Asaga: Asynchronous Parallel Saga

Rémi Leblond
INRIA - Sierra Project-team
École normale supérieure, Paris

Fabian Pedregosa
INRIA - Sierra Project-team
École normale supérieure, Paris

Simon Lacoste-Julien
Department of CS & OR (DIRO)
Université de Montréal, Montréal

Abstract

We describe Asaga, an asynchronous parallel version of the incremental gradient algorithm Saga that enjoys fast linear convergence rates. Through a novel perspective, we revisit and clarify a subtle but important technical issue present in a large fraction of the recent convergence rate proofs for asynchronous parallel optimization algorithms, and propose a simplification of the recently introduced "perturbed iterate" framework that resolves it. We thereby prove that Asaga can obtain a theoretical linear speedup on multi-core systems even without sparsity assumptions. We present results of an implementation on a 40-core architecture illustrating the practical speedup as well as the hardware overhead.

1 Introduction

We consider the unconstrained optimization problem of minimizing a *finite sum* of smooth convex functions:

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

where each f_i is assumed to be convex with L-Lipschitz continuous gradient, f is μ-strongly convex and n is large (for example, the number of data points in a regularized empirical risk minimization setting). We define a condition number for this problem as $\kappa := {}^{L}/_{\mu}$. A flurry of randomized incremental algorithms (which at each iteration select i at random and process only one gradient f_i') have recently been proposed to solve (1) with a fast[1] linear convergence rate, such as Sag (Le Roux et al., 2012), Sdca (Shalev-Shwartz and Zhang, 2013), Svrg (Johnson and Zhang, 2013) and Saga (Defazio et al., 2014). These algorithms can be interpreted as variance reduced versions of the stochastic gradient descent (Sgd) algorithm, and they have demonstrated both theoretical and practical improvements over Sgd (for the *finite sum* optimization problem (1)).

In order to take advantage of the multi-core architecture of modern computers, the aforementioned optimization algorithms need to be adapted to the asynchronous parallel setting, where multiple threads work concurrently. Much work has been devoted recently in proposing and analyzing asynchronous parallel variants of algorithms such as Sgd (Niu et al., 2011), Sdca (Hsieh et al., 2015) and Svrg (Reddi et al., 2015; Mania et al., 2015; Zhao and Li, 2016). Among the incremental gradient algorithms with fast linear convergence rates that can optimize (1) in its general form, only Svrg has had an asynchronous parallel version proposed.[2] No such adaptation has been attempted yet for Saga, even though one could argue that it is a more natural candidate as, contrarily to Svrg, it is not epoch-based and thus has no synchronization barriers at all.

Contributions. In Section 2, we present a novel sparse variant of Saga that is more adapted to the parallel setting than the original Saga algorithm. In Section 3, we present Asaga, a lock-free asynchronous parallel version of Sparse Saga that does not require consistent reads. We propose a simplification of the "perturbed iterate" framework from Mania et al. (2015) as a basis for our convergence analysis. At the same time, through a novel perspective, we revisit and clarify a technical problem present in a large fraction of the literature on randomized asynchronous parallel algorithms (with the exception of Mania et al. (2015), which also highlights this issue): namely, they all assume unbiased gradient estimates, an assumption that is inconsistent with their proof technique without fur-

[1] Their complexity in terms of gradient evaluations to reach an accuracy of ϵ is $O((n+\kappa)\log({}^{1}/_{\epsilon}))$, in contrast to $O(n\kappa\log({}^{1}/_{\epsilon}))$ for batch gradient descent in the worst case.

Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54. Copyright 2017 by the author(s).

[2] We note that Sdca requires the knowledge of an explicit μ-strongly convex regularizer in (1), whereas Sag / Saga are adaptive to any local strong convexity of f (Schmidt et al., 2016; Defazio et al., 2014). This is also true for a variant of Svrg (Hofmann et al., 2015).

ther synchronization assumptions. In Section 3.3, we present a tailored convergence analysis for ASAGA. Our main result states that ASAGA obtains the same geometric convergence rate per update as SAGA when the overlap bound τ (which scales with the number of cores) satisfies $\tau \leq \mathcal{O}(n)$ and $\tau \leq \mathcal{O}(\frac{1}{\sqrt{\Delta}} \max\{1, \frac{n}{\kappa}\})$, where $\Delta \leq 1$ is a measure of the sparsity of the problem, notably implying that a linear speedup is theoretically possible even without sparsity in the well-conditioned regime where $n \gg \kappa$. In Section 4, we provide a practical implementation of ASAGA and illustrate its performance on a 40-core architecture, showing improvements compared to asynchronous variants of SVRG and SGD.

Related Work. The seminal textbook of Bertsekas and Tsitsiklis (1989) provides most of the foundational work for parallel and distributed optimization algorithms. An asynchronous variant of SGD with constant step size called HOGWILD was presented by Niu et al. (2011); part of their framework of analysis was re-used and inspired most of the recent literature on asynchronous parallel optimization algorithms with convergence rates, including asynchronous variants of coordinate descent (Liu et al., 2015), SDCA (Hsieh et al., 2015), SGD for non-convex problems (De Sa et al., 2015; Lian et al., 2015), SGD for stochastic optimization (Duchi et al., 2015) and SVRG (Reddi et al., 2015; Zhao and Li, 2016). These papers make use of an unbiased gradient assumption that is not consistent with the proof technique, and thus suffers from technical problems[3] that we highlight in Section 3.2.

The "perturbed iterate" framework presented in Mania et al. (2015) is to the best of our knowledge the only one that does not suffer from this problem, and our convergence analysis builds heavily from their approach, while simplifying it. In particular, the authors assumed that f was both strongly convex and had a bound on the gradient, two *inconsistent* assumptions in the unconstrained setting that they analyzed. We overcome these difficulties by using tighter inequalities that remove the requirement of a bound on the gradient. We also propose a more convenient way to label the iterates (see Section 3.2). The sparse version of SAGA that we propose is also inspired from the sparse version of SVRG proposed by Mania et al. (2015). Reddi et al. (2015) presents a hybrid algorithm called HSAG that includes SAGA and SVRG as special cases. Their asynchronous analysis is epoch-based though, and thus does not handle a fully asynchronous version of SAGA as we do. Moreover, they require consistent reads and do not propose an efficient sparse implementation for SAGA, in contrast to ASAGA.

[3]Except Duchi et al. (2015) that can be easily fixed by incrementing their global counter *before* sampling.

Notation. We denote by $\mathbb{E}$ a full expectation with respect to all the randomness, and by $\mathbf{E}$ the *conditional* expectation of a random i (the index of the factor f_i chosen in SGD-like algorithms), conditioned on all the past, where "past" will be clear from the context. $[x]_v$ is the coordinate v of the vector $x \in \mathbb{R}^d$. x^+ represents the updated parameter vector after one algorithm iteration.

2 Sparse SAGA

Borrowing our notation from Hofmann et al. (2015), we first present the original SAGA algorithm and then describe a novel sparse variant that is more appropriate for a parallel implementation.

Original SAGA Algorithm. The standard SAGA algorithm (Defazio et al., 2014) maintains two moving quantities to optimize (1): the current iterate x and a table (memory) of historical gradients $(\alpha_i)_{i=1}^n$.[4] At every iteration, the SAGA algorithm samples uniformly at random an index $i \in \{1, \ldots, n\}$, and then executes the following update on x and α (for the unconstrained optimization version):

$$x^+ = x - \gamma\big(f_i'(x) - \alpha_i + \bar{\alpha}\big); \qquad \alpha_i^+ = f_i'(x), \tag{2}$$

where γ is the step size and $\bar{\alpha} := {}^1\!/_n \sum_{i=1}^n \alpha_i$ can be updated efficiently in an online fashion. Crucially, $\mathbf{E}\alpha_i = \bar{\alpha}$ and thus the update direction is unbiased ($\mathbf{E}x^+ = x - \gamma f'(x)$). Furthermore, it can be proven (see Defazio et al. (2014)) that under a reasonable condition on γ, the update has vanishing variance, which enables the algorithm to converge linearly with a constant step size.

Motivation for a Variant. In its current form, every SAGA update is dense even if the individual gradients are sparse due to the historical gradient ($\bar{\alpha}$) term. Schmidt et al. (2016) introduced a special implementation with lagged updates where every iteration has a cost proportional to the size of the support of $f_i'(x)$. However, this subtle technique is not easily adaptable to the parallel setting (see App. F.2). We therefore introduce Sparse SAGA, a novel variant which explicitly takes sparsity into account and is easily parallelizable.

Sparse SAGA Algorithm. As in the Sparse SVRG algorithm proposed in Mania et al. (2015), we obtain Sparse SAGA by a simple modification of the parameter update rule in (2) where $\bar{\alpha}$ is replaced by a sparse version equivalent in expectation:

$$x^+ = x - \gamma(f_i'(x) - \alpha_i + D_i\bar{\alpha}), \tag{3}$$

where D_i is a diagonal matrix that makes a weighted projection on the support of f_i'. More precisely, let S_i

[4]For linear predictor models, the memory α_i^0 can be stored as a scalar. Following Hofmann et al. (2015), α_i^0 can be initialized to any convenient value (typically 0), unlike the prescribed $f_i'(x_0)$ analyzed in Defazio et al. (2014).

be the support of the gradient f_i' function (i.e., the set of coordinates where f_i' can be nonzero). Let D be a $d \times d$ diagonal reweighting matrix, with coefficients $1/p_v$ on the diagonal, where p_v is the probability that dimension v belongs to S_i when i is sampled uniformly at random in $\{1, ..., n\}$. We then define $D_i := P_{S_i} D$, where P_{S_i} is the projection onto S_i. The normalization from D ensures that $\mathbf{E} D_i \bar{\alpha} = \bar{\alpha}$, and thus that the update is still unbiased despite the projection.

Convergence Result for (Serial) Sparse Saga. For clarity of exposition, we model our convergence result after the simple form of Hofmann et al. (2015, Corollary 3) (note that the rate for Sparse Saga is the same as Saga). The proof is given in Appendix B.

Theorem 1. *Let $\gamma = \frac{a}{5L}$ for any $a \leq 1$. Then Sparse Saga converges geometrically in expectation with a rate factor of at least $\rho(a) = \frac{1}{5} \min\left\{\frac{1}{n}, a\frac{1}{\kappa}\right\}$, i.e., for x_t obtained after t updates, we have $\mathbb{E}\|x_t - x^*\|^2 \leq (1-\rho)^t C_0$, where $C_0 := \|x_0 - x^*\|^2 + \frac{1}{5L^2}\sum_{i=1}^n \|\alpha_i^0 - f_i'(x^*)\|^2$.*

Comparison with Lagged Updates. The lagged updates technique in Saga is based on the observation that the updates for component $[x]_v$ can be delayed until this coefficient is next accessed. Interestingly, the expected number of iterations between two steps where a given dimension v is involved in the partial gradient is p_v^{-1}, where p_v is the probability that v is involved. p_v^{-1} is precisely the term which we use to multiply the update to $[x]_v$ in Sparse Saga. Therefore one may view the Sparse Saga updates as *anticipated* Saga updates, whereas those in the Schmidt et al. (2016) implementation are *lagged*.

Although Sparse Saga requires the computation of the p_v probabilities, this can be done during a first pass through the data (during which constant step size Sgd may be used) at a negligible cost. In our experiments, both Sparse Saga and Saga with lagged updates had similar convergence in terms of number of iterations, with the Sparse Saga scheme being slightly faster in terms of runtime. We refer the reader to Schmidt et al. (2016) and Appendix F for more details.

3 Asynchronous Parallel Sparse Saga

As most recent parallel optimization contributions, we use a similar hardware model to Niu et al. (2011). We have multiple cores which all have read and write access to a shared memory. They update a central parameter vector in an asynchronous and lock-free fashion. Unlike Niu et al. (2011), we *do not* assume that the vector reads are consistent: multiple cores can read and write different coordinates of the shared vector at the same time. This means that a full vector read for a core might not correspond to any consistent state in the shared memory at any specific point in time.

3.1 Perturbed Iterate Framework

We first review the "perturbed iterate" framework recently introduced by Mania et al. (2015) which will form the basis of our analysis. In the sequential setting, stochastic gradient descent and its variants can be characterized by the following update rule:

$$x_{t+1} = x_t - \gamma g(x_t, i_t), \tag{4}$$

where i_t is a random variable independent from x_t and we have the unbiasedness condition $\mathbf{E} g(x_t, i_t) = f'(x_t)$ (recall that $\mathbf{E}$ is the relevant-past conditional expectation with respect to i_t).

Unfortunately, in the parallel setting, we manipulate stale, inconsistent reads of shared parameters and thus we do not have such a straightforward relationship. Instead, Mania et al. (2015) proposed to separate $\hat{x}_t$, the actual value read by a core to compute an update, with x_t, a "virtual iterate" that we can analyze and is *defined* by the update equation: $x_{t+1} := x_t - \gamma g(\hat{x}_t, i_t)$. We can thus interpret $\hat{x}_t$ as a noisy (perturbed) version of x_t due to the effect of asynchrony. In the specific case of (Sparse) Saga, we have to add the additional read memory argument $\hat{\alpha}^t$ to our update:

$$\begin{aligned} x_{t+1} &:= x_t - \gamma g(\hat{x}_t, \hat{\alpha}^t, i_t); \\ g(\hat{x}_t, \hat{\alpha}^t, i_t) &:= f_{i_t}'(\hat{x}_t) - \hat{\alpha}_{i_t}^t + D_{i_t}\left(1/n \sum_{i=1}^n \hat{\alpha}_i^t\right). \end{aligned} \tag{5}$$

We formalize the precise meaning of x_t and $\hat{x}_t$ in the next section. We first note that all the papers mentioned in the related work section that analyzed asynchronous parallel randomized algorithms assumed that the following unbiasedness condition holds:

$$\begin{bmatrix} \text{unbiasedness} \\ \text{condition} \end{bmatrix} \quad \mathbf{E}[g(\hat{x}_t, i_t) | \hat{x}_t] = f'(\hat{x}_t). \tag{6}$$

This condition is at the heart of most convergence proofs for randomized optimization methods.[5] Mania et al. (2015) correctly pointed out that most of the literature thus made the often implicit assumption that i_t is independent of $\hat{x}_t$. But as we explain below, this assumption is incompatible with a non-uniform asynchronous model in the analysis approach used in most of the recent literature.

3.2 On the Difficulty of Labeling the Iterates

Formalizing the meaning of x_t and $\hat{x}_t$ highlights a subtle but important difficulty arising when analyzing

[5] A notable exception is Sag (Le Roux et al., 2012) which has biased updates, yielding a significantly more complex convergence proof. Making Sag unbiased leads to Saga (Defazio et al., 2014) and a much simpler proof.

randomized parallel algorithms: what is the meaning of t? This is the problem of *labeling* the iterates for the purpose of the analysis, and this labeling can have randomness itself that needs to be taken in consideration when interpreting the meaning of an expression like $\mathbb{E}[x_t]$. In this section, we contrast three different approaches in a unified framework. We notably clarify the dependency issues that the labeling from Mania et al. (2015) resolves and propose a new, simpler labeling which allows for much simpler proof techniques. We consider algorithms that execute in parallel the following four steps, where t is a global labeling that needs to be defined:

1. Read the information in shared memory ($\hat{x}_t$).
2. Sample i_t. (7)
3. Perform some computations using $(\hat{x}_t, i_t)$.
4. Write an update to shared memory.

The "After Write" Approach. We call the "after write" approach the standard global labeling scheme used in Niu et al. (2011) and re-used in all the later papers that we mentioned in the related work section, with the notable exceptions of Mania et al. (2015) and Duchi et al. (2015). In this approach, t is a (virtual) global counter recording the number of *successful writes* to the shared memory x (incremented after step 4 in (7)); x_t thus represents the (true) content of the shared memory after t updates. The interpretation of the crucial equation (5) then means that $\hat{x}_t$ represents the (delayed) local copy value of the core that made the $(t+1)^{\text{th}}$ successful update; i_t represents the factor sampled by this core for this update. Notice that in this framework, the value of $\hat{x}_t$ and i_t is unknown at "time t"; we have to wait to the later time when the next core writes to memory to finally determine that its local variables are the ones labeled by t. We thus see that here $\hat{x}_t$ and i_t are not necessarily independent – they share dependence through the t label assignment. In particular, if some values of i_t yield faster updates than others, it will influence the label assignment defining $\hat{x}_t$. We illustrate this point with a concrete problematic example in Appendix A that shows that in order to preserve the unbiasedness condition (6), the "after write" framework makes the implicit assumption that the computation time for the algorithm running on a core is independent of the sample i chosen. This assumption seems overly strong in the context of potentially heterogeneous factors f_i's, and is thus a fundamental flaw for analyzing non-uniform asynchronous computation.

The "Before Read" Approach. Mania et al. (2015) addresses this issue by proposing instead to increment the global t counter just *before* a new core starts to *read* the shared memory (before step 1 in (7)). In their framework, $\hat{x}_t$ represents the (inconsistent) read that was made by this core in this computational block, and i_t represents the picked sample. The update rule (5) represents a *definition* of the meaning of x_t, which is now a "virtual iterate" as it does not necessarily correspond to the content of the shared memory at any point. The real quantities manipulated by the algorithm in this approach are the $\hat{x}_t$'s, whereas x_t is used only for the analysis – the critical quantity we want to see vanish is $\mathbb{E}\|\hat{x}_t - x^*\|^2$. The independence of i_t with $\hat{x}_t$ can be simply enforced in this approach by making sure that the way the shared memory x is read does not depend on i_t (e.g. by reading all its coordinates in a fixed order). Note that this means that we have to read all of x's coordinates, regardless of the size of f_{i_t}'s support. This is a much weaker condition than the assumption that all the computation in a block does not depend on i_t as required by the "after write" approach, and is thus more reasonable.

A New Global Ordering: the "After Read" Approach. The "before read" approach gives rise to the following complication in the analysis: $\hat{x}_t$ can depend on i_r for $r > t$. This is because t is a global time ordering only on the assignment of computation to a core, not on when $\hat{x}_t$ was finished to be read. This means that we need to consider both the "future" and the "past" when analyzing x_t. To simplify the analysis (which proved crucial for our Asaga proof), we thus propose a third way to label the iterates: $\hat{x}_t$ represents the $(t+1)^{\text{th}}$ *fully completed read* (t incremented after step 1 in (7)). As in the "before read" approach, we can ensure that i_t is independent of $\hat{x}_t$ by ensuring that how we read does not depend on i_t. But unlike in the "before read" approach, t here now does represent a global ordering on the $\hat{x}_t$ iterates – and thus we have that i_r is independent of $\hat{x}_t$ for $r > t$. Again using (5) as the definition of the virtual iterate x_t as in the perturbed iterate framework, we then have a very simple form for the value of x_t and $\hat{x}_t$ (assuming atomic writes, see Property 3 below):

$$x_t = x_0 - \gamma \sum_{u=0}^{t-1} g(\hat{x}_u, \hat{\alpha}^u, i_u)\,;$$

$$[\hat{x}_t]_v = [x_0]_v - \gamma \sum_{\substack{u=0 \\ \text{u s.t. coordinate } v \text{ was written} \\ \text{for } u \text{ before } t}}^{t-1} [g(\hat{x}_u, \hat{\alpha}^u, i_u)]_v\,. \tag{8}$$

The main idea of the perturbed iterate framework is to use this handle on $\hat{x}_t - x_t$ to analyze the convergence for x_t. In this paper, we can instead give directly the convergence of $\hat{x}_t$, and so unlike in Mania et al. (2015), we do not require that there exists a T such that x_T lives in shared memory.

3.3 Analysis setup

We describe Asaga, a sparse asynchronous parallel implementation of Sparse Saga, in Algorithm 1 in the

Algorithm 1 Asaga (analyzed algorithm)

```
1: Initialize shared variables $x$ and $(\alpha_i)_{i=1}^n$
2: keep doing in parallel
3:     $\hat{x}$ = inconsistent read of $x$
4:     $\forall j$, $\hat{\alpha}_j$ = inconsistent read of $\alpha_j$
5:     Sample $i$ uniformly at random in $\{1, ..., n\}$
6:     Let $S_i$ be $f_i$'s support
7:     $[\bar{\alpha}]_{S_i} = 1/n \sum_{k=1}^n [\hat{\alpha}_k]_{S_i}$
8:     $[\delta x]_{S_i} = -\gamma(f_i'(\hat{x}) - \hat{\alpha}_i + D_i[\bar{\alpha}]_{S_i})$
9:
10:    for $v$ in $S_i$ do
11:        $[x]_v \leftarrow [x]_v + [\delta x]_v$                // atomic
12:        $[\alpha_i]_v \leftarrow [f_i'(\hat{x})]_v$
13:        // ('$\leftarrow$' denotes a shared memory update.)
14:    end for
15: end parallel loop
```

Algorithm 2 Asaga (implementation)

```
1: Initialize shared variables $x$, $(\alpha_i)_{i=1}^n$ and $\bar{\alpha}$
2: keep doing in parallel
3:     Sample $i$ uniformly at random in $\{1, ..., n\}$
4:     Let $S_i$ be $f_i$'s support
5:     $[\hat{x}]_{S_i}$ = inconsistent read of $x$ on $S_i$
6:     $\hat{\alpha}_i$ = inconsistent read of $\alpha_i$
7:     $[\bar{\alpha}]_{S_i}$ = inconsistent read of $\bar{\alpha}$ on $S_i$
8:     $[\delta\alpha]_{S_i} = f_i'([\hat{x}]_{S_i}) - \hat{\alpha}_i$
9:     $[\delta x]_{S_i} = -\gamma([\delta\alpha]_{S_i} + D_i[\bar{\alpha}]_{S_i})$
10:    for $v$ in $S_i$ do
11:        $[x]_v \leftarrow [x]_v + [\delta x]_v$                // atomic
12:        $[\alpha_i]_v \leftarrow [\alpha_i]_v + [\delta\alpha]_v$          // atomic
13:        $[\bar{\alpha}]_v \leftarrow [\bar{\alpha}]_v + 1/n[\delta\alpha]_v$        // atomic
14:    end for
15: end parallel loop
```

theoretical form that we analyze, and in Algorithm 2 as its practical implementation. Before stating its convergence, we highlight some properties of Algorithm 1 and make one central assumption.

Property 1 (independence)**.** *Given the "after read" global ordering, i_r is independent of $\hat{x}_t$ $\forall r \geq t$.*

We enforce the independence for $r = t$ in Algorithm 1 by having the core read all the shared data parameters and historical gradients before starting their iterations. Although this is too expensive to be practical if the data is sparse, this is required by the theoretical Algorithm 1 that we can analyze. As Mania et al. (2015) stress, this independence property is assumed in most of the parallel optimization literature. The independence for $r > t$ is a consequence of using the "after read" global ordering instead of the "before read" one.

Property 2 (Unbiased estimator)**.** *The update, $g_t := g(\hat{x}_t, \hat{\alpha}^t, i_t)$, is an unbiased estimator of the true gradient at $\hat{x}_t$ (i.e. (5) yields (6) in conditional expectation).*

This property is crucial for the analysis, as in most related literature. It follows by the independence of i_t with $\hat{x}_t$ and from the computation of $\bar{\alpha}$ on line 7 of Algorithm 1, which ensures that $\mathbb{E}\hat{\alpha}_i = 1/n \sum_{k=1}^n [\hat{\alpha}_k]_{S_i} = [\bar{\alpha}]_{S_i}$, making the update unbiased. In practice, recomputing $\bar{\alpha}$ is not optimal, but storing it instead introduces potential bias issues in the proof (as detailed in Appendix G.3).

Property 3 (atomicity)**.** *The shared parameter coordinate update of $[x]_v$ on line 11 is atomic.*

Since our updates are additions, this means that there are no overwrites, even when several cores compete for the same resources. In practice, this is enforced by using *compare-and-swap* semantics, which are heavily optimized at the processor level and have minimal overhead. Our experiments with non-thread safe algorithms (i.e. where this property is not verified, see Figure 6 of Appendix G) show that compare-and-swap is necessary to optimize to high accuracy.

Finally, as is standard in the literature, we make an assumption on the maximum delay that asynchrony can cause – this is the *partially asynchronous* setting as defined in Bertsekas and Tsitsiklis (1989):

Assumption 1 (bounded overlaps)**.** *We assume that there exists a uniform bound, called τ, on the maximum number of iterations that can overlap together. We say that iterations r and t overlap if at some point they are processed concurrently. One iteration is being processed from the start of the reading of the shared parameters to the end of the writing of its update. The bound τ means that iterations r cannot overlap with iteration t for $r \geq t + \tau + 1$, and thus that every coordinate update from iteration t is successfully written to memory before the iteration $t + \tau + 1$ starts.*

Our result will give us conditions on τ subject to which we have linear speedups. τ is usually seen as a proxy for p, the number of cores (which lowerbounds it). However, though τ appears to depend linearly on p, it actually depends on several other factors (notably the data sparsity distribution) and can be orders of magnitude bigger than p in real-life experiments. We can upper bound τ by $(p - 1)R$, where R is the ratio of the maximum over the minimum iteration time (which encompasses theoretical aspects as well as hardware overhead). More details can be found in Appendix E.

Explicit effect of asynchrony. By using the overlap Assumption 1 in the expression (8) for the iterates, we obtain the following explicit effect of asynchrony that is crucially used in our proof:

$$\hat{x}_t - x_t = \gamma \sum_{u=(t-\tau)_+}^{t-1} S_u^t g(\hat{x}_u, \hat{\alpha}^u, i_u), \tag{9}$$

where S_u^t are $d \times d$ diagonal matrices with terms in $\{0, +1\}$. We know from our definition of t and x_t that every update in $\hat{x}_t$ is already in x_t – this is the 0 case. Conversely, some updates might be late: this is the +1 case. $\hat{x}_t$ may be lacking some updates from the "past" in some sense, whereas given our global ordering definition, it cannot contain updates from the "future".

3.4 Convergence and speedup results

We now state our main theoretical results. We give an outline of the proof in Section 3.5 and its full details in Appendix C. We first define a notion of problem sparsity, as it will appear in our results.

Definition 1 (Sparsity)**.** *As in Niu et al. (2011), we introduce* $\Delta_r := \max_{v=1..d} |\{i : v \in S_i\}|$. Δ_r *is the maximum right-degree in the bipartite graph of the factors and the dimensions, i.e., the maximum number of data points with a specific feature. For succinctness, we also define* $\Delta := \Delta_r/n$. *We have* $1 \leq \Delta_r \leq n$, *and hence* $1/n \leq \Delta \leq 1$.

Theorem 2 (Convergence guarantee and rate of Asaga)**.** *Suppose* $\tau < n/10$.[6] *Let*

$$a^*(\tau) := \frac{1}{32\left(1+\tau\sqrt{\Delta}\right)\xi(\kappa,\Delta,\tau)}$$
$$\text{where } \xi(\kappa,\Delta,\tau) := \sqrt{1+\frac{1}{8\kappa}\min\{\frac{1}{\sqrt{\Delta}},\tau\}} \tag{10}$$

(note that $\xi(\kappa,\Delta,\tau) \approx 1$ *unless* $\kappa < 1/\sqrt{\Delta}$ $(\leq \sqrt{n})$*).*

For any step size $\gamma = \frac{a}{L}$ *with* $a \leq a^*(\tau)$, *the inconsistent read iterates of Algorithm 1 converge in expectation at a geometric rate of at least:* $\rho(a) = \frac{1}{5}\min\left\{\frac{1}{n}, a\frac{1}{\kappa}\right\}$, *i.e.,* $\mathbb{E}f(\hat{x}_t) - f(x^*) \leq (1-\rho)^t\,\tilde{C}_0$, *where* $\tilde{C}_0$ *is a constant independent of* t *(*$\approx \frac{n}{\gamma}C_0$ *with* C_0 *as defined in Theorem 1).*

This result is very close to Saga's original convergence theorem, but with the maximum step size divided by an extra $1+\tau\sqrt{\Delta}$ factor. Referring to Hofmann et al. (2015) and our own Theorem 1, the rate factor for Saga is $\min\{1/n, 1/\kappa\}$ up to a constant factor. Comparing this rate with Theorem 2 and inferring the conditions on the maximum step size $a^*(\tau)$, we get the following conditions on the overlap τ for Asaga to have the same rate as Saga (comparing upper bounds).

Corollary 3 (Speedup condition)**.** *Suppose* $\tau \leq \mathcal{O}(n)$ *and* $\tau \leq \mathcal{O}(\frac{1}{\sqrt{\Delta}}\max\{1, \frac{n}{\kappa}\})$. *Then using the step size* $\gamma = a^*(\tau)/L$ *from (10), Asaga converges geometrically with rate factor* $\Omega(\min\{\frac{1}{n}, \frac{1}{\kappa}\})$ *(similar to Saga), and is thus linearly faster than its sequential counterpart up to a constant factor. Moreover, if* $\tau \leq \mathcal{O}(\frac{1}{\sqrt{\Delta}})$, *then a universal step size of* $\Theta(\frac{1}{L})$ *can be used for Asaga to be adaptive to local strong convexity with a similar rate to Saga (i.e., knowledge of* κ *is not required).*

[6] Asaga can actually converge for any τ, but the maximum step size then has a term of $\exp(\tau/n)$ in the denominator with much worse constants. See Appendix C.8.

Interestingly, in the well-conditioned regime ($n > \kappa$, where Saga enjoys a range of stepsizes which all give the same contraction ratio), Asaga can get the same rate as Saga even in the non-sparse regime ($\Delta = 1$) for $\tau < \mathcal{O}(n/\kappa)$. This is in contrast to the previous work on asynchronous incremental gradient methods which required some kind of sparsity to get a theoretical linear speedup over their sequential counterpart (Niu et al., 2011; Mania et al., 2015). In the ill-conditioned regime ($\kappa > n$), sparsity is required for a linear speedup, with a bound on τ of $\mathcal{O}(\sqrt{n})$ in the best-case (though degenerate) scenario where $\Delta = 1/n$.

Comparison to related work.

- We give the first convergence analysis for an asynchronous parallel version of Saga (note that Reddi et al. (2015) only covers an epoch based version of Saga with random stopping times, a fairly different algorithm).
- Theorem 2 can be directly extended to a parallel extension of the Svrg version from Hofmann et al. (2015), which is adaptive to the local strong convexity with similar rates (see Appendix C.2).
- In contrast to the parallel Svrg analysis from Reddi et al. (2015, Thm. 2), our proof technique handles inconsistent reads and a non-uniform processing speed across f_i's. Our bounds are similar (noting that Δ is equivalent to theirs), except for the adaptivity to local strong convexity: Asaga does not need to know κ for optimal performance, contrary to parallel Svrg (see App. C.2 for more details).
- In contrast to the Svrg analysis from Mania et al. (2015, Thm. 14), we obtain a better dependence on the condition number in our rate ($1/\kappa$ vs. $1/\kappa^2$ for them) and on the sparsity (they get $\tau \leq \mathcal{O}(\Delta^{-1/3})$), while we remove their gradient bound assumption. We also give our convergence guarantee on $\hat{x}_t$ *during* the algorithm, whereas they only bound the error for the "last" iterate x_T.

3.5 Proof outline

We give here the outline of our proof. Its full details can be found in Appendix C.

Let $g_t := g(\hat{x}_t, \hat{\alpha}^t, i_t)$. By expanding the update equation (5) defining the virtual iterate x_{t+1} and introducing $\hat{x}_t$ in the inner product term, we get:

$$\begin{aligned}\|x_{t+1}-x^*\|^2 = \|x_t - x^*\|^2 - 2\gamma\langle\hat{x}_t - x^*, g_t\rangle \\ + 2\gamma\langle\hat{x}_t - x_t, g_t\rangle + \gamma^2\|g_t\|^2.\end{aligned} \tag{11}$$

In the sequential setting, we require i_t to be independent of x_t to get unbiasedness. In the perturbed iterate framework, we instead require that i_t is independent of $\hat{x}_t$ (see Property 1). This crucial property enables us to use the unbiasedness condition (6) to write: $\mathbb{E}\langle\hat{x}_t - x^*, g_t\rangle = \mathbb{E}\langle\hat{x}_t - x^*, f'(\hat{x}_t)\rangle$. We thus take the expectation of (11) that allows us to use the μ-strong convexity of f:[7]

$$\langle \hat{x}_t - x^*, f'(\hat{x}_t)\rangle \geq f(\hat{x}_t) - f(x^*) + \frac{\mu}{2}\|\hat{x}_t - x^*\|^2.$$

With further manipulations on the expectation of (11), including the use of the standard inequality $\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$ (see Section C.3), we obtain our basic recursive contraction inequality:

$$\begin{aligned} a_{t+1} \leq & (1 - \frac{\gamma\mu}{2})a_t + \gamma^2\mathbb{E}\|g_t\|^2 - 2\gamma e_t \\ & \underbrace{+\gamma\mu\mathbb{E}\|\hat{x}_t - x_t\|^2 + 2\gamma\mathbb{E}\langle \hat{x}_t - x_t, g_t\rangle}_{\text{additional asynchrony terms}}, \end{aligned} \qquad (12)$$

where $a_t := \mathbb{E}\|x_t - x^*\|^2$ and $e_t := \mathbb{E}f(\hat{x}_t) - f(x^*)$.

In the sequential setting, one crucially uses the negative suboptimality term $-2\gamma e_t$ to cancel the variance term $\gamma^2\mathbb{E}\|g_t\|^2$ (thus deriving a condition on γ). Here, we need to bound the additional asynchrony terms using the same negative suboptimality in order to prove convergence and speedup for our parallel algorithm – thus getting stronger constraints on the maximum step size.

The rest of the proof then proceeds as follows:

- Lemma 1: we first bound the additional asynchrony terms in (12) in terms of past updates $(\mathbb{E}\|g_u\|^2, u \leq t)$. We achieve this by crucially using the expansion (9) for $x_t - \hat{x}_t$, together with the sparsity inequality (44) (which is derived from Cauchy-Schwartz, see Appendix C.4).
- Lemma 2: we then bound the updates $\mathbb{E}\|g_u\|^2$ with respect to past suboptimalities $(e_v)_{v\leq u}$. From our analysis of Sparse Saga in the sequential case:

  $$\mathbb{E}\|g_t\|^2 \leq 2\mathbb{E}\|f'_{i_t}(\hat{x}_t) - f'_{i_t}(x^*)\|^2 + 2\mathbb{E}\|\hat{\alpha}^t_{i_t} - f'_{i_t}(x^*)\|^2$$

  We bound the first term by $4Le_t$ using Hofmann et al. (2015, Equation (8)). To express the second term in terms of past suboptimalities, we note that it can be seen as an expectation of past first terms with an adequate probability distribution which we derive and bound.
- By substituting Lemma 2 into Lemma 1, we get a master contraction inequality (28) in terms of a_{t+1}, a_t and $e_u, u \leq t$.
- We define a novel Lyapunov function $\mathcal{L}_t = \sum_{u=0}^{t}(1-\rho)^{t-u}a_u$ and manipulate the master inequality to show that $\mathcal{L}_t$ is bounded by a contraction, subject to a maximum step size condition on γ (given in Lemma 3, see Appendix C.1).
- Finally, we unroll the Lyapunov inequality to get the convergence Theorem 2.

[7]Here is our departure point with Mania et al. (2015) who replaced the $f(\hat{x}_t) - f(x^*)$ term with the lower bound $\frac{\mu}{2}\|\hat{x}_t - x^*\|^2$ in this relationship (see their Equation (2.4)), yielding an inequality too loose to get fast rates for Svrg.

4 Empirical results

We now present the main results of our empirical comparison of asynchronous Saga, Svrg and Hogwild. Additional results, including convergence and speedup figures with respect to the number of iteration and measures on the τ constant are available in the appendix.

4.1 Experimental setup

Models. Although Asaga can be applied more broadly, we focus on logistic regression, a model of particular practical importance. The associated objective function takes the following form: $\frac{1}{n}\sum_{i=1}^{n}\log\left(1+\exp(-b_i a_i^\intercal x)\right) + \frac{\lambda}{2}\|x\|^2$, where $a_i \in \mathbb{R}^p$ and $b_i \in \{-1,+1\}$ are the data samples.

Datasets. We consider two sparse datasets: RCV1 (Lewis et al., 2004) and URL (Ma et al., 2009); and a dense one, Covtype (Collobert et al., 2002), with statistics listed in the table below. As in Le Roux et al. (2012), Covtype is standardized, thus 100% dense. Δ is $\mathcal{O}(1)$ in all datasets, hence not very insightful when relating it to our theoretical results. Deriving a less coarse sparsity bound remains an open problem.

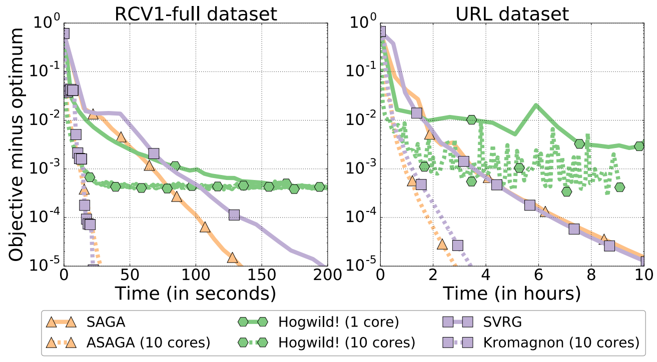
	n	d	density	L
RCV1	697,641	47,236	0.15%	0.25
URL	2,396,130	3,231,961	0.004%	128.4
Covtype	581,012	54	100%	48428

Hardware and software. Experiments were run on a 40-core machine with 384GB of memory. All algorithms were implemented in Scala. We chose this high-level language despite its typical 20x slowdown compared to C (when using standard libraries, see Appendix G.2) because our primary concern was that the code may easily be reused and extended for research purposes (to this end, we have made all our code available at `https://github.com/RemiLeblond/ASAGA`).

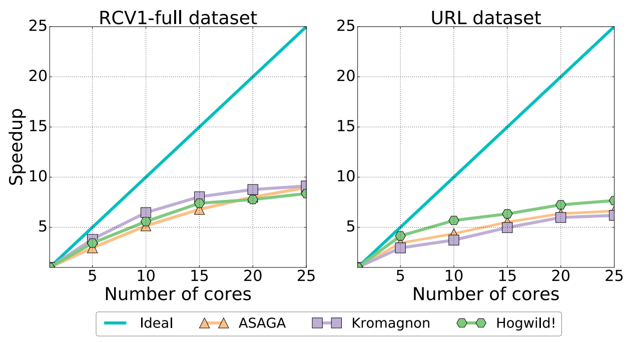
4.2 Implementation details

Exact regularization. Following Schmidt et al. (2016), the amount of regularization used was set to $\lambda = 1/n$. In each update, we project the gradient of the regularization term (we multiply it by D_i as we also do with the vector $\bar{\alpha}$) to preserve the sparsity pattern while maintaining an unbiased estimate of the gradient. For squared ℓ_2, the Sparse Saga updates becomes: $x^+ = x - \gamma(f'_i(x) - \alpha_i + D_i\bar{\alpha} + \lambda D_i x)$.

Comparison with the theoretical algorithm. The algorithm we used in the experiments is fully detailed in Algorithm 2. There are two differences with Algorithm 1. First, in the implementation we pick i_t at random *before* we read data. This enables us to only read the necessary data for a given iteration (i.e. $[\hat{x}_t]_{S_i}, [\hat{\alpha}^t_i], [\bar{\alpha}^t]_{S_i}$). Although this violates



(a) Suboptimality as a function of time.



(b) Speedup as a function of the number of cores

Figure 1: **Convergence and speedup for asynchronous stochastic gradient descent methods**. We display results for RCV1 and URL. Results for Covtype can be found in Appendix D.2.

Property 1, it still performs well in practice.

Second, we maintain $\bar{\alpha}^t$ in memory. This saves the cost of recomputing it at every iteration (which we can no longer do since we only read a subset data). Again, in practice the implemented algorithm enjoys good performance. But this design choice raises a subtle point: the update is not guaranteed to be unbiased in this setup (see Appendix G.3 for more details).

4.3 Results

We first compare three different asynchronous variants of stochastic gradient methods on the aforementioned datasets: ASAGA, presented in this work, KROMAGNON, the asynchronous sparse SVRG method described in Mania et al. (2015) and HOGWILD (Niu et al., 2011). Each method had its step size chosen so as to give the fastest convergence (up to 10^{-3} in the special case of HOGWILD). The results can be seen in Figure 1a: for each method we consider its asynchronous version with both one (hence sequential) and ten processors. This figure reveals that the asynchronous version offers a significant speedup over its sequential counterpart.

We then examine the speedup relative to the increase in the number of cores. The speedup is measured as time to achieve a suboptimality of 10^{-5} (10^{-3} for HOGWILD) with one core divided by time to achieve the same suboptimality with several cores, averaged over 3 runs. Again, we choose step size leading to fastest convergence (see Appendix G.2 for information about the step sizes). Results are displayed in Figure 1b.

As predicted by our theory, we observe linear "theoretical" speedups (i.e. in terms of number of iterations, see Appendix D.2). However, with respect to running time, the speedups seem to taper off after 20 cores. This phenomenon can be explained by the fact that our hardware model is by necessity a simplification of reality. As noted in Duchi et al. (2015), in a modern machine there is no such thing as *shared memory*. Each core has its own levels of cache (L1, L2, L3) in addition to RAM. The more cores are used, the lower in the memory stack information goes and the slower it gets. More experimentation is needed to quantify that effect and potentially increase performance.

5 Conclusions and future work

We have described ASAGA, a novel sparse and fully asynchronous variant of the incremental gradient algorithm SAGA. Building on the recently proposed "perturbed iterate" framework, we have introduced a novel analysis of the algorithm and proven that under mild conditions ASAGA is linearly faster than SAGA. Our empirical benchmarks confirm speedups up to 10x.

Our proof technique accommodates more realistic settings than is usually the case in the literature (e.g. inconsistent reads/writes and an unbounded gradient); we obtain tighter conditions than in previous work. In particular, we show that sparsity is not always necessary to get linear speedups. Further, we have proposed a novel perspective to clarify an important technical issue present in most of the recent convergence rate proofs for asynchronous parallel optimization algorithms.

Schmidt et al. (2016) have shown that SAG enjoys much improved performance when combined with non-uniform sampling and line-search. We have also noticed that our Δ_r constant (being essentially a maximum) sometimes fails to accurately represent the full sparsity distribution of our datasets. Finally, while our algorithm can be directly ported to a distributed master-worker architecture, its communication pattern would have to be optimized to avoid prohibitive costs. Limiting communications can be interpreted as artificially increasing the delay, yielding an interesting trade-off between delay influence and communication costs.

A final interesting direction for future analysis is the further exploration of the τ term, which we have shown encompasses more complexity than previously thought.

Acknowledgments

We would like to thank Xinghao Pan for sharing with us their implementation of KROMAGNON. This work was partially supported by a Google Research Award and the MSR-Inria Joint Center. FP acknowledges financial support from from the chaire *Économie des nouvelles données* with the *data science* joint research initiative with the *fonds AXA pour la recherche*.

References

D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of svms for very large scale problems. *Neural Comput.*, 14:1105–1114, 2002.

C. De Sa, C. Zhang, K. Olukotun, and C. Ré. Taming the wild: a unified analysis of Hogwild!-style algorithms. In *NIPS*, 2015.

A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS*, 2014.

J. C. Duchi, S. Chaturapruek, and C. Ré. Asynchronous stochastic convex optimization. In *NIPS*, 2015.

T. Hofmann, A. Lucchi, S. Lacoste-Julien, and B. McWilliams. Variance reduced stochastic gradient descent with neighbors. In *NIPS*, 2015.

C.-J. Hsieh, H.-F. Yu, and I. Dhillon. PASSCoDe: Parallel ASynchronous Stochastic dual Co-ordinate Descent. In *ICML*, 2015.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, 2013.

J. Konecny and P. Richtarik. Semi-stochastic gradient descent methods. *arXiv:1312.1666*, 2013.

N. Le Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS*, 2012.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *JMLR*, 5:361–397, 2004.

X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *NIPS*, 2015.

J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *JMLR*, 16:285–322, 2015.

C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtarik, and M. Takac. Adding vs. averaging in distributed primal-dual optimization. In *ICML*, 2015.

J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious URLs: an application of large-scale online learning. In *ICML*, 2009.

H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *arXiv:1507.06970v2*, 2015.

F. Niu, B. Recht, C. Re, and S. Wright. Hogwild: a lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

S. J. Reddi, A. Hefny, S. Sra, B. Póczos, and A. Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *NIPS*, 2015.

M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *F. Math. Program.*, 2016.

S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss. *JMLR*, 14:567–599, 2013.

S.-Y. Zhao and W.-J. Li. Fast asynchronous parallel stochastic gradient descent. In *AAAI*, 2016.