THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Distributed Block Coordinate Descent for Minimizing Partially Separable Functions

Citation for published version:
Marecek, J, Richtarik, P & Takac, M 2014 'Distributed Block Coordinate Descent for Minimizing Partially Separable Functions' ArXiv. <http://arxiv.org/abs/1406.0238>

Link:
Link to publication record in Edinburgh Research Explorer

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 27. Jul. 2024

arXiv:1406.0238v1 [math.OC] 2 Jun 2014

Distributed Block Coordinate Descent for Minimizing Partially Separable Functions

Jakub Mareček[1], Peter Richtárik[2], and Martin Takáč[3]
[1] IBM Research, Dublin, Ireland, jakub@marecek.cz
[2] School of Mathematics, University of Edinburgh, UK, peter.richtarik@ed.ac.uk
[3] Dept. of Industrial & Systems Engineering, Lehigh University, USA, takac.mt@gmail.com

June 3, 2014

Abstract

In this work we propose a distributed randomized block coordinate descent method for minimizing a convex function with a huge number of variables/coordinates. We analyze its complexity under the assumption that the smooth part of the objective function is partially block separable, and show that the degree of separability directly influences the complexity. This extends the results in [22] to a distributed environment. We first show that partially block separable functions admit an expected separable overapproximation (ESO) with respect to a distributed sampling, compute the ESO parameters, and then specialize complexity results from recent literature that hold under the generic ESO assumption. We describe several approaches to distribution and synchronization of the computation across a cluster of multi-core computers and provide promising computational results.

1 Introduction

With the ever increasing availability of data, there is need to solve ever larger instances of data science and machine learning problems, many of which turn out to be convex optimization problems in huge dimensions. A single machine is unable to store the complete data of big enough problems in its main memory. This suggests the need for efficient algorithms, which can benefit from distributed computing, where only a part of the input is stored on each of the nodes of a cluster and both the computation and communication are designed accordingly.

The central focus of this work is the optimization of the form:

$$\min_{x\in\mathbf{R}^N} \left[F(x) := f(x) + \Omega(x)\right], \tag{1}$$

where f is a smooth, convex and partially block separable function, and Ω is a possibly nonsmooth, convex, block separable and "simple" extended real valued function. The technical definitions of these assumptions are given in Section 2.

1.1 Contributions

We propose and study the performance of a *distributed block coordinate descent method* applied to problem (1).

In our method, the blocks of coordinates are first partitioned among C computers of a cluster. Likewise, data associated with these blocks are partitioned accordingly and stored in a distributed way. In each of the subsequent iterations, each computer chooses τ blocks out of those stored locally, uniformly at random. Then, each computer computes and applies an update to the selected blocks, in parallel, out of information available to it locally. A residual is then transmitted to other computers, which require it at the beginning of the next iteration in a reduce all (supported by our theory) or asynchronous (we depart from theory here) manner, so that the next parallel iteration can commence.

The main contributions of this paper are, in no particular order:

1. **Connection to previous results.** Distributed coordinate descent methods were developed and studied recently by Richtárik and Takáč [23] (algorithm: Hydra) and Fercoq et al. [4] (Algorithms: Hydra^2). None of these papers consider blocks, and both focus on a different class of functions f (convex functions admitting a quadratic upper bound with a fixed Hessian). In contrast, here we focus on partially separable functions f and on distributed *block*-coordinate descent. Our work can be most directly seen as an extension of the setting in [22], which focuses on parallel block coordinate descent, to the distributed setting.
2. **New ESO.** Our analysis is based on the development of an ESO (expected separable overapproximation) inequality for partialy separable functions and distributed samplings (Theorem 1 in Section 4).
3. **Iteration complexity.** We show that the iteration complexity of the method is directly governed by the degree of block separability of f: with more separable problems, the method requires fewer iterations. Interestingly, more separability usually results in a smaller cost of each iteration. Hence, more separable problems enjoy a double acceleration effect. The complexity results are stated in two theorems in Section 5. In particular, we state linear $O(\log(1/\varepsilon))$ and sublinear $O(1/\varepsilon)$ rates for strongly and weakly convex F, respectively.
4. **NUMA.** Our method and results are valid not only for a cluster setting, where there really are C computers which do not share any memory, and hence have to communicate by sending messages to each other, but also for computers using Non-Uniform Memory Access (NUMA) architecture. In NUMA, as a name suggests, the memory access time depends on the memory location relative to a processor, and accessing local memory is much faster than accessing memory elsewhere. Hence, even in NUMA architectures, it is beneficial to split the problem among NUMA regions and let each processor use primarily the local memory, although the communication between processors may be replaced by the use of shared memory.
5. **Efficient implementation.** We replaced a natural synchronous communications between computers (which follows our theory) with asynchronous communication (which departs from our theory) to great effect. An efficient open-source implementation of the algorithm is available as part of the package `http://code.google.com/p/ac-dc/`.

1.2 Coordinate descent methods

Here we give a brief overview of some existing literature on randomized coordinate descent; for more reference we refer the reader to [22, 5].

Block-coordinate descent. Block-coordinate descent is a simple iterative optimization strategy, where two subsequent iterates differ only in a single block of coordinates. In the very common special case, each block consists of a single coordinate. The choice of the block can be deterministic, e.g. cyclic ([27]), greedy ([21]), or randomized. Recent theoretical guarantees of randomized coordinate descent algorithms can be found in [18, 25, 6, 13, 16, 10]. Coordinate descent algorithms are also closely related to coordinate relaxation, linear and non-linear Gauss-Seidel methods, subspace correction, and domain decomposition (see [2] for references). For classical references in the optimization literature on non-randomized variants we refer to the work of Tseng [14, 39, 38, 37].

Parallel block-coordinate descent. Clearly, one can parallelize coordinate descent by updating several blocks, in parallel. Complexity issues were studied by various authors. Richtárik and Takáč studied a broad class of parallel methods for the same problem we study in this paper, and introduced the concept of ESO [22]. The complexity was improved by Tappenden et al [36]. An efficient accelerated version was introduced by Fercoq and Richtárik [5], an inexact version was studied in [35], an asynchronous variant was studied by Liu and Wright [12], nonuniform variant by Richtárik and Takáč [24] and a way of handling nonsmooth functions was described in [6]. Some further approaches can be found in [29, 34, 41, 19, 26].

Distributed block-coordinate descent. Distributed coordinate descent was first proposed and analyzed by Richtárik and Takáč [23] and an accelerated version thereof by Fercoq et al [4]. The literature on this topic is both very sparse and recent.

2 Notation and assumptions

In this section we introduce the notation used in the rest of the paper and give formal assumptions on f and Ω already described informally in the introduction.

Block structure.[1] We decompose $\mathbf{R}^N$ into n subspaces as follows. Let $U \in \mathbf{R}^{N\times N}$ be the $N \times N$ identity matrix and further let $U = [U_1, U_2, \ldots, U_n]$ be a column decomposition of U into n submatrices, with U_i being of size $N \times N_i$, where $\sum_i N_i = N$. It is easy to observe that any vector $x \in \mathbf{R}^N$ can be written uniquely as $x = \sum_{i=1}^n U_i x^{(i)}$, where $x^{(i)} \in \mathbf{R}^{N_i}$. Moreover, $x^{(i)} = U_i^T x$. In view of the above, from now on we write $x^{(i)} := U_i^T x \in \mathbf{R}^{N_i}$, and refer to $x^{(i)}$ as the *i-th block* of x.

Projection onto a set of blocks. For a set of blocks $S \subseteq [n]$ and $x \in \mathbf{R}^N$ we let $x_{[S]}$ be the vector in $\mathbf{R}^N$ whose blocks $i \in S$ are identical to those of x, but whose other blocks are zeroed out. Block-by-block, we thus have $(x_{[S]})^{(i)} = x^{(i)}$ for $i \in S$ and $(x_{[S]})^{(i)} = 0 \in \mathbf{R}^{N_i}$, otherwise. It will be more useful to us however to write

$$x_{[S]} := \sum_{i\in S} U_i x^{(i)}, \tag{2}$$

where we adopt the convention that if $S = \emptyset$, the sum is equal $0 \in \mathbf{R}^N$.

Norms. Spaces $\mathbf{R}^{N_i}$, $i \in [n]$, are equipped with a pair of conjugate norms: $\|t\|_{(i)}$ and $\|t\|_{(i)}^* := \max_{\|s\|_{(i)}\leq 1} \langle s,t\rangle$, $t \in \mathbf{R}^{N_i}$. For $w \in \mathbf{R}^n_{++}$, define a pair of conjugate norms in $\mathbf{R}^N$ by

$$\|x\|_w = \left[\sum_{i=1}^n w_i \|x^{(i)}\|_{(i)}^2\right]^{1/2}, \quad \|y\|_w^* := \max_{\|x\|_w\leq 1} \langle y,x\rangle = \left[\sum_{i=1}^n w_i^{-1} (\|y^{(i)}\|_{(i)}^*)^2\right]^{1/2}. \tag{3}$$

We shall assume throughout the paper that f has the following properties.

Assumption 1 (Properties of f)**.** *Function $f : \mathbf{R}^N \to \mathbf{R}$ satisfies:*

1. ***Partial separability.*** *Function f is of the form*

$$f(x) = \sum_{J\in\mathscr{J}} f_J(x), \tag{4}$$

where $\mathscr{J}$ is a collection of subsets of $[n]$ and function f_J depends on x through blocks $x^{(i)}$ for $i \in J$ only. The quantity $\omega := \max_{j\in\mathscr{J}} |J|$ is the degree of separability of f.

2. ***Convexity.*** *Functions f_J, $j \in \mathscr{J}$ in (4) are convex.*

3. ***Smoothness.*** *The gradient of f is block Lipschitz, uniformly in x, with positive constants $L_1, \ldots, L_n$. That is, for all $x \in \mathbf{R}^N$, $i \in [n]$ and $t \in \mathbf{R}^{N_i}$,*

$$\|\nabla_i f(x+U_i t) - \nabla_i f(x)\|_{(i)}^* \leq L_i \|t\|_{(i)}, \tag{5}$$

where $\nabla_i f(x) := (\nabla f(x))^{(i)} = U_i^T \nabla f(x) \in \mathbf{R}^{N_i}$.

Let us now offer a few remarks:

1. Note that every function f is trivially of the form (4): we can always assume that $\mathscr{J}$ contains just the single set $J = [n]$ and let $f_J = f$. In this case we would have $\omega = n$. However, many functions appearing in applications can naturally be decomposed as a sum of a number of functions each of which depends on a small number of blocks of x only. That is, many functions have degree of separability ω that is much smaller than n.

2. Note that since f_J are convex, so is f. While it is possible to lift this assumption and provide an analysis in the nonconvex case, this is beyond the scope of this paper.

3. An important consequence of (5) is the following standard inequality [17]:

$$f(x+U_i t) \leq f(x) + \langle \nabla_i f(x), t\rangle + \tfrac{L_i}{2}\|t\|_{(i)}^2. \tag{6}$$

[1] We aim to keep our notation consistent with that of Nesterov [18] and Richtárik & Takáč [22].

Assumption 2 (Properties of Ω). *We assume that $\Omega : \mathbf{R}^N \to \mathbf{R} \cup \{+\infty\}$ is (block) separable, i.e., that it can be decomposed as follows:*

$$\Omega(x) = \sum_{i=1}^{n} \Omega_i(x^{(i)}), \tag{7}$$

where the functions $\Omega_i : \mathbf{R}^{N_i} \to \mathbf{R} \cup \{+\infty\}$ are convex and closed.

3 Distributed block coordinate descent method

In this section we describe our distributed block coordinate descent method (Algorithm 1). It is designed to solve convex optimization problems of the form (1) where the data describing the problem is so large that it is impossible to store it in memory of a single computer.

Algorithm 1: Distributed Block Coordinate Descent

1 choose $x_0 \in \mathbf{R}^N$
2 $k \longleftarrow 0$
3 **while** *termination criteria are not satisfied* **do**
4 $x_{k+1} \longleftarrow x_k$
5 **for each** *computer* $c \in \{1, \dots, C\}$ **in parallel do**
6 sample a set of coordinates $Z_k^{(c)} \subseteq P^{(c)}$ of size τ, uniformly at random
7 **for each** $i \in Z_k^{(c)}$ **in parallel do**
8 compute an update $h^{(i)}(x_k)$
9 $x_{k+1} \longleftarrow x_{k+1} + U_i h^{(i)}(x_k)$
10 $k \longleftarrow k+1$

Pre-processing. Before the method is run, the set of blocks is partitioned into C sets $P^{(c)}$, $c = 1, 2, \dots, C$. Each computer "owns" one partition and will only store and update blocks of x it owns. That is, the blocks $i \in P^{(c)}$ of x are stored on and updated by computer c only. Likewise, "all data" relevant to these blocks is stored on computer c. However, in the above description of the methods we abstract from details of this sort as they vary from problem to problem. We will deal with these issues in Section 6.

Distributed sampling of blocks. In Step 6, each computer c chooses a random subset $Z_k^{(c)}$ of blocks from its partition $P^{(c)}$. We assume that $|Z_k^{(c)}| = \tau$, and that it is chosen uniformly at random from all subsets of $P^{(c)}$ of cardinality τ. Moreover, we assume the choice is done independently from all history and from what the other computers do in the same iteration. Formally, we say that the set of blocks chosen by all computers in iteration k, i.e., $Z_k = \cup_{c=1}^{C} Z_k^{(c)}$, is a (C, τ)-*distributed sampling.*

For easier reference in the rest of the paper we formalize the setup described above as Assumption 3 at the end of this section (where we drop the subscript k, since the samplings are independent of k).

Computing and applying block updates. In Steps 7-9, each computer c first computes and then applies updates to blocks $i \in Z_k^{(c)}$ to x_k. This is done on each computer in parallel. Hence, we have two levels of parallelism: across the nodes/computers and within each computer. The update to block i is denoted by $h^{(i)}(x_k)$, and arises as a solution of an optimization problem in the lower dimensional space R^{N_i}:

$$h^{(i)}(x_k) \leftarrow \arg\min_{t \in \mathbf{R}^{N_i}} \langle \nabla_i f(x_k), t \rangle + \frac{\beta w_i}{2} \|t\|_{(i)}^2 + \Omega_i(x_k^{(i)} + t). \tag{8}$$

Our method is most effective when this optimization problem has a closed form solution, which is the case in many applications. Note that *nearly all* information that describes problem (8) for $i \in P^{(c)}$

is available at node c. In particular, as we said before, $x_k^{(i)}$ is stored on c. Moreover, we can store the description of Ω_i, norm $\|\cdot\|_{(i)}$ and the pair (β, w_i), for $i \in P^{(c)}$, on node c (and only there).

Note that we did not specify yet the values of the parameters β and $w = (w_1, \dots, w_n)$. These depend on the properties of f and sampling $\hat{Z}$. We shall give theoretically justified formulas for these parameters in Section 4.

Communication. Finally, note in order to find $h^{(i)}(x_k)$, each computer needs to be able to compute $\nabla_i f(x_k)$ for blocks $i \in Z_k^{(c)} \subseteq P^{(c)}$. This is the only information that the computers can *not* obtain from data stored locally – communication among the nodes/computers is necessary. We shall describe an efficient communication protocol that allows each node to compute $\nabla_i f(x_k)$ in Section 6.

Assumption 3 (Distributed sampling)**.** *We make the following assumptions:*

1. ***Balanced partitioning.*** *The set of blocks is partitioned into C groups $P^{(1)}, \dots, P^{(C)}$, each of size $s := n/C$. That is,*
   (a) $\{1, 2, \dots, n\} = \cup_{c=1}^{C} P^{(c)}$,
   (b) $P^{(c')} \cap P^{(c'')} = \emptyset$ *for* $c' \neq c''$,
   (c) $|P^{(c)}| =: s$ *for all* c.
2. ***Sampling.*** *For each $c \in \{1, \dots, C\}$, the set $\hat{Z}^{(c)}$ is a random subset of $P^{(c)}$ of size $\tau \in \{1, 2, \dots, s\}$, where each subset of size τ is chosen with equal probability.*

We refer to the random set-valued mapping $\hat{Z} := \cup_{c=1}^{C} \hat{Z}^{(c)}$ by the name (C, τ)-distributed sampling.

4 Expected Separable Overapproximation (ESO)

The following concept was first defined in [22]. It plays a key role in the complexity analysis of randomized coordinate descent methods.

Definition 1 (ESO)**.** *Let $\hat{Z}$ be any uniform sampling, i.e., a random sampling of blocks for which* $\mathbf{Prob}(i \in \hat{Z}) = \mathbf{Prob}(j \in \hat{Z})$ *for all $i, j \in [n]$. We say that function f admits an ESO with respect to sampling $\hat{Z}$, with parameters $\beta > 0$ and $w \in \mathbf{R}^n_{++}$, if the following inequality holds for all $x, h \in \mathbf{R}^N$:*

$$\mathbf{E}[f(x + h_{[\hat{Z}]})] \leq f(x) + \frac{\mathbf{E}[|\hat{Z}|]}{n} \left(\langle \nabla f(x), h \rangle + \frac{\beta}{2} \|h\|_w^2 \right). \tag{9}$$

For simplicity, we will sometimes write $(f, \hat{Z}) \sim ESO(\beta, w)$.

In the rest of this section we derive an ESO inequality for f satisfying Assumption 1 (smooth, convex, partially separable) and for sampling $\hat{Z}$ satisfying Assumption 3 ((C, τ)-distributed sampling). This has not been done before in the literature. In particular, we give simple closed-form formulas for parameters β and w, which we shall use in Section 5 to shed light on the performance of the method.

We first need to establish an auxiliary result.

Lemma 1. *Let $\hat{Z} = \cup_{c=1}^{C} \hat{Z}^{(c)}$ be the (C, τ)-distributed sampling. Pick $J \subseteq [n]$ and assume that $|P^{(c)} \cap J| = \xi$ for some $\xi \geq 1$ and all c. Let $\kappa = \kappa(|\hat{Z} \cap J|, i)$ be any function that depends on $|\hat{Z} \cap J|$ and $i \in [n]$ only. Then*

$$\mathbf{E}\left[\sum_{i \in \hat{Z} \cap J} \kappa(|\hat{Z} \cap J|, i) \right] = \mathbf{E}\left[\frac{|\hat{Z} \cap J|}{C\xi} \sum_{i \in J} \kappa(|\hat{Z} \cap J|, i) \right]. \tag{10}$$

Proof. Let us denote by $J^{(c)} = J \cap P^{(c)}$, $\zeta = |\hat{Z} \cap J|$ and $\zeta^{(c)} = |\hat{Z} \cap J^{(c)}|$. Then

$$\begin{aligned}
\mathbf{E}\left[\sum_{i\in\hat{Z}\cap J}\kappa(\zeta,i)\right] &= \mathbf{E}\left[\mathbf{E}\left[\sum_{i\in\hat{Z}\cap J}\kappa(\zeta,i)\mid\zeta\right]\right] \\
&= \mathbf{E}\left[\mathbf{E}\left[\mathbf{E}\left[\sum_{i\in\hat{Z}\cap J}\kappa(\sum_{c=1}^{C}\zeta^{(c)},i)\mid\zeta^{(1)},\ldots,\zeta^{(C)},\sum_{c=1}^{C}\zeta^{(c)}=\zeta\right]\mid\zeta\right]\right] \\
&= \mathbf{E}\left[\mathbf{E}\left[\mathbf{E}\left[\sum_{c=1}^{C}\sum_{i\in\hat{Z}^{(c)}\cap J^{(c)}}\kappa(\zeta,i)\mid\zeta^{(1)},\ldots,\zeta^{(C)}\right]\mid\sum_{c=1}^{C}\zeta^{(c)}=\zeta\right]\right] \\
&= \mathbf{E}\left[\mathbf{E}\left[\sum_{c=1}^{C}\frac{\zeta^{(c)}}{\xi}\sum_{i\in\cap J^{(c)}}\kappa(\zeta,i)\mid\sum_{c=1}^{C}\zeta^{(c)}=\zeta\right]\right] \\
&= \mathbf{E}\left[\sum_{c=1}^{C}\frac{\zeta}{\xi C}\sum_{i\in\cap J^{(c)}}\kappa(\zeta,i)\right] = \mathbf{E}\left[\frac{\zeta}{\xi C}\sum_{i\in J}\kappa(\zeta,i)\right]. \qquad \square
\end{aligned}$$

□

The main technical result of this paper follows. This is a generalization of a result from [22] for partially separable f and τ-nice sampling to the distributed ($c > 1$) case. Notice that for $C = 1$ we have $\xi = \omega$.

Theorem 1 (ESO)**.** *Let f satisfy Assumption 1 and $\hat{Z}$ satisfy Assumption 3. Let* [2] *as $\xi := \max\{|P^{(c)} \cap J| : c \in \{1,\ldots,C\}, J \in \mathscr{J}\}$. Then $(f,\hat{Z})$ admits ESO with parameters β and w given by*

$$\beta = 1 + \frac{(\xi-1)(\tau-1)}{\max\{1,s-1\}} + (C-1)\frac{\xi\tau}{s}, \tag{11}$$

and $w_i = L_i$, $i = 1,2,\ldots,n$

Proof. For fixed $x \in \mathbf{R}^N$, define $\phi(h) := f(x+h) - f(x) - \langle\nabla f(x), h\rangle$. Likewise, for all $J \in \mathscr{J}$ we define $\phi_J(h) := f_J(x+h) - f_J(x) - \langle\nabla f_J(x), h\rangle$. Note that

$$\phi(h) = \sum_{J\in\mathscr{J}}\phi_J(h). \tag{12}$$

Also note that the functions ϕ_J and ϕ are convex and minimized at $h = 0$, were they attain the value of 0. For any uniform sampling, and hence for $\hat{Z}$ in particular, and any $a \in \mathbf{R}^N$, one has

$$\mathbf{E}[\langle a, h_{[\hat{Z}]}\rangle] = \tfrac{\mathbf{E}[|\hat{Z}|]}{n}\langle a, h\rangle, \tag{13}$$

and therefore $\mathbf{E}[\phi(h_{[\hat{Z}]})] = \mathbf{E}[f(x+h_{[\hat{Z}]})] - f(x) - \frac{\mathbf{E}[|\hat{Z}|]}{n}\langle\nabla f(x), h\rangle$. Because of this, and in view of (9) and the fact that as $\mathbf{E}[\hat{Z}] = C\tau$ (in fact, $|\hat{Z}| = C\tau$ with probability 1), we only need to show that

$$\mathbf{E}[\phi(h_{[\hat{Z}]})] \le \tfrac{C\tau}{n}\tfrac{\beta}{2}\|h\|_w^2. \tag{14}$$

Our starting point in establishing (14) will be the observation that from (6) used with $t = h^{(i)}$ we get

$$\phi(U_i h^{(i)}) \le \tfrac{L_i}{2}\|h^{(i)}\|_{(i)}^2, \quad i \in [n]. \tag{15}$$

To simplify the proof, we shall without loss of generality assume that $|P^{(c)} \cap J| = \xi$ for all $c \in \{1,2,\ldots,C\}$ and $J \in \mathscr{J}$ for some constant $\xi > 1$. This can be achieved by extending the sets $J \in \mathscr{J}$ by introducing dummy dependencies (note that the assumptions of the theorem are still satisfied after this change). For brevity, let us write $\theta_{J,\hat{Z}} := |J \cap \hat{Z}|$ and $h_{[i]} := U_i h^{(i)}$. Fixing $J \in \mathscr{J}$ and $h \in \mathbf{R}^N$, we can estimate:

[2] Note that $\xi \in \{\lceil\frac{\omega}{C}\rceil,\ldots,\omega\}$.

$$\begin{aligned}
\mathbf{E}[\phi_J(h_{[\hat{Z}]})] &\overset{(2)}{=} \mathbf{E}\left[\phi_J\left(\sum_{i\in\hat{Z}} h_{[i]}\right)\right] = \mathbf{E}\left[\phi_J\left(\sum_{i\in\hat{Z}\cap J} h_{[i]}\right)\right] \\
&= \mathbf{E}\left[\phi_J\left(\tfrac{1}{\theta_{J,\hat{Z}}}\sum_{i\in\hat{Z}\cap J}\theta_{J,\hat{Z}}h_{[i]}\right)\right] \le \mathbf{E}\left[\tfrac{1}{\theta_{J,\hat{Z}}}\sum_{i\in\hat{Z}\cap J}\phi_J\left(\theta_{J,\hat{Z}}h_{[i]}\right)\right] \\
&\overset{(10)}{=} \mathbf{E}\left[\tfrac{1}{\theta_{J,\hat{Z}}}\left(\tfrac{\theta_{J,\hat{Z}}}{C\xi}\sum_{i\in J}\phi_J\left(\theta_{J,\hat{Z}}h_{[i]}\right)\right)\right] = \frac{1}{C\xi}\mathbf{E}\left[\sum_{i\in J}\phi_J\left(\theta_{J,\hat{Z}}h_{[i]}\right)\right] \\
&= \frac{1}{C\xi}\mathbf{E}\left[\sum_{i\in[n]}\phi_J\left(\theta_{J,\hat{Z}}h_{[i]}\right)\right]. \qquad (16)
\end{aligned}$$

In the second equation above we have used the assumption that ϕ_J depends on blocks $i \in J$ only. The only inequality above follows from convexity of ϕ_J. Note that this step can only be performed if the sum is over a nonempty index set, which happens precisely when $\theta_{J,\hat{Z}} \ge 1$. This technicality can be handled at the expense of introducing a heavier notation (which we shall not do here), and (16) still holds. Finally, in one of the last steps we have used (10) with $\kappa(|\hat{Z}\cap J|, i) \leftarrow \phi_J(\theta_{J,\hat{Z}}h_{[i]})$.

By summing up inequalities (16) for $J \in \mathscr{J}$, we get

$$\begin{aligned}
\mathbf{E}\left[\phi(h_{[\hat{Z}]})\right] &\overset{(12)}{=} \sum_{J\in\mathscr{J}}\mathbf{E}\left[\phi_J(h_{[\hat{Z}]})\right] \overset{(16)}{\le} \frac{1}{C\xi}\sum_{J\in\mathscr{J}}\mathbf{E}\left[\sum_{i\in[n]}\phi_J\left(\theta_{J,\hat{Z}}h_{[i]}\right)\right] \\
&\overset{(12)}{=} \frac{1}{C\xi}\mathbf{E}\left[\sum_{i\in[n]}\phi\left(\theta_{J,\hat{Z}}h_{[i]}\right)\right] \overset{(15)}{\le} \frac{1}{C\xi}\mathbf{E}\left[\sum_{i\in[n]}\tfrac{L_i}{2}\|\theta_{J,\hat{Z}}h^{(i)}\|^2_{(i)}\right] \\
&= \frac{1}{2C\xi}\mathbf{E}\left[\theta^2_{J,\hat{Z}}\sum_{i\in[n]}L_i\|h^{(i)}\|^2_{(i)}\right] \overset{(3)}{=} \frac{1}{2C\xi}\|h\|^2_w\mathbf{E}\left[\theta^2_{J,\hat{Z}}\right]. \qquad (17)
\end{aligned}$$

We now need to compute $\mathbf{E}[\theta^2_{J,\hat{Z}}]$. Note that the random variable $\theta_{J,\hat{Z}}$ is the sum of C independent random variables $\theta_{J,\hat{Z}} = \sum_{c=1}^{C}\theta_{J,\hat{Z}^{(c)}}$, where $\theta_{J,\hat{Z}^{(c)}}$ has the simple law

$$\mathbf{Prob}(\theta_{J,\hat{Z}^{(c)}} = k) = \binom{\xi}{k}\binom{s-\xi}{\tau-k}\Big/\binom{s}{\tau}.$$

We therefore get

$$\begin{aligned}
\mathbf{E}[\theta^2_{J,\hat{Z}}] &= \mathbf{E}[(\sum_{c=1}^{C}\theta_{J,\hat{Z}^{(c)}})^2] = C\mathbf{E}[(\theta_{J,\hat{Z}^{(c)}})^2] + C(C-1)(\mathbf{E}[\theta_{J,\hat{Z}^{(c)}}])^2 \\
&= C\tfrac{\xi\tau}{s}\left(1+\tfrac{(\xi-1)(\tau-1)}{\max\{1,s-1\}}\right) + C(C-1)\left(\tfrac{\xi}{s}\tau\right)^2. \qquad (18)
\end{aligned}$$

It only remains to combine (17) and (18). □ □

Note that ESO inequalities have recently been used in the analysis of distributed coordinate descent methods by Richtárik and Takáč [23] and Fercoq et al [4]. However, their assumptions on f and derivation of ESO are very different and hence our results apply to a different class of functions.

5 Iteration complexity

In this section, we state two iteration complexity results for Algorithm 1. Theorem 2 deals with non-strongly convex objective and shows that the algorithm achieves sub-linear rate of convergence $\mathscr{O}(\frac{1}{\varepsilon})$. Theorem 3 shows Algorithm 1 achieves linear convergence rate $\mathscr{O}(\log\frac{1}{\varepsilon})$ for a strongly convex objective.

However, we wish to stress that in high dimensional settings, and especially in applications where low or medium accuracy solutions are acceptable, the dependence of the method on ε is somewhat less important than its dependence on data size through quantities such as dimension N and the number of blocks n, and on quantities such as the number of computers C and number of parallel updates per computer τ, which is related to the number of cores.

Notice that once the ESO is established by Theorem 1, the complexity results, Theorems 2 and 3, follow from the generic complexity results in [36] and [22], respectively.

5.1 Convex case

Theorem 2 (Based on [36]). *Let f satisfy Assumption 1 and sampling $\hat{Z}$ satisfy Assumption 3. Let x_k be the iterates of Algorithm 1 applied to problem* (1)*, where parameters β and w are chosen as in Theorem 1 and the random sets Z_k are iid, following the law of $\hat{Z}$. Then for all $k \geq 1$,*

$$\mathbf{E}[F(x_k) - F^*] \leq \frac{n}{n + C\tau k}\left(\frac{\beta}{2}\|x_0 - x^*\|_w^2 + F(x_0) - F^*\right). \tag{19}$$

Note that the leading term in the bound decreases as the number of blocks updated in a single (parallel) iteration, $C\tau$, increases. However, notice that the parameter β also depends on C and τ. We shall investigate this phenomenon in Section 5.3 and show that the level of speedup one gets by increasing C and/or τ depends (where by speedup we mean the decrease of the upper bound established by the theorem) the degree of separability ω of f. The smaller ω is, the more speedup one obtains.

5.2 Strongly convex case

If we assume that F is strongly convex with respect to the norm $\|\cdot\|_w$ then the following theorem shows that $F(x_k)$ converges to F^* linearly, with high probability.

Definition 2 (Strong convexity). *Function $\phi : \mathbf{R}^N \to \mathbf{R} \cup \{+\infty\}$ is strongly convex with respect to the norm $\|\cdot\|_w$ with convexity parameter $\mu_\phi(w) \geq 0$ if*

$$\phi(y) \geq \phi(x) + \langle \phi'(x), y - x\rangle + \tfrac{\mu_\phi(w)}{2}\|y - x\|_w^2, \quad \forall x, y \in dom\phi, \tag{20}$$

where $\phi'(x)$ is any subgradient of ϕ at x. The case with $\mu_\phi(w) = 0$ reduces to convexity.

Strong convexity of F may come from f or Ω (or both); we write $\mu_f(w)$ (resp. $\mu_\Omega(w)$) for the (strong) convexity parameter of f (resp. Ω). It follows from (20) that $\mu_F(w) \geq \mu_f(w) + \mu_\Omega(w)$.

Theorem 3 (Based on [22]). *Let us adopt the same assumptions as in Theorem 2. Moreover, assume that F is strongly convex with $\mu_f(w) + \mu_\Omega(w) > 0$. Choose initial point $x_0 \in \mathbf{R}^N$, target confidence level $0 < \rho < 1$, target accuracy level $0 < \varepsilon < F(x_0) - F^*$ and*

$$K \geq \frac{n}{C\tau}\frac{\beta + \mu_\Omega(w)}{\mu_f(w) + \mu_\Omega(w)}\log\left(\frac{F(x_0) - F^*}{\varepsilon\rho}\right). \tag{21}$$

If $\{x_k\}$ are the random points generated by Algorithm 1, then $\mathbf{Prob}(F(x_K) - F^* \leq \varepsilon) \geq 1 - \rho$.

Notice that now both ε and ρ appear inside a logarithm. Hence, it is easy to obtain accurate solutions with high probability.

5.3 Parallelization speedup is governed by sparsity

If we assume that $\|x_0 - x^*\|_w^2 \gg F(x_0) - F^*$, then in view of Theorem 2, the number of iterations required by our method to get an ε solution in expectation is $O(\frac{\beta}{C\tau})$. Hence, the smaller $\frac{\beta}{C\tau}$ is, the fewer iterations are required. If β were a constant independent of C and τ, one would achieve linear speedup by increasing workload (i.e., by increasing $C\tau$). However, this is the case for $C = 1$ and

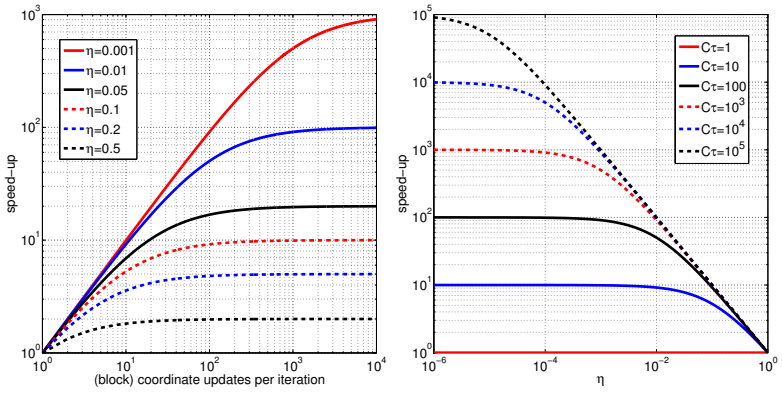

Figure 1: Speedup gained from updating more blocks per iteration is almost linear initially, and depending on sparsity level η, breaks down to significant sublinear.

$\omega = 1$ only (see Theorem 1). Let us look at the general case. If we write $\eta := \frac{\xi}{s}$ (this a measure of sparsity of the partitioned data), then

$$\frac{\beta}{C\tau} \overset{(11)}{=} \frac{1+\frac{(\xi-1)(\tau-1)}{\max\{1,s-1\}}+(C-1)\frac{\xi\tau}{s}}{C\tau} \leq \frac{1+\frac{\xi(\tau-1)}{s}+(C-1)\frac{\xi\tau}{s}}{C\tau}$$
$$= \frac{1+\eta(\tau-1)+(C-1)\eta\tau}{C\tau} = \frac{1+\eta(C\tau-1)}{C\tau} = \frac{1}{C\tau}+\eta\left(1-\frac{1}{C\tau}\right).$$

As expected, the first term represents linear speedup. The second term represents a penalty for the lack of sparsity (correlations) in the data. As $C\tau$ increases, the second term becomes increasingly dominant, and hence slows the speed-up from almost linear to none. Notice that for fixed η, the ratio $\frac{\beta}{C\tau}$ as a function of $C\tau$ is decreasing and hence we always get *some* speedup by increasing $C\tau$.

Figure 1 (left) shows the speedup factor ($\frac{C\tau}{\beta}$; high values are good) as a function of $C\tau$ for different sparsity levels η. One can observe that sparse problems achieve almost linear speed-up even for bigger value of $C\tau$, whereas for, e.g., $\eta = 0.2$, almost linear speedup is possible only up to $C\tau = 10$. For sparser data with $\eta = 0.01$, linear speedup can be achieved up to $C\tau = 100$. For $\eta = 0.001$, we can use $C\tau = 10^3$. The right part of Figure 1 shows how sparsity affects speedup for a fixed number of updates $C\tau$. Again, the break-point of almost linear speedup is visibly present.

Similar observations in the non-distributed setting were reported in [22]. The phenomenon is not merely a byproduct of our theoretical analysis, it also appears in practice.

5.4 The cost of distribution

Notice that in a certain intuitive sense, variants of Algorithm 1, which at each iteration update the same number of blocks, are comparable. This is because we can vary C and τ while keeping the product constant. In particular, let us consider two situations:

1. We use C computers, each updating τ blocks in parallel, and
2. We use 1 computer, updating $C\tau$ blocks in each iteration in parallel.

By placing these two variants of our method side by side for a comparison, we are implicitly assuming that the underlying problem is small enough so that it can be stored on and solved by a single computer. Nevertheless, let us see where this leads us.

Assume that F is strongly convex, $\mu(\Omega) = 0$ and $s = \frac{n}{C} \geq 2$. Similar comparisons can be made in different settings, but for simplicity we do the comparison in this case only. When comparing the complexity bounds (21), we notice that the only difference is in the value of β. Let β_1 be the β parameter in the first situation, and β_2 be the β parameter in the second situation. In this situation, the ratio of the complexity bounds (21) is equal to the ratio of the β parameters:

$$\frac{\beta_1}{\beta_2} = (1+\tfrac{(\xi-1)(\tau-1)}{s-1}+(C-1)\tfrac{\xi\tau}{s})/(1+\tfrac{(\omega-1)(C\tau-1)}{Cs-1}).$$

Notice that $\frac{\omega}{C} \leq \xi \leq \omega$ and that the ratio β_1/β_2 is increasing in ξ. We thus obtain the following bounds:

$$\text{LB} := \frac{1+\frac{(\omega-C)(\tau-1)}{n-C}+(C-1)\frac{\omega\tau}{n}}{1+\frac{(\omega-1)(C\tau-1)}{n-1}} \leq \frac{\beta_1}{\beta_2} \leq \frac{1+\frac{(\omega-1)(C\tau-C)}{n-C}+(C-1)\frac{\omega C\tau}{n}}{1+\frac{(\omega-1)(C\tau-1)}{n-1}} =: \text{UB}.$$

Table 1 presents the values of LB and UB for various parameter choices and problem sizes. We observe that the ratio β_1/β_2 is small, at most 2 in the table, which means that *by distributing the computation, the method will at most double the number of iterations.* However, larger factors are possible for different settings of the parameters. A *theoretical* result of this type (with a factor of 2) was obtained for a different class of functions f in [23] (the latter result was recently improved in [4] to the factor $1+1/(\tau-1)$ whenever $\tau > 1$).

Table 1: Lower and upper bounds on β_1/β_2 for a selection parameters n, ω, C and τ.

n	ω	C	τ	β_2	LB	UB
10^6	10^2	10	50	1.049	1.0000086	1.4279673
10^7	10^2	10	50	1.005	1.0000009	1.0446901
10^8	10^2	100	100	1.009	1.0000009	1.9801989

Of course, if the problem size exceeds available memory of a single computer, the second option is not available to us. However, it is reassuring to know that the price we pay, in terms of the number of iterations, is limited. Of course, the major cost associated with a distributed method is in communication. We shall discuss communication issues in the next section.

6 Implementation and communication

In this section we give a brief overview of the distributed architecture we consider. Afterwards, we will discuss a particular implementation of the (block) coordinate descent algorithm in a such environment. This brings up numerous difficulties and questions, which we focus on.

Message Passing Interface (MPI). MPI is am established interface for passing data from one computer (MPI process) to another computer (MPI process).[3] Communication can involve any subset of computers. It can be both blocking ("synchronous") or non-blocking ("asynchronous"). As the name suggests, if one of the computers in an asynchronous transfer is busy, e.g., the receiving one is busy receiving some data, the other computer need not wait, but can do some other work instead. If more than two computers are involved in the communication, it is called a *collective* operation. An example of a collective operation can be, e.g., a *barrier*, where computers are waiting until all of them reach the same point in the algorithm, or the commonly used *reduce all* operation. To explain a *reduce all* operation, let us assume that each computer stores a vector $\delta g^{(c)} \in \mathbf{R}^m$ and the goal is to sum it up, i.e. to compute $\delta g^{(1,\ldots,C)} = \sum_{c=1}^{C} \delta g^{(c)}$ and to make this result available on each computer. Figure 2 shows a standard approach which leads to the desired result. Let us note that from the performance point of view, the use of this operation should be minimized, as it not only involves an implicit synchronisation, but also leaves most of the computers idle throughout the operation. Nevertheless, we present a naïve implementation approach using *reduce all* first, before we improve upon the implementation.

6.1 Sparse optimization and SVM dual

Although our algorithm and results apply to a rather broad class of non-smooth convex functions, we focus on two important problems in statistics and machine learning in describing our computational experience.

Sparse optimization via L1 regularization. In a number of problems in these fields, it is beneficial to find a solution x with only a few non-zero elements, not least to aid its interpretability.

[3] See [32] for the reference documentation.

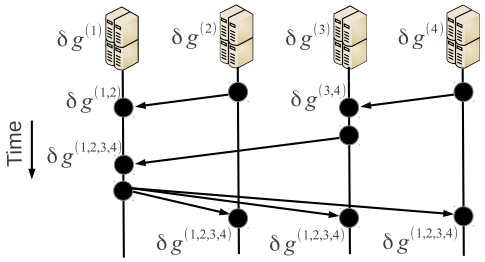

Figure 2: Schematic diagram of a standard reduce all implementation. The goal is to compute $\sum_{c=1}^{C} \delta g^{(c)}$. The arrows show data flow between computers.

It has been recognized that the inclusion of the number of non-zero elements, $\|x\|_0$, in the objective function raises the complexity of many efficiently solvable problems to NP-Hard [15, 7]. Recently, there are versions of randomized coordinate descent methods which try to handle ℓ_0 norm directly, however, only convergence to local minimum can be guaranteed [20]. Fortunately, the inclusion of the sum of absolute values, $\|x\|_1$, provides a provably good proxy, which is also known as ℓ_1 regularization. There is a large and growing body of work on both practical solvers for non-smooth convex problems, obtained by such a regularisation, and their convergence properties, when one restricts oneself to a single computer storing the complete input. Such solvers are, however, most useful in high-dimensional applications, e.g., in applications of machine learning to Genomics and Proteomics. There, the size of the data sets often exceeds the capacity of random-access memory of any single computer available today. We hence present a distributed method for a class of problems, which includes many ℓ_1 regularized problems.

Given a matrix $A \in \mathbf{R}^{m\times N}$, a compatible vector $y \in \mathbf{R}^m$, and constant $\gamma > 0$, the goal is to find a vector $x \in \mathbf{R}^N$ which is a solution of following optimization problem:

$$\min_{x\in\mathbf{R}^N} F(x) := \underbrace{\gamma\|x\|_1}_{\Omega(x)} + \underbrace{\sum_{j=1}^{m} \mathscr{L}(x, A_{j:}, y^{(j)})}_{f(x)}, \tag{22}$$

where $A_{j:}$ denotes j-th row of matrix A and $\mathscr{L}$ is a loss function, such as

$$\mathscr{L}_{SL}(x, A_{j:}, y^{(j)}) := \frac{1}{2}(y^{(j)} - A_{j:}x)^2, \qquad \text{square loss,} \tag{SL}$$

$$\mathscr{L}_{LL}(x, A_{j:}, y^{(j)}) := \log(1 + e^{-y^{(j)}A_{j:}x}), \qquad \text{logistic loss,} \tag{LL}$$

$$\mathscr{L}_{HL}(x, A_{j:}, y^{(j)}) := \frac{1}{2}\max\{0, 1 - y^{(j)}A_{j:}x\}^2, \qquad \text{hinge square loss.} \tag{HL}$$

The input (A, y) is often referred to as the training data. Rows of matrix A represent observations of N features each. For binary classifications problem, $y \in \{-1, 1\}^m$ and (22) is solved with (LL) or (HL) loss. The goal is to find a hyperplane balancing "sparsity" and separation of the differently-classified observations. Notice that in this settings there are blocks of dimension 1. Sparse support vector machines (SVMs), also known as ℓ_1-norm SVMs, ℓ_1-regularized SVMs and ℓ_1-regularized linear classification [40], have proven essential in deriving interpretable results in applications with many irrelevant features [**?**], such as Genomics and Proteomics.

Dual of SVM with hinge loss. In machine learning, hinge loss is popular choice of $\mathscr{L}$, but is not smooth. It is well know that the dual has the form [8, 31, 33]:

$$\min_{x\in\mathbf{R}^m} F(x) := \underbrace{\frac{1}{2\lambda m^2}x^T Q x - \frac{1}{m}x^T \mathbf{1}}_{f(x)} + \underbrace{\sum_{i=1}^{m} \Phi_{[0,1]}(x^{(i)})}_{\Omega(x)}, \tag{SVM-DUAL}$$

where $\Phi_{[0,1]}$ is an indicator function of interval $[0, 1]$ and $Q \in \mathbf{R}^{m\times m}$ is the Gram matrix of the data, i.e. $Q_{i,j} = y^{(i)}y^{(j)}A_{i:}A_{j:}^T$. If x^* is optimal solution of (SVM-DUAL) then $w^* = w^*(x^*) =$

$\frac{1}{\lambda m}\sum_{i=1}^{m} y^{(i)}(x^*)^{(i)}A_{i:}^T$ is optimal solution of the primal problem

$$\min_{w\in\mathbf{R}^N} P(w) := \frac{1}{N}\sum_{i=1}^{N}\mathscr{L}(w,A_{i:},y^{(i)}) + \frac{\lambda}{2}\|w\|^2, \tag{23}$$

where $\mathscr{L}(w,A_{i:},y^{(i)}) = \max\{0,1-y^{(i)}A_{i:}w\}$.

6.2 Implementation details for sparse least squares

In this section, we focus on implementing the distributed coordinate descent for sparse (L1 regularized) least squares. The key components needed by Algorithm 1 are the computation of L_i, $\nabla_i f(x_k)$ and then solving of a 1D minimization problem. Note that $\nabla_i f(x) = \sum_{j=1}^{m} -A_{j,i}(y^{(j)} - A_{j:}x)$ and $L_i = \|A_{:i}\|_2^2$. The only difficulty is that given the data partition $\{P^{(c)}\}_{c=1}^{C}$, no single computer c is able to compute $\nabla_i f(x)$ for any $i \in P^{(c)}$. The reasoning follows from a simple observation: if we wanted to compute $\nabla_i f(x_k)$ for a given x_k from scratch, we would have to access all coordinates of x_k, vector y, and all non-zero elements of the input matrix A. This could be avoided by introducing an auxiliary vector $g_k := g(x_k)$ defined as

$$g_k := Ax_k - y. \tag{24}$$

Once the value of $g_k = g(x_k)$ is available, a new iterate is defined as

$$x_{k+1} = x_k + \sum_{c=1}^{C}\sum_{i\in Z_k^{(c)}} U_i h^{(i)}(x_k) \tag{25}$$

then $g_{k+1} = g(x_{k+1})$ can be easily expressed as

$$g_{k+1} = g_k + \sum_{c=1}^{C}\underbrace{\sum_{i\in Z_k^{(c)}} h^{(i)}(x_k)A_{:i}}_{\delta g^{(c)}}. \tag{26}$$

Note that the value $\delta g^{(c)}$ can be computed on computer c as all required data are available on computer c. Using a *reduce all* operation, as discussed above, we can obtain g_{k+1}. Subsequently, the formula for $\nabla_i f(x)$ will take following form $\nabla_i f(x) = A_{:i}^T g = \sum_{j=1}^{m} A_{j,i}g^{(j)}$.[4] Once we know how to compute $\nabla_i f(x)$ and L_i, all which remains to be done is to solve the 1D problem

$$\min_{t\in\mathbf{R}} a + bt + \frac{c}{2}t^2 + \lambda|d+t|, \tag{27}$$

where $a,b,d \in \mathbf{R}$ and $c,\lambda \in \mathbf{R}_{++}$, which is given by a *soft-thresholding* formula $t^* = \mathrm{sgn}(\zeta)(|\zeta| - \frac{1}{c})_+ - d$, where $\zeta = d - \frac{b}{c}$.

6.3 Implementation details for training sparse SVM

In this section, we focus on a distributed coordinate descent for sparse support vector machines (SVM). In this case, we define

$$g_k := \frac{1}{\lambda m}\sum_{i=1}^{m} x_k^{(i)} y^{(i)} A_{i:}^T. \tag{28}$$

Then

$$\nabla_i f(x) = \frac{y^{(i)}A_{i:}g_k - 1}{m}, \qquad L_i = \frac{\|A_{i:}\|^2}{\lambda m^2}. \tag{29}$$

[4] The formula for (LL) and (HL) are similar and can be found e.g. in [23].

The optimal step length is then solution of the following one dimensional problem:

$$h^{(i)}(x_k) = \arg\min_{t\in\mathbf{R}} \nabla_i f(\alpha)t + \frac{\beta}{2}L_i t^2 + \Phi_{[0,1]}(\alpha^{(i)}+t) \tag{30}$$

$$= \mathrm{clip}_{[-\alpha^{(i)},1-\alpha^{(i)}]}\left(\frac{\lambda m(1-y^{(i)}A_{i:}g_k)}{\beta\|A_{i:}\|^2}\right), \tag{31}$$

where for $a<b$

$$\mathrm{clip}_{[a,b]}(\zeta) = \begin{cases} a, & \text{if } \zeta < a, \\ b, & \text{if } \zeta > b, \\ \zeta, & \text{otherwise.} \end{cases}$$

The new value of the auxiliary vector $g_{k+1} = g(x_{k+1})$ is given by

$$g_{k+1} = g_k + \sum_{c=1}^{C} \underbrace{\sum_{i\in Z_k^{(c)}} \frac{1}{\lambda m} h^{(i)}(x_k) y^{(i)} A_{i:}^T}_{\delta g^{(c)}} \tag{32}$$

and the duality gap $G(x_k) = P(g_k) + F(x_k)$ can be easily obtained as

$$G(x_k) = \frac{1}{m}\sum_{i=1}^{m}(\mathscr{L}(g_k, A_{i:}, y^{(i)}) - x_k^{(i)}) + \lambda\|g_k\|^2.$$

7 Runtime complexity

Due to the trick with an auxiliary vector g_k, which has been introduced in the previous subsection, Algorithm 1 has two alternating and time consuming sub-procedures, namely

1. *computation of an update:* $\sum_{i\in Z_k^{(c)}} U_i h^{(i)}(x_k)$ and the corresponding *accumulation of* g_k*:* $\delta g^{(c)}$,
2. updating g_k to g_{k+1}.

Let us denote the runtime of the first sub-procedure by $\mathscr{T}_1(\tau)$ (note that this depends on τ) and by $\mathscr{T}_2$ the runtime of a second one. We will neglect the rest of the runtime cost, such as managing a loop, evaluation of termination criteria, measuring a computation time, etc. The total runtime cost $\mathscr{T}_T$ is hence given by

$$\mathscr{T}_T = \mathscr{O}\left(\frac{\beta}{C\tau}(\mathscr{T}_1(\tau) + \mathscr{T}_2)\right). \tag{33}$$

Let us now for simplicity assume that the first sub-procedure is linear in τ, i.e. $\mathscr{T}_1(\tau) = \tau\mathscr{T}_1(1) =: \tau\mathscr{T}_1$. Then

$$\mathscr{T}_T = \mathscr{O}\left(\frac{\beta}{C\tau}(\tau\mathscr{T}_1 + \mathscr{T}_2)\right). \tag{34}$$

Numerical values of $\mathscr{T}_1$ and $\mathscr{T}_2$ could be estimated, given problem sparsity and underlying hardware, or can be measured during the run.

Optimal choice of sampling parameter τ. In the previous paragraph, we gave an estimate of the complexity of a single iteration. In this paragraph, we answer a question how to choose a τ given times $\mathscr{T}_1, \mathscr{T}_2$. For variable β, we have more options, but we stick to the most general one given in (11). Given that $s \geq 2$, we have

$$\mathscr{T}_T = \mathscr{O}\left(\frac{1+\frac{(\xi-1)(\tau-1)}{s-1}+(C-1)\frac{\xi\tau}{s}}{C}\left(r_{1,2}+\frac{1}{\tau}\right)\mathscr{T}_2\right) \lesssim \left(\frac{s}{\xi C}+\tau\right)\left(r_{1,2}+\frac{1}{\tau}\right) \tag{35}$$

where $r_{1,2} = \frac{\mathscr{T}_1}{\mathscr{T}_2}$ is a work to communication ratio. The optimal parameter τ^* can be obtain by minimizing (35) and is given by

$$\tau^* = \sqrt{\frac{s}{r_{1,2}\ \xi\ C}}. \tag{36}$$

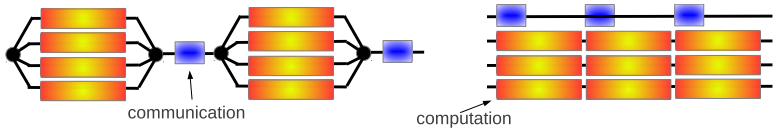

Figure 3: An illustration of naïve (PS) approach (left) which alternate between parallel regions where computations take place with serial regions dedicated to MPI communications with other computers. An alternative (FP) approach (right) dedicates the communication task to one thread and use other threads to computation.

Therefore, smaller values of $r_{1,2}$ imply that we should do more work in each iteration, and hence bigger values of τ should be chosen. This is quite natural, as one should tune the parameters in such a way that time spend in communication should be in comparable with that of effective computation.

The naïve implementation discussed in previous paragraphs has a number of drawbacks. Let us mention them in turn and suggest improvements:

Alternating Parallel and Serial regions (PS). The naïve implementation alternates two subprocedures. One, which is computationally heavy and done in parallel, but with no MPI communication, and another one, which is purely communicational. As an easy fix, one can dedicate 1 thread to the communication and other threads within the same computer for computation. We call this approach **Fully Parallel** (FP)). Figure 3 compares the naïve strategy (left) with the proposed one (right),

Reduce All (RA). As mentioned above, the use of *reduce all* operations significantly decreases the performance of many distributed algorithms. It is, however, the preferred form of communication between computers close to each other in the computer network, such as computers directly connected by a network cable. The use of asynchronous methods is also preferred over synchronous methods.

Asynchronous StreamLined (ASL). We propose another pattern of communication, where each computer in one iteration sends only one message to the closest computer, asynchronously, and receives only one message from another computer close-by, asynchronously. The communication hence takes place in an ring. This tweak, however, requires a significant change in the algorithm. Figure 4 illustrates the data flow of messages at the end of k-th iteration for $C = 4$.

We fix an order of computers in a ring, denoting $\mathrm{pred}_R(c)$ and $\mathrm{succ}_R(c)$ the two computers neighbouring computer c along the two directions on the ring. Computer c always receives data only from computer $\mathrm{pred}_R(c)$ and sends data only to computer $\mathrm{succ}_R(c)$. Let us denote by $\delta G_k^{(c)}$ the data, which computer c sends to computer $\mathrm{succ}_R(c)$ at the end of iteration k. When computer c starts an iteration k, it already received by the end of previous iteration $\delta G_{k-1}^{(\mathrm{pred}_R(c))}$.[5] Hence the data, which will be send at the end of k-th iteration by computer c, is given as follows

$$\delta G_k^{(c)} = \delta G_{k-1}^{(\mathrm{pred}_R(c))} - \delta g_{k-C}^{(c)} + \delta g_k^{(c)}. \tag{37}$$

It should be noticed that at the end of each iteration in the ASL procedure, each computer has different vector g_k, which we denote $g_k^{(c)}$. The update rule is hence given by

$$g_{k+1}^{(c)} = g_k^{(c)} + \delta g_k^{(c)} + \delta G_k^{(\mathrm{pred}_R(c))} - \delta g_{k-C+1}^{(c)}. \tag{38}$$

The clear advantage of ASL method is the decrease of a communication time. On the other hand it comes with a cost of slower propagation of information. Indeed, it takes $C-1$ iterations to propagate information to all computers. It also comes with bigger storage requirements, as at iteration k, we have to have all vectors $\delta g_l^{(c)}$ for $k-C \le l \le k$ stored on computer c.

Asynchronous Torus (AST). There is a compromise solution, though, which inherits many desirable features of both RA and SLA. The asynchronous torus uses the fact that modern high-performance computng often assumes the network topology, which is known as a "torus", and is

[5] For the start of the algorithm we define $\delta g_l^{(c)} = \delta G_l^{(c)} = \mathbf{0}$ for all $l < 0$.

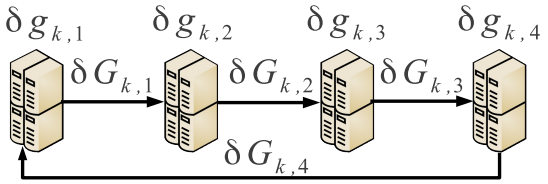

Figure 4: Illustration of ASL method for $C = 4$. During k-th iteration, computer c obtains its contribution $\delta g_k^{(c)}$ but asynchronically sends an accumulated update $\delta G_k^{(c)}$ to his successor.

Table 2: Summary of additional memory and computation requirements for strategies RA, SLA, AST.

strategy	memory for g's	communication	extra computation
RA	$2m$	$\mathscr{T}_{ra}$	0
SLA	$(2+C)m$	$\mathscr{T}_{p2p}$	$4m$ additions
AST	$(2+C/r)m$	$\mathscr{T}_{p2p} + \mathscr{T}_{ra}/r$	$8m$ additions

inherent to InfiniBand [9], a standard. Let us assume that C is a multiple of $r \in N$, where r represents a width of a torus, i.e. C computers are partitioned into subsets R_i each with size r. Each group R_i has a root computer. These root computers aggregate updates from their respective groups, e.g. using a local *reduce all* operation, in each iteration and exchange those update in an asynchronous ring with two other adjacent root computers. That is the communication between the root nodes follows the ASL communication pattern. The AST approach decreases the propagation time from C to $\frac{C}{r}$, additional storage is also decrease by factor r, and the overall communication complexity remains low.

Runtime and Storage Complexity. Changing from FP approach to PS approach does not require much computational or storage overhead, but can reduce the idle time of processors. However, changing from RA to SLA or AST brings significant storage requirements, while it reduces both communication and idle time significantly. Table 2 summarize maximum memory requirements on each single node of the cluster, time spent in communication, and amount of data transferred over the network. Once the time spent in communication is measured or estimated, one can pick the most appropriate strategy. Notice that the time of the *reduce all* operation, $\mathscr{T}_{ra}$, which is of the order of $\mathscr{T}_{ra} = \mathscr{O}(\log C) \cdot \mathscr{T}_{p2p}$.

8 Numerical experiments

In this section we present numerical evidence of efficiency of distributed (block) coordinate descent method. First, we describe further details of the implementation. Then, we present results for a huge instance of the sparse least squares problem ("LASSO", SL), whose data matrix has more than 3 TB. We also test SVM problems for various large ML datasets.

The implementation details. Distributed (block) coordinate descent solver is part of our AC-DC library, available at `http://code.google.com/p/ac-dc/`. The library is written in C++ using Boost::MPI and OpenMP. The use of templates and Boost.Serialization make it easy to change the composite objective functions and the precision of the computation. A particular care has been taken to measure the resource utilization faithfully. Both wall-clock and CPU-time has been measured using Boost::Timers, which achieve nano-second accuracy on recent processors running recent versions of Linux.

The facility. Our empirical tests were conducted in high performance computing facility (HECToR) equipped with multi-core computers connected using Infiniband [9]. In the larger Hector Phase 3 facility, we have used up to 128 nodes in a Cray XE6 cluster, equipped with two AMD Opteron Interlagos 16-core processors and 32 GB of memory each (giving 4,096 cores in total). The computers ran Cray Linux Environment, which is based on SuSE Linux. The computers are connected using Cray Gemini routers in a 3D torus. Every two computers share a Gemini router, which is connected to both the processors and the random-access memory directly via the HyperTransport links, and

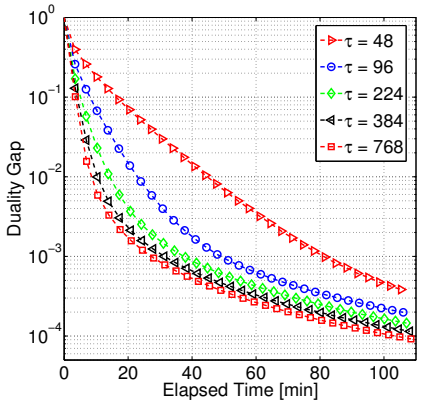

Figure 5: Evolution of duality gap for the WebSpam dataset for various choices of τ.

every router is connected to ten other routers. In practice, the latency is about 1-1.5 microseconds and the capacity of each link is 8 GBs^{-1}.

Support Vector Machine (SVM). In machine learning problems, the size of data can easily grow to terabytes. Parallel algorithms are hence a necessity. Parallel version of LIBSVM using MPI was studied [28] and there is also the PSVM library (see [3]), where parallel row-based incomplete Cholesky Factorization (CF) is performed in an interior point approach. A MapReduce-based distributed algorithm for SVM was found to be effective in automatic image annotation [1].

One of the goals of this paper it to train huge sparse SVM which doesn't fit to the memory of a single computer. In machine learning literature, one often performs experiments on instances of moderate size, e.g. 100 MBs (see e.g. [30, 8, 33]). Well-known instances include, e.g., *CCAT, Astro-ph, COV*. In this Section, we focus on larger datasets, namely on the WebSpam dataset from [11]. This dataset has 350,000 observations (rows) and 16,609,143 features (columns). The size of input data is 25 GBs. Figure 5 show the execution time and duality gap for WebSpam dataset. In this case we used $C = 16$ MPI processes and each process used 8 threads. represents how many coordinates were updated by one MPI process during one iteration. As expected, the main runtime cost it not computing the updates, but updating g. Let us remark that ε is usually not particularly small in machine learning community. In experimenting with small ε, we just wanted to demonstrate that our algorithm is able to close the duality gap within the limits of machine precision. The truly important measures of the performance of the classifier, e.g., 0-1 loss or prediction error, are actually within 10 % after the first minute, already, which is the first time we compute it. Hence, the duality gap of 0.1 or 0.01 can be sufficient for machine learning problems, in practice.

Sparse least squares (LASSO). We now move on to solving an artificial big data LASSO problem with matrix A in block angular form, depicted in (39). We have used 128 nodes, where there were 4 MPI processes on each node, and each MPI process run 8 OpenMP threads, giving the total of 4,096 (hardware) threads. The data matrix:

$$A = \left(\begin{array}{c|c|c|c} A_{loc}^{(1)} & 0 & \cdots & 0 \\ 0 & A_{loc}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hline A_{glob}^{(1)} & A_{glob}^{(2)} & \cdots & A_{glob}^{(C)} \end{array}\right) \tag{39}$$

has $n = 10^9$ rows, $d = 510^8$ columns, and amounts to 3 TB. Such matrices often arise in stochastic optimization. Each node c stores two matrices: $A_{loc}^{(c)} \in \mathbf{R}^{1,952,148\times 976,562}$ and $A_{glob}^{(c)} \in \mathbf{R}^{500,224\times 976,562}$. The average number of nonzero elements per row in the local part of $A^{(c)}$ is 175, and $1,000$ for the global part. Optimal solution x^* has exactly $160,000$ nonzero elements. Figure 6 compares the evolution of $F(x_k) - F^*$ for ASL-FP and RA-FP.

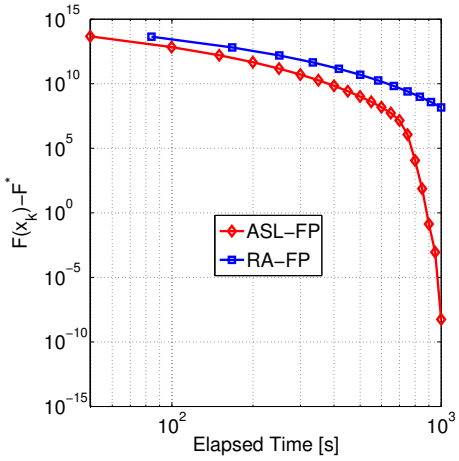

Figure 6: Evolution of $F(x_k) - F^*$ in time. ASL-FP significantly outperforms RA-FP. The loss F is pushed down by 25 degrees of magnitude in less than 30 minutes (3TB problem).

Remark: When communicating $g_k^{(c)}$, only entries corresponding to the global part of $A^{(c)}$ need to be communicated, and hence in RA, a *reduce all* operation is applied to vectors $\delta g_{glob}^{(c)} \in \mathbf{R}^{500,224}$. In ASL, vectors with the same length are sent.

References

[1] Nasullah Khalid Alham, Maozhen Li, Yang Liu, and Suhel Hammoud. A mapreduce-based distributed SVM algorithm for automatic image annotation. *Computers & Mathematics with Applications*, 62(7):2801 – 2811, 2011.

[2] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, September 1999.

[3] Edward Y. Chang, Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, and Hang Cui. PSVM: Parallelizing support vector machines on distributed computers. In *NIPS*, 2007.

[4] Olivier Fercoq, Zheng Qu, Peter Richtárik, and Martin Takáč. Fast distributed coordinate descent for non-strongly convex losses. *arXiv:1405.5300*, 2014.

[5] Olivier Fercoq and Peter Richtárik. Accelerated, parallel and proximal coordinate descent. *arXiv:1312.5799*, 2013.

[6] Olivier Fercoq and Peter Richtárik. Smooth minimization of nonsmooth functions with parallel coordinate descent methods. *arXiv:1309.5885*, 2013.

[7] Dongdong Ge, Xiaoye Jiang, and Yinyu Ye. A note on the complexity of ℓ_p minimization. *Mathematical Programming*, 129(2):285–299, 2011.

[8] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and S Sundararajan. A dual coordinate descent method for large-scale linear SVM. *ICML*, pages 408–415, 2008.

[9] InfiniBand Trade Association. *InfiniBand Architecture Specification, Volume 1, Release 1.0*. 2005.

[10] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *FOCS*, pages 147–156. IEEE, 2013.

[11] Libsvm. *Datasets*, 2014. http://www.csie.ntu.edu.tw/~cjlin/ libsvmtools/datasets/binary.html.

[12] Ji Liu, Stephen J Wright, Christopher Ré, and Victor Bittorf. An asynchronous parallel stochastic coordinate descent algorithm. *arXiv:1311.1873*, 2013.

[13] Zhaosong Lu and Lin Xiao. On the complexity analysis of randomized block-coordinate descent methods. *arXiv preprint arXiv:1305.4723*, 2013.

[14] Z. Q. Luo and Paul Tseng. A coordinate gradient descent method for nonsmooth separable minimization. *J. Optim. Theory Appl.*, 72(1), January 2002.

[15] Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2):227–234, 1995.

[16] Ion Necoara and Dragos Clipici. Distributed coordinate descent methods for composite minimization. *arXiv:1312.5302*, 2013.

[17] Yurii Nesterov. *Introductory lectures on convex optimization*, volume 87 of *Applied Optimization*. Kluwer, 2004.

[18] Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[19] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *NIPS*, 24:693–701, 2011.

[20] Andrei Patrascu and Ion Necoara. Random coordinate descent methods for ℓ_0 regularized convex optimization. *arXiv:1403.6622*, 2014.

[21] Peter Richtárik and Martin Takáč. Efficient serial and parallel coordinate descent methods for huge-scale truss topology design. In *Operations Research Proceedings 2011*, pages 27–32. Springer, 2012.

[22] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *arXiv:1212.0873*, 2012.

[23] Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data. *arXiv:1310.2059*, 2013.

[24] Peter Richtárik and Martin Takáč. On optimal probabilities in stochastic coordinate descent methods. *arXiv:1310.3438*, 2013.

[25] Peter Richtárik and Martin Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.

[26] Peter Richtárik and Martin Takáč. Randomized lock-free methods for minimizing sparse convex functions. 2012.

[27] Ankan Saha and Ambuj Tewari. On the finite time convergence of cyclic coordinate descent methods. *SIAM Journal of Optimization*, 23(1):576–601, 2013.

[28] Nur Shakirah Md Salleh, Azizah Suliman, and Abdul Rahim Ahmad. Parallel execution of distributed svm using mpi (codlib). In *Information Technology and Multimedia (ICIM)*, pages 1–4. IEEE, 2011.

[29] Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David Haglin. Feature clustering for accelerating parallel coordinate descent. *NIPS*, pages 28–36, 2012.

[30] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

[31] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *JMLR*, 14(1):567–599, 2013.

[32] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI-The Complete Reference, Volume 1: The MPI Core*. MIT Press, Cambridge, MA, USA, 2nd. (revised) edition, 1998.

[33] Martin Takáč, Avleen Singh Bijral, Peter Richtárik, and Nathan Srebro. Mini-batch primal and dual methods for SVMs. *ICML*, 2013.

[34] Rachael Tappenden, Peter Richtárik, and Burak Büke. Separable approximations and decomposition methods for the augmented lagrangian. *arXiv:1308.6774, to appear in Optimization Methods and Software*, 2013.

[35] Rachael Tappenden, Peter Richtárik, and Jacek Gondzio. Inexact coordinate descent: complexity and preconditioning. *arXiv:1304.5530*, 2013.

[36] Rachael Tappenden, Peter Richtárik, and Martin Takáč. Improved complexity analysis of parallel coordinate descent methods. 2014.

[37] Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *J. Optim. Theory Appl.*, 109(3):475–494, 2001.

[38] Paul Tseng and Sangwoon Yun. Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. *J. Optim. Theory Appl.*, 140:513–535, 2009.

[39] Paul Tseng and Sangwoon Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117:387–423, 2009.

[40] Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A comparison of optimization methods and software for large-scale ℓ_1-regularized linear classification. *JMLR*, 9999:3183–3234, 2010.

[41] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling. *arXiv:1401.2753*, 2014.