Journal of Machine Learning Research 17 (2016) 1-25 Submitted 1/15; Published 2/16

Distributed Coordinate Descent Method for Learning with Big Data

Peter Richtárik PETER.RICHTARIK@ED.AC.UK
School of Mathematics
University of Edinburgh
6317 James Clerk Maxwell Building
Peter Guthrie Tait Road
Edinburgh, EH9 3FD

Martin Takáč TAKAC.MT@GMAIL.COM
Industrial and Systems Engineering Department
Lehigh University
H.S. Mohler Laboratory
200 West Packer Avenue
Bethlehem, PA 18015, USA

Editor: Koby Crammer

Abstract

In this paper we develop and analyze Hydra: HYbriD cooRdinAte descent method for solving loss minimization problems with big data. We initially partition the coordinates (features) and assign each partition to a different node of a cluster. At every iteration, each node picks a random subset of the coordinates from those it owns, independently from the other computers, and in parallel computes and applies updates to the selected coordinates based on a simple closed-form formula. We give bounds on the number of iterations sufficient to approximately solve the problem with high probability, and show how it depends on the data and on the partitioning. We perform numerical experiments with a LASSO instance described by a 3TB matrix.

Keywords: stochastic methods, parallel coordinate descent, distributed algorithms, boosting

1. Introduction

Randomized coordinate descent methods (CDMs) are increasingly popular in many learning tasks, including boosting, large scale regression and training linear support vector machines. CDMs update a single randomly chosen coordinate at a time by moving in the direction of the negative partial derivative (for smooth losses). Methods of this type, in various settings, were studied by several authors, including Hsieh et al. (2008); Shalev-Shwartz and Tewari (2009); Nesterov (2012); Richtárik and Takáč (2014); Necoara et al. (2012); Tappenden et al. (2013); Shalev-Shwartz and Zhang (2013b); Lu and Xiao (2015).

It is clear that in order to utilize modern shared-memory parallel computers, more coordinates should be updated at each iteration. One way to approach this is via partitioning the coordinates into blocks, and operating on a single randomly chosen block at a time, utilizing parallel linear algebra libraries. This approach was pioneered by Nesterov (2012) for smooth losses, and was extended to regularized problems in (Richtárik and Takáč, 2014). Another popular approach involves working with a random subset of coordinates (Bradley et al., 2011). These approaches can be combined, and theory was developed for methods that update a random subset of blocks of coordinates at a time Richtárik and Takáč (2015); Fercoq and Richtárik (2013). Further recent works on parallel coordinate descent include (Richtárik and Takáč, 2012; Mukherjee et al., 2013; Fercoq, 2013; Tappenden et al., 2015; Shalev-Shwartz and Zhang, 2013a).

However, none of these methods are directly scalable to problems of sizes so large that a single computer is unable to store the data describing the instance, or is unable to do so efficiently (e.g., in memory). In a big data scenario of this type, it is imperative to split the data across several nodes (computers) of a cluster, and design efficient methods for this memory-distributed setting.

Hydra. In this work we design and analyze the first distributed coordinate descent method: *Hydra: HYbriD cooRdinAte descent.* The method is "hybrid" in the sense that it uses parallelism at two levels: i) across a number of nodes in a cluster and ii) utilizing the parallel processing power of individual nodes[1].

Assume we have c nodes (computers) available, each with parallel processing power. In Hydra, we initially partition the coordinates $\{1, 2, \dots, d\}$ into c sets, $\mathcal{P}_1, \dots, \mathcal{P}_c$, and assign each set to a single computer. For simplicity, we assume that the partition is balanced: $|\mathcal{P}_k| = |\mathcal{P}_l|$ for all k, l. Each computer *owns* the coordinates belonging to its partition for the duration of the iterative process. Also, these coordinates are stored locally. The data matrix describing the problem is partitioned in such a way that all data describing features belonging to $\mathcal{P}_l$ is stored at computer l. Now, at each iteration, each computer, independently from the others, chooses a random subset of τ coordinates from those they own, and computes and applies updates to these coordinates. Hence, once all computers are done, $c\tau$ coordinates will have been updated. The resulting vector, stored as c vectors of size $s = d/c$ each, in a distributed way, is the new iterate. This process is repeated until convergence. It is important that the computations are done locally on each node, with minimum communication overhead. We comment on this and further details in the text.

The main insight. We show that the parallelization potential of Hydra, that is, its ability to accelerate as τ is increased, depends on two data-dependent quantities: i) the *spectral norm of the data (σ)* and ii) a *partition-induced norm of the data (σ').* The first quantity completely describes the behavior of the method in the $c = 1$ case. If σ is small, then utilization of more processors (i.e., increasing τ) leads to nearly linear speedup. If σ is large, speedup may be negligible, or there may be no speedup whatsoever. Hence, the size of σ suggests whether it is worth to use more processors or not. The second quantity, σ', characterizes the effect of the initial partition on the algorithm, and as such is relevant in the $c > 1$ case. Partitions with small σ' are preferable. We show that, surprisingly, that as long as $\tau \geq 2$, the effect of a bad partitioning is that it most doubles the number of iterations of Hydra. Hence, data partitioning can be used to optimize for different aspects of the method, such as reducing communication complexity, if needed.

For all of these quantities we derive easily computable and interpretable estimates (ω for σ and ω' for σ'), which may be used by practitioners to gauge, a-priori, whether their problem of interest is likely to be a good fit for Hydra or not. We show that for strongly convex losses, Hydra outputs an ϵ-accurate solution with probability at least $1 - \rho$ after

$$\frac{d\beta}{c\tau\mu} \log\left(\frac{1}{\epsilon\rho}\right)$$

iterations (we ignore some small details here), where a single iteration corresponds to changing of τ coordinates by each of the c nodes; β is a stepsize parameter and μ is a strong convexity constant.

1. We like to think of each node of the cluster as one of the many heads of the mythological Hydra.

square loss (SL)	$\frac{1}{2}(y^j - \mathbf{A}_{j:}x)^2$
logistic loss (LL)	$\log(1 + \exp(-y^j \mathbf{A}_{j:}x))$
square hinge loss (HL)	$\frac{1}{2}\max\{0, 1 - y^j \mathbf{A}_{j:}x\}^2$

Table 1: Examples of loss functions ℓ covered by our analysis.

Outline. In Section 2 we describe the structure of the optimization problem we consider in this paper and state assumptions. We then proceed to Section 3, in which we describe the method. In Section 4 we prove bounds on the number of iterations sufficient for Hydra to find an approximate solution with arbitrarily high probability. A discussion of various aspects of our results, as well as a comparison with existing work, can be found in Section 5. Implementation details of our distributed communication protocol are laid out in Section 6. Finally, we comment on our computational experiments with a big data (3TB matrix) L1 regularized least-squares instance in Section 7.

2. The Problem

We study the problem of minimizing regularized loss,

$$\min_{x\in\mathbb{R}^d} L(x) := f(x) + R(x), \tag{1}$$

where f is a smooth convex loss, and R is a convex (and possibly nonsmooth) regularizer.

2.1 Loss Function f

We assume that there exists a positive definite matrix $\mathbf{M} \in \mathbb{R}^{d\times d}$ such that for all $x, h \in \mathbb{R}^d$,

$$f(x+h) \le f(x) + (f'(x))^T h + \tfrac{1}{2}h^T\mathbf{M}h, \tag{2}$$

and write $\mathbf{M} = \mathbf{A}^T\mathbf{A}$, where $\mathbf{A}$ is some n-by-d matrix.

Example. These assumptions are natural satisfied in many popular problems. A typical loss function has the form

$$f(x) = \sum_{j=1}^{n} \ell(x, \mathbf{A}_{j:}, y^j), \tag{3}$$

where $\mathbf{A} \in \mathbb{R}^{n\times d}$ is a matrix encoding n examples with d features, $\mathbf{A}_{j:}$ denotes j-th row of $\mathbf{A}$, ℓ is some loss function acting on a single example and $y \in \mathbb{R}^n$ is a vector of labels. For instance, in the case of the three losses ℓ in Table 1, assumption (2) holds with $\mathbf{M} = \mathbf{A}^T\mathbf{A}$ for SL and HL, and $\mathbf{M} = \frac{1}{4}\mathbf{A}^T\mathbf{A}$ for LL (Bradley et al., 2011).

2.2 Regularizer R

We assume that R is separable, i.e., that it can be decomposed as $R(x) = \sum_{i=1}^{d} R_i(x^i)$, where x^i is the i-th coordinate of x, and the functions $R_i : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$ are convex and closed.

Example. The choice $R_i(t) = 0$ for $t \in [0,1]$ and $R_i(t) = +\infty$, otherwise, effectively models bound constraints, which are relevant for SVM dual. Other popular choices are $R(x) = \lambda\|x\|_1$ (L1-regularizer) and $R(x) = \frac{\lambda}{2}\|x\|_2^2$ (L2-regularizer).

3. Distributed Coordinate Descent

We consider a setup with c computers (nods) and first partition the d coordinates (features) into c sets $\mathcal{P}_1, \dots, \mathcal{P}_c$ of equal cardinality, $s := d/c$, and assign set $\mathcal{P}_l$ to node l. Hydra is described in Algorithm 1. Hydra's convergence rate depends on the partition; we comment on this later in Sections 4 and 5. Here we simply assume that we work with a fixed partition. We now comment on the steps.

Algorithm 1: Hydra: HYbriD cooRdinAte descent

```
1 Parameters: $x_0 \in \mathbb{R}^d$; $\{\mathcal{P}_1, \dots, \mathcal{P}_c\}$; $\beta > 0$, $\tau$; $k \leftarrow 0$;
2 repeat
3     $x_{k+1} \leftarrow x_k$ ;
4     for each computer $l \in \{1, \dots, c\}$ in parallel do
5         Pick a random set of coordinates $\hat{S}_l \subseteq \mathcal{P}_l$ , $|\hat{S}_l| = \tau$
6         for each $i \in \hat{S}_l$ in parallel do
7             $h_k^i \leftarrow \arg\min_t f_i'(x_k)t + \frac{M_{ii}\beta}{2}t^2 + R_i(x_k^i + t)$ ;
8             Apply the update: $x_{k+1}^i \leftarrow x_{k+1}^i + h_k^i$ ;
9 until happy;
```

Step 3. Here we are just establishing a way of labeling the iterates. Starting with x_k, all c computers modify $c\tau$ entries of x_k in total, in a distributed way, and the result is called x_{k+1}. No computer is allowed to proceed before all computers are done with computing their updates. The resulting vector, x_{k+1}, is the new iterate. Note that, due to this, our method is inherently *synchronous.* In practice, a carefully designed asynchronous implementation will be faster, and our experiments in Section 7 are done with such an implementation.

Steps 4–5. At every iteration, each of the c computers picks a random subset of τ features from those that it owns, uniformly at random, independently of the choice of the other computers. Let $\hat{S}_l$ denote the set picked by node l . More formally, we require that i) $\hat{S}_l \subseteq \mathcal{P}_l$, ii) $\mathbf{Prob}(|\hat{S}_l| = \tau) = 1$, where $1 \leq \tau \leq s$, and that iii) all subsets of $\mathcal{P}_l$ of cardinality τ are chosen equally likely. In summary, at every iteration of the method, features belonging to the random set $\hat{S} := \cup_{l=1}^{c} \hat{S}_l$ are updated. Note that $\hat{S}$ has size $c\tau$, but that, as a sampling from the set $\{1, 2, \dots, d\}$, it does not choose all cardinality $c\tau$ subsets of $\{1, 2, \dots, d\}$ with equal probability. Hence, the analysis of parallel coordinate descent methods of Richtárik and Takáč (2015) does not apply. We will say that $\hat{S}$ is a τ*-distributed sampling* with respect to the partition $\{\mathcal{P}_1, \dots, \mathcal{P}_c\}$.

Step 6. Once computer l has chosen its set of τ coordinates to work on in Step 5, it will *in parallel* compute (Step 7) and apply (Step 8) updates to them.

Step 7. This is a critical step where updates to coordinates $i \in \hat{S}_l$ are computed. By $f'_i(x)$ we denote the i-th partial derivative of f at x. Notice that the formula is very simple as it involves one dimensional optimization.

Closed-form formulas. Often, h^i_k can be computed in closed form. For $R_i(t) = \lambda_i|t|$ (weighted L1 regularizer), h^i_k is the point in the interval

$$\left[\frac{-\lambda_i - f'_i(x_k)}{\mathbf{M}_{ii}\beta}, \frac{\lambda_i - f'_i(x_k)}{\mathbf{M}_{ii}\beta}\right]$$

which is closest to $-x^i_k$. If $R_i(t) = \frac{\lambda_i}{2}t^2$ (weighted L2 regularizer), then $h^i_k = -\frac{f'_i(x_k)+\lambda_i x^i_k}{\lambda_i + \mathbf{M}_{ii}\beta}$.

Choice of β. The choice of the step-size parameter β is of paramount significance for the performance of the algorithm, as argued for different but related algorithms by Richtárik and Takáč (2015); Takáč et al. (2013); Fercoq and Richtárik (2013). We will discuss this issue at length in Sections 4 and 5.

Implementation issues: Note that computer l needs to know the partial derivatives of f at x_k for coordinates $i \in \hat{S}_l \subseteq \mathcal{P}_l$. However, x_k, as well as the data describing f, is distributed among the c computers. One thus needs to devise a fast and communication efficient way of computing these derivatives. This issue will be dealt with in Section 6.

Step 8. Here all the τ updates computed in Step 7 are applied to the iterate. Note that the updates are *local*: computer l only updates coordinates it owns, which are stored locally. Hence, this step is communication-free.

4. Convergence Rate Analysis

Notation: For any $\mathbf{G} \in \mathbb{R}^{d\times d}$, let $D^{\mathbf{G}} = \mathrm{Diag}(\mathbf{G})$. That is, $D^{\mathbf{G}}_{ii} = \mathbf{G}_{ii}$ for all i and $D^{\mathbf{G}}_{ij} = 0$ for $i \neq j$. Further, let $B^{\mathbf{G}} \in \mathbb{R}^{d\times d}$ be the block diagonal of $\mathbf{G}$ associated with the partition $\{\mathcal{P}_1, \dots, \mathcal{P}_c\}$. That is, $B^{\mathbf{G}}_{ij} = \mathbf{G}_{ij}$ whenever $i, j \in \mathcal{P}_l$ for some l, and $B^{\mathbf{G}}_{ij} = 0$ otherwise.

4.1 Four Important Quantities: $\sigma', \omega', \sigma, \omega$

Here we define two quantities, σ' and σ, which, as we shall see, play an important role in the computation of the stepsize parameter β of Algorithm 1, and through it, in understanding its rate of convergence and potential for speedup by parallelization and distribution. As we shall see, these quantities might not be easily computable. We therefore also provide each with an easily computable and interpretable upper bound, ω' for σ' and ω for σ.

Let

$$\mathbf{Q} := (D^{\mathbf{M}})^{-1/2}\mathbf{M}(D^{\mathbf{M}})^{-1/2}, \tag{4}$$

and notice that, by construction, $\mathbf{Q}$ has ones on the diagonal. Since M is positive definite, $\mathbf{Q}$ is as well. For each $l \in \{1, \dots, c\}$, let $\mathbf{A}_l \in \mathbb{R}^{n\times s}$ be the column submatrix of $\mathbf{A}$ corresponding to coordinates $i \in \mathcal{P}_l$. The diagonal blocks of $B^{\mathbf{Q}}$ are the matrices $\mathbf{Q}^{ll}$, $l = 1, 2, \dots, c$, where

$$\mathbf{Q}^{kl} := (D^{\mathbf{A}_k^T\mathbf{A}_k})^{-1/2}\mathbf{A}_k^T\mathbf{A}_l(D^{\mathbf{A}_l^T\mathbf{A}_l})^{-1/2} \in \mathbb{R}^{s\times s} \tag{5}$$

for each $k, l \in \{1, 2, \dots, c\}$. We now define

$$\sigma' := \max\{x^T\mathbf{Q}x \ : \ x \in \mathbb{R}^d,\ x^T B^{\mathbf{Q}} x \leq 1\}, \tag{6}$$

$$\sigma := \max\{x^T \mathbf{Q} x \;:\; x \in \mathbb{R}^d,\; x^T x \le 1\}. \tag{7}$$

A useful consequence of (6) is the inequality

$$x^T(\mathbf{Q} - B^{\mathbf{Q}})x \le (\sigma' - 1)x^T B^{\mathbf{Q}} x. \tag{8}$$

4.1.1 SPARSITY

Let a_{rl} be the r-th row of $\mathbf{A}_l$, and define

$$\omega' := \max_{1 \le r \le n} \left\{ \omega'(r) := |\{l \;:\; l \in \{1,\dots,c\},\; a_{rl} \ne 0\}| \right\},$$

where $\omega'(r)$ is the number of matrices $\mathbf{A}_l$ with a nonzero in row r. Likewise, define

$$\omega := \max_{1 \le r \le n} \left\{ \omega(r) := |\{l \;:\; l \in \{1,\dots,c\},\; \mathbf{A}_{rl} \ne 0\}| \right\},$$

where $\omega(r)$ is the number of nonzeros in the r-th row of $\mathbf{A}$.

Lemma 1. *The following relations hold:*

$$\max\{1, \tfrac{\sigma}{s}\} \le \sigma' \le \omega' \le c, \qquad 1 \le \sigma \le \omega \le d. \tag{9}$$

The proof can be found in the appendix.

4.2 Choice of the Stepsize Parameter β

We analyze Hydra with stepsize parameter $\beta \ge \beta^*$, where

$$\begin{aligned} \beta^* &:= \beta_1^* + \beta_2^*, \\ \beta_1^* &:= 1 + \frac{(\tau-1)(\sigma-1)}{s_1}, \\ \beta_2^* &:= \left(\frac{\tau}{s} - \frac{\tau-1}{s_1}\right)\frac{\sigma'-1}{\sigma'}\sigma, \end{aligned} \tag{10}$$

and $s_1 = \max(1, s-1)$.

As we shall see in Theorem 5, fixing c and τ, the number of iterations needed by Hydra find a solution is proportional to β. Hence, we would wish to use β which is as small as possible, but not smaller than the safe choice $\beta = \beta^*$, for which convergence is guaranteed. In practice, β can often be chosen smaller than the save but conservative value of β^*, leading to larger steps and faster convergence.

If the quantities σ and σ' are hard to compute, then one can replace them by the easily computable upper bounds ω and ω', respectively. However, there are cases when σ can be efficiently approximated and is much smaller than ω. In some ML data sets with $\mathbf{A} \in \{0,1\}^{n\times d}$, σ is close to the average number of nonzeros in a row of $\mathbf{A}$, which can be significantly smaller than the maximum, ω. On the other hand, if σ is difficult to compute, ω may provide a good proxy. Similar remarks apply to σ'.

More importantly, if $\tau \ge 2$ (which covers all interesting uses of Hydra), we may ignore β_2^* altogether, as implied by the following result.

Lemma 2. *If $\tau \geq 2$, then $\beta^* \leq 2\beta_1^*$.*

Proof. It is enough to argue that $\beta_2^* \leq \beta_1^*$. Notice that β_2^* is increasing in σ'. On the other hand, from Lemma 1 we know that $\sigma' \leq c = \frac{d}{s}$. So, it suffices to show that

$$\left(\frac{\tau}{s} - \frac{\tau-1}{s-1}\right)\left(1 - \frac{s}{d}\right)\sigma \leq 1 + \frac{(\tau-1)(\sigma-1)}{s-1}.$$

After straightforward simplification we observe that this inequality is equivalent to $(s-\tau) + (\tau - 2)\sigma + \frac{\sigma}{d}(s+\tau) \geq 0$, which clearly holds. □

Clearly, $\beta^* \geq \beta_1^*$. Hence, if in Hydra we instead of $\beta = \beta^*$ (best/smallest value prescribed by our theory) use $\beta = 2\beta_1^*$ —eliminating the need to compute σ'—the number of iterations will *at most double.* Since σ', present in β_2^*, captures the effect of the initial partition on the iteration complexity of the algorithm, we conclude that this effect is under control.

4.3 Separable Approximation

We first establish a useful identity for the expected value of a random quadratic form obtained by sampling the rows and columns of the underlying matrix via the distributed sampling $\hat{S}$. Note that the result is a direct generalization of Lemma 1 in (Takáč et al., 2013) to the $c > 1$ case.

For $x \in \mathbb{R}^d$ and $\emptyset \neq S \subseteq [d] := \{1, 2, \dots, d\}$, we write $x^S := \sum_{i \in S} x^i e_i$, where e_i is the i-th unit coordinate vector. That is, x^S is the vector in $\mathbb{R}^d$ whose coordinates $i \in S$ are identical to those of x, but are zero elsewhere.

Lemma 3. *Fix arbitrary $\mathbf{G} \in \mathbb{R}^{d \times d}$ and $x \in \mathbb{R}^d$ and let $s_1 = \max(1, s-1)$. Then*

$$\mathbf{E}[(x^{\hat{S}})^T \mathbf{G} x^{\hat{S}}] = \frac{\tau}{s}\left[\alpha_1 x^T D^{\mathbf{G}} x + \alpha_2 x^T \mathbf{G} x + \alpha_3 x^T (\mathbf{G} - B^{\mathbf{G}}) x\right], \tag{11}$$

where $\alpha_1 = 1 - \frac{\tau-1}{s_1}$, $\alpha_2 = \frac{\tau-1}{s_1}$, $\alpha_3 = \frac{\tau}{s} - \frac{\tau-1}{s_1}$.

Proof. In the $s = 1$ case the statement is trivially true. Indeed, we must have $\tau = 1$ and thus $\mathbf{Prob}(\hat{S} = \{1, 2, \dots, d\}) = 1$, $h^{\hat{S}} = h$, and hence

$$\mathbf{E}\left[(h^{\hat{S}})^T \mathbf{Q} h^{\hat{S}}\right] = h^T \mathbf{Q} h.$$

This finishes the proof since $\frac{\tau-1}{s_1} = 0$.

Consider now the $s > 1$ case. From Lemma 3 in Richtárik and Takáč (2015) we get

$$\mathbf{E}\left[(h^{\hat{S}})^T \mathbf{Q} h^{\hat{S}}\right] = \mathbf{E}\left[\sum_{i \in \hat{S}} \sum_{j \in \hat{S}} \mathbf{Q}_{ij} h^i h^j\right] = \sum_{i=1}^d \sum_{j=1}^d p_{ij} \mathbf{Q}_{ij} h^i h^j, \tag{12}$$

where $p_{ij} = \mathbf{Prob}(i \in \hat{S} \;\&\; j \in \hat{S})$. One can easily verify that

$$p_{ij} = \begin{cases} \frac{\tau}{s}, & \text{if } i = j, \\ \frac{\tau(\tau-1)}{s(s-1)}, & \text{if } i \neq j \text{ and } i \in \mathcal{P}_l,\ j \in \mathcal{P}_l \text{ for some } l, \\ \frac{\tau^2}{s^2}, & \text{if } i \neq j \text{ and } i \in \mathcal{P}_k,\ j \in \mathcal{P}_l \text{ for } k \neq l. \end{cases}$$

In particular, the first case follows from Eq (32) and the second from Eq (37) in Richtárik and Takáč (2015). It only remains to substitute p_{ij} into (12) and transform the result into the desired form. □

We now use the above lemma to compute a separable quadratic upper bound on $\mathbf{E}[(h^{\hat{S}})^T \mathbf{M} h^{\hat{S}}]$.

Lemma 4. *For all $h \in \mathbb{R}^d$,*

$$\mathbf{E}\left[(h^{\hat{S}})^T \mathbf{M} h^{\hat{S}}\right] \leq \frac{\tau}{s}\beta^* \left(h^T D^{\mathbf{M}} h\right). \tag{13}$$

Proof. For $x := (D^{\mathbf{M}})^{1/2}h$, we have $(h^{\hat{S}})^T \mathbf{M} h^{\hat{S}} = (x^{\hat{S}})^T \mathbf{Q} x^{\hat{S}}$. Taking expectations on both sides, and applying Lemma 3, we see that $\mathbf{E}[(h^{\hat{S}})^T \mathbf{M} h^{\hat{S}}]$ is equal to (11) for $\mathbf{G} = \mathbf{Q}$. It remains to bound the three quadratics in (11). Since $D^{\mathbf{Q}}$ is the identity matrix, $x^T D^{\mathbf{Q}} x = h^T D^{\mathbf{M}} h$. In view of (7), the 2nd term is bounded as $x^T \mathbf{Q} x \leq \sigma x^T x = \sigma h^T D^{\mathbf{M}} h$. Finally,

$$\begin{aligned}
x^T(\mathbf{Q} - B^{\mathbf{Q}})x &= \frac{\sigma' - 1}{\sigma'} x^T(\mathbf{Q} - B^{\mathbf{Q}})x + \frac{1}{\sigma'} x^T(\mathbf{Q} - B^{\mathbf{Q}})x \\
&\overset{(8)}{\leq} \frac{\sigma' - 1}{\sigma'} x^T(\mathbf{Q} - B^{\mathbf{Q}})x + \frac{\sigma' - 1}{\sigma'} x^T B^{\mathbf{Q}} x \\
&= \frac{\sigma' - 1}{\sigma'} x^T \mathbf{Q} x \\
&\overset{(7)}{\leq} \frac{\sigma' - 1}{\sigma'} \sigma x^T x \\
&= \frac{\sigma' - 1}{\sigma'} \sigma h^T D^{\mathbf{M}} h.
\end{aligned}$$

It only remains to plug in these three bounds into (11). □

Inequalities of type (13) were first proposed and studied by Richtárik and Takáč (2015)—therein called Expected Separable Overapproximation (ESO)—and were shown to be important for the convergence of parallel coordinate descent methods. However, they studied a different class of loss functions f (convex smooth and partially separable) and different types of random samplings $\hat{S}$, which did not allow them to propose an efficient distributed sampling protocol leading to a distributed algorithm. An ESO inequality was recently used by Takáč et al. (2013) to design a mini-batch stochastic dual coordinate ascent method (parallelizing the original SDCA methods of Hsieh et al. (2008)) and mini-batch stochastic subgradient descent method (Pegasos of Shalev-Shwartz et al. (2011)), and give bounds on how mini-batching leads to acceleration. While it was long observed that mini-batching often accelerates Pegasos in practice, it was only shown with the help of an ESO inequality that this is so also in theory. Recently, Fercoq and Richtárik (2013) have derived ESO inequalities for smooth approximations of nonsmooth loss functions and hence showed that parallel coordinate descent methods can accelerate on their serial counterparts on a class of structured nonsmooth convex losses. As a special case, they obtain a parallel randomized coordinate descent method for minimizing the logarithm of the exponential loss. Again, the class of losses considered in that paper, and the samplings $\hat{S}$, are different from ours. None of the above methods are distributed.

4.4 Fast Rates for Distributed Learning with Hydra

Let x_0 be the starting point of Algorithm 1, x_* be an optimal solution of problem (1) and let $L^* = L(x_*)$. Further, define $\|x\|_{\mathbf{M}}^2 := \sum_{i=1}^d \mathbf{M}_{ii}(x^i)^2$ (a weighted Euclidean norm on $\mathbb{R}^d$) and assume that f and R are strongly convex with respect to this norm with convexity parameters μ_f and μ_R, respectively. A function ϕ is strongly convex with parameter $\mu_\phi > 0$ if for all $x, h \in \mathbb{R}^d$,

$$\phi(x+h) \geq \phi(x) + (\phi'(x))^T h + \frac{\mu_\phi}{2}\|h\|_{\mathbf{M}}^2,$$

where $\phi'(x)$ is a subgradient (or gradient) for ϕ at x.

We now show that Hydra decreases strongly convex L with an exponential rate in ϵ.

Theorem 5. *Assume L is strongly convex with respect to the norm $\|\cdot\|_{\mathbf{M}}$, with $\mu_f + \mu_R > 0$. Choose $x_0 \in \mathbb{R}^d$, $0 < \rho < 1$, $0 < \epsilon < L(x_0) - L^*$ and*

$$T \quad \geq \quad \frac{d}{c\tau} \times \frac{\beta + \mu_R}{\mu_f + \mu_R} \times \log\left(\frac{L(x_0) - L^*}{\epsilon\rho}\right), \tag{14}$$

where $\beta \geq \beta^*$ and β^* is given by (10). *If $\{x_k\}$ are the random points generated by Hydra (Algorithm 1), then*

$$\mathbf{Prob}(L(x_T) - L^* \leq \epsilon) \geq 1 - \rho.$$

Proof. We first claim that for all $x, h \in \mathbb{R}^d$,

$$\mathbf{E}\left[f(x + h^{\hat{S}})\right] \leq f(x) + \frac{\mathbf{E}[|\hat{S}|]}{d}\left((f'(x))^T h + \frac{\beta}{2} h^T D^{\mathbf{M}} h\right). \tag{15}$$

To see this, substitute $h \leftarrow h^{\hat{S}}$ into (2), take expectations on both sides and then use Lemma 4 together with the fact that for any vector a, $\mathbf{E}[a^T h^{\hat{S}}] = \frac{\mathbf{E}[|\hat{S}|]}{d} = \frac{\tau c}{sc} = \frac{\tau}{s}$. The rest follows by following the steps in the proof in (Richtárik and Takáč, 2015, Theorem 20). □

A similar result, albeit with the weaker rate $O(\frac{s\beta}{\tau\epsilon})$, can be established in the case when neither f nor R are strongly convex. In big data setting, where parallelism and distribution is unavoidable, it is much more relevant to study the dependence of the rate on parameters such as τ and c. We shall do so in the next section.

5. Discussion

In this section we comment on several aspects of the rate captured in (14) and compare Hydra to selected methods.

5.1 Insights Into the Convergence Rate

Here we comment in detail on the influence of the various design parameters (c = # computers, s = # coordinates owned by each computer, and τ = # coordinates updated by each computer in each iteration), instance-dependent parameters ($\sigma, \omega, \mu_R, \mu_f$), and parameters depending both on the instance and design (σ', ω'), on the stepsize parameter β, and through it, on the convergence rate described in Theorem 5.

special case	β^*	$\beta^*/(c\tau)$
any c $\tau = 1$	$1 + \frac{\sigma}{s}\left(\frac{\sigma' - 1}{\sigma'}\right)$	$s + \sigma\left(\frac{\sigma' - 1}{\sigma'}\right)$
$c = 1$ any τ	$1 + \frac{(\tau-1)(\sigma-1)}{d-1}$	$\frac{d}{\tau}\left(1 + \frac{(\tau-1)(\sigma-1)}{d-1}\right)$
$\tau c = d$	σ	σ

Table 2: Stepsize parameter $\beta = \beta^*$ and the leading factor in the rate (14) (assuming $\mu_R = 0$) for several special cases of Hydra.

5.2 Strong Convexity

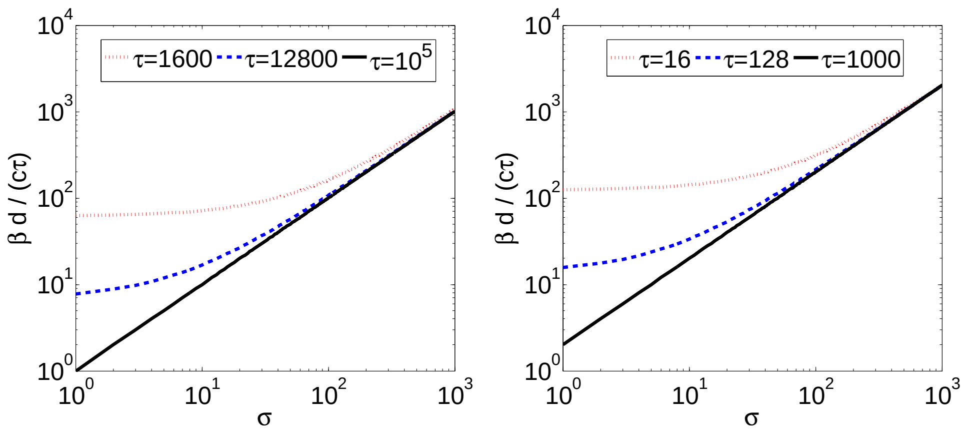
Notice that the size of $\mu_R > 0$ mitigates the effect of a possibly large β on the bound (14). Indeed, for large μ_R, the factor $(\beta + \mu_R)/(\mu_f + \mu_R)$ approaches 1, and the bound (14) is dominated by the term $\frac{d}{c\tau}$, which means that Hydra enjoys linear speedup in c and τ. In the following comments we will assume that $\mu_R = 0$, and focus on studying the dependence of the leading term $d\frac{\beta}{c\tau}$ on various quantities, including τ, c, σ and σ'.

5.3 Search for Small but Safe β

As shown by Takáč et al. (2013, Section 4.1), mini-batch SDCA might *diverge* in the setting with $\mu_f = 0$ and $R(x) \equiv 0$, even for a simple quadratic function with $d = 2$, provided that $\beta = 1$. Hence, small values of β need to be avoided. However, in view of Theorem 5, it is good if β is as small as possible. So, there is a need for a "safe" formula for a small β. Our formula (10), $\beta = \beta^*$, is serving that purpose. For a detailed introduction into the issues related to selecting a good β for parallel coordinate descent methods, we refer the reader to the first 5 pages of (Fercoq and Richtárik, 2013).

5.4 The Effect of σ'

If $c = 1$, then by Lemma 9, $\sigma' = c = 1$, and hence $\beta_2^* = 0$. However, for $c > 1$ we *may* have $\beta_2^* > 0$, which can hence be seen as a price we need to pay for using more nodes. The price depends on the way the data is partitioned to the nodes, as captured by σ'. In favorable circumstances, $\sigma' \approx 1$ even if $c > 1$, leading to $\beta_2^* \approx 0$. However, in general we have the bound $\sigma' \geq \frac{c\sigma}{d}$, which gets worse as c increases and, in fact, σ' can be as large as c. Note also that ξ is decreasing in τ, and that $\xi(s, s) = 0$. This means that by choosing $\tau = s$ (which effectively removes randomization from Hydra), the effect of β_2^* is eliminated. This may not be always possible as often one needs to solve problems with s vastly larger than the number of updates that can be performed on any given node in parallel. If $\tau \ll s$, the effect of β_2^* can be controlled, to a certain extent, by choosing a partition



(a) $(c,s) = (1, 10^5)$ (b) $(c,s) = (10^2, 10^3)$

Figure 1: In terms of the number of iterations, very little is lost by using $c > 1$ as opposed to $c = 1$.

with small σ'. Due to the way σ' is defined, this may not be an easy task. However, it may be easier to find partitions that minimize ω', which is often a good proxy for σ'. Alternatively, we may ignore estimating σ' altogether by setting $\beta = 2\beta_1^*$, as mentioned before, at the price of at most doubling the number of iterations.

5.5 Speedup by Increasing τ

Let us fix c and compare the quantities $\gamma_\tau := \frac{\beta^*}{c\tau}$ for $\tau = 1$ and $\tau = s$. We now show that $\gamma_1 \geq \gamma_s$, which means that if all coordinates are updated at every node, as opposed to one only, then Hydra run with $\beta = \beta^*$ will take fewer iterations. Comparing the 1st and 3rd row of Table 2, we see that $\gamma_1 = s + \sigma\frac{\sigma'-1}{\sigma'}$ and $\gamma_s = \sigma$. By Lemma 1, $\gamma_1 - \gamma_s = s - \frac{\sigma}{\sigma'} \geq 0$.

5.6 Price of Distribution

For illustration purposes, consider a problem with $d = 10^5$ coordinates. In Figure 1(a) we depict the size of $\frac{d\beta_1^*}{c\tau}$ for $c = 1$ and several choices of τ, as a function of σ. We see that Hydra works better for small values of σ and that with increasing σ, the benefit of using updating more coordinates diminishes. In Figure 1(a) we consider the same scenario, but with $c = 100$ and $s = 1000$, and we plot $\frac{d2\beta_1^*}{c\tau}$ on the y axis. Note that the red dotted line in both plots corresponds to a parallel update of 1600 coordinates. In (a) all are updated on a single node, whereas in (b) we have 100 nodes, each updating 16 coordinates at a time. Likewise, the dashed blue dashed and solid black lines are also comparable in both plots. Note that the setup with $c = 10$ has a slightly weaker performance, the lines are a bit lower. This is the price we pay for using c nodes as opposed to a single node (obviously, we are ignoring communication cost here). However, in big data situations one simply has no other choice but to utilize more nodes.

5.7 Comparison with Other Methods

While we are not aware of any other *distributed* coordinate descent method, Hydra in the $c = 1$ case is closely related to several existing parallel coordinate descent methods.

5.7.1 HYDRA VS SHOTGUN

The Shotgun algorithm (parallel coordinate descent) of Bradley et al. (2011) is similar to Hydra for $c = 1$. Some of the differences: Bradley et al. (2011) only consider R equal to the $L1$ norm and their method works in dimension $2d$ instead of the native dimension d. Shotgun was not analyzed for strongly convex f, and convergence in expectation was established. Moreover, Bradley et al. (2011) analyze the step-size choice $\beta = 1$, fixed independently of the number of parallel updates τ, and give results that hold only in a "small τ" regime. In contrast, our analysis works for any choice of τ.

5.7.2 HYDRA VS PCDM

For $c = 1$, Hydra reduces to the parallel coordinate descent method (PCDM) of Richtárik and Takáč (2015), but with a *better* stepsize parameter β. We were able to achieve smaller β (and hence better rates) because we analyze a different and more specialized class of loss functions (those satisfying (2)). In comparison, Richtárik and Takáč (2015) look at a general class of partially separable losses. Indeed, in the $c = 1$ case, our distributed sampling $\hat{S}$ reduces to the sampling considered in (Richtárik and Takáč, 2015) (τ-nice sampling). Moreover, our formula for β (see Table 2) is essentially identical to the formula for β provided in (Richtárik and Takáč, 2015, Theorem 14), with the exception that we have σ where they have ω. By (9), we have $\sigma \leq \omega$, and hence our β is smaller.

5.7.3 HYDRA VS SPCDM

SPCDM of Fercoq and Richtárik (2013) is PCDM applied to a smooth approximation of a nonsmooth convex loss; with a special choice of β, similar to β_1. As such, it extends the reach of PCDM to a large class of nonsmooth losses, obtaining $O(\frac{1}{\epsilon^2})$ rates. It is possible to develop accelerated Hydra with $O(\frac{1}{\epsilon})$ rates by combining ideas from this paper, Fercoq and Richtárik (2013) with the APPROX method of Fercoq and Richtárik (2015).

5.7.4 HYDRA VS MINI-BATCH SDCA

Takáč et al. (2013) studied the performance of a mini-batch stochastic dual coordinate ascent for SVM dual ("mini-batch SDCA"). This is a special case of our setup with $c = 1$, convex quadratic f and $R_i(t) = 0$ for $t \in [0, 1]$ and $R_i(t) = +\infty$ otherwise. Our results can thus be seen as a generalization of the results in that paper to a larger class of loss functions f, more general regularizers R, and most importantly, to the distributed setting ($c > 1$). Also, we give $O(\log \frac{1}{\epsilon})$ bounds under strong convexity, whereas (Takáč et al., 2013) give $O(\frac{1}{\epsilon})$ results without assuming strong convexity. However, Takáč et al. (2013) perform a primal-dual analysis, whereas we do not.

6. Distributed Computation of the Gradient

In this section we describe some important elements of our distributed implementation.

loss function ℓ	$f'_i(x)$	$\mathbf{M}_{ii}$
SL	$\sum_{j=1}^{m} \quad -\mathbf{A}_{ji}(y^j - \mathbf{A}_{j:}x)$	$\lVert\mathbf{A}_{:i}\rVert_2^2$
LL	$\sum_{j=1}^{m} \quad -y^j\mathbf{A}_{ji}\frac{\exp(-y^j\mathbf{A}_{j:}x)}{1+\exp(-y^j\mathbf{A}_{j:}x)}$	$\frac{1}{4}\lVert\mathbf{A}_{:i}\rVert_2^2$
HL	$\sum_{j\,:\,y^j\mathbf{A}_{j:}x<1} \quad \big(-y^j\mathbf{A}_{ji}(1-y^j\mathbf{A}_{j:}x)\big)$	$\lVert\mathbf{A}_{:i}\rVert_2^2$

Table 3: Information needed in Step 5 of Hydra for f given by (3) in the case of the three losses ℓ from Table 1.

Note that in Hydra, x_k is stored in a distributed way. That is, the values x_k^i for $i \in \mathcal{P}_l$ are stored on computer l. Moreover, Hydra partitions $\mathbf{A}$ columnwise as $\mathbf{A} = [\mathbf{A}_1, \dots, \mathbf{A}_c]$, where $\mathbf{A}_l$ consists of columns $i \in \mathcal{P}_l$ of $\mathbf{A}$, and stores $\mathbf{A}_l$ on computer l. So, $\mathbf{A}$ is chopped into smaller pieces with stored in a distributed way in fast memory (if possible) across the c nodes. Note that this allows the method to work with large matrices.

At Step 5 of Hydra, node l at iteration $k+1$ needs to know the partial derivatives $f'_i(x_{k+1})$ for $i \in \hat{S}_l \subseteq \mathcal{P}_l$. We now describe several efficient distributed protocols for the computation of $f'_i(x_{k+1})$ for functions f of the form (3), in the case of the three losses ℓ given in Table 1 (SL, LL, HL). The formulas for $f'_i(x)$ are summarized in Table 3 ($\mathbf{A}_{j:}$ refers to the j-th row of $\mathbf{A}$). Let $D^y := \mathrm{Diag}(y)$.

6.1 Basic Protocol

If we write $h_k^i = 0$ if i is not updated in iteration k, then

$$x_{k+1} = x_k + \sum_{l=1}^{c}\sum_{i\in\hat{S}_l} h_k^i e_i. \tag{16}$$

Now, if we let

$$g_k := \begin{cases} \mathbf{A}x_k - y, & \text{for SL,} \\ -D^y\mathbf{A}x_k, & \text{for LL and HL,} \end{cases} \tag{17}$$

then by combining (16) and (17), we get

$$g_{k+1} = g_k + \sum_{l=1}^{c}\delta g_{k,l}, \qquad \text{where}$$

$$\delta g_{k,l} = \begin{cases} \sum_{i\in\hat{S}_l} h_k^i\mathbf{A}_{:i}, & \text{for SL,} \\ \sum_{i\in\hat{S}_l} -h_k^i D^y\mathbf{A}_{:i}, & \text{for LL and HL.} \end{cases}$$

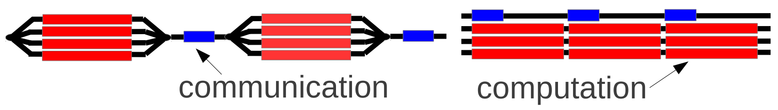


Figure 2: Parallel-serial (PS; left) vs Fully Parallel (FP; right) approach.

Note that the value $\delta g_{k,l}$ can be computed on node l as all the required data is stored locally. Hence, we let each node compute $\delta g_{k,l}$, and then use a *reduce all* operation to add up the updates to obtain g_{k+1}, and pass the sum to all nodes. Knowing g_{k+1}, node l is then able to compute $f_i'(x_{k+1})$ for any $i \in \mathcal{P}_l$ as follows:

$$f_i'(x_{k+1}) = \begin{cases} \mathbf{A}_{:i}^T g_{k+1} = \sum_{j=1}^n \mathbf{A}_{ji} g_{k+1}^j, & \text{for SL,} \\ \sum_{j=1}^n y^j \mathbf{A}_{ji} \frac{\exp(g_{k+1}^j)}{1+\exp(g_{k+1}^j)}, & \text{for LL,} \\ \sum_{j \,:\, g_{k+1}^j > -1} y^j \mathbf{A}_{ji}(1 + g_{k+1}^j), & \text{for HL.} \end{cases}$$

6.2 Advanced Protocols

The basic protocol discussed above has obvious drawbacks. Here we identify them and propose modifications leading to better performance.

- *alternating Parallel and Serial regions (PS):* The basic protocol alternates between two procedures: i) a computationally heavy one (done in parallel) with no MPI communication, and ii) MPI communication (serial). An easy fix would be to dedicate 1 thread to deal with communication and the remaining threads within the same computer for computation. We call this protocol *Fully Parallel (FP)*. Figure 2 compares the basic (left) and FP (right) approaches.

- *Reduce All (RA):* In general, reduce all operations may significantly degrade the performance of distributed algorithms. Communication taking place only between nodes close to each other in the network, e.g., nodes directly connected by a cable, is more efficient. Here we propose the *Asynchronous StreamLined (ASL)* communication protocol in which each node, in a given iteration, sends only 1 message (asynchronously) to a nearby computer, and also receives only one message (asynchronously) from another nearby computer. Communication hence takes place in an *Asynchronous Ring*. This communication protocol requires significant changes in the algorithm. Figure 3 illustrates the flow of messages at the end of the k-th iteration for $c = 4$.

  We order the nodes into a ring, denoting l_- and l_+ the two nodes neighboring node l. Node l only receives data from l_-, and sends data to l_+. Let us denote by $\delta G_{k,l}$ the data sent by node l to l_+ at the end of iteration k. When l starts iteration k, it already knows $\delta G_{k-1,l_-}$.[2] Hence, data which will be sent at the end of the k-th iteration by node l is given by

$$\delta G_{k,l} = \delta G_{k-1,l_-} - \delta g_{k-c,l} + \delta g_{k,l}. \tag{18}$$

  This leads to the update rule

$$g_{k+1,l} = g_{k,l} + \delta g_{k,l} + \delta G_{k,l_-} - \delta g_{k-c+1,l}.$$

2. Initially, we let $\delta g_{k,l} = \delta G_{k,l} = 0$ for all $k \leq 0$.

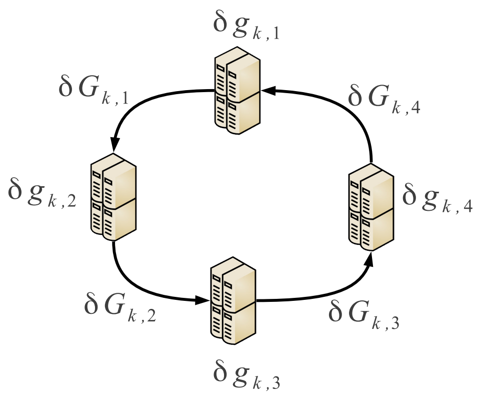


Figure 3: ASL protocol with $c = 4$ nodes. In iteration k, node l computes $\delta g_{k,l}$, and sends $\delta G_{k,l}$ to l_+.

ASL needs less communication per iteration. On the other hand, information is propagated more slowly to the nodes through the ring, which may adversely affect the number of iterations till convergence (note that we do not analyze Hydra with this communication protocol). Indeed, it takes $c - 1$ iterations to propagate information to all nodes. Also, storage requirements have increased: at iteration k we need to store the vectors $\delta g_{t,l}$ for $k - c \leq t \leq k$ on computer l.

7. Experiments

In this section we present numerical evidence that Hydra is capable to efficiently solve big data problems. We have a C++ implementation, using Boost::MPI and OpenMP. Experiments were executed on a Cray XE6 cluster with 128 nodes; with each node equipped with two AMD Opteron Interlagos 16-core processors and 32 GB of RAM.

7.1 Performance of Communication Protocols

In this experiment we consider a LASSO problem, i.e., f given by (3) with ℓ being the square loss (SL) and $R(x) = \|x\|_1$. In order to to test Hydra under controlled conditions, we adapted the LASSO generator proposed by Nesterov (2013, Section 6); modifications were necessary as the generator does not work well in the big data setting.

As discussed in Section 6, the advantage of the RA protocol is the fact that Theorem 5 was proved in this setting, and hence can be used as a safe benchmark for comparison with the advanced protocols.

Table 4 compares the average time per iteration for the 3 approaches and 3 choices of τ. We used 128 nodes, each running 4 MPI processes (hence $c = 512$). Each MPI process runs 8 OpenMP threads, giving 4,096 cores in total. The data matrix $\mathbf{A}$ has $n = 10^9$ rows and $d = 5 \times 10^8$ columns, and has 3 TB, double precision. One can observe that in all cases, ASL-FP yields largest gains compared to the benchmark RA-PS protocol. Note that ASL has some overhead in each iteration, and hence in cases when computation per node is small ($\tau = 10$), the speedup is only 1.62. When $\tau = 10^2$ (in this case the durations of computation and communication were comparable), ASL-FP

τ	comm. protocol	organization	avg. time	speedup
10	RA	PS	0.040	—
10	RA	FP	0.035	1.15
10	ASL	FP	0.025	1.62
10^2	RA	PS	0.100	—
10^2	RA	FP	0.077	1.30
10^2	ASL	FP	0.032	3.11
10^3	RA	PS	0.321	—
10^3	RA	FP	0.263	1.22
10^3	ASL	FP	0.249	1.29

Table 4: Duration of a single Hydra iteration for 3 communication protocols. The basic RA-PS protocol is always the slowest, but follows the theoretical analysis. ASL-FP can be 3× faster.

is 3.11 times faster than RA-PS. But the gain becomes again only moderate for $\tau = 10^3$; this is because computation now takes much longer than communication, and hence the choice of strategy for updating the auxiliary vector g_k is less significant. Let us remark that the use of larger τ requires larger $beta$, and hence possibly more iterations (in the worst case).

We now move on to solving an artificial big data LASSO problem with matrix $\mathbf{A}$ in block angular form, depicted in (19).

$$\mathbf{A} = \left(\begin{array}{c:c:c:c} \mathbf{A}_1^{loc} & 0 & \cdots & 0 \\ 0 & \mathbf{A}_2^{loc} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hdashline \mathbf{A}_1^{glob} & \mathbf{A}_2^{glob} & \cdots & \mathbf{A}_c^{glob} \end{array}\right). \tag{19}$$

Such matrices often arise in stochastic optimization. Each Hydra head (=node) l owns two matrices: $\mathbf{A}_l^{loc} \in \mathbb{R}^{1,952,148 \times 976,562}$ and $\mathbf{A}_l^{glob} \in \mathbb{R}^{500,224 \times 976,562}$. The average number of nonzero elements per row in the local part of $\mathbf{A}_l$ is 175, and $1,000$ for the global part. Optimal solution x_* has exactly $160,000$ nonzero elements. Figure 4 compares the evolution of $L(x_k) - L^*$ for ASL-FP and RA-FP.

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

7.2 Updating All Coordinates on Each Node in Each Iteration Might not be Optimal

In this section we give an experimental demonstration that it may not be optimal for each node of Hydra to update all coordinates it owns (in each iteration). That is, we will show that the seemingly ideal choice $\tau = s$ is not necessarily optimal.

In the experiment, we use the *astro-ph* data set consisting of abstracts of physics papers (see Shalev-Shwartz et al. (2011)). This data set consists of $d = 29,882$ samples with $n = 32,487$ fea-

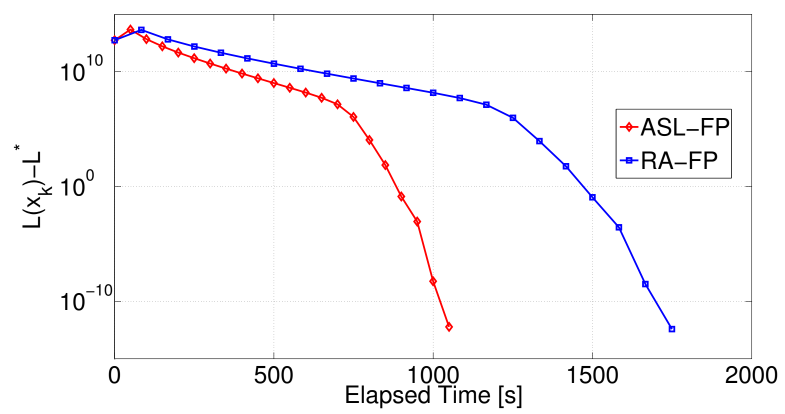


Figure 4: Evolution of $L(x_k) - L^*$ in time. ASL-FP significantly outperforms RA-FP. The loss L is pushed down by 25 degrees of magnitude in less than 30 minutes (3TB problem).

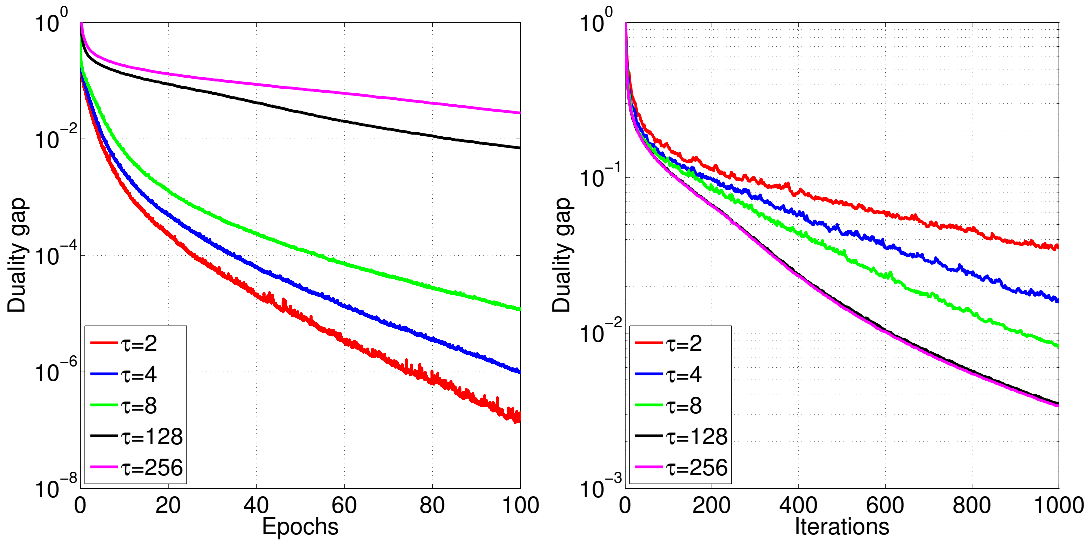


Figure 5: Evolution of the duality gap for several choices of τ.

tures; and hence is a relatively small data set. We have chosen a random partition of the coordinates (=samples) to $c = 32$ nodes, with each partition consisting of $s = 933$ samples (the last partition consists of 959 samples).

In Figure 5 we depict the evolution of the duality gap as a function of effective passes over data (epochs) (left plot) and as a function of iterations (right right). In each plot, we depict the performance of Hydra run with various choices of τ.

With larger τ, i.e., when each node updates more coordinates in a single iteration, more epochs are needed to obtain a solution of any given accuracy. However, if increasing τ by a fair amount only leads to a small increase in computation time within a node (say, because, each node utilizes a multicore processor and τ is proportional to the number of processors), then the increased number of passes over data will not be matched with a similar increase in compute time. The reverse side

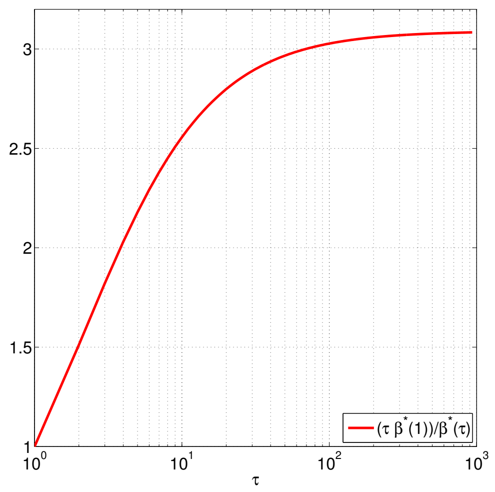


Figure 6: Evolution of the duality gap for several choices of τ.

of this situation is depicted in the right plot. That is, as τ increases, one needs fewer iterations to obtain any given accuracy.

In Figure 6 we depict the theoretical speed-up guarantee, i.e., the fraction $\frac{\tau}{\beta^*}$. Looking at the right plot of Figure 5, we see that the choice $\tau = 256$ does not lead to further speedup when compared to the $\tau = 128$ choice. This phenomenon is also captured in Figure 6, where the line becomes quite flat above $\tau \approx 10^2$.

.

Remark: We have used 10 iterations of the power method in order to estimate parameters σ and σ' which were used to obtain β^* using (10) (note that, in view of Lemma 2, we could have also used $\beta = 2\beta_1^*$). Their values are: $\sigma = 440.61$ and $\sigma' = 29.65$.

7.3 A Simple Model for Communication-Computation Trade-Off: Search for Optimal τ

Assume all c nodes of the cluster are identical. Also assume $s \geq 2$. Further, assume there is a constant $1 \leq K \leq s$ such that while each node is able to compute K coordinate updates in time T_1, the computation of jK updates, for $j = 2, 3, \dots$ takes jT_1 units of time. This will be approximately true in reality for K sufficiently large, typically a small multiple of the number of cores. It may be useful to think that K is, in fact, equal to the number of cores of each node. Further, assume that the time of updating g_k to g_{k+1} is T_2 (note that this involves communication).

Bases on this simple model, a single iteration of Hydra run with $\tau = jK$ coordinate updates on each node (for $1 \leq j \leq s/K$) takes

$$jT_1 + T_2 \tag{20}$$

units of time.

In what follows, we will assume, for simplicity, that $\mu_R = 0$ (i.e., the regularizer is not strongly convex). The computations simplify with this assumption, but similar computations can be performed in the $\mu_R > 0$ case.

Focusing on the terms which depend on τ, Theorem 5 says that to obtain an ϵ-solution, Hydra needs $\mathcal{O}(\frac{\beta}{\tau})$ iterations. Combining this with the time it takes to perform a single iteration, the overall

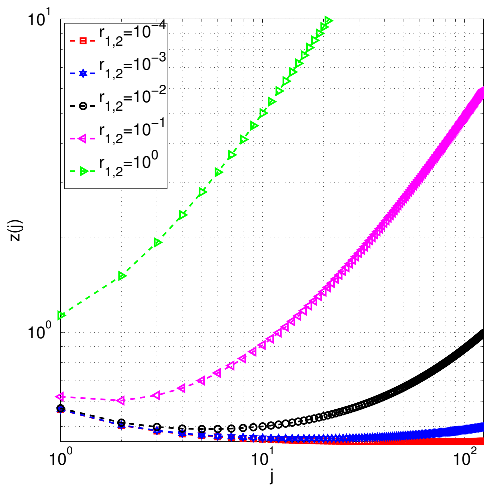


Figure 7: The dependence of the total "time complexity" $T(j)$ on j for various values of the computation-communication ratio $r_{1,2}$.

time, as a function of j, is proportional to

$$\begin{aligned} T(j) &:= \frac{\beta}{jK}(jT_1+T_2) \\ &\overset{(10)}{=} \frac{1+\frac{(jK-1)(\sigma-1)}{s_1}+\left(\frac{jK}{s}-\frac{jK-1}{s_1}\right)\frac{\sigma'-1}{\sigma'}\sigma}{jK}(\mathcal{T}_2+\mathcal{T}_1 j) \\ &= \frac{1+\frac{(jK-1)(\sigma-1)}{s_1}+\left(\frac{jK}{s}-\frac{jK-1}{s_1}\right)\frac{\sigma'-1}{\sigma'}\sigma}{jK}(1+jr_{1,2}), \end{aligned}$$

where

$$r_{1,2} := \frac{T_1}{T_2}.$$

Note that $r_{1,2}$ is high if computation is expensive relative to communication, and vice versa. This ratio can be estimated in any particular distributed system.

In Figure 7 we plot $T(j)$ as a function of j for $r_{1,2} \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$, $K = 8$ and the rest of the parameters appearing in the definition of $T(j)$ identical to those used in Section 7.2. For large enough $r_{1,2}$, i.e., if communication is relatively cheap, then T is an increasing function of j. This means that it is optimal to choose $j = 1$, which means that it is optimal to only update $\tau = jK$ coordinates on each node in each iteration. This observation is informally summarized as follows:

If communication is inexpensive relative to computation, it is optimal for each node to perform a small number of coordinate updates in each iteration. That is, if communication is cheap, do less computation in each iteration.

For small enough $r_{1,2}$, however, the optimal point j is strictly larger than 1 and smaller than s/K. As $r_{1,2}$ decreases, optimal j increases. This observation is informally summarized as:

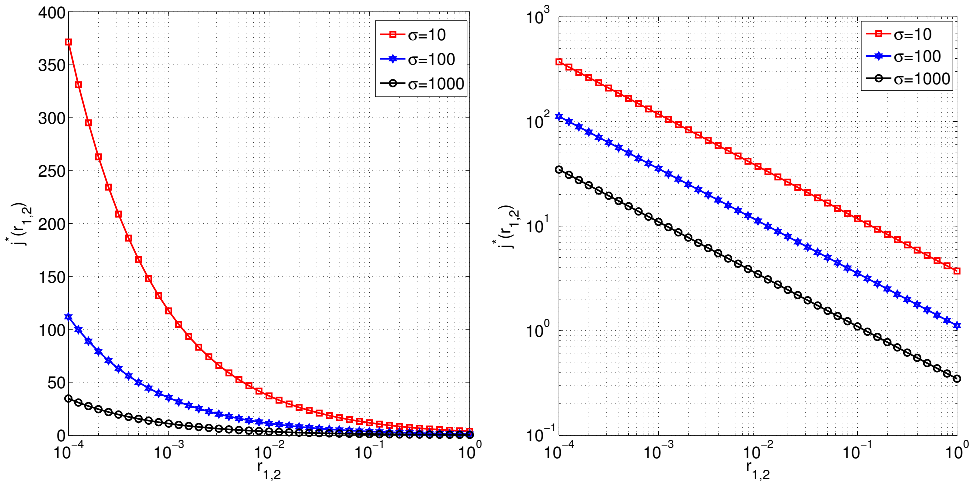


Figure 8: Comparison of $j^*(r_{1,2})$ for various values of parameter σ (left: semilog plot; right: loglog plot).

As communication becomes more expensive relative to computation, it is optimal for each node to perform a larger number of coordinate updates in each iteration. That is, if communication is more expensive, do more computation in each iteration.

Since T is a simple function of j, it is possible to find j^* which minimizes $T(j)$ in closed form (if we relax the requirement of keeping j integral, which is not very important):

$$j^* := \arg\min_j T(j) = \sqrt{\frac{s(s\sigma' - \sigma)}{r_{1,2}K(s\sigma'(\sigma-1) + \sigma - \sigma'\sigma)}}.$$

In Figure 8 we plot j^* as a function of $r_{1,2}$ for $\sigma = 10, 100, 1000$. Recall that smaller σ means less correlated data. This means that less synchronization is needed, and eventually allows us to do more coordinate updates in a single iteration (i.e., j^* is larger). If communication is relatively expensive ($r_{1,2}$ is small), then j^* is big. However, as communication gets cheaper, j^* gets smaller.

7.4 Experiments with a Real Data Set in a Real Distributed Environment

In this section we report on experiments with the WebSpam data set Libsvm. This data set encodes a binary classification problem with $d = 350,000$ samples and $n = 16,609,143$ features. The total size of the data set describing the problem exceeds 23 GB. We split this data into $c = 16$ balanced groups, each containing $s = 21,875$ samples and applied Hydra to the problem (the dual of SVM with hinge loss) for various choices of τ. The results are depicted in Figure 9. As each node in our experiments had an 8-core processor, we have chosen τ in multiples of 8: $\tau \in \{8, 80, 160, 320, 640, 1280, 2560, 5120\}$.

In the plot on the left we have run the RA variant with $\beta := 2\beta_1^*$ and in the plot on the right we have run the RA with $\beta := \frac{2\beta_1^*}{100}$. By comparing the plots, we can observe that for this particular data set, the theoretically safe choice of β ($\beta := 2\beta_1^*$) is too conservative. Indeed, massive speedups

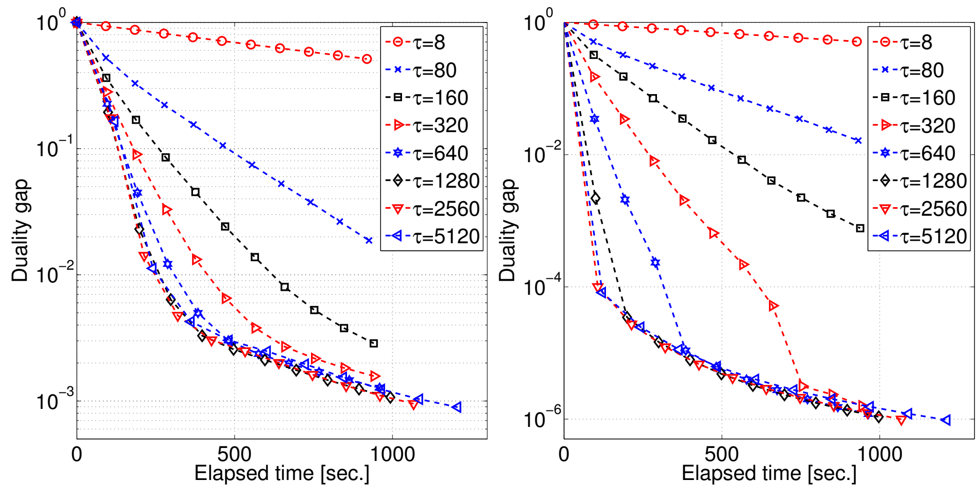


Figure 9: The duality gap evolving in time for Hydra run with various levels of parallelism (as given by τ) within each node.

can be obtained by using a more aggressive stepsize strategy, i.e., by choosing β hundred times smaller than the one recommended by theory. While it is to be expected that particular data sets will tolerate such aggressive stepsize strategies, there are data sets where such stepsizes might lead to a diverging algorithm. However, it is clear that Hydra would benefit from a line-search procedure for the selection of β.

One can also observe from Figure 9 that with beyond some point, increasing τ does not bring much benefit, and only increases the runtime.

8. Extensions

Our results can be extended to the setting where coordinates are replaced by blocks of coordinates, as in (Nesterov, 2012), and to partially separable losses, as in (Richtárik and Takáč, 2015). We expect the block setup to potentially lead to further significant speedups in a practical implementation due to the fact that this will allow us to design data-dependent block norms, which will in turn enable Hydra to use more curvature information the information contained in the diagonal of M. In such a setup, in Step 7 we will instead have a problem of a larger dimension, and the quadratic term can involve a submatrix of M.

Acknowledgements

The first author would like to acknowledge support from the EPSRC Grant EP/K02325X/1, “Accelerated Coordinate Descent Methods for Big Data Optimization” and EPSRC Grant EP/I017127/1, “Mathematics for Vast Digital Resources”. The second author was supported by the Centre for Numerical Algorithms and Intelligent Software (funded by EPSRC grant EP/G036136/1 and the Scottish Funding Council).

Appendix A. Proof Lemma 1

1. The inequality $\omega' \le c$ is obviously true. By considering x with zeroes in all coordinates except those that belong to $\mathcal{P}_k$ (where k is an arbitrary but fixed index), we see that $x^T \mathbf{Q} x = x^T B^{\mathbf{Q}} x$, and hence $\sigma' \ge 1$.

2. We now establish that $\sigma' \le \omega'$. Let $\phi(x) = \frac{1}{2} x^T \mathbf{Q} x$, $x \in \mathbb{R}^d$; its gradient is
$$\phi'(x) = \mathbf{Q}x. \tag{21}$$
For each $k = 1, 2, \dots, c$, define a pair of conjugate norms on $\mathbb{R}^s$ as follows:
$$\|v\|^2_{(k)} := \langle \mathbf{Q}^{kk} v, v\rangle, \qquad (\|v\|^*_{(k)})^2 := \max_{\|v'\|_{(k)} \le 1} \langle v', v\rangle = \langle (\mathbf{Q}^{kk})^{-1} v, v\rangle. \tag{22}$$
Let $\mathbf{U}_k$ be a column submatrix of the d-by-d identity matrix corresponding to columns $i \in \mathcal{P}_k$. Clearly, $\mathbf{A}_k = \mathbf{A}\mathbf{U}_k$ and $\mathbf{U}_k^T \mathbf{Q} \mathbf{U}_k$ is the k-th diagonal block of $\mathbf{Q}$, i.e.,
$$\mathbf{U}_k^T \mathbf{Q} \mathbf{U}_k \overset{(4)}{=} \mathbf{Q}^{kk}. \tag{23}$$
Moreover, for $x \in \mathbb{R}^d$ and $k \in \{1, 2, \dots, c\}$, let $x^{(k)} = \mathbf{U}_k^T x$ and, fixing positive scalars $w_1, \dots, w_c$, define a norm on $\mathbb{R}^d$ as follows:
$$\|x\|_w := \left(\sum_{k=1}^c w_k \|x^{(k)}\|^2_{(k)} \right)^{1/2}. \tag{24}$$
Now, we claim that for each k,
$$\|\mathbf{U}_k^T \phi'(x + \mathbf{U}_k h^{(k)}) - \mathbf{U}_k^T \phi'(x)\|^*_{(k)} \le \|h^{(k)}\|_{(i)}.$$
This means that ϕ' is block Lipschitz (with blocks corresponding to variables in $\mathcal{P}_k$), with respect to the norm $\|\cdot\|_{(k)}$, with Lipschitz constant 1. Indeed, this is, in fact, satisfied with equality:
$$\begin{aligned} \|\mathbf{U}_k^T \phi'(x + \mathbf{U}_k h^{(k)}) - \mathbf{U}_k^T \phi'(x)\|^*_{(k)} &\overset{(21)}{=} \|\mathbf{U}_k^T \mathbf{Q}(x + \mathbf{U}_k h^{(k)}) - \mathbf{U}_k \mathbf{Q} x\|^*_{(k)} \\ &= \|\mathbf{U}_k^T \mathbf{Q} \mathbf{U}_k h^{(k)}\|^*_{(k)} \\ &\overset{(23)}{=} \|\mathbf{Q}^{kk} h^{(k)}\|^*_{(k)} \\ &\overset{(22)}{=} \langle (\mathbf{Q}^{kk})^{-1} \mathbf{Q}^{kk} h^{(k)}, \mathbf{Q}^{kk} h^{(k)}\rangle \\ &\overset{(22)}{=} \|h^{(k)}\|_{(k)}. \end{aligned}$$
This is relevant because then, by Richtárik and Takáč (2015, Theorem 7; see comment 2 following the theorem), it follows that ϕ' is Lipschitz with respect to $\|\cdot\|_w$, where $w_k = 1$ for all $k = 1, \dots, c$, with Lipschitz constant ω' (ω' is the degree of partial block separability of ϕ with respect to the blocks $\mathcal{P}_k$). Hence,
$$\tfrac{1}{2} x^T \mathbf{Q} x = \phi(x) \le \phi(0) + (\phi'(0))^T x + \frac{\omega'}{2}\|x\|^2_w \overset{(22)+(24)}{=} \frac{\omega'}{2} \sum_{k=1}^c \langle \mathbf{Q}^{kk} x^{(k)}, x^{(k)}\rangle = \frac{\omega'}{2}(x^T B^{\mathbf{Q}} x),$$
which establishes the inequality $\sigma' \le \omega'$.

3. We now show that $\frac{\sigma}{s} \leq \sigma'$. If we let $\theta := \max\{x^T B^{\mathbf{Q}} x : x^T x \leq 1\}$, then $x^T B^{\mathbf{Q}} x \leq \theta x^T x$ and hence $\{x \ : \ x^T x \leq 1\} \subseteq \{x \ : \ x^T B^{\mathbf{Q}} x \leq \theta\}$. This implies that

$$\sigma = \max_x \{x^T \mathbf{Q} x \ : \ x^T x \leq 1\} \leq \max_x \{x^T \mathbf{Q} x \ : \ x^T B^{\mathbf{Q}} x \leq \theta\} = \theta \sigma'.$$

   It now only remains to argue that $\theta \leq s$. For $x \in \mathbb{R}^d$, let $x^{(k)}$ denote its subvector in $\mathbb{R}^s$ corresponding to coordinates $i \in \mathcal{P}_k$ and $\Delta = \{p \in \mathbb{R}^c : p \geq 0, \ \sum_{k=1}^c p_k = 1\}$. We can now write

$$\begin{aligned}
\theta &= \max_x \left\{ \sum_{k=1}^c (x^{(k)})^T \mathbf{Q}^{kk} x^{(k)} \ : \ \sum_{k=1}^c (x^{(k)})^T x^{(k)} \leq 1 \right\} \\
&= \max_{p \in \Delta} \sum_{k=1}^c \left\{ \max (x^{(k)})^T \mathbf{Q}^{kk} x^{(k)} \ : \ (x^{(k)})^T x^{(k)} = p_k \right\} \\
&= \max_{p \in \Delta} \sum_{k=1}^c p_k \max \left\{ (x^{(k)})^T \mathbf{Q}^{kk} x^{(k)} \ : \ (x^{(k)})^T x^{(k)} = 1 \right\} \\
&= \max_{1 \leq k \leq c} \max \left\{ (x^{(k)})^T \mathbf{Q}^{kk} x^{(k)} \ : \ (x^{(k)})^T x^{(k)} = 1 \right\} \\
&\leq s.
\end{aligned}$$

   In the last step we have used the fact that $\sigma(\mathbf{Q}) = \sigma \leq c = \dim(\mathbf{Q})$, proved in steps 1 and 2, applied to the setting $\mathbf{Q} \leftarrow \mathbf{Q}^{kk}$.

4. The chain of inequalities $1 \leq \sigma \leq \omega \leq c$ is obtained as a special case of the chain $1 \leq \sigma' \leq \omega' \leq d$ (proved above) when $c = d$ (and hence $\mathcal{P}_l = \{l\}$ for $l = 1, \ldots, d$). Indeed, in this case $B^{\mathbf{Q}} = D^{\mathbf{Q}}$, and so $x^T B^{\mathbf{Q}} x = x^T D^{\mathbf{Q}} x = x^T x$, which means that $\sigma' = \sigma$ and $\omega' = \omega$.

References

J. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *ICML*, 2011.

O. Fercoq. Parallel coordinate descent for the AdaBoost problem. In *ICMLA*, 2013.

O. Fercoq and P. Richtárik. Smooth minimization of nonsmooth functions with parallel coordinate descent methods. *arXiv:1309.5885*, 2013.

O. Fercoq and P. Richtárik. Accelerated, parallel and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

C-J. Hsieh, K-W. Chang, C-J. Lin, S.S. Keerthi, , and S. Sundarajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, 2008.

Libsvm. *Datasets*. http://www.csie.ntu.edu.tw/~cjlin/ libsvmtools/datasets/binary.html.

Z. Lu and L. Xiao. On the complexity analysis of randomized block-coordinate descent methods. *Mathematical Programming*, 152(1):615–642, 2015.

I. Mukherjee, Y. Singer, R. Frongillo, and K. Canini. Parallel boosting with momentum. In *ECML*, 2013.

I. Necoara, Yu. Nesterov, and F. Glineur. Efficiency of randomized coordinate descent methods on optimization problems with linearly coupled constraints. Technical report, 2012.

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

Yu. Nesterov. Gradient methods for minimizing composite objective function. *Mathematical Programming*, 140(1):125–161, 2013.

P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, pages 1–52, 2015.

P. Richtárik and M. Takáč. Efficient serial and parallel coordinate descent methods for huge-scale truss topology design. In *Operations Research Proceedings*, pages 27–32. Springer, 2012.

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

S. Shalev-Shwartz and A. Tewari. Stochastic methods for ℓ_1 regularized loss minimization. In *ICML*, 2009.

S. Shalev-Shwartz and T. Zhang. Accelerated mini-batch stochastic dual coordinate ascent. In *NIPS*, pages 378–385, 2013a.

S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599, 2013b.

S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

M. Takáč, A. Bijral, P. Richtárik, and N. Srebro. Mini-batch primal and dual methods for SVMs. In *ICML*, 2013.

R. Tappenden, P. Richtárik, and J. Gondzio. Inexact coordinate descent: complexity and preconditioning. *arXiv:1304.5530*, 2013.

R. Tappenden, P. Richtárik, and B. Büke. Separable approximations and decomposition methods for the augmented Lagrangian. *Optimization Methods and Software*, 30(3):643–668, 2015.