Semester Thesis - MLO lab, EPFL
Asynchronous updates for stochastic gradient descent

Supervisors: Dr. Sebastian Stich, Prof. Martin Jaggi
Author: Alberto Silvio Chiappa

July 11, 2018

Abstract

Finding convergence rates for numerical optimization algorithms is an important task, because it gives a justification to their use in solving practical problems, while also providing a way to compare their efficiency. This is especially useful in an asynchronous environment, because the algorithms are often proven to be more efficient than their synchronous counterparts by experience, but they lack the theory that justifies this property. Furthermore, analyzing the various issues that can arise when inconsistency is taken into consideration, it is possible to obtain a clearer picture of the downsides of inexact implementations one should be aware of.
This work tries to address the problem of finding an expected convergence rate for the asynchronous version of the widely popular stochastic gradient descent algorithm, applied to the common class of problems that present a cost function with a sum structure. It follows a similar approach to the one suggested by R. Leblond, F. Pedregosa and S. Lacoste-Julien in "ASAGA: Asynchronous Parallel SAGA" (2016) [RLLJ16], also borrowing their formalization of asynchronicity.
The main achievement of this work is a bound on the constant step size that guarantees convergence in expectation of the algorithm. The relative convergence rate is also obtained. The result is also partially validated by sequential models of an asynchronous environment.
We hope that this can be a basis for future applications of the same approach to more specific algorithms and that numerical experiments on real multiprocessor architecture can be performed in the future to further validate the convergence rates.

1 Introduction

The parallelization of stochastic optimization algorithms like Stochastic Gradient Descent, Stochastic Coordinate Descent and others is a necessary result in order for large machine learning problems to exploit the computation power offered by distributed memory clusters. However, making it efficient has proven to be a challenging task, as most of these algorithms are intrinsically sequential. Exact parallelization algorithms have been proposed (see, for instance, [BT93]), but, while working correctly, they have efficiency problems, mainly because of the large amount of time spent in locking the threads to assure consistency. In fact, the rates achievable with multicore systems can be incredibly high (for instance, the just released cluster Fidis at Epfl has a peak performance of 401.3 TFLOPs [http://scitas.epfl.ch/hardware/fidis]), which moves the bottlenecks in parallel computation to the locking, which is needed for synchronization. Indeed, in order for the correctness of the algorithm to be guaranteed, some threads must "lock" at certain critical points, waiting for the other to complete their computations before starting to execute again. The time wasted by the cores in both waiting and communicating hinders scalability and harms performance.
To make up for this issue, lock-free algorithms have been proposed. They are versions of well known optimization algorithms that manage to avoid most of the (if not any) locking, at the cost of introducing some inconsistency in the computation. A popular example of this class of algorithms is "HOGWILD!" [FNW11]. Experiments show that they manage to reach a notable speedup if compared to their exact counterpart, but the theoretical convergence guarantees are usually proven for sparse problems. This means that the hypothesis according to which individual update steps only modify a small part of the solution has to be introduced. Under this assumption, memory overwrites can be considered as rare events, so their influence on the convergence rate can

be neglected.
However, depending on the application, sparsity can be condition that is not easy to guarantee, if not impossible at all. For instance, one can simply think about a logistic regression problem. It is characterized by the following cost function:

$$f(w) = -\sum_{n=1}^{N} \left[y_n \log(\sigma(x_n^T w)) + (1 - y_n) \log(1 - \sigma(x_n^T w)) \right], \tag{1}$$

where $x_n \in \mathbb{R}^d$ are the data relative to outcome $y_n \in \{0, 1\}$, while $\sigma : \mathbb{R} \to \mathbb{R}$ is the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ and w is the weight vector (the solution we have to find).
Its minimization is one of the simplest examples of problems that can be addressed with Stochastic Gradient Descent. The stochastic gradient can be expressed as

$$\nabla_s f(w) = (\sigma(x_n^T w) - y_n) x_n \tag{2}$$

Despite only needing the data relative to one sampled observation, the resulting stochastic gradient is not likely to present any sparsity, as it has the same number of zero elements as the data vector. In order to be able to deal with such problems, we follow the same approach that is described in [RLLJ16], in which the authors are able to obtain linear speedups also for non-sparse problems (assuming they are not ill-conditioned). The first part of the asynchronous convergence estimate is largely inspired by their work, as SGD can be seen as a simpler implementation of SAGA. However, given the also simpler formulation, a cleaner convergence estimate could be found, that guarantees convergence for larger step sizes.

2 Synchronous convergence estimate

First of all, we introduce the synchronous SGD problem and its convergence estimate for strongly convex functions with Lipschitz-continuous gradient. This is not only a preliminary result to the following part, but also a good reference for the subsequent results. In addition, it introduces some of the techniques that will be used afterwards.
As a reminder, SGD is based on moving from the solution at step t to the solution at step $t + 1$ with the following update rule:

$$x_{t+1} = x_t - \gamma \nabla_s f_i(x_t) \tag{3}$$

in which γ is the step size and f_i is one of the functions whose sum forms the global cost. A sum structure for the cost function is a fundamental requirement for this algorithm.

Theorem 1. *Given the optimization problem*

$$\min_{x \in \mathbb{R}^d} f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) \tag{4}$$

where f_i are μ-strongly convex functions with an L-Lipschitz continuous gradient, the Stochastic Gradient descent algorithm performed with a constant step size $\gamma = \frac{1}{L}$ has a guaranteed convergence rate of $\rho = \frac{\mu}{L}$, which means that

$$\mathbb{E}\left[\|x_{t+1} - x^*\|^2 \right] \leq (1 - \rho) \, \mathbb{E}\left[\|x_t - x^*\|^2 \right] \tag{5}$$

where x^* denotes the point in which the minimum is reached and x_t the solution at iteration t.

Proof. Given the update rule of the stochastic gradient descent

$$x_{t+1} = x_t - \gamma g_t \tag{6}$$

where $g_t = \nabla_s f_t$ is the stochastic gradient of f, we can express the error at iteration $t+1$ as follows:

$$\|x_{t+1} - x^*\|^2 = \|x_t - \gamma g_t - x^*\|^2 = \|x_t - x^*\|^2 + \gamma^2 \|g_t\|^2 - 2\gamma \langle x_t - x^*, g_t \rangle \tag{7}$$

Then we use the following important results due to strong convexity and Lipschitz-continuity of the gradient of f_i:

$$\langle x_t - x^*, \nabla f(x_t) \rangle \geq f(x_t) - f(x^*) \frac{\mu}{2} \|x_t - x^*\|^2 \tag{8}$$

and

$$\|\nabla f(x_t)\|^2 \leq 2L\left(f(x_t) - f(x^*)\right) \tag{9}$$

Using these two inequalities[1] and considering the inequality in expectation over the randomly sampled data used to compute the stochastic gradient we get:

$$\begin{aligned}\mathbb{E}\left[\|x_{t+1} - x^*\|^2\right] &\leq \mathbb{E}\left[\|x_t - x^*\|^2\right] + (2L\gamma^2 - 2\gamma)\left(\mathbb{E}\left[f(x_t)\right] - f(x^*)\right) - \gamma\mu\|x_t - x^*\|^2 \\ &\leq (1-\gamma\mu)\mathbb{E}\left[\|x_t - x^*\|^2\right] + 2\gamma(\gamma L - 1)\left(\mathbb{E}\left[f(x_t)\right] - f(x^*)\right)\end{aligned} \tag{10}$$

Imposing $\gamma = \frac{1}{L}$ concludes the proof. □

3 Asynchronous convergence estimate

In this section we introduce the concept of asynchronicity and how it can be mathematically modeled. As anticipated, a major issue for parallel implementations of SGD is the time spent on communication between processors, which greatly hurts performance. Therefore, algorithms that never lock the processes during execution have been successfully tested [FNW11]. They give up consistency when reading the solution at the current time step in exchange for a faster execution. Obviously, this introduces an error. The aim of this section is to find a convergence estimate for SGD that takes into consideration the inconsistency of the reading step, but still manages to guarantee a decrease in the expectation of the error.

3.1 Global ordering of the updates

A natural concept for iterative schemes is the ordering of the updates. While in a synchronous framework its definition is obvious, in [RLLJ16] is pointed out for the first time (to our knowledge) a quite important problem, that seems to have been neglected so far: in an asynchronous framework, not all definitions of a global ordering are equivalent. Most important, some even popular definition do not have the properties one would expect (and desire for the proofs). Following the scheme outlined in [citation], we list the most popular definitions.

The "After Write" approach

The global counter records the number of successful writes into the shared memory. This means that x_t stands for the real content of the memory after t updates. On the other side, $\hat{x}_t$ is the local copy of the core that made the $(t+1)^{th}$ update. These definitions make it impossible to know what $\hat{x}_t$ and i_t (which is the index sampled for the corresponding update, that leads to the solution x_{t+1}) will be at time t. Indeed, they depend on which core is going to write the update x_{t+1} in the shared memory.
What is really problematic about this approach is that a dependence between the computation time for the update and the sampled index i_t also leads to a dependency between $\hat{x}_t$ and i_t. The label of the update can be larger or smaller depending on how long it took to compute it. As this independence is needed for a rigorous proof, it has to be guaranteed imposing that all the updates have the same computation time. If this is too restrictive, another definition for the global ordering had to be formulated.

The "Before Read" approach

According to this approach, the global t counter is updated before a core starts to read in the shared memory. In this framework, $\hat{x}_t$ is the t^{th} inconsistent read and i_t the sampled index for the update. Independence between the sampled index and the inconsistent read can be easily obtained keeping the reading process the same for each update. It is clear that this requirement is much weaker than a constant update time.
However, also this approach presents a downside. As the label is assigned before a core begins the reading process, it may be corrupted not only by previous updates, but also by subsequent ones. This means that $\hat{x}_t$ can depend on i_r for $r > t$

[1] **Erratum 07/2018 SS**: And under the assumption $\mathbb{E}\left[\|g_t\|^2\right] \leq 2L(f(x_t) - f(x^*))$. This inequality is assumed *throughout* this work (in particular in Theorem 2). It holds for instance for functions where the stochastic gradients vanish at the optimum, $\nabla f_i(x^*) = 0$, $\forall i \in [n]$.

The "After Read" approach

Defining $\hat{x}_t$ as the $(t+1)^{th}$ fully completed read makes up for the problem of the "Before Read" approach, while also guaranteeing the independence of i_r from $\hat{x}_t$ for $r > t$. This is due to the fact that this new definition really defines a global ordering on the $\hat{x}_t$ iterates.

3.2 Model for asynchronicity

The numerical model of inconsistent update for SGD consists in considering that other processors can write their computed solution in the shared memory while one other processor is reading. In principle, any coordinate could be an element of a solution computed for any update before the reading is performed. To ease the procedure, we make the assumption that at most τ updates before the current one can interfere with the reading process. We have the following expression for the difference between the "after read" partial solution and its inconsistent counterpart stored in the local memory of a processor:

$$\hat{x}_t - x_t = \gamma \sum_{u=(t-\tau)_+}^{t-1} I_t(u) g\left(\hat{x}_u\right) \tag{11}$$

where x_t is the solution at step t, $\hat{x}_t$ is its inconsistent read, γ is the step size, $I_t(u)$ is the indicator function of the updates u that have changed the data in the shared memory while the reading t was taking place.
It is important to observe that the "after read" definition for the reference solution at step t allows us to neglect the dependency from the sampled data in the computation of the gradient, as it only influences it through the time the reading process takes. It is extensively explained (in the paper) how this approach makes the reading time independent of the sampled data.

Theorem 2. *Given the optimization problem*

$$\min_{x \in \mathbb{R}^d} f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) \tag{12}$$

where f_i are μ-strongly convex functions with an L-Lipschitz continuous gradient, the Asynchronous Stochastic Gradient descent algorithm with a constant step size

$$\gamma \leq \frac{1}{2L} \left(\frac{p\tau^2}{1 - \rho\tau} + 1 \right)^{-1} \tag{13}$$

has a guaranteed convergence rate of

$$\rho = \frac{\mu}{2L(1 + p\tau^2) + \mu(1 + \tau)} \tag{14}$$

This means that

$$\mathbb{E}\left[\left\| x_{t+1} - x^* \right\|^2 \right] \leq (1 - \rho)\, \mathbb{E}\left[\left\| x_t - x^* \right\|^2 \right] \tag{15}$$

where x^* denotes the point in which the minimum is reached and x_t the solution at iteration t.

In order to keep the proof of this theorem compact, the explanation of most results is delayed to a few lemmas, that will be proven afterwards.

Proof. As it was done for the synchronous convergence estimate, we begin expressing the dependence of the solution at step $t+1$ from the solution at step t. To do this, we only have to use the update rule for a gradient descent algorithm, according to which

$$x_{t+1} = x_t - \gamma g_t \tag{16}$$

where γ is the step size and g_t is the stochastic gradient. This allows us to proceed as follows:

$$\begin{aligned} \left\| x_{t+1} - x^* \right\|^2 &= \left\| x_t - \gamma g_t - x^* \right\|^2 = \left\| x_t - x^* \right\|^2 + \gamma^2 \left\| g_t \right\|^2 - 2\gamma \langle \hat{x}_t - x^*, g_t \rangle \\ &= \left\| x_t - x^* \right\|^2 + \gamma^2 \left\| g_t \right\|^2 - 2\gamma \langle \hat{x}_t - x^*, g_t \rangle + 2\gamma \langle \hat{x}_t - x_t, g_t \rangle \end{aligned} \tag{17}$$

In (17) x_t is the "after read" solution at iteration t, x^* is the exact minimizer of the functional, $g_t = g(\hat{x}_t)$ is the stochastic gradient computed from the inconsistent read of x_t and γ is the step size. The steps simply consist in computing the square of the norm and adding and subtracting the inconsistent read $\hat{x}_t$ inside the inner product.
It should be observed that the inconsistent reading process generates an additional term (compared to the same passage of the proof for the sequential algorithm):

$$2\gamma\langle\hat{x}_t - x_t, g_t\rangle \tag{18}$$

whose sign is unknown. Bounding this term and isolating the negative contribution of

$$2\gamma\langle\hat{x}_t - x^*, g_t\rangle \tag{19}$$

will be the central aims of the proof.
From now on, we consider a batch size equal to 1. Therefore, the stochastic gradient g_t is simply $\nabla f_i(\hat{x}_t)$, with $i = i_t$ is the index sampled at random at step t.
We can proceed with the manipulation of (17). Thanks to the definition of strong convexity applied to f_i:

$$\langle\hat{x}_t - x^*, g_t\rangle = \langle\hat{x}_t - x^*, \nabla f_i(\hat{x}_t)\rangle \geq f_i(\hat{x}_t) - f_i(x^*) + \frac{\mu}{2}\left\|\hat{x}_t - x^*\right\|^2 \tag{20}$$

By using (20) in (17) we get

$$\begin{aligned}\left\|x_{t+1} - x^*\right\|^2 \leq \left\|x_t - x^*\right\|^2 + \gamma^2\left\|\nabla f_i(\hat{x}_t)\right\|^2 - 2\gamma\left(f_i(\hat{x}_t) - f_i(x^*) + \frac{\mu}{2}\left\|\hat{x}_t - x^*\right\|^2\right) + \\ + 2\gamma^2\sum_{u=(t-\tau)_+}^{t-1}\left\langle I_t(u)\nabla f_{k(u)}\left(\hat{x}_u\right), \nabla f_i(\hat{x}_t)\right\rangle\end{aligned} \tag{21}$$

To obtain (21) the expression for the difference between the solution x_t and its inconsistent read $\hat{x}_t$, which is

$$\hat{x}_t - x_t = \gamma\sum_{u=(t-\tau)_+}^{t-1} I_t(u) g\left(\hat{x}_u\right) \tag{22}$$

has been used.
At this point a clarification about the notation has to be made. Until this moment we have stressed the fact that the stochastic gradient is computed only with one of the functions f_i, sampled at random. The sampled index i is, in principle, a function of the global index t. However, as pointed out before, the "after read" definition of the global ordering does eliminate this dependency. Given this important property, which will be fundamental to find an estimate in expectation, from now on we just indicate the stochastic gradient as ∇_s, without specifying the sampled index every time.
Thanks to the results presented in Lemma 1, Lemma 2 and 3, inequality (21) can be further bounded by

$$\begin{aligned}\left\|x_{t+1} - x^*\right\|^2 \leq \left(1 - \frac{\gamma\,\mu}{1+\alpha}\right)\left\|x_t - x^*\right\|^2 + \tau\,\gamma^2\left(\frac{\gamma\mu}{\alpha} + \eta\right)\sum_{u=(t-\tau)_+}^{t-1} I_t(u)\left\|\nabla_s f\left(\hat{x}_u\right)\right\|^2 + \\ + \gamma^2\left(1 + \frac{1}{\eta}\right)\left\|\nabla_s f(\hat{x}_t)\right\|^2 - 2\gamma\left(f(\hat{x}_t) - f(x^*)\right)\end{aligned} \tag{23}$$

For the following steps of the proof it is convenient to fix the values of the positive constants coming from the previous inequalities: $\alpha = \gamma\mu$ and $\eta = 1$. However, this particular choice of α will make it impossible to recover the estimate for the synchronous algorithm for $\tau \to 0$, as it will be discussed later. This choice of the constants leads to the inequality

$$\begin{aligned}\left\|x_{t+1} - x^*\right\|^2 \leq \left(1 - \frac{\gamma\,\mu}{1+\gamma\mu}\right)\left\|x_t - x^*\right\|^2 + 2\gamma^2\tau\sum_{u=(t-\tau)_+}^{t-1} I_t(u)\left\|\nabla_s f\left(\hat{x}_u\right)\right\|^2 + \\ + 2\gamma^2\left\|\nabla_s f(\hat{x}_t)\right\|^2 + -2\gamma\left(f(\hat{x}_t) - f(x^*)\right)\end{aligned} \tag{24}$$

To be able to keep the notation more synthetic, we define:

$$a_t = \mathbb{E}\left[\left\|x_t - x^*\right\|^2\right] \qquad e_t = \mathbb{E}\left[f(\hat{x}_t)\right] - f(x^*) \tag{25}$$

This allows us to express the expectation of the previous result in a compact form:

$$a_{t+1} \leq \left(1 - \frac{\gamma\mu}{1+\gamma\mu}\right) a_t + 2\gamma^2\tau\mathbb{E}\left[\sum_{u=(t-\tau)_+}^{t-1} I_t(u) \left\|\nabla_s f(\hat{x}_u)\right\|^2\right] + 2\gamma^2\mathbb{E}\left[\left\|\nabla_s f(\hat{x}_t)\right\|^2\right] - 2\gamma e_t \quad (26)$$

Thanks to Lemma 5 and Lemma 6, which use the Lipschitz continuity of the gradient of each f_i to bound the stochastic gradient in terms of f, the previous terms can be bounded and regrouped to get the following simpler inequality:

$$a_{t+1} \leq \left(1 - \frac{\gamma\mu}{1+\gamma\mu}\right) a_t + 4pL\gamma^2\tau \sum_{u=(t-\tau)_+}^{t-1} e_u + (4\gamma^2 L - 2\gamma)e_t \quad (27)$$

In order to be able to express a convergence result getting rid of the historical updates we use the same trick as in [RLLJ16].
We first define an auxiliary function, that considers all the errors (in expectation) up to step t:

$$\mathcal{L}_t = \sum_{u=0}^{t} (1-\rho)^{t-u} a_u \qquad 0 < \rho < 1 \quad (28)$$

Now we are able to regroup the terms to get:

$$\mathcal{L}_{t+1} \leq (1-\rho)^{t+1} a_0 + \left(1 - \frac{\gamma\mu}{1+\gamma\mu}\right)\mathcal{L}_t + \left(4\gamma^2 L \frac{p\tau^2}{1-\rho\tau} + 4\gamma^2 L - 2\gamma\right)\sum_{u=0}^{t}(1-\rho)^{t-u} e_u \quad (29)$$

By imposing

$$\gamma = \frac{1}{2L}\left(\frac{p\tau^2}{1-\rho\tau} + 1\right)^{-1} \quad (30)$$

we get

$$\mathcal{L}_{t+1} \leq (1-\rho)^{t+1} a_0 + \left(1 - \frac{\gamma\mu}{1+\gamma\mu}\right)\mathcal{L}_t \quad (31)$$

which leads to:

$$a_t \leq (1-\rho)^{t+1} a_0 + \sum_{u=0}^{t}\left[\left(1 - \frac{\gamma\mu}{1+\gamma\mu}\right)(1-\rho)^{t-u} - (1-\rho)^{t+1-u}\right] a_u \quad (32)$$

We finally impose that:

$$\rho = \frac{\gamma\mu}{1+\gamma\mu} \quad (33)$$

Relation (33) clearly shows that ρ satisfies the condition $0 < \rho < 1$. This way we get the final convergence estimate

$$a_t \leq \left(1 - \frac{\gamma\mu}{1+\gamma\mu}\right)^{t+1} a_0 \quad (34)$$

We still have to find an explicit expression for the convergence rate. The previous result is not a guarantee by itself, because the bound on γ depends on ρ, which is itself a function of γ. This causes (33) to define a nonlinear relation between ρ and γ, which is quadratic. Simple computations performed on equations (30) and (33) show that $\rho = \rho(\gamma)$ is implicitly defined by the following quadratic equation:

$$\tau(2L+\mu)\rho^2 - (2L + \mu + 2p\tau^2 L + \tau\mu)\rho + \mu = 0 \quad (35)$$

Lemma 7 shows how to solve the equation, isolates the only acceptable solution and finds a weaker but simpler convergence rate certificate:

$$\rho \geq \frac{\mu}{2L + \mu + 2p\tau^2 L + \mu\tau} = \frac{\mu}{2L(1 + p\tau^2) + \mu(1 + \tau)} \tag{36}$$

The previous result, while finding an explicit expression for the convergence rate, also guarantees its existence. This concludes the proof. □

List of lemmas

Here follows the list of lemmas that are used in the proof of Theorem 2. These are not to be intended as an effort to generalize some results that could be useful in other contexts. We decided to organize the proof in this way because it is long enough for the reader to loose sight of the problem as a whole. As all the lemmas are very closely linked to the main theorem, the notation will stay the same and will not be defined at the beginning of each one.
Here is the list of the quantities that will be most frequently used:

- $f_i : \mathbb{R}^d \to \mathbb{R}$ are μ-strictly convex functions of class C^1 with an L-Lipschitz continuous gradient.
- $x_t \in \mathbb{R}^d$ is the update computed using $\hat{x}_{t-1}$ inconsistent read.
- $\hat{x}_t \in \mathbb{R}^d$ is the inconsistent read at step t, defined with the "After Read" global ordering.
- $I_t(u)$ is an indicator function that assumes value 1 if the update u corrupts the shared memory when the read $\hat{x}_t$ is being performed, and 0 otherwise. p is the probability that each update in range $[t - \tau, t - 1]$ has to interfere with the reading process.
- γ is the step size of the descent.
- ∇_s denotes the stochastic gradient. When applied to f, it implies a random sampling of an index i and the computation of the gradient of f_i.
- $\langle \cdot, \cdot \rangle$ denotes the standard $\mathbb{R}^d$ inner product and $\|\cdot\|$ the induced norm.

Also, we repeatedly use some popular inequalities:

- Cauchy-Schwartz: $\forall a,\, b \in \mathbb{R}^d$ it holds that $\langle a, b \rangle \leq \|a\| \, \|b\|$
- Young: $\forall a,\, b \in \mathbb{R}^d, \forall \alpha \in \mathbb{R}_+$ it holds that $\langle a, b \rangle \leq \frac{\alpha}{2} \|a\|^2 + \frac{1}{2\alpha} \|b\|^2$
- Bernoulli: $\forall x \in \mathbb{R}, \forall n \in \mathbb{N}, x > -1$ it holds that $(1 + x)^n \geq 1 + nx$
- Generalized Bernoulli: $\forall x \in \mathbb{R}, \forall r \in \mathbb{R}, x > -1, 0 \leq r \leq 1$ it holds that $(1 + x)^r \leq 1 + rx$

As it was done for the proof of Theorem 2, in order to keep the notation more synthetic, the quantities a_t and e_t will be used. They are defined as follows:

$$a_t = \mathbb{E}\left[\|x_t - x^*\|^2\right] \qquad e_t = \mathbb{E}\left[f(\hat{x}_t)\right] - f(x^*) \tag{37}$$

Lemma 1. *For any $\alpha > 0$ the following inequality holds:*

$$-\|\hat{x}_t - x^*\|^2 \leq \frac{1}{\alpha} \|x_t - \hat{x}_t\|^2 - \frac{1}{1 + \alpha} \|x_t - x^*\|^2 \tag{38}$$

Proof. This result follows from an application of Young's inequality. Thanks to the non-negativity of the square of a binomial it is easy to show that, for any couple of vectors $a,\, b \in \mathbb{R}^d$ and for any $\alpha \in \mathbb{R}_+$,

$$2\langle a, b \rangle \leq \alpha \|a\|^2 + \frac{1}{\alpha} \|b\|^2 \tag{39}$$

We can apply this inequality to the square of the sum of two vectors:

$$\|a + b\|^2 = \|a\|^2 + \|b\|^2 + 2\langle a, b \rangle \leq (1 + \alpha) \|a\|^2 + (1 + \frac{1}{\alpha}) \|b\|^2 \tag{40}$$

Simply rearranging the inequality leads to:

$$-\|a\|^2 \leq \frac{1}{\alpha}\|b\|^2 - \frac{1}{1+\alpha}\|a+b\|^2 \tag{41}$$

By replacing

$$a = \hat{x}_t - x^* \qquad a+b = x_t - x^* \qquad b = x_t - \hat{x}_t \tag{42}$$

we obtain the thesis of this lemma. □

Lemma 2. *The following inequality holds:*

$$\sum_{u=(t-\tau)_+}^{t-1} I_t(u)\langle \nabla_s f(\hat{x}_u), \nabla_s f(\hat{x}_t)\rangle \leq \frac{\eta\tau}{2} \sum_{u=(t-\tau)_+}^{t-1} I_t(u)\|\nabla_s f(\hat{x}_u)\|^2 + \frac{1}{2\eta}\|\nabla_s f(\hat{x}_t)\|^2 \tag{43}$$

Proof. First of all, the following preliminary result has to be proven.

$$\left\|\sum_{i=1}^{N} a_i\right\|^2 \leq N\sum_{i=1}^{N}\|a_i\|^2 \tag{44}$$

The following bound can be shown:

$$\left\|\sum_{i=1}^{N} a_i\right\|^2 = \sum_{i=1}^{N}\|a_i\|^2 + 2\sum_{i=1}^{N}\sum_{j>i}\langle a_i, a_j\rangle \leq \sum_{i=1}^{N}\|a_i\|^2 + 2\sum_{i=1}^{N}\sum_{j>i}\left(\frac{1}{2}\|a_i\|^2 + \frac{1}{2}\|a_j\|^2\right) \tag{45}$$

One should observe that each a_i appears in the inner sum i times as first member and $N-i-1$ times as second member, for a total of $N-1$ occurrences. This fully justifies inequality (44). Thanks again to Young's inequality and the just mentioned result, we can proceed as follows towards the claim:

$$\begin{aligned}\sum_{u=(t-\tau)_+}^{t-1} I_t(u)\langle \nabla_s f(\hat{x}_u), \nabla_s f(\hat{x}_t)\rangle &\leq \frac{\eta}{2}\left\|\sum_{u=(t-\tau)_+}^{t-1} I_t(u)\nabla_s f(\hat{x}_u)\right\|^2 + \frac{1}{2\eta}\|\nabla_s f(\hat{x}_t)\|^2 \\ &\leq \frac{\eta\tau}{2}\sum_{u=(t-\tau)_+}^{t-1} I_t(u)\|\nabla_s f(\hat{x}_u)\|^2 + \frac{1}{2\eta}\|\nabla_s f(\hat{x}_t)\|^2\end{aligned} \tag{46}$$

where the last passage was possible because the terms of the sum are at most τ □

Lemma 3. *The following inequality holds:*

$$\left\|\sum_{u=(t-\tau)_+}^{t-1} I_t(u)\nabla_s f\left(\hat{x}_u\right)\right\|^2 \leq \tau\sum_{u=(t-\tau)_+}^{t-1} I_t(u)\|\nabla_s f\left(\hat{x}_u\right)\|^2 \tag{47}$$

Proof. The result directly follows from (44). □

Lemma 4. *Let $f:\mathbb{R}^d\to\mathbb{R}$ be a convex functions with a L-Lipschitz continuous gradient. For any $x\in\mathbb{R}^d$ and $y\in\mathbb{R}^d$ the following inequality holds:*

$$f(y) \geq f(x) + \langle \nabla f(x), y-x\rangle + \frac{1}{2L}\|\nabla f(x) - \nabla f(y)\|^2 \tag{48}$$

Proof. First of all, we introduce the function

$$\Phi(y) = f(y) - \langle \nabla f(x), y\rangle \tag{49}$$

This function can be shown to have the following properties:

- $\Phi(y)$ is convex
- $\nabla\Phi(y) = \nabla f(y) - \nabla f(x)$

- $\min_{y \in \mathbb{R}^d} \Phi(y) = f(x) - \langle \nabla f(x), x \rangle$

The convexity is very easy to check, as $\Phi(y)$ is the sum of a convex and a linear (thus convex) function. The expression for the gradient can be obtained by observing that $\langle \nabla f(x), y \rangle$ defines a linear function of y. Given the differentiability and convexity of Φ, the first order optimality condition is both necessary and sufficient. It is clearly satisfied for $y = x$, which gives the expression for the minimum of Φ.
Additionally, one should remember that a characterization of the L-Lipschitz continuity of the gradient is provided by the following relation:

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2 \tag{50}$$

Now we have all the ingredients that are necessary to prove the result. We proceed as follows:

$$\begin{aligned} f(x) - \langle \nabla f(x), x \rangle &= \min_{y \in \mathbb{R}^d} \Phi(y) \leq \Phi\left(y - \frac{1}{L} \nabla \Phi(y)\right) \\ &\leq \Phi(y) - \langle \nabla \Phi(y), \frac{1}{L} \nabla \Phi(y) \rangle + \frac{L}{2} \left\| \frac{\nabla \Phi(y)}{L} \right\|^2 \\ &= \Phi(y) - \frac{1}{2L} \|\nabla \Phi(y)\|^2 \\ &= f(y) - \langle \nabla f(x), y \rangle - \frac{1}{2L} \|\nabla f(y) - \nabla f(x)\|^2 \end{aligned} \tag{51}$$

Simply rearranging the terms we obtain

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{1}{2L} \|\nabla f(x) - \nabla f(y)\|^2 \tag{52}$$

which proves the claim. □

Lemma 5. *The following inequality holds for a convex function with a Lipschitz continuous gradient:*

$$\mathbb{E}\left[\|\nabla_s f(\hat{x}_t)\|^2\right] \leq 2Le_u \tag{53}$$

Proof. We start from lemma 4. If we choose $y = \hat{x}_u$ and $x = x^*$ and we use the fact that at the minimum point, under the assumption of smooth objective function, the first derivative is 0, we can write:

$$\mathbb{E}\left[\|\nabla_s f(\hat{x}_u)\|^2\right] = \mathbb{E}\left[\|\nabla_s f(\hat{x}_u) - \nabla_s f(x^*)\|^2\right] \leq 2L \left(\mathbb{E}\left[f(\hat{x}_u)\right] - f(x^*)\right) = 2Le_u \tag{54}$$

□

Lemma 6. *If each update previous to the one labeled as t, up to the update $t - \tau$, has a constant probability p to interfere with the reading process, then*

$$\mathbb{E}\left[\sum_{u=(t-\tau)_+}^{t-1} I_t(u) \|\nabla_s f(\hat{x}_u)\|^2\right] \leq 2pL \sum_{u=(t-\tau)_+}^{t-1} e_u \tag{55}$$

Proof. This result is a simple generalization of Lemma 5. We first need to observe that we do not expect the duration of the writing process to change based on which index was sampled, and neither on which is the current solution that is being written. Under these assumptions, the expectation of the indicator function $I_t(u)$ is the constant probability p. Therefore:

$$\begin{aligned} \mathbb{E}\left[\sum_{u=(t-\tau)_+}^{t-1} I_t(u) \|f'(\hat{x}_u)\|^2\right] &= \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\left[I_t(u) \|f'(\hat{x}_u)\|^2\right] = \sum_{u=(t-\tau)_+}^{t-1} p_t(u) \mathbb{E}\left[\|f'(\hat{x}_u)\|^2\right] \\ &= p \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\left[\|f'(\hat{x}_u) - f'(x^*)\|^2\right] \leq 2pL \sum_{u=(t-\tau)_+}^{t-1} e_u \end{aligned} \tag{56}$$

□

Lemma 7. *Let $\mathcal{L}_t$ defined as*

$$\mathcal{L}_t = \sum_{u=0}^{t}(1-\rho)^{t-u}a_u \qquad 0<\rho<1 \tag{57}$$

with a_t satisfying the relation

$$a_{t+1} \leq \left(1-\frac{\gamma\mu}{1+\gamma\mu}\right)a_t + 4pL\gamma^2\tau \sum_{u=(t-\tau)_+}^{t-1} e_u + (4\gamma^2 L - 2\gamma)e_t. \tag{58}$$

Then

$$\mathcal{L}_{t+1} \leq (1-\rho)^{t+1}a_0 + \left(1-\frac{\gamma\mu}{1+\gamma\mu}\right)\mathcal{L}_t + \left(4\gamma^2 L\frac{p\tau^2}{1-\rho\tau} + 4\gamma^2 L - 2\gamma\right)\sum_{u=0}^{t}(1-\rho)^{t-u}e_u \tag{59}$$

Proof. Due to (58) we can write:

$$\begin{aligned}
\mathcal{L}_{t+1} &= \sum_{u=0}^{t+1}(1-\rho)^{t+1-u}a_u = (1-\rho)^{t+1}a_0 + \sum_{v=0}^{t}(1-\rho)^{t-v}a_{v+1} \\
&\leq (1-\rho)^{t+1}a_0 + \sum_{v=0}^{t}(1-\rho)^{t-v}\left[\left(1-\frac{\gamma\mu}{1+\gamma\mu}\right)a_v + 4pL\gamma^2\tau\sum_{u=(v-\tau)_+}^{v-1}e_u + (4\gamma^2L-2\gamma)e_v\right] \\
&= (1-\rho)^{t+1}a_0 + \left(1-\frac{\gamma\mu}{1+\gamma\mu}\right)\mathcal{L}_t + 4pL\gamma^2\tau\sum_{v=0}^{t}\sum_{u=(v-\tau)_+}^{v-1}(1-\rho)^{t-v}e_u + \\
&+ (4\gamma^2L-2\gamma)\sum_{v=0}^{t}(1-\rho)^{t-v}e_v
\end{aligned} \tag{60}$$

The next step is to rearrange the terms of the double sum so that they can be grouped with the negative sub-optimality term.

$$\sum_{v=0}^{t}\sum_{u=(v-\tau)_+}^{v-1}(1-\rho)^{t-v}e_u = \sum_{u=0}^{t}\sum_{v=u+1}^{\min(u+\tau,t)}(1-\rho)^{t-v}e_u \leq \frac{\tau}{(1-\rho)^\tau}\sum_{u=0}^{t}(1-\rho)^{t-u}e_u \tag{61}$$

The last inequality is obtained by substituting of the index v in the internal summation with $u+\tau$ and multiplying everything by τ. This is fine because the terms of the sum are increasing and they are at most τ, so their sum is lesser or equal than τ times the last one.
Thanks to Bernoulli's inequality:

$$(1-\rho)^\tau \geq 1-\rho\tau \qquad (\rho<1) \tag{62}$$

which implies

$$\frac{1}{(1-\rho)^\tau} \leq \frac{1}{1-\rho\tau} \tag{63}$$

we are able to regroup the terms to get:

$$\mathcal{L}_{t+1} \leq (1-\rho)^{t+1}a_0 + \left(1-\frac{\gamma\mu}{1+\gamma\mu}\right)\mathcal{L}_t + \left(4\gamma^2 L\frac{p\tau^2}{1-\rho\tau} + 4\gamma^2 L - 2\gamma\right)\sum_{u=0}^{t}(1-\rho)^{t-u}e_u \tag{64}$$

□

Lemma 8. *The only solution ρ of the quadratic equation*

$$\tau(2L+\mu)\rho^2 - (2L+\mu+2p\tau^2L+\tau\mu)\rho + \mu = 0 \tag{65}$$

such that $0<\rho<1$ is bound by

$$\rho \geq \frac{\mu}{2L+\mu+2p\tau^2L+\mu\tau} = \frac{\mu}{2L(1+p\tau^2)+\mu(1+\tau)} \tag{66}$$

Proof. The well known rule to find the two roots gives the following result:

$$\rho = \frac{1}{2\tau(2L+\mu)}\left(2L+\mu+2p\tau^2L+\tau\mu \pm \sqrt{(2L+\mu+2p\tau^2L+\tau\mu)^2-4\tau\mu(2L+\mu)}\right) \tag{67}$$

Some observations:

- The term under the square root is always positive. It can be regrouped as follows:

$$(2L+\mu-\tau\mu)^2+(2p\tau^2L)^2+4p\tau^2L(2L+\mu+\tau\mu) \tag{68}$$

- The root corresponding to the sign $(-)$ has to be chosen, because for a very wide range of parameters the one corresponding to the sign $(+)$ is larger than 1.
- It can be shown that the solution corresponding to the sign $(-)$ respects the bounds on ρ

We now have a convergence certificate, but its expression is quite complicated and not easy to interpret. However, using again Bernoulli's inequality, we have:

$$\begin{aligned}
&\sqrt{(2L+\mu+2p\tau^2L+\tau\mu)^2-4\tau\mu(2L+\mu)} \\
&= (2L+\mu+2p\tau^2L+\tau\mu)\left[1-\frac{4\tau\mu(2L+\mu)}{2L+\mu+2p\tau^2L+\tau\mu)^2}\right]^{\frac{1}{2}} \\
&\leq (2L+\mu+2p\tau^2L+\tau\mu)\left[1-\frac{1}{2}\frac{4\tau\mu(2L+\mu)}{2L+\mu+2p\tau^2L+\tau\mu)^2}\right] \\
&= (2L+\mu+2p\tau^2L+\tau\mu)-\frac{1}{2}\frac{4\tau\mu(2L+\mu)}{2L+\mu+2p\tau^2L+\tau\mu)}
\end{aligned} \tag{69}$$

Now we use this result in (67) to find a weaker but simpler convergence rate certificate:

$$\rho \geq \frac{\mu}{2L+\mu+2p\tau^2L+\mu\tau} = \frac{\mu}{2L(1+p\tau^2)+\mu(1+\tau)} \tag{70}$$

□

4 Numerical experiments

The objective of the previous section was to give a convergence certificate for SGD that also applies to those problems, such as linear or logistic regression, in which even considering just one of the terms of the cost function to compute an update produces a full gradient vector. This means that this kind of algorithms are the ones that should be most sensitive to inconsistent readings or delayed writings. To see these effects in action, we wrote some code that simulates what could go wrong in a non-locking algorithm. It is obviously a very difficult task to simulate perfectly a parallel code in a sequential environment, but we think these results can provide an idea of what is really harmful to performance and what is not.

4.1 Multiple processors simulation

Before presenting the results obtained with our simulations, here we describe by means of a pseudo code the algorithms we have used to model the asynchronicity in a sequential framework. This way the reader can built himself an idea about which are the aspects of the problem that have been modeled and the ones that have been neglected, also understanding to which extent our simulations reproduce what could happen in a real application scenario.
First of all, we start from the synchronous algorithm that only models a multiprocessor execution of the stochastic gradient descent. This sample algorithm can be applied to all the problems that present a cost function with the special sum structure previously described (see, for instance, Theorem 1). A real problem that can be taken as a reference is, for instance, the optimization of the cost function associated to a logistic regression.

4.1.1 Synchronous multiprocessor stochastic gradient descent

With this first implementation we address the problem of modeling the behavior of a multiprocessor architecture with a sequential code, when the locking ensures the correctness of the algorithm. The only difference with a purely sequential implementation of a stochastic gradient descent is that all the cores compute an update at the same time, so their updates are all computed on the same partial result. In practice, such an implementation behaves in the same way as a sequential code using a batch size equal to the number of cores (or, if each core is already considering a batch size larger than 1, let's say B, this synchronous algorithm acts like the batch size was $P \times B$, P being the number of processors). The reason behind this equivalence is that the sum structure of the objective function also holds for the gradient.
Each iteration of the algorithm consists of two steps: firstly, an inner cycle of the same size as the number of processors computes all the updates, based on the same partial solution, and stores them in a matrix; after the completion of this step, the partial solution is updated with all these computed gradients. The step size γ can be either constant or adaptive. However, our theoretical analysis, that we want to validate, is based on a constant step size.

```
Data: Data matrix 'tx', labels 'y', step size 'γ'
Result: weights 'w', cost
Initialize 'w', 'storedGradients', 'iter', 'proc' ;
while iter < maxIter do
    if proc < numProc then
        sample y and tx at random;
        grad = computeGradient(sampledTx, sampledY, w) ;
        update storedGradients;
        proc = proc +1 ;
    else
        for grad in storedGradients do
            w = γ grad|;
            loss = computeLoss(y, tx, w);
            iter = iter +1;
        end
    end
end
```

Algorithm 1: Locking multiprocessor SGD

It should be noticed that, in order to apply the previous scheme to different problems, it is enough to change the definition of the two functions "computeGradient" and "computeLoss".

4.1.2 Asynchronous models

These two following implementations are different efforts to model the asynchronicity of a parallel environment in a sequential one. The first we are going to present is based on the observation that, if no locking is imposed, the updates can be performed in an order that is different from the one of the reading process. This happens when the computation of an update takes longer than the one of another. It is a quite common circumstance if the cost function is composed by terms that make use of data of different natures, that can potentially have different sizes. It is important for the aim of this work to model such a possibility, because one of the strengths of the "After Read" approach in defining a global ordering is that of giving an estimate that does not rely on homogeneous computation time for the updates.
The implementation of the concepts we have just introduced has been made by storing a list of the historical partial solutions in a matrix. At the time of computing the update of the solution, one of them is chosen at random, so that it is impossible to foresee the moment when the update computed with a certain partial solution will be used. The number of solutions that are stored has a similar meaning to the τ variable of Theorem 2, that was a limit to how old an update that might interfere with the reading process could be. Here is a pseudo code of the said algorithm.

```
Data: Data matrix 'tx', labels 'y', step size 'γ'
Result: weights 'w', cost
Initialize 'historicalW', 'storedGradients', 'iter', 'proc' ;
while iter < maxIter do
    if proc < numProc then
        sample y and tx at random;
        sample w in historicalW at random;
        grad = computeGradient(sampledTx, sampledY, w) ;
        update storedGradients;
        proc = proc +1 ;
    else
        for grad in storedGradients do
            w = γ grad|;
            loss = computeLoss(y, tx, w);
            iter = iter +1;
        end
    end
end
```

Algorithm 2: Random update multiprocessor SGD

Although this first algorithm grasps a part of the problem, it completely neglects another aspect. In fact, if we are allowing all the processors to freely access the shared memory at will, we also have to consider the fact that more than one processor could access it at the same time. This is especially true when the number of processors is large, which is the situation we are most interested in. Such a circumstance can generate what we have named in Theorem 2 as inconsistent reads: for instance, while a processor is reading the solution that is present in the shared memory at that moment, some other processors could be computing an update. This means that not only could the update of the solution have been computed on a partial solution relatively old, but also that some of its coordinates came from different partial solutions. This inconsistency we are talking about has been translated in our theorems into the following relation:

$$\hat{x}_t - x_t = \gamma \sum_{u=(t-\tau)_+}^{t-1} I_t(u) g\left(\hat{x}_u\right) \tag{71}$$

In order to model this event in our code, a modification to the previous code has been made: instead of sampling one whole solution of the historical record, at each update computation the partial solution needed for the computation is created sampling each coordinate at random among the corresponding coordinates of the recorded solutions. This means that each coordinate could potentially come from a different partial solution. One might think that such a model is a little too extreme, because is should be a relatively rare that so many processors update the solution during a single reading process. This observation is true, but what we would argue is that the algorithm still performs quite well, especially if the step size is not too large, which should be a strong guarantee that, under certain conditions, asynchronicity has a smaller impact on the performance per iteration than one could expect.

```
Data: Data matrix 'tx', labels 'y', step size 'γ'
Result: weights 'w', cost
Initialize 'historicalW', 'storedGradients', 'iter', 'proc' ;
while iter < maxIter do
    if proc < numProc then
        sample y and tx at random;
        initialize w;
        for coordinate in w do
            sample coordinate in historicalW at random;
        end
        grad = computeGradient(sampledTx, sampledY, w) ;
        update storedGradients;
        proc = proc +1 ;
    else
        for grad in storedGradients do
            w = γ grad|;
            loss = computeLoss(y, tx, w);
            iter = iter +1;
        end
    end
end
```

Algorithm 3: Random update multiprocessor SGD

4.2 Experiments with a simple model

In order to further test the convergence estimate that was obtained in Theorem 2, we now consider a very simple optimization problem, for which it is possible to obtain the parameters we need. It is a quadratic optimization problem. It can be stated as follows:

$$\min_{x\in\mathbb{R}^d} \quad f(x) = \frac{1}{2}x^T Ax - b^T x \tag{72}$$

where $A \in M^{d\times d}$ is a symmetric and positive definite matrix and $b \in \mathbb{R}^d$ is a vector.
This problem is a very good sample case, because it has all the properties required by Theorem 2 to be applied and al the quantities of interest can be easily computed.

Strong convexity

f(x) is of class C^1, so the following characterization of strong convexity is equivalent to the definition:

$$\forall x,\, y \in \mathbb{R}^d \quad \langle \nabla f(x) - \nabla f(y), x - y\rangle \geq \mu \left\|x - y\right\|^2 \tag{73}$$

Given the very special expression of f we have that

$$\nabla f(x) = Ax - b \tag{74}$$

The symmetry of A is required for the last relation to be true. This means that the characterization of strong convexity applied to f leads to the following expression:

$$(x - y)^T A(x - y) \geq \mu \left\|x - y\right\|^2 \tag{75}$$

Given that A is positive definite, the inequality is satisfied if $\mu = \lambda_{min}$, where λ_{min} is the smallest eigenvalue of A. Also, μ cannot be larger than λ_{min}, because equality holds if $x-y$ is an eigenvector relative to λ_{min}.

Lipschitz continuous gradient

The gradient of f is linear, which obviously makes it Lipschitz continuous. To find out the continuity constant, we use again a characterization of this property:

$$\langle \nabla f(x) - \nabla f(y), x - y\rangle \geq \frac{1}{L} \left\|\nabla f(x) - \nabla f(y)\right\|^2 \tag{76}$$

We use again the special expression of the considered f in order to get:

$$\begin{aligned}(x-y)^T A(x-y) &\geq \frac{1}{L}\left\|Ax-Ay\right\|^2 \\ &= \frac{1}{L}(x-y)^T A^T A(x-y) \\ &= \frac{1}{L}(x-y)^T A^2(x-y)\end{aligned} \tag{77}$$

Now we use the property of symmetric matrices of being diagonalizable through orthogonal matrices to express A^2 as $Q^T D^2 Q$, where D^2 is given by the elementwise square of the diagonal of D. The previous expression can therefore be rewritten as

$$(x-y)^T Q^T D\, Q\,(x-y) \geq \frac{1}{L}(x-y)^T Q^T D^2\, Q\,(x-y) \tag{78}$$

Introducing a vector $z := Q\,(x-y)$ the inequality becomes

$$z^T\left(D - \frac{1}{L}D^2\right) z \geq 0 \tag{79}$$

This is true for any $z \in \mathbb{R}^d$ if and only if $D - (1/L)D^2$ is positive semidefinite. The condition for this to be true is that each of its eigenvalues is positive. Namely:

$$\lambda_i - \frac{1}{L}\lambda_i^2 \geq 0 \quad \forall i = 1...d \tag{80}$$

We already know that $\lambda_i > 0$ because A is positive definite. We only have to impose that $\lambda_i \leq L\,\forall i$, which is satisfied if $L = \lambda_{max}$.

Experiments

In order to gain control over the constants, we have obtained a fixed dimension orthogonal matrix with the QR decomposition of a randomly generated matrix. Then we have obtained a symmetric positive definite matrix with given eigenvalues through the product $Q\,D\,Q^T$, where D is a diagonal matrix with predefined diagonal elements. Then we have solved the optimization problem previously analyzed, where λ_{max} and λ_{min} are L and μ, respectively.
The simulation have been run with the previously described sequential code that models the effects of inconsistent reads in a parallel framework. As for the considered problem the update is also sparse, the probability of interference p has been set to $1/d$, with d the length of the gradient vector. According to Theorem 2, the step size has been set to

$$\gamma \leq \frac{1}{2L}\left(\frac{p\tau^2}{1-\rho\tau} + 1\right)^{-1} \tag{81}$$

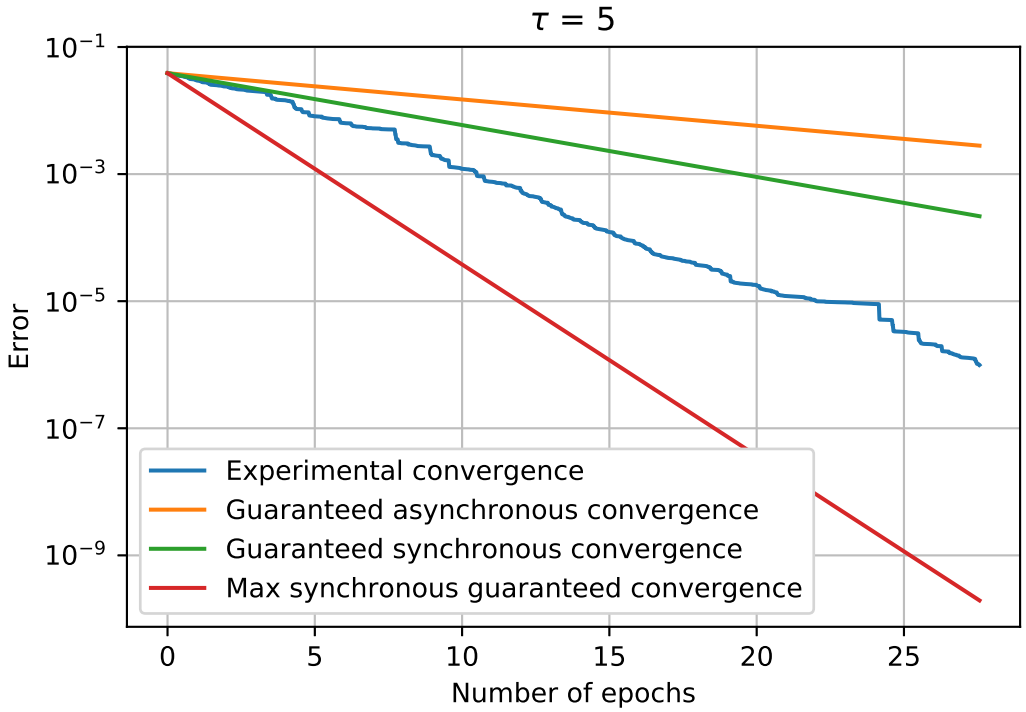
For the given γ, the theorem also provides a lower bound for the expected convergence rate:

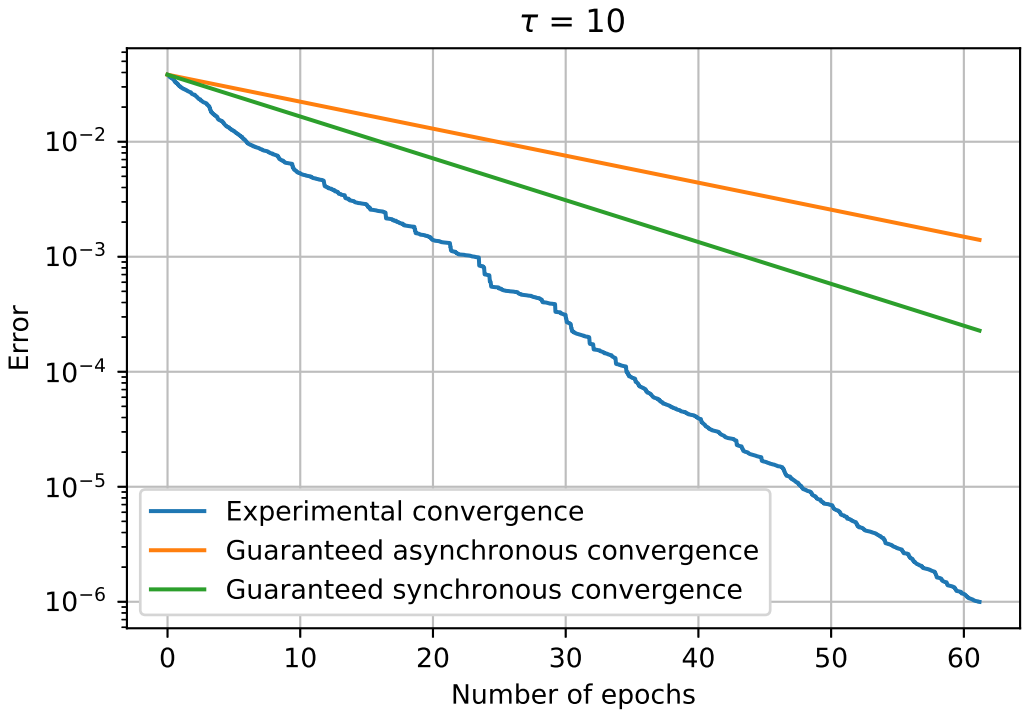
$$\rho = \frac{\mu}{2L(1+p\tau^2) + \mu(1+\tau)} \tag{82}$$

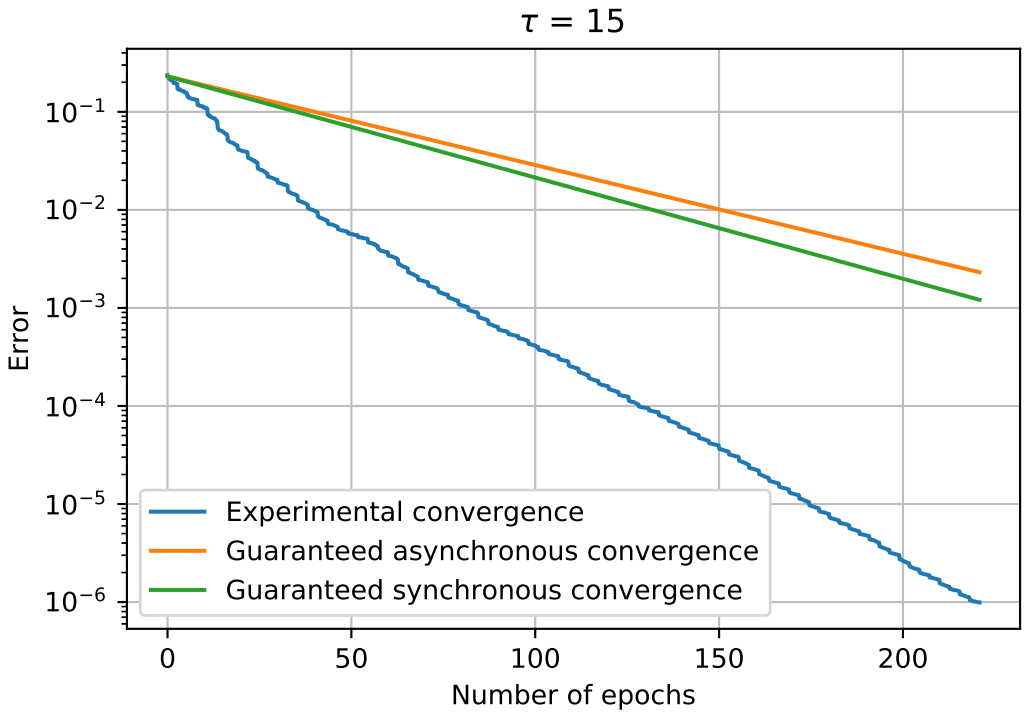
The following plots show a comparison between the convergence rate obtained in the experiment and the theoretical lower bound. The parameter τ, which states how many old update can corrupt the current reading process, has been progressively increased. In all the figures we have also plotted the theoretical lower bound obtained in Theorem 1 for the synchronous algorithm, which is

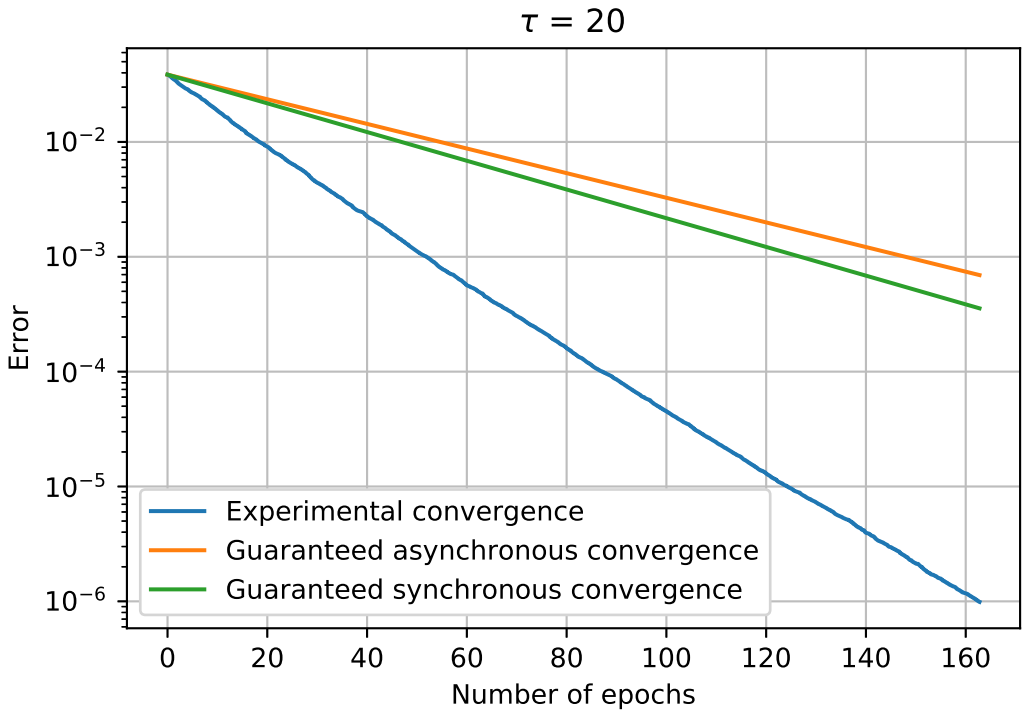
$$\rho = \gamma\mu \tag{83}$$

Only in the first figure we have also plotted the convergence rate that we would expect from a synchronous algorithm if a step size equal to $1/L$ was chosen. For the other figures we have decided to omit this graph, because the step size that guarantees a convergence for the given value of τ is too much smaller than $1/L$ and a comparison does not seem sensible.


$\tau = 5$
Error
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
0
5
10
15
20
25
Number of epochs
Experimental convergence
Guaranteed asynchronous convergence
Guaranteed synchronous convergence
Max synchronous guaranteed convergence


$\tau = 10$
Error
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
10
20
30
40
50
60
Number of epochs
Experimental convergence
Guaranteed asynchronous convergence
Guaranteed synchronous convergence


$\tau = 15$
Error
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
50
100
150
200
Number of epochs
Experimental convergence
Guaranteed asynchronous convergence
Guaranteed synchronous convergence


$\tau = 20$
Error
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
20
40
60
80
100
120
140
160
Number of epochs
Experimental convergence
Guaranteed asynchronous convergence
Guaranteed synchronous convergence

The algorithm we have chosen to test the theoretical results is the one described in the pseudo code 3, as it seems the most conservative of the three. Although it seemed to overestimate the side effects of asynchronicity, the theoretical convergence rate is respected: it actually even outperforms the guaranteed synchronous convergence rate. The reason behind this better efficiency can be that the convergence estimate we have obtained is a little loose and that the results we actually obtain outperform the predicted behavior. This could be especially true for the considered example, in which the problem is ideal under any point of view. The matrix is very well conditioned, the Lipschitz constant L and the strong convexity constant μ have the same order of magnitude (they have been fixed to 2 and 1, respectively). These desired properties are not so likely to be found in practice, but it is still good news that the estimate works for a very basic problem. However, in order to have a stronger validation of the theoretical result, we think that it needs to be checked on a real size problem, with all the connected issues.
Furthermore, although we have tried to be conservative as far as the asynchronicity is concerned, it is difficult to say if there is nothing else that could go wrong when running a gradient descent in a real parallel processor. The loss of control on the inconsistency errors could also lead to a worse convergence rate in practice.
Another observation is that the algorithm outperforms the theoretical estimates more and more as τ gets larger. This may also suggest that maybe the way we have chosen to model this parameter in our code does not match perfectly its abstraction in the theorem. Moreover, in a real application, we are not sure how to estimate this parameter: we think that the number of processors can be a reasonable upper bound to it, but it probably overestimates τ too much, leading to the choice of a too small γ. The algorithm would still converge, but at a slower rate.
Finally, we have the confirmation that the step size is small enough to make the algorithm converge. Again, some additional experiments showed that it was possible to choose a quite larger step size without making the algorithm diverge. Again, this can explained with the fact that the problem is very easy and almost everything seems to work with it. What we suggest is, however, to adopt an even elementary algorithm to adjust the step size dynamically, as it is desirable to try to keep the step size as large as possible. The fact that a small enough step size makes the algorithm converge is still a good guarantee.

4.3 Experiments with a slightly more complex model

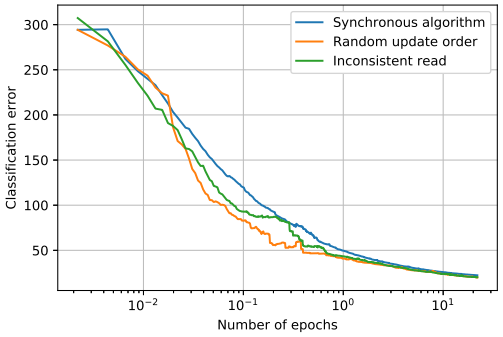
In this last section we consider a problem that involves the solution of a classification problem with logistic regression. The function we want to minimize has the following form:

$$f(w) = -\sum_{n=1}^{N} \left[y_n \log(\sigma(x_n^T w)) + (1 - y_n) \log(1 - \sigma(x_n^T w)) \right] \tag{84}$$

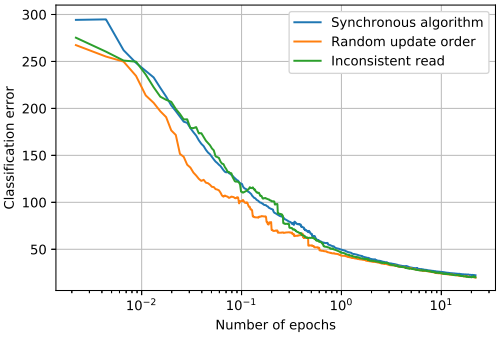
In equation 84 x_n denotes the data relative to the observed outcome y_n. The variable y_n can assume either value 0 or 1, and $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function, while w is the weight vector. We want to find the vector w that minimizes $f(w)$, so that the misclassification error is as small as possible. As already discussed in the introduction, the stochastic gradient can be expressed as

$$\nabla_s f(w) = (\sigma(x_n^T w) - y_n)\, x_n \tag{85}$$

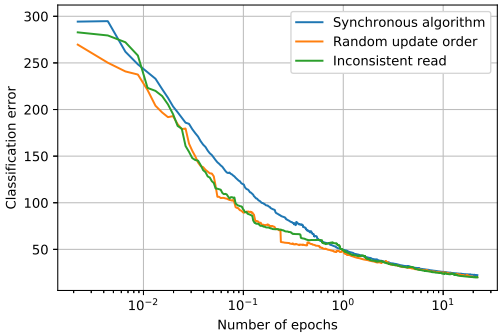
We want to compare the performance of the synchronous algorithm with that of the two asnynchronous models. As we are only simulating a parallel environment, it is impossible to draw conclusions about the running time of the different algorithms on a real parallel cluster. However, we can compare the performance per iteration, namely how large is the difference in convergence rate between the synchronous and the asynchronous versions. We expect that if the convergence rate is not too different, the benefit from running the algorithm asynchronously can be significant. The sample dataset that has been used to test the regression algorithm is known as "Breast Cancer Wisconsin" [Lic13] (available here: https://archive.ics.uci.edu/ml/datasets).



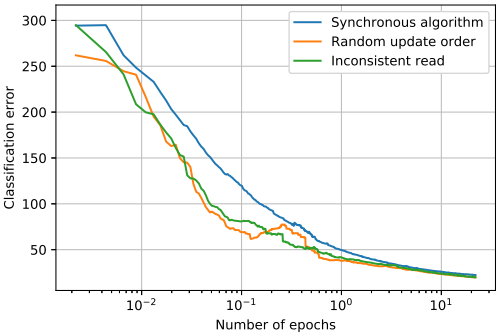
(a) $\tau = 10$.



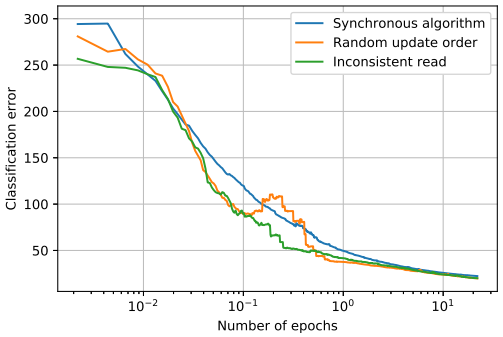
(b) $\tau = 20$.



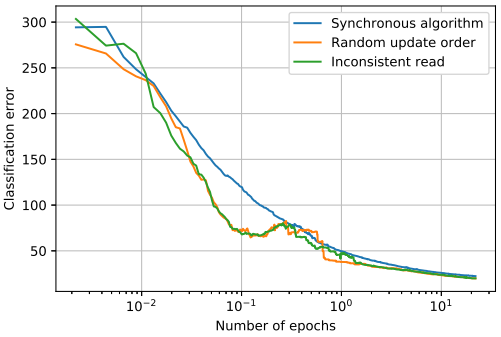
(c) $\tau = 30$



(d) $\tau = 40$



(e) $\tau = 50$



(f) $\tau = 60$

Figure 1: results obtained with the sinchronous and the two asynchronous models for different values of τ

As we can observe in 1, form our simulation it is not possible to observe a clear difference in efficiency per iteration when an asynchronous algorithm is used instead of a synchronous one, if they use the same step size. What is not expected is that the convergence rate is almost the same even increasing the parameter τ, which is the indicator of how much inconsistency is caused by the asynchronicity. It is probably a limit of our simulation, because we expect it to have a certain impact on the performance.
As a final remark, we want to underline the fact that the strongest limitation to the convergence rate that asynchronicity introduces is the upper bound on the step size. If the simulation is very large and the updates are not sparse, this can be a strong limit compared to a synchronous

simulation. Keeping the step size as large as possible while still achieving convergence is one aspect that should not be neglected when implementing an asynchronous algorithm.

5 Conclusions and further development

The main achievement of this work is to give a proof for the convergence rate of SGD, one of the most basic optimization algorithms, in an asynchronous framework, without any sparsity assumption. We hope that the outlined procedure can be applied to more specific algorithms in order to achieve similar convergence results.
Furthermore, another research topic could be to modify the model for asynchronicity and to try to still find a convergence estimate: for instance, we considered that all the updates previous to the considered one had the same probability to interfere with the reading process. However, we expect that the further two updates are, the smaller is the probability they interfere with each other. If a sufficiently fast decreasing probability distribution is chosen, it may be even possible to prove similar results while getting rid of the parameter τ. As it represents an abstract concept, it is hard to estimate in practice.
Finally, one aspect of the convergence result of Theorem (2) that could be improved is that the synchronous convergence rate is not recovered for $\tau = 0$. In fact, $\rho = \mu/(2L + \mu)$ is a weaker result than $\rho = \mu/L$. The reason behind this difference is that, in (24), we choose $\alpha = \gamma\mu$, but, in order to retrieve the synchronous estimate, the parameter α must be set to 0 when $\tau = 0$ (if $\tau \neq 0$ it is not possible to do so, because α appears as the denominator of a fraction). An additional free constant could be used to make up for this issue.

Acknowledgement

I would like to thank Dr. Stich for the extensive help he gave me during our meetings and Prof. Jaggi for assigning me this topic to work on, and also for putting me in touch with Prof. Lacoste-Julien, who was able to give me the right hints to conclude the proof.

References

[BT93] D. P. Bertsekas and J. N. Tsitsiklis. Parallel and Distributed Computation: Numerical Methods. *TUGBoat*, 14(3):342–351, 1993.

[FNW11] Christopher Ré Feng Niu, Benjamin Recht and Stephen J. Wright. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. 2011.

[Lic13] M. Lichman. UCI machine learning repository, 2013.

[RLLJ16] F. Pedregosa R. Leblond and S. Lacoste-Julien. ASAGA: Asynchronous parallel SAGA. 2016.