P}{\mu/P}\right)\frac{1}{k}. \tag{16}$$

The execution time ratio is decreasing as $1/k$, thus using more than one thread is not profitable. The previous upper bound also holds for PR-SGD. Unless μ is small, this bound is very limiting, and we need to overcome it.

7 Data Loading and Processing Speed

7.1 Observations and Better Understanding

To measure how computationally expensive page loading can be, we tested the PR-SGD and PR-SGD-PIP algorithms with 3 workers on a large, very high-dimensional dataset, and we visualized the workers' activities in Figure 9.

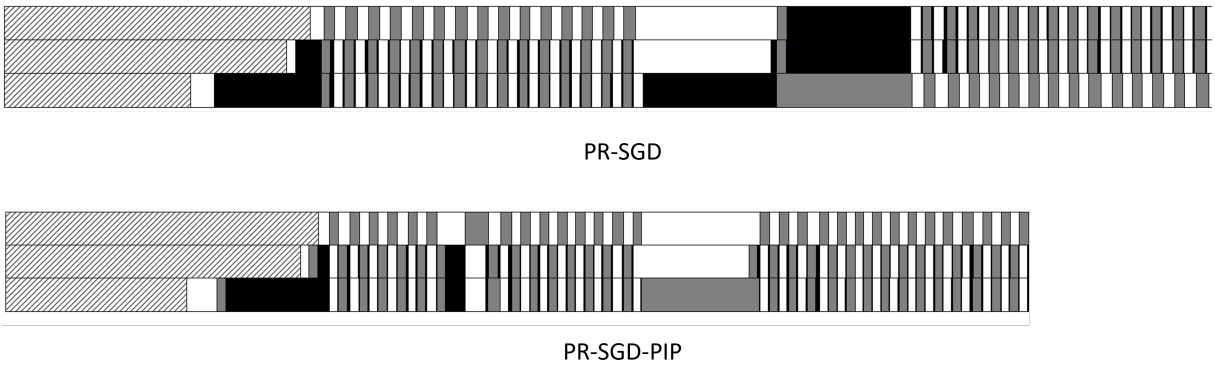

Figure 9: Activity of the workers during time for PR-SGD and PR-SGD-PIP, with $k = 3, I = 100$ and $N = 2P$ (two pages processed). Idles are colored in black, while white and gray rectangles show each the execution time of a request. The hashed rectangles show preprocessing times spent by the workers before starting their first requests.

We notice first that there is a large preprocessing time at the beginning. This is the time needed to initialize the vector of parameters, load the first page of data, ... This preprocessing time seems to be random and generates important waiting times at the beginning, but this is not a problem since it is constant and can be ignored when N is sufficiently large.
The second interesting remark is that we have some tasks taking a much longer time than the others. They are those where workers load a new page of data. As explained in the proof of Lemma 10, the delay generated by the loading of a page is a random variable with values in $[\mu, 2\mu]$ where μ is the time needed to a single page by one worker: if all the workers load the page in the same step then we have a delay of μ, otherwise they would all load it on two consecutive steps and we have therefore a delay at least equal to $2\mu - 2(\beta - \alpha)$. Data loading appears on the figure to be the main cause of the idles.
PR-SGD-PIP behaves as we want and eliminates a part of the idles by allowing the workers to work on two different steps, and clearly accelerates the execution compared to PR-SGD but we still have he performance limitation shown in Inequality 16.

7.2 One Worker Loading Data

The Algorithm To overcome the limitation of page loading, we propose another variant of PR-SGD that we call PR-SGD-OLD (PR-SGD- One Loading Data), where the workers read data from a common memory, and only one of them will be in charge of loading the data while the others will process it.
More in detail, initially the worker w_k for example loads the first page, and then when it finishes the other workers $w_1, \dots, w_{k-1}$ start processing it, w_k will at this moment start loading the second page, so that when the other workers finish processing the first one, they can directly move on to the the second, and the memory storing the first page can be freed for w_k to start loading the third page and so on.
Since the operation NextObs does the page loading when necessary, the algorithm can be implemented simply by guaranteeing a lead of P observations to the worker w_k over the others throughout the all execution.

All this is explained in Algorithm 3, and for a more visual illustration of how it works we refer to Figure 10. We use in Algorithm 3 an asynchronous function Skip(w_i, ℓ) that takes as input a worker w_i and a positive integer ℓ and performs ℓ times the operation NextObs(w_i).

Algorithm 3: PR-SGD-OLD($\theta_0, T, I, \lambda, P$): Parallel Restarted SGD - One Loading Data

Input : Initial parameters vector θ_0, a positive integer T, a learning rate λ, a synchronization interval I and the number of observations per page P
Output: a parameters vector θ_T

w_k loads the first page, wait until the loading is finished;
$q \leftarrow 0$;
for *t=1, . . . , T* **do**
  for [Parallel] $i = 1, \dots, k-1$ **do**
    stRequest($w_i, \theta_{t-1}, I, \lambda$);
  end for
  Wait for the workers $w_1, \dots, w_{k-1}$ to finish executing their requests;
  $\theta_t \leftarrow \frac{1}{k-1} \sum_{i=1}^{k-1} \theta_t^i$;
  $q \leftarrow q + (k-1)I$ [mod P];
  if $(k-1)I \leq q < 2(k-1)I$ **then**
    Wait for w_k to finish the last Skip request if it is not the case yet;
    Skip(w_k, P); // Do not wait for this operation to finish
  end if
end for
return θ_T

The variable q counts the number of observations processed in the current page m. Initially $q = 0$ and after each iteration, the workers $w_1, \dots, w_{k-1}$ process each I observations, hence q is incremented to $q + (k-1)I$. When q becomes larger than P this means that we have switched to the next page, and q is replaced by $q - P$ (the "[mod P]" operation).
The inequality $(k-1)I \leq q < 2(k-1)I$ happens at the first or second step of the processing of each page, and it ensures that the workers $w_1, \dots, w_{k-1}$ have all switched to the page $m+1$, w_k can thus erase the page m from the memory and start loading the page $m+2$, and this is done by having w_k skip P observations: it goes to the first observation of the next page and must therefore load it. This algorithm requires having at least $2(k-1)I$ observations per page.

Strenghts and Limitations For our testing example, PR-SGD-OLD compares to PR-SGD and PR-SGD-PIP as shown in Figure 9.

We can do many interesting observations. First the global initial preprocessing time becomes much shorter: having a shared memory and having only one worker loading the first page of data shortcuts

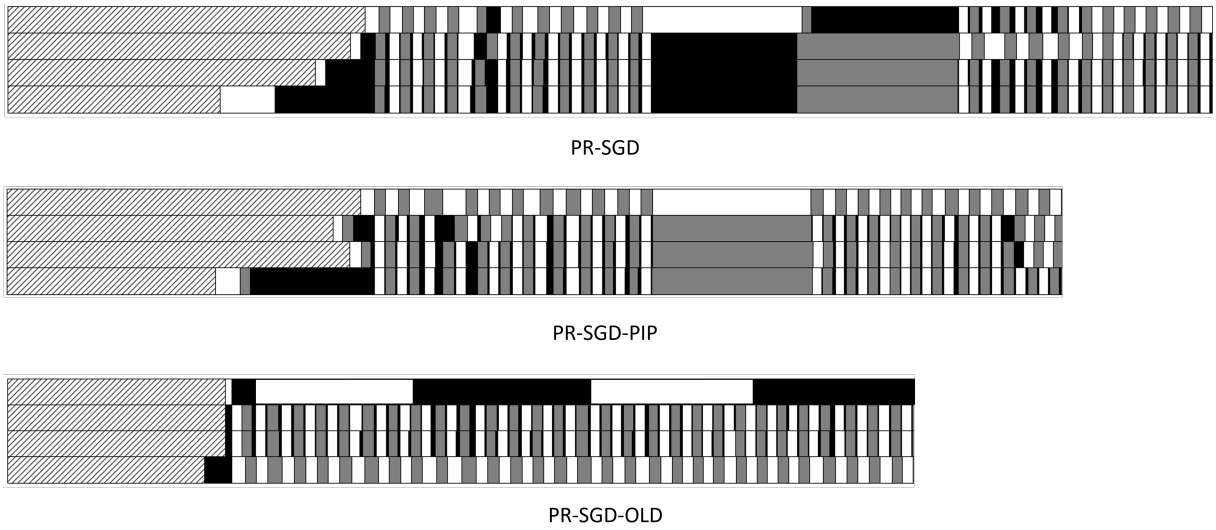

Figure 10: Activity of the workers during time for PR-SGD, PR-SGD-PIP and PR-SGD-OLD, with $k = 4, I = 100$ and $N = 2P$. Idles are colored in black, while white and gray rectangles show each the execution time of a request. The hashed rectangles show preprocessing times spent by the workers before starting their first requests.

many unnecessary operations. Secondly, even though the observations are treated only by 3 workers in the case of PR-SGD-OLD instead of 4 for PR-SGD, having the pages loading done independently makes the whole process faster.

PR-SGD-OLD allows indeed to cancel the idles due to the pages loading for the workers $w_1, \dots, w_{k-1}$, but we have consequently an important waiting time for w_k.
It is difficult to assign other tasks to w_k because we need to respect the reproducibility constraint. However, we can minimize its waiting time by choosing an optimal number of workers $k^\star$ such that the time needed for loading a page is in expectation the same time needed for processing it.
$k < k^\star$, w_k will lead to w_k staying inactive while waiting for the others to finish processing the page, and $k > k^\star$ will result in having the time for loading a page being larger than the time for processing it, thus the workers $w_1, \dots, w_{k-1}$ will have to wait for w_k to finish the loading before continuing.

Convergence Rate The convergence rate with Algorithm 3 is given by Theorem 9 but replacing k by $k-1$. PR-SGD has therefore a slightly better convergence rate but we hope that PR-SGD-OLD will compensate this with its faster execution time and have a better performance overall .

7.3 Execution Time

Let $T = N/((k-1)I)$ be the number of iterations needed by PR-SGD-OLD to process N observations, and let $M = N/P$ be the number of pages loaded during these iterations.
For $m \in \{1, \dots, M\}$, we define Z_m the time needed for processing the observations of page m by $k-1$ workers. Z_m is a random variable and its expectation is given by

$$\begin{aligned}
\mathbb{E}[Z_m] &= \mathbb{E} \sum_{t=(m-1)P/((k-1)I)+1}^{mP/((k-1)I)} \left(\max_{1 \le i \le k-1} \{\tau_{i,t}\} + \gamma(k-1) \right) \\
&= \left(\mathbb{E}\left[\max_{1 \le i \le k-1} \{\tau_{i,1}\} \right] + \gamma(k-1) \right) \frac{P}{(k-1)I} \\
&= \left(\frac{\alpha + (k-1)\beta}{k} + \gamma(k-1) \right) \frac{P}{(k-1)I}.
\end{aligned}$$

The last equality is due to Lemma 32.
Note that γ is only multiplied by $k-1$ and not by k because we will only need to average the models of $k-1$ workers at each synchronization.
The total runtime of the algorithm is nevertheless not $\sum_{m=1}^{M} Z_m$ because before processing each page, w_k should have finish loading it for $m < M$:

$$\mathcal{T}(\text{PR-SGD-OLD}; N) = \sum_{m=1}^{M-1} \max\{Z_m, \mu\} + Z_M.$$

The real execution time contains an additional time μ for loading the first page, but it is not taken into account here because it is part of the preprocessing also made by PR-SGD and PR-SGD-PIP. We will now only study the expectation of this runtime in some particular interesting cases.

Proposition 21. *If $\mu > \gamma P/I$ then we have*

$$\mathbb{E}[\mathcal{T}(\textit{PR-SGD-OLD}; N)] = \begin{cases} \left(\frac{\alpha+(k-1)\beta}{k} + \gamma(k-1)\right) T & \text{if } k \leq k_1, \\ (k-1)\frac{IT}{P}\mu - (\mu - \mathbb{E}[Z_1]) & \text{if } k \geq k_2, \end{cases}$$

with $k_1 := 1 + \frac{\alpha/I}{\mu/P}$ and $k_2 := 1 + \frac{\beta}{\mu I/P - \gamma}$.

Proof. The condition $\mu > \gamma P/I$ only means that the time needed for loading a page is superior to the time needed for doing all the synchronizations when processing a page. This inequality is largely verified in practice (and thus not restrictive), and it is used in our proof only for the case $k \geq k_2$, in the case $k \leq k_1$ we only need to have $\mu > 0$. The proof is immediate: if $k \leq k_1$ then

$$\mu \leq \frac{\alpha N}{(k-1)MI} = \frac{\alpha P}{(k-1)I} \leq (\alpha + \gamma(k-1))\frac{P}{(k-1)I} \leq Z_m \qquad \forall 1 \leq m \leq M$$

because Z_m is the sum of $P/((k-1)I)$ terms all larger than $\alpha + \gamma(k-1)$. Therefore we have

$$\mathcal{T}(\text{PR-SGD-OLD}; N) = \sum_{m=1}^{M-1} \max\{Z_m, \mu\} + Z_M. = \sum_{m=1}^{M} Z_m.$$

We deduce that $\mathbb{E}[\mathcal{T}(\text{PR-SGD-OLD}; N)] = M\mathbb{E}[Z_1]$ which we computed in the beginning of the Section. Given that $N = (k-1)IT = MP$ we immediately have the result.
On the other hand, if $k \geq k_2 = 1 + \frac{\beta P/I}{\mu - \gamma P/I}$ then similarly

$$\mu \geq \frac{\beta P}{(k-1)I} + \frac{\gamma P}{I} \geq (\beta + \gamma(k-1))\frac{P}{(k-1)I} \geq Z_m, \qquad \forall 1 \leq m \leq M$$

thus

$$\mathcal{T}(\text{PR-SGD-OLD}; N) = \sum_{m=1}^{M-1} \max\{Z_m, \mu\} + Z_M. = (M-1)\mu + Z_M = M\mu - (\mu - Z_M).$$

and we having the result by taking the expectation. □

Note that in the case $k \geq k_2$ we can also write

$$\mathbb{E}[\mathcal{T}(\text{PR-SGD-OLD}; N)] = \left(\frac{N}{P} - 1\right)\mu + \mathbb{E}[Z_1],$$

which means that for N fixed, the runtime only depends on $\mathbb{E}[Z_1]$ that is a decreasing function of k, and the execution time is lower bounded by $(N/P - 1)\mu$ that is the time for loading the data. Figure 11 gives a visualization of the execution when increasing the number of workers from 1 to 12.

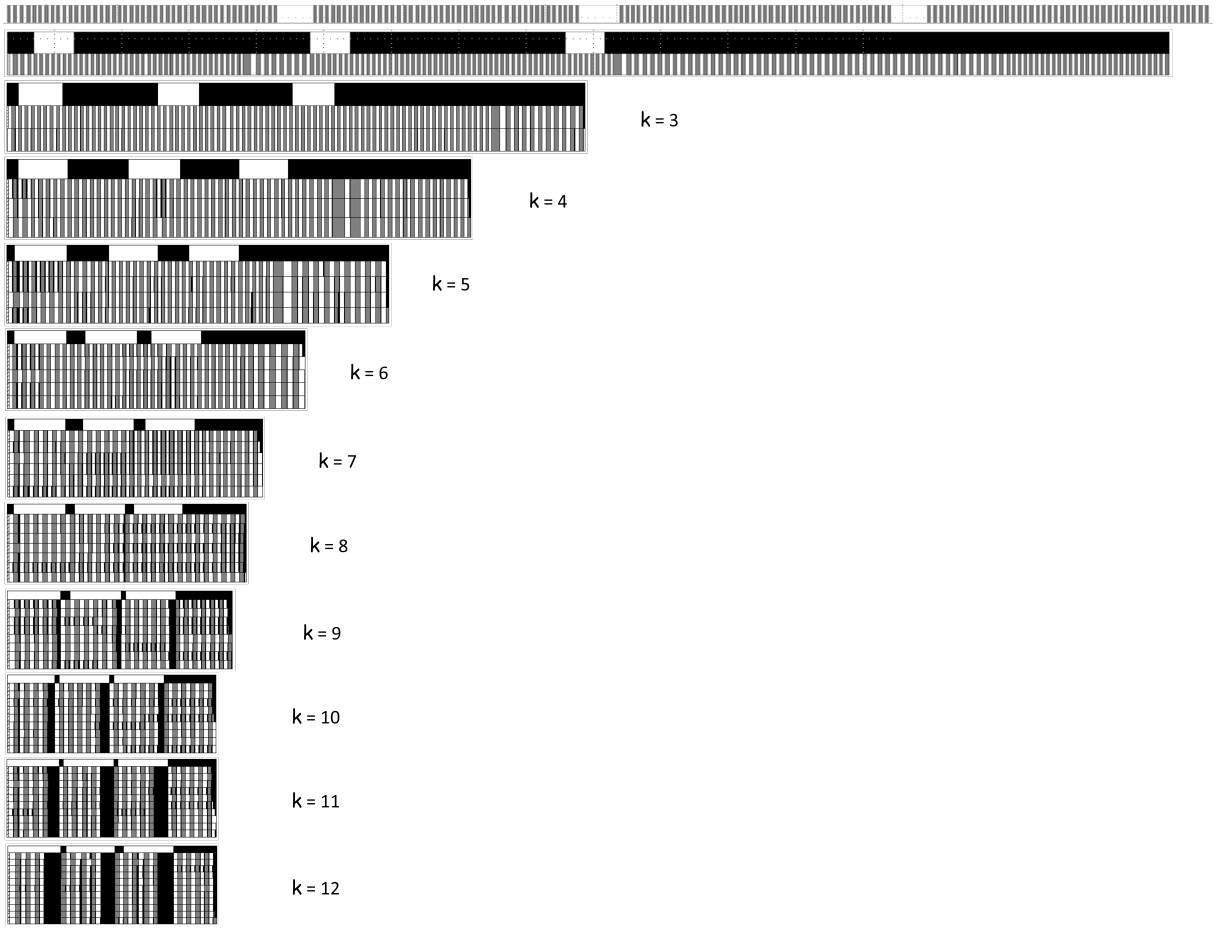

Figure 11: Activity of the workers during time for PR-SGD-OLD, with $N = 4P$, $k \in \{1, \ldots, 12\}$ and $I = 100$. Idles are colored in black, while white and gray rectangles show each the execution time of a request.

We can see that having two workers instead of one does not have a huge impact, as it simply removes the page loading and assigns it to the second worker. But with more workers we have a great reduction in time as long as the page loading time remains lower than the page processing time. This would mean that PR-SGD-OLD is only more performing than PR-SGD starting from a certain number of workers k_0. We show this later in Proposition 25.

For $k = 9$ we begin to have the undesirable effects of idles generated because the large number of workers makes the processing of a page faster loading the next one, the workers $w_1, \ldots, w_k$ are thus obliged to wait. This effect becomes even more pronounced as k increases further.

Let us now analyze the Execution Time Ratio (Definition 3) of PR-SGD-OLD.

Proposition 22. *If $k \leq k_1$ then we have the two expressions*

$$\begin{aligned}\frac{1}{\mathrm{TR}(\textit{PR-SGD-OLD})} &= 1 + \frac{1}{\frac{\alpha+\beta}{2} + \frac{\mu I}{P}} \left(\frac{\beta - \alpha}{2} \left(1 - \frac{2}{k} \right) + \frac{\mu I}{P} (\mathbb{E}[Z_1] - \mu) + \gamma(k-1) \right) \\ &= 1 + \frac{1}{\frac{\alpha+\beta}{2} + \frac{\mu I}{P}} \left(\frac{\beta - \alpha}{2} + \alpha \left(\frac{1}{k-1} - \frac{1}{k_1 - 1} \right) + \gamma k \right).\end{aligned}$$

Remark 23. *We give in this proposition two different expressions, the first one explains better the reasons why* TR(*PR-SGD-OLD*) *cannot reach the value 1, while the second shows the value we can achieve by choosing a suitable value of k, and this is very interesting because k is practically the only parameter in*

the previous equation that we can control.
Let us focus on the first expression, it shows us why PR-SGD-OLD cannot be k times faster than SGD:

- $\frac{\beta-\alpha}{2}\left(1-\frac{2}{k}\right)$*: translates the presence of idles before synchronizations, it concerns the workers* $w_1,\dots,w_{k-1}$*. It is due to having* $\beta\neq\alpha$ *and having more than two workers* $k\geq 3$*. If we have* $k=2$ *then one worker will load data to the memory and one will process it, this we will not have any idles before synchronizations.*

- $\frac{\mu I}{P}(\mathbb{E}[Z_1]-\mu)$*: translates the presence of a second type of idles: the ones of the worker* w_k *after it finishes loading a page. In Proposition 22 we place ourselves in the case* $k\leq k_1$ *and thus loading a page always takes less time than processing it as shown in the proof of Proposition 21,* w_k *will therefore wait for a time* $Z_{m-1}-\mu$ *after loading each page* m*,*

- $\gamma(k-1)$*: this last term is a result of the necessary synchronization between the workers. It is linear in* k *because the synchronization is an operation that we do not do in the case of non-distributed SGD: its cost increases with the number of workers. But we do not need to worry about it as in practice we have* $\gamma\approx 0$*, and the number of workers we use is relatively small (*$k\leq 256$*) because they are processors of a same machine.*

Proof. The proof is immediate and only computational. For processing N observations PR-SGD-OLD needs $T:=N/((k-1)I)$ iterations. For the first expression we have from Proposition 21, with $M=N/P$ again the number of pages loaded during a run of the algorithm

$$\begin{aligned}k\mathbb{E}[\mathcal{T}(\text{PR-SGD-OLD};N)] &= (k-1)\mathbb{E}[\mathcal{T}(\text{PR-SGD-OLD};N)]+M\mathbb{E}[Z_1]\\ &= \left(\beta-\frac{\beta-\alpha}{k}+\gamma(k-1)+\frac{I}{P}\mathbb{E}[Z_1]\right)\frac{N}{I}.\end{aligned}$$

because $\mathbb{E}[\mathcal{T}(\text{PR-SGD-OLD};N)]=M\mathbb{E}[Z_1]$ in the case of $k\leq k_1$ as shown earlier in the proof of Proposition 21. Thus we have

$$\begin{aligned}\frac{1}{\text{TR(PR-SGD-OLD)}}-1 &= \lim_{N\to\infty}\frac{k\mathbb{E}[\mathcal{T}(\text{PR-SGD-OLD};N)]}{\mathbb{E}[\mathcal{T}(\text{SGD};N)]}-1\\ &= \frac{1}{\frac{\alpha+\beta}{2}+\frac{\mu I}{P}}\left(\beta-\frac{\beta-\alpha}{k}+\gamma(k-1)+\frac{I}{P}\mathbb{E}[Z_1]-\frac{\alpha+\beta}{2}-\frac{\mu I}{P}\right)\\ &= \frac{1}{\frac{\alpha+\beta}{2}+\frac{\mu I}{P}}\left(\frac{\beta-\alpha}{2}\left(1-\frac{2}{k}\right)+\frac{I\mu}{P}(\mathbb{E}[Z_1]-\mu)+\gamma(k-1)\right).\end{aligned}$$

For the second expression we have also immediately from Proposition 21

$$k\mathbb{E}[\mathcal{T}(\text{PR-SGD-OLD};N)]=\left(\frac{\alpha+(k-1)\beta}{k}+\gamma(k-1)\right)\frac{kN}{(k-1)I}=\left(\frac{\alpha}{k-1}+\beta+\gamma k\right)\frac{N}{I},$$

and thus

$$\begin{aligned}\frac{1}{\text{TR(PR-SGD-OLD)}}-1 &= \lim_{N\to\infty}\frac{k\mathbb{E}[\mathcal{T}(\text{PR-SGD-OLD};N)]}{\mathbb{E}[\mathcal{T}(\text{SGD};N)]}-1\\ &= \frac{1}{\frac{\alpha+\beta}{2}+\frac{\mu I}{P}}\left(\frac{\alpha}{k-1}+\beta+\gamma k-\frac{\alpha+\beta}{2}-\frac{\mu I}{P}\right)\\ &= \frac{1}{\frac{\alpha+\beta}{2}+\frac{\mu I}{P}}\left(\frac{\beta-\alpha}{2}+\alpha\left(\frac{1}{k-1}-\frac{\mu I}{\alpha P}\right)+\gamma k\right)\\ &= \frac{1}{\frac{\alpha+\beta}{2}+\frac{\mu I}{P}}\left(\frac{\beta-\alpha}{2}+\alpha\left(\frac{1}{k-1}-\frac{1}{k_1-1}\right)+\gamma k\right).\end{aligned}$$

□

Strengths and Limitations Considering that $\gamma = 0$, which is almost the case in practice ($\gamma \ll \alpha$) and the number of workers is bounded, the previous proposition shows that for $k = k_1$ we can achieve an execution time ratio

$$\text{TR(PR-SGD-OLD)} = \frac{1}{1 + (\beta - \alpha)/(\alpha + \beta + 2\mu I/P)},$$

which is actually a very good result, because the only thing preventing us from reaching the ideal value 1 if the randomness of the runtime of a request ($\beta \neq \alpha$).
This might look surprising because the computations of the gradient descent are only distributed on $k - 1$ while we use k workers, and thus we might think that we do not fully exploit the computation power of all our workers, but this is due to removing the pages loading from the equation thanks to w_k. If we imagine that $\alpha = \beta$, then for $k = k_1$ the time for loading a page with w_k is exactly the time for processing the previous page with the $k - 1$ other workers, thus there will be no idles and no additional operations compared to SGD, which is not the case with PR-SGD because the operation of loading a page is duplicated by each worker.
Even for $k < k_1$, we no longer have the problem revealed for PR-SGD and PR-SGD-OLD by Inequality 16. In fact, the execution time ratio of PR-SGD-OLD depends on k as $\Theta\left(\frac{1}{1+1/k}\right)$, which means that we almost achieve linear speed up using k workers.

The limitation we have is that Proposition 22 holds only for $k \leq k_1$, i.e when the time for loading a page is smaller than the time needed for processing it by $k - 1$ workers. To guarantee this for a large range of values of k we need to have μ as small as possible, and this can be done by compressing the data before running PR-SGD-OLD.

8 Experiments and Results

We will now evaluate and compare the algorithms we described in the previous sections. We will start by theoretically comparing their ε-performances (Definition 4) in the case of $I = 1$, before moving to the simulations, that were made using private data borrowed from Lokad.

8.1 Performance Comparison for $I = 1$

We will now compare the ε-performance of the algorithms we have described before. In the convergence analyses of the algorithms we have shown upper bounds on

$$\frac{1}{T}\sum_{t=1}^{T} \|\nabla F(\theta_{t-1})\|^2,$$

For the theoretical study of the performance we will study the times necessary for these bounds to reach a level ε. This means on the one hand to consider that these bounds are tight, and on the other hand to consider that our algorithms will stop at a randomly chosen iteration in $\{1, \dots, T\}$, which would make that the level of $\|\nabla F\|^2$ reached at the end of the algorithm can be any $\|\nabla F(\theta_{t-1})\|^2$ with t chosen uniformly in $\{1, \dots, T\}$, and so in expectation it will be equal to the average of these terms.

Since in the convergence proofs we assumed that $I = 1$, we assume that it is the case in this section too to use the previous results. Let us first compute the performance of PR-SGD-OLD.

Proposition 24. *Assuming that $I = 1$, if $k \leq k_1$ then we have*

$$\begin{aligned}\frac{1}{\mathcal{P}_\varepsilon(\textit{PR-SGD-OLD})} &= 1 + \frac{1}{\frac{\alpha+\beta}{2} + \frac{\mu I}{P}}\left(\frac{\beta - \alpha}{2}\left(1 - \frac{2}{k}\right) + \frac{\mu}{P}(\mathbb{E}[Z_1] - \mu) + \gamma(k-1)\right) \\ &= 1 + \frac{1}{\frac{\alpha+\beta}{2} + \frac{\mu}{P}}\left(\frac{\beta - \alpha}{2} + \alpha\left(\frac{1}{k-1} - \frac{1}{k_1 - 1}\right) + \gamma k\right).\end{aligned}$$

Proof. The proof and the computations are exactly the same as in Proposition 22, we simply need to see from Theorems 6 and 9 that SGD reaches an error ε after A^2/ε^2 iterations while PR-SGD-OLD needs $A^2/((k-1)\varepsilon^2)$, with $A = 2L(F(\theta_0) - F(\theta^\star)) + \sigma^2$ and then using Propositions 5 and 21 we obtain the same quotients as in Proposition 22. □

Proposition 25. *Assume that $I = 1$. Let $k_0 := 1 + \sqrt{\frac{\beta P}{\mu}}$. If $\beta \leq \frac{\alpha^2 P}{\mu}$ then for any $\varepsilon > 0$ and for any k such that $k_0 \leq k \leq k_1$ we have*

$$\mathcal{P}_\varepsilon(\textit{PR-SGD-OLD}) > \mathcal{P}_\varepsilon(\textit{PR-SGD}),$$

where k_1 is defined in Proposition 21.

This proposition confirms the observations made after Figure 11, where we explain that with very few workers, PR-SGD-OLD is not as efficient as PR-SGD, typically in the case $k = 2$ because it only removes the time of data loading compared to the non-distributed SGD.

Proof. The condition $\beta \leq \frac{\alpha^2 P}{\mu}$ guarantees that $k_0 \leq k_1$. We consider $\theta_0 \in \mathbb{R}^d$ to be a constant vector, and the same given to all the algorithms. Let us denote $A := 2L(F(\theta_0) - F(\theta^\star)) + \sigma^2$, Theorem 9 shows that $\frac{1}{T}\sum_{t=1}^{T} \|\nabla F(\theta_{t-1})\|^2$ is bounded by $\frac{A}{\sqrt{kT}}$ in the case of PR-SGD and by $\frac{A}{\sqrt{k(T-1)}}$ for PR-SGD-OLD. For these bounds to reach a level ε we need respectively $T_\varepsilon = \frac{A^2}{k\varepsilon^2}$ $T_\varepsilon^{\text{OLD}} = \frac{A^2}{(k-1)\varepsilon^2}$.
For PR-SGD, Proposition 11 shows that the time needed for running T_ε iterations is

$$\begin{aligned}
\mathbb{E}[\mathcal{T}_\varepsilon(\text{PR-SGD})] &= \left(\frac{\alpha + k\beta}{k+1} + \gamma k + (1 + \mathbb{E}[\varepsilon])\frac{\mu k}{P}\right)\frac{A^2}{k\varepsilon^2} \\
&\geq \left(\frac{\alpha + k\beta}{k+1} + \gamma k + \frac{\mu k}{P}\right)\frac{A^2}{k\varepsilon^2},
\end{aligned}$$

and Proposition 21 shows that for PR-SGD-OLD with $k \leq k_1$, running $T_\varepsilon^{\text{OLD}}$ iterations requires a time

$$\begin{aligned}
\mathbb{E}[\mathcal{T}_\varepsilon(\text{PR-SGD-OLD})] &= \left(\frac{\alpha + (k-1)\beta}{k} + \gamma(k-1)\right)\frac{A^2}{(k-1)\varepsilon^2} \\
&= \left(\frac{\alpha}{k-1} + \beta + \gamma k\right)\frac{A^2}{k\varepsilon^2}.
\end{aligned}$$

Thus by definition of the ε-performance we have

$$\begin{aligned}
&\frac{1}{\mathcal{P}_\varepsilon(\text{PR-SGD})} - \frac{1}{\mathcal{P}_\varepsilon(\text{PR-SGD-OLD})} \\
&\quad= \frac{1}{\mathbb{E}[\mathcal{T}_\varepsilon(\text{SGD})]}(k\mathbb{E}[\mathcal{T}_\varepsilon(\text{PR-SGD})] - \mathbb{E}[\mathcal{T}_\varepsilon(\text{PR-SGD-OLD})]) \\
&\quad\geq \frac{A^2}{\varepsilon^2\mathbb{E}[\mathcal{T}_\varepsilon(\text{SGD})]}\left(\frac{\alpha + k\beta}{k+1} + \gamma k + \frac{\mu k}{P} - \frac{\alpha}{k-1} - \beta - \gamma k\right) \\
&\quad= \frac{A^2}{\varepsilon^2\mathbb{E}[\mathcal{T}_\varepsilon(\text{SGD})]}\left(\frac{\mu k}{P} - \frac{\alpha}{k-1} - \frac{\beta - \alpha}{k+1}\right) \\
&\quad> \frac{A^2}{\varepsilon^2\mathbb{E}[\mathcal{T}_\varepsilon(\text{SGD})]}\left(\frac{\mu(k-1)}{P} - \frac{\alpha}{k-1} - \frac{\beta - \alpha}{k-1}\right) \\
&\quad= \frac{A^2}{\varepsilon^2\mathbb{E}[\mathcal{T}_\varepsilon(\text{SGD})]} \cdot \frac{\mu}{(k-1)P}\left((k-1)^2 - \frac{\beta P}{\mu}\right) \geq 0.
\end{aligned}$$

□

Remark 26. *This Proposition shows that PR-SGD-OLD is more performing than PR-SGD if we correctly choose k. Even in the case $k \geq k_2$ we can show that PR-SGD-OLD outperforms PR-SGD under certain conditions but we are not interested in this case because clearly none of the two algorithms is efficient because the time for loading the data exceeds the time for processing it.*

Comparison to PR-SGD-PIP Concerning PR-SGD-PIP, we can easily see that in the case $\alpha = \beta$, PR-SGD and PR-SGD-PIP are equivalent, hence we have using the previous Proposition that PR-SGD-OLD is more performing than PR-SGD-PIP in that case for k well chosen

$$\mathcal{P}_\varepsilon(\text{PR-SGD-OLD})_{|\alpha=\beta} - \mathcal{P}_\varepsilon(\text{PR-SGD-PIP})_{|\alpha=\beta} > 0.$$

We can argue that $\mathcal{P}_\varepsilon(\text{PR-SGD-PIP})$ is a continuous function of (α, β), and $\mathcal{P}_\varepsilon(\text{PR-SGD-OLD})$ is clearly continuous by Proposition 22, it yields by continuity that PR-SGD-OLD has a better ε-performance than PR-SGD-PIP when α and β are sufficiently close to each other. It is most probably the case for any α, β.

We do not give a proof of this claim, but we will see experimentally that the convergence of PR-SGD-PIP is relatively bad, i.e. the gain it brings in terms of the runtime does not compensate for the delay of $O(h^2)$ it generates in the convergence rate (Theorem 18).

8.2 Choosing the Task Size

As explained in Section 5, choosing the task size I is not easy: choosing it too small will increase the synchronization cost, and choosing it too big will reduce the quality of the convergence.
We show in Figure 12 the evolution of the loss with the number of epochs and with time for SGD and for PR-SGD with $I \in \{10, 15, 50, 100\}$.

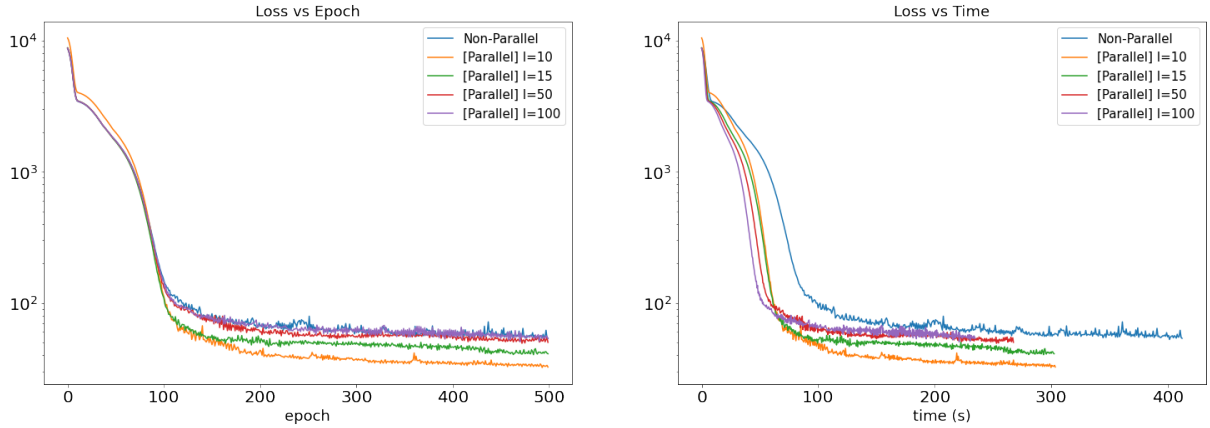

Figure 12: Loss during 500 epochs of learning represented in logarithmic scale, using PR-SGD different take sizes $I \in \{10, 15, 50, 100\}$ with $k = 4$ workers on a data set with 50000 observations.

Let us focus on the loss with the number of epochs: we see that choosing $I = 100$ gives a level of convergence very similar to the non-parallel mode, while choosing I even smaller than 100 gives a better convergence. In fact, synchronizing the results often leads to having less noise and thus converge better to the close local optimum. On the right figure we observe the convergence as well as the runtimes with different values of I.

Remark 27. *As we can expect, the distributed setting gives a faster runtime than the non-parallel SGD, however the time ratio seems to be hardly 1.5 or 2 while we are using 4 threads, this is mainly due to the time of loading pages as shown previously in Figure 9.*

Smaller I, Longer Runtime For the PR-SGD runs, we see that the runtime is decreasing when choosing larger I, this is partially because we have more synchronization steps, but mostly because the idles are more important when I is bigger. In fact, if we see closer the time $\tau_{i,t}$ spent by w_i on the step t, i.e the time for processing the observations a_l with $((t-1)k + (i-1))I + 1 \leq l \leq ((t-1)k + i)I$ (Proposition 7), we can write

$$\tau_{i,t} = \sum_{l=((t-1)k+(i-1))I+1}^{((t-1)k+i)I} \eta_{i,l},$$

where $\eta_{i,l}$ is the time needed by w_i to process observation l. Let us consider $I, J \in \mathbb{N}^*$ such that $Q := I/J \in \mathbb{N}$. The time needed for processing the first I observations if we choose a take size I is

$$\tau^I = \max_{1 \leq i \leq k} \left\{ \sum_{l=1}^{I} \eta_{i,l} \right\} \tag{17}$$

and if we choose a take size J then the time for processing the first I observations is

$$\tau^J = \sum_{u=1}^{Q} \max_{1 \leq i \leq k} \left\{ \sum_{l=uJ+1}^{(u+1)J} \eta_{i,l} \right\}. \tag{18}$$

We have immediately that for any $i \in \{1, \ldots, k\}$

$$\sum_{l=1}^{I} \eta_{i,l} = \sum_{u=1}^{Q} \sum_{l=uJ+1}^{(u+1)J} \eta_{i,l} \leq \sum_{u=1}^{Q} \max_{1 \leq i \leq k} \left\{ \sum_{l=uJ+1}^{(u+1)J} \eta_{i,l} \right\} = \tau^J,$$

taking the maximum on $i \in \{1, \ldots, k\}$ yields

$$\tau^I \leq \tau^J.$$

Moreover, $\tau^J - \tau^I$ is in the worse case linear with I/J: in fact, let $\tau^J_{i,u} := \sum_{l=uJ+1}^{(u+1)J} \eta_{i,l}$ the time needed for processing J observations, as assumed during all the report it takes uniformly distributed random values in some interval $[\alpha^J, \beta^J]$. Equations 17 and 18 can be written as

$$\tau^J = \sum_{u=1}^{Q} \max_{1 \leq i \leq k} \tau^J_{i,u}, \qquad \text{and} \qquad \tau^I = \max_{1 \leq i \leq k} \left\{ \sum_{u=1}^{Q} \tau^J_{i,u} \right\}.$$

In the setting

$$\tilde{\tau}^J_{i,u} = \begin{cases} \beta^J & \text{if } u = i[\text{mod } k] \\ \alpha^J & \text{otherwise,} \end{cases}$$

we have

$$\max_{1 \leq i \leq k} \tilde{\tau}^J_{i,u} = \beta^J \quad \forall u \in \{1, \ldots, Q\}$$

and (assuming that k divides Q)

$$\sum_{u=1}^{Q} \tilde{\tau}^J_{i,u} = \frac{Q}{k}\beta^J + (Q - \frac{Q}{k})\alpha^J \qquad \forall i \in \{1, \ldots, k\}$$

which gives $\tau^J = Q\beta^J$ and $\tau^I = \left(\frac{\beta^J}{k} + \left(1 - \frac{1}{k}\right)\alpha^J\right) Q$. Hence the difference of the two gives

$$\begin{aligned} \tau^J - \tau^I &= \left(1 - \frac{1}{k}\right)(\beta^J - \alpha^J)Q \\ &= \left(1 - \frac{1}{k}\right)(\beta^J - \alpha^J)\frac{I}{J}. \end{aligned}$$

Choice of I The previous development gives a good idea why choosing smaller take sizes can induce a longer runtime even without taking into account the synchronization.
On the particular example we showed in Figure 12, $I = 10$ seems to be the best choice of the take size because it reaches smaller loss values faster. We chose however for out next simulations $I = 150$ because it gives a convergence curve very similar to SGD, and thus it will enable evaluating the ε-performance by just observing the execution time ratio.

8.3 Convergence and Runtime

We saw in Section 6 that PR-SGD-PIP has an increased execution speed compared to PR-SGD, but the convergence rate is slower. We ran both algorithms as well as SGD on many datastes, and we always observe that PR-SGD-PIP cannot reach the same loss level as SGD or PR-SGD, even though its execution time is a little bit shorter than PR-SGD's.
As for PR-SGD-OLD, it gives a convergence rate very similar to PR-SGD's and SGD's, but it has a much faster execution time, which makes it the most performing algorithm. We show in Figure 13 an example of how these algorithms compare to each other with $I = 150$ and $k = 4$.

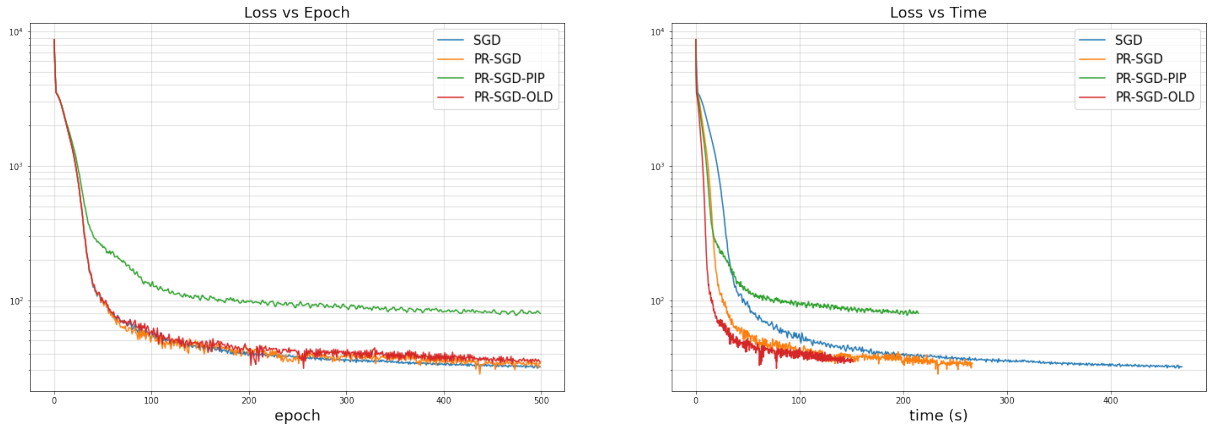

Figure 13: Comparing SGD, PR-SGD and PR-SGD-PIP on dataset with 50000 observations over 500 epochs, with $I = 150$ and $k = 4$.

Note that in Figure 13, PR-SGD-PIP brings down the loss from $\approx 9 \times 10^3$ to ≈ 80, which means that it is a performing optimization algorithm, on the right Figure we even observe that it outperforms PR-SGD during the first epochs, but its weakness is revealed when we need optimize on tight valleys of the objective function, and the gradients need to be a as precise as possible.

8.4 Execution Time Ratio

We will now see how the number of workers influences execution time ratio of our algorithms. The number of observations per page is $P = 13700$ and $N = 550000$, so we have 40 pages. The loading time of a page is on average $\mu = 125ms$, we notice experimentally that μ has a very small variance, the hypothesis that it is a constant is thus legitimate. Finally, for a choice of $I = 150$, we obtain $\alpha = 6.70$ ms and $\beta = 11.26$ ms, and we have $\gamma = 0.0005$ ms ≈ 0 ms. Processing one page using worker should take a time in order of magnitude $\alpha P/I = \approx 611$. The distribution of the execution time τ^I of requests of size I does not really follow a uniform distribution on $[\alpha, \beta]$. The values of α, β we gave were chosen such that

$$\mathbb{E}[\tau^I] = \frac{\alpha + \beta}{2} \text{ and } \text{Var}[\tau^I] = \frac{(\beta - \alpha)^2}{12}.$$

thus if we consider that it is a uniform distribution we obtain the correct mean and variance.

To test the execution time ratio in Proposition 22, we run SGD on the dataset we have described, and then we run the three algorithms with different values of k. Each experiment is repeated 10 times to compute an average of the runtime. We will plot the curve of the function

$$k \mapsto \frac{\mathbb{E}[\mathcal{T}(\text{SGD}; N)]}{k \cdot \mathbb{E}[\mathcal{T}(\text{PR-SGD-OLD}_k; N)]}$$

with $N = 550000$, that approximates the execution time ratio which is in reality the limit when N tends to infinity. In Figure 14 we plot the results of this experiment as well as the theoretical curve of

$k \mapsto \text{TR(PR-SGD-OLD)}(k)$ given by Proposition 22 for $k \leq \lceil k_1 \rceil$, with k_1 given in Proposition 21:

$$k_1 = 1 + \frac{\alpha P}{\mu I} = 5.60.$$

Figure 14: The execution time ratios with the setting described above, for $k \in \{2, \ldots, 12\}$.

The first observation that comes up is that PR-SGD-OLD has a much better execution time ratio than PR-SGD or PR-SGD-PIP. In fact, PR-SGD is better when using 2 workers, but its efficiency decreases when adding more workers, while PR-SGD-OLD becomes even better. Its efficiency also starts dropping after using more than 6 workers because the loading time becomes longer than the processing time as we explained previously. We also observe that PR-SGD-PIP has a better execution time than PR-SGD, but as we saw in the previous section PR-SGD stays better because it reaches smaller loss values.

Concerning PR-SGD-OLD, the theoretical curve is close to the experimental one for $k \leq 6$, the theoretical curve even seems to be pessimistic with regards to the real results. The difference between them is certainly due to the assumption that $\tau^I \sim \mathcal{U}[\alpha, \beta]$, which is not true in reality. What is also remarkable is the value 0.9 of TR(PR-SGD-OLD) reached for $k = 6$. It means that using 6 threads we have a 5.37 speed up, which is very impressive !

As for the convergence rate, Figure 15 shows that increasing the number of workers reduces the loss

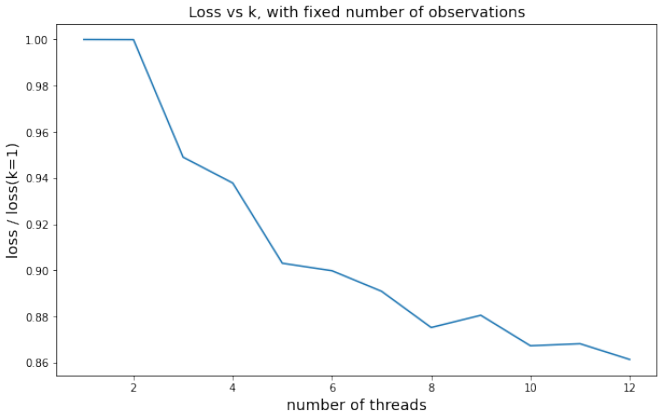

Figure 15: the ratio $\frac{\text{loss}(k)}{\text{loss}(k=1)}$, where loss$(k)$ is the value of F obtained after processing 27.5×10^6 observations with PR-SGD-OLD using k processors.

obtained after N iterations. The difference is though not very significant. Anyways, it confirms that

having more workers does not have a bad impact on the convergence, and thus the ε-performance of the algorithm when varying k can be lower bounded by its execution time ratio.

9 Conclusion

In the work we have done, we have first reviewed the PR-SGD algorithm, which is a distributed gradient descent algorithm, but in a more simplified framework than in the original paper. We also estimated its execution time and reviewed its proof of convergence. PR-SGD has the advantage of being a reproducible algorithm, unlike other distributed gradient descent algorithms such as HogWild!. It has however the disadvantage of having idles to the synchronization between the workers.
This problem is accentuated in the case of very high dimensional data that cannot be loaded on the RAM at once, because the data loading is a costly operation that takes a long time, and that will hence generate important idles.
A first solution to overcome this is to allow the workers to start their next tasks without waiting for all the others to finish. But to keep the reproducibility of the algorithm, we were forced to not use the results computed at step $t-1$ at step t. We therefore proposed an algorithm PR-SGD-PIP, and we showed that mathematically it is equivalent to a gradient descent using delayed gradients, a problem that has already been studied in the literature. The execution time is thus improved compared to PR-SGD but the convergence rate is slowed down by a factor $O(h^2)$.
Since all the processors have access to the same database we proposed yet another variant PR-SGD-OLD of PR-SGD, where one worker will take care of loading the data on a shared memory in advance while the other workers will perform the gradient descent on the already loaded data. Given the large data loading time in our cases, we have shown that this algorithm is much more efficient than PR-SGD or PR-SGD-PIP, and achieves very high speedup as long as the number of workers does not exceed a certain critical value k_1. The limiting and blocking point of this algorithm is not the gradient descent or the synchronization between workers, but rather the data loading time. This could be improved by taking a more technical look at how the data is represented and stored, and trying to compress it in order to speed up its reading.
We proved PR-SGD-OLD to be more efficient than PR-SGD when the data loading takes a time comparable to the time for processing it. This is typically the case when we high very dimensional data and each gradient is computed only using few features of an observation. It is only the case when using machines that have powerful processors but slow RAM.

References

[1] Hao Yu, Sen Yang, and Shenghuo Zhu. "Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model Averaging Works for Deep Learning". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 5693–5700. DOI: 10.1609/aaai.v33i01.33015693. URL: https://doi.org/10.1609/aaai.v33i01.33015693.

[2] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[3] Ofer Dekel et al. "Optimal Distributed Online Prediction Using Mini-Batches". In: *J. Mach. Learn. Res.* 13 (2012), pp. 165–202. DOI: 10.5555/2503308.2188391. URL: https://dl.acm.org/doi/10.5555/2503308.2188391.

[4] Tao Lin, Sebastian U. Stich, and Martin Jaggi. "Don't Use Large Mini-Batches, Use Local SGD". In: *CoRR* abs/1808.07217 (2018). arXiv: 1808.07217. URL: http://arxiv.org/abs/1808.07217.

[5] Blake E. Woodworth et al. "Is Local SGD Better than Minibatch SGD?" In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 10334–10343. URL: http://proceedings.mlr.press/v119/woodworth20a.html.

[6] Sebastian U. Stich. "Local SGD Converges Fast and Communicates Little". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=S1g2JnRcFX.

[7] Martin Zinkevich et al. "Parallelized Stochastic Gradient Descent". In: *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*. Ed. by John D. Lafferty et al. Curran Associates, Inc., 2010, pp. 2595–2603. URL: https://proceedings.neurips.cc/paper/2010/hash/abea47ba24142ed16b7d8fbf2c740e0d-Abstract.html.

[8] Benjamin Recht et al. "Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent". In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*. Ed. by John Shawe-Taylor et al. 2011, pp. 693–701. URL: https://proceedings.neurips.cc/paper/2011/hash/218a0aefd1d1a4be65601cc6ddc1520e-Abstract.html.

[9] Yossi Arjevani, Ohad Shamir, and Nathan Srebro. "A Tight Convergence Analysis for Stochastic Gradient Descent with Delayed Updates". In: *Algorithmic Learning Theory, ALT 2020, 8-11 February 2020, San Diego, CA, USA*. Ed. by Aryeh Kontorovich and Gergely Neu. Vol. 117. Proceedings of Machine Learning Research. PMLR, 2020, pp. 111–132. URL: http://proceedings.mlr.press/v117/arjevani20a.html.

[10] Martin Zinkevich, Alexander J. Smola, and John Langford. "Slow Learners are Fast". In: *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*. Ed. by Yoshua Bengio et al. Curran Associates, Inc., 2009, pp. 2331–2339. URL: https://proceedings.neurips.cc/paper/2009/hash/b55ec28c52d5f6205684a473a2193564-Abstract.html.

[11] Sai Praneeth Karimireddy et al. “Error Feedback Fixes SignSGD and other Gradient Compression Schemes”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 3252–3261. URL: http://proceedings.mlr.press/v97/karimireddy19a.html.

[12] Hongchang Gao, Gang Wu, and Ryan A. Rossi. “Provable Distributed Stochastic Gradient Descent with Delayed Updates”. In: *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021, Virtual Event, April 29 - May 1, 2021*. Ed. by Carlotta Demeniconi and Ian Davidson. SIAM, 2021, pp. 441–449. DOI: 10.1137/1.9781611976700.50. URL: https://doi.org/10.1137/1.9781611976700.50.

[13] Hongchang Gao, Gang Wu, and Ryan A. Rossi. “Provable Distributed Stochastic Gradient Descent with Delayed Updates”. In: *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021, Virtual Event, April 29 - May 1, 2021*. Ed. by Carlotta Demeniconi and Ian Davidson. SIAM, 2021, pp. 441–449. DOI: 10.1137/1.9781611976700.50. URL: https://doi.org/10.1137/1.9781611976700.50.

A Mathematical Tools

Reminders of Smoothness and Strong Convexity

We remind here the definition of a L-smooth function and the classical inequality it verifies.

Definition 28 (L-smoothness)**.** *Let $F : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. F is said to be L-smooth if its gradient is L-Lipschitz, i.e for any $x, y \in \mathbb{R}^d$ we have*

$$\|\nabla F(x) - \nabla F(y)\| \leq L\|x - y\|.$$

Proposition 29. *If $F : \mathbb{R}^d \to \mathbb{R}$ is a L-smooth function then for all $x, y \in \mathbb{R}^n$ we have*

$$F(y) \leq F(x) + \langle \nabla F(x), y - x\rangle + \frac{L}{2}\|x - y\|^2.$$

Computational Results

We prove here some elementary results that were used in previous proofs.

Lemma 30. *If Z is a random vector in $\mathbb{R}^d$ then*

$$\mathbb{E}[\|Z\|^2] = \mathbb{E}[\|Z - \mathbb{E}[Z]\|^2] + \|\mathbb{E}[Z]\|^2.$$

Proof. Let $Z = (Z_1, \ldots, Z_d)$ a random vector in $\mathbb{R}^d$. By linearity of the expectation we have that $\mathbb{E}[Z] = (\mathbb{E}[Z_u])_{1 \leq u \leq d}$ and

$$\begin{aligned}\mathbb{E}[\|Z - \mathbb{E}[Z]\|^2] &= \sum_{u=1}^{d} \mathbb{E}[(Z_u - \mathbb{E}[Z_u])^2] \\ &= \sum_{u=1}^{d} \mathbb{E}[Z_u^2] - \sum_{u=1}^{d} \mathbb{E}[Z_u]^2 = \mathbb{E}[\|Z\|^2] + \|\mathbb{E}[Z]\|^2.\end{aligned}$$

□

Lemma 31. *If $Z_1, \ldots, Z_k$ are k independent random vectors in $\mathbb{R}^d$ then*

$$\mathbb{E}[\|\sum_{i=1}^{k}(Z_i - \mathbb{E}[Z_i])\|^2] = \sum_{i=1}^{k} \mathbb{E}[\|Z_i - \mathbb{E}[Z_i]\|^2].$$

Proof. Without loss of generality, we can assume that $\mathbb{E}[Z_i] = 0$ for all $i \in \{1, \ldots, k\}$. We have then

$$\mathbb{E}[\|\sum_{i=1}^{k} Z_i\|^2] = \sum_{i=1}^{k} \mathbb{E}[\|Z_i\|^2] + 2 \sum_{1 \leq i < j \leq k} \mathbb{E}[\langle Z_i, Z_j\rangle],$$

and by independence we have for any $1 \leq i < j \leq k$ that $\mathbb{E}[\langle Z_i, Z_j\rangle] = \langle \mathbb{E}[Z_i], \mathbb{E}[Z_j]\rangle = 0$, which gives the result. □

Lemma 32. *If $u_1, \ldots, u_k$ are i.i.d random variables following a uniform random distribution in $[0, 1]$ then*

$$\mathbb{E}\left[\max_{1 \leq i \leq k}\{u_i\}\right] = \frac{k}{k+1}.$$

Proof. Let $U := \max_{1 \leq i \leq k}\{u_i\}$. We have that $U \in [0, 1]$ a.s. and for any $x \in [0, 1]$

$$\Pr[U \leq x] = \Pr[\forall 1 \leq i \leq k : \ u_i \leq x] = \Pr[u_1 \leq x]^k = x^k.$$

U has therefore a density function $x \in [0, 1] \mapsto kx^{k-1}$ and thus

$$\mathbb{E}[U] = \int_0^1 x \cdot kx^{k-1} dx = \frac{k}{k+1}.$$

□