Fixing the Convergence Problems in Parallel Asynchronous Dual Coordinate Descent

Huan Zhang
Dept. of Electrical and Computer Engineering
University of California, Davis
Davis, CA 95616, USA
Email: ecezhang@ucdavis.edu

Cho-Jui Hsieh
Depts. of Computer Science and Statistics
University of California, Davis
Davis, CA 95616, USA
Email: chohsieh@ucdavis.edu

***Abstract*—Solving L2-regularized empirical risk minimization (e.g., linear SVMs and logistic regression) using multiple cores has become an important research topic. Among all the existing algorithms, Parallel ASynchronous Stochastic dual Co-Ordinate DEscent (*PASSCoDe*) demonstrates superior performance compared with other methods. Although *PASSCoDe* is fast when it converges, the algorithm has been observed to diverge on several cases especially when a relatively large number of threads are used. This is mainly due to the delayed parameter access problem—the parameters used for the current update may be delayed and are not the latest ones. In theory, the algorithm converges only when the delay is small enough, but in practice the delay depends on the underlying parallel computing environment and cannot be guaranteed. In this work, we propose a simple and computational efficient way to fix the convergence problem of *PASSCoDe*. Instead of allowing all worker threads to conduct asynchronous updates wildly, we add periodic check points to the procedure, where all workers need to stop and refine the current solution at each check point. The resulting "semi-asynchronous" algorithm is guaranteed to converge for any problem even when *PASSCoDe* diverges, and for the cases where *PASSCoDe* converges they have almost identical speed.**

***Keywords*—Coordinate descent, Asynchronous algorithm**

I. Introduction

Many machine learning problems involve solving L2-regularized Empirical Risk Minimization (ERM) problems. Famous examples include Support Vector Machines (SVMs) [3] and logistic regression. Many optimization techniques have been developed for solving these problems [15], [23], [10], [6]. Among these algorithms, Stochastic Dual Coordinate Descent (*DCD*) [6], [16] outperforms others for solving large-scale linear SVMs and logistic regression, and has been implemented as the default solver in LIBLINEAR [5].

Due to the increasing need of handling big data, parallelizing stochastic dual coordinate descent has become a very important research topic. For multi-core shared memory systems, Parallel Asynchronous Stochastic Dual Coordinate Descent (*PASSCoDe*) proposed in [7] is a simple but efficient way to parallelize *DCD*. In this algorithm, all the threads conduct dual coordinate descent updates asynchronously in parallel, and the communication is implicitly done by reading and writing variables stored in the shared memory space. Since each thread does not need to idle and wait for other threads, *PASSCoDe* enjoys good speedup and outperforms existing

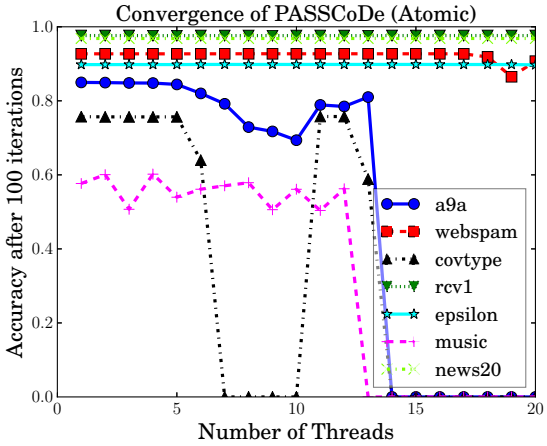


Fig. 1: We run *PASSCoDe-Atomic* for SVM with squared hinge loss on 7 classification datasets on a dual-socket 20-core machine, and collect prediction accuracy after 100 iterations. Accuracy drops to 0 when *PASSCoDe-Atomic* outputs `NaN`, indicating it has severely diverged. We vary the number of threads from 1 to 20, and on 3 of the 7 datasets (a9a, covtype, music) *PASSCoDe-Atomic* diverges before we reach the maximum available parallelism. *PASSCoDe-Atomic* becomes less stable on webspam when using 20 threads. Also, when using 11 threads, *PASSCoDe-Atomic* behaves quite differently than using 10 threads on a9a and covtype. This is because starting from the 11-th thread, threads are allocated to the second CPU socket.

parallel algorithms. However, the convergence properties of *PASSCoDe* is non-trivial since asynchronous updates lead to *delayed parameter access*—the parameters used for the current update may not be the latest ones. Fortunately, Hsieh et al. [7] theoretically proves that the algorithm will converge *if this "delay" is small enough*. It has been shown in [7] that the algorithm performs very well on many large-scale datasets.

Although *PASSCoDe* is fast when it converges, it is hard to know whether the "delay" is small enough in practice. Since this delay is determined by the underlying hardware and cannot be easily adjusted, we never know whether the algorithm will converge. Figure 1 shows that *PASSCoDe* diverges on some real-life classification datasets, and it can diverge when only 6 threads are used (in covtype). Also, *PASSCoDe*'s performance is sensitive to the underlying parallel computing environment, because we observe that for a9a and covtype, *PASSCoDe* behaves quite differently when the second CPU socket starts being used. With this issue, it is questionable whether asynchronous dual coordinate descent algorithm is

useful in practice.

In this paper, we propose a simple and computational efficient way to fix this convergence problem in parallel asynchronous stochastic dual coordinate descent (*PASSCoDe*). Our proposed algorithm, *PASSCoDe-fix*, is as fast as *PASSCoDe* but is guaranteed to converge to the optimal solution. Our contributions can be summarized below:

- To avoid divergence, we design a procedure to periodically adjust the dual coordinate descent updates by scaling the update vector, and this scaling factor can be computed very efficiently by exploiting the primal-dual relationship.
- In some cases the updates are in totally wrong directions due to a huge delay, and no scaling factor can sufficiently decrease the dual objective. When this is the case, we use a modified *PASSCoDe* algorithm with a conservative step size $\gamma < 1$ for future iterations. We reduce γ every time when this situation is detected, and this allows the value of γ to be automatically adjusted.
- The resulting algorithm, *PASSCoDe-fix*, is simple, efficient, and guaranteed to converge to the optimal solution. We show empirically that *PASSCoDe-fix* converges for all datasets even when *PASSCoDe* diverges, and for the cases where *PASSCoDe* converges they have almost identical speed.

The rest of the paper is organized as follows: in Section II we summarize related work. In Section III we introduce the *PASSCoDe* algorithm proposed in [7]. In Section IV we detail our proposed algorithm and show the algorithm is guaranteed to converge. In Section V we conduct experiments on real-world datasets. The work is concluded in Section VI, and all the detailed proofs are in Appendix (Section VII).

II. Related Work

Stochastic (Dual) Coordinate Descent. Coordinate descent is a classical optimization algorithm that updates a single variable at a time and has been studied extensively. Recently, several papers showed coordinate descent is very efficient for solving machine learning problems when the variable-to-update is chosen randomly [12], [6]. In this paper, we focus on the family of Dual Coordinate Descent (*DCD*) algorithms that applies stochastic coordinate descent to solve the dual problem while maintaining the primal solution via the primal-dual relationship. This was first used for solving linear SVMs in [6] and discussed as a general framework in [16]. It has been shown that *DCD* is one of the most efficient algorithms for solving L2-regularized empirical risk minimization problems including SVMs [6], logistic regression [19], multi-class classification [8] and many others.

Parallel Asynchronous (Dual) Coordinate Descent. Due to the success of stochastic (dual) coordinate descent, its parallel counterpart has become an important research topic (see [2], [14]). In a multi-core shared memory system, parallel asynchronous coordinate descent ([11], [1]) is the most efficient approach, where each thread conducts updates independently, and the communication is implicitly done by accessing variables in the shared memory space. Recently, [7] (*PASSCoDe*) extended the algorithm to dual coordinate descent and demonstrates good speedup on large datasets.

Other Parallel Asynchronous Algorithms. Due to the efficiency of asynchronous updates, there are many other asynchronous algorithms proposed recently. [13], [22] proposed an asynchronous stochastic gradient descent algorithm and has been widely used. [4] applied an asynchronous algorithm on distributed system for deep learning. [20], [18] proposed asynchronous algorithms for matrix completion and Latent Dirichlet Allocation (LDA). [9], [17] developed parameter servers for conducting asynchronous updates on distributed systems. In this paper, we focus only on fixing the convergence problem of asynchronous dual coordinate descent, but the problem of delayed parameter access happens in almost all the asynchronous algorithms, so it might be interesting to extend our idea to fix or improve the convergence of other algorithms.

III. Background

We first review parallel asynchronous stochastic dual coordinate descent proposed in [7]. Given training samples $\{\boldsymbol{x}_i\}_{i=1}^n$ and corresponding labels $\{y_i\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and y_i can be either ± 1 or real numbers, we focus on the following ℓ_2-regularized Empirical Risk Minimization (ERM) problem:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} P(\boldsymbol{w}) := \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^n \ell_i(\boldsymbol{w}^T\boldsymbol{x}_i), \tag{1}$$

where $\ell_i(\cdot)$ is the loss function that may depend on labels, and $\|\cdot\|$ is the 2-norm. Many machine learning problems belong to this category, including SVM (hinge loss), L2-SVM (squared hinge loss), logistic regression (logistic loss) and ridge regression (square loss). The dual problem of (1) can be written as

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} D(\boldsymbol{\alpha}) := \frac{1}{2}\left\|\sum_{i=1}^n \alpha_i\boldsymbol{x}_i\right\|^2 + \sum_{i=1}^n \ell_i^*(-\alpha_i), \tag{2}$$

where $\ell_i^*(\cdot)$ is the conjugate of the loss function $\ell_i(\cdot)$, defined by $\ell_i^*(u) = \max_z(zu - \ell_i(z))$. For each dual vector $\boldsymbol{\alpha}$, we can define the corresponding primal vector $\boldsymbol{w}$ by the following primal-dual relationship (see [16] for more details):

$$\boldsymbol{w}(\boldsymbol{\alpha}) = \sum\nolimits_{i=1}^n \alpha_i\boldsymbol{x}_i. \tag{3}$$

At the optima, $\boldsymbol{w}(\boldsymbol{\alpha}^*) = \boldsymbol{w}^*$ and $P(\boldsymbol{w}^*) = -D(\boldsymbol{\alpha}^*)$ where $\boldsymbol{w}^*, \boldsymbol{\alpha}^*$ are the primal and dual optimal solutions respectively.

A. Stochastic Dual Coordinate Descent

Stochastic Dual Coordinate Descent (*DCD*) is one of the most widely-used approach to solve the L2-regularized ERM problem and is implemented in LIBLINEAR [5]. It solves the dual form of ERM problem (2) while in the same time maintaining a primal solution by Eq (3).

At each iteration, *DCD* randomly selects a dual variable α_i and updates its value by $\alpha_i \leftarrow \alpha_i + \delta^*$, where

$$\begin{aligned}\delta^* &= \arg\min_\delta D(\boldsymbol{\alpha} + \delta\boldsymbol{e}_i) \\ &= \arg\min_\delta \frac{1}{2}(\delta + \frac{\boldsymbol{w}^T\boldsymbol{x}_i}{\|\boldsymbol{x}_i\|^2})^2 + \frac{1}{\|\boldsymbol{x}_i\|^2}\ell_i^*(-(\alpha_i+\delta)).\end{aligned} \tag{4}$$

This is just a single-variate proximal operator, so can often be solved in $O(1)$ time if $\boldsymbol{w}^T\boldsymbol{x}_i$ and $\|\boldsymbol{x}_i\|$ are known. $\|\boldsymbol{x}_i\|$ can be pre-computed and remains unchanged during the whole optimization procedure. Therefore, the main cost of computing δ^* is to compute $\boldsymbol{w}^T\boldsymbol{x}_i$. Without storing $\boldsymbol{w}$, we have to re-compute $\boldsymbol{w}$ by (3) every time, which requires $O(\text{nnz})$ time complexity where nnz is the total number of nonzero elements in the training data. The main trick proposed in *DCD* ([6], [16]) is to maintain $\boldsymbol{w}$ in the memory, so the computational cost of (4) can be reduced to $O(\bar{d})$, where $\bar{d} = (\text{nnz})/n$ is the averaged number of nonzero elements in each training sample. As a result, the computational complexity is exactly the same as primal-SGD, while *DCD* converges faster and there is no need to select the step size. *DCD* is presented in Algorithm 1.

Algorithm 1 Stochastic Dual Coordinate Descent (*DCD*)

Require: Initial $\boldsymbol{\alpha}$ and $\boldsymbol{w} = \sum_{i=1}^n \alpha_i \boldsymbol{x}_i$

1: **while** not converged **do**
2: Randomly pick i
3: Update $\alpha_i \leftarrow \alpha_i + \delta^*$, where

$$\delta^* \leftarrow \arg\min_{\delta} \frac{1}{2}(\delta + \frac{\boldsymbol{w}^T\boldsymbol{x}_i}{\|\boldsymbol{x}_i\|^2})^2 + \frac{1}{\|\boldsymbol{x}_i\|^2}\ell_i^*(-(\alpha_i+\delta)).$$

4: Update $\boldsymbol{w}$ by $\boldsymbol{w} \leftarrow \boldsymbol{w} + \delta^*\boldsymbol{x}_i$
5: **end while**

B. Parallel Asynchronous Dual Coordinate Descent

Next we discuss the Parallel Asynchronous Dual Coordinate Descent (*PASSCoDe*) algorithm proposed in [7] that parallelizes *DCD* in multi-core shared memory systems. In this algorithm, each thread repeatedly conducts coordinate updates (step 2 to 4 in Algorithm 1) in parallel. Threads do not need to explicitly synchronize the parameters because $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ will be read from and written to the same shared memory space.

In [7], the authors proposed two ways[1] to maintain $\boldsymbol{w}$—*atomic* or *wild*. In the wild version, the updates of $\boldsymbol{w}$ in step 4 of Algorithm 1 are unprotected and can be overwritten by other threads, so it cannot converge to the optimal solution of (2). In the atomic version, each update in step 4 uses an *atomic read-modify-write* operation, which is slightly slower than wild updates but ensures that the writes are not overwritten by other threads. Therefore, the atomic version is more stable and has been further used in practice. This algorithm is presented in Algorithm 2.

C. Convergence Properties of PASSCoDe-Atomic

To discuss the convergence properties of Algorithm 2, [7] assigns a global counter j for the updates and use $\{\boldsymbol{\alpha}^1, \boldsymbol{\alpha}^2, \cdots\}$ to denote the sequence of dual solutions. The $\hat{\boldsymbol{w}}$ used for computing the update from $\boldsymbol{\alpha}^j$ to $\boldsymbol{\alpha}^{j+1}$ is the "delayed" $\boldsymbol{w}$ but the delay will be smaller than τ (see the definition in (11)). This means all the updates before iteration $(j-\tau)$ are finished while updates between iteration $(j-\tau+1)$ and $(j-1)$ may or may not be finished. In this case, [7] proves the following lemma, showing that *PASSCoDe-Atomic* converges to the optimal solution when τ is small enough:

[1]The authors of [7] also discussed another way of using explicit locking but they showed that it is even slower than the single-thread version, so we do not discuss it here.

Algorithm 2 Parallel Asynchronous Stochastic dual Coordinate Descent (*PASSCoDe-Atomic*)

Require: Initial $\boldsymbol{\alpha}$ and $\boldsymbol{w} = \sum_{i=1}^n \alpha_i \boldsymbol{x}_i$

Each thread repeatedly performs the following updates:
step 1: Randomly pick i
step 2: Update $\alpha_i \leftarrow \alpha_i + \delta^*$, where

$$\delta^* \leftarrow \arg\min_{\delta} \frac{1}{2}(\delta + \frac{\boldsymbol{w}^T\boldsymbol{x}_i}{\|\boldsymbol{x}_i\|^2})^2 + \frac{1}{\|\boldsymbol{x}_i\|^2}\ell_i^*(-(\alpha_i+\delta)).$$

step 3: For each $j \in N(i) := \{t \mid (\boldsymbol{x}_i)_t \neq 0\}$
Update $w_j \leftarrow w_j + \delta^*(\boldsymbol{x}_i)_j$ **atomically**

Lemma 1 (A simplified version of Theorem 1 in [7])**.** *Assume the objective function admits the global error bound (see Definition 1 in [7]) and is Lipschitz continuous. Then there exist constants A, η such that when $\tau < A$, PASSCoDe-Atomic converges to the global optimum with a linear rate:*

$$E[D(\boldsymbol{\alpha}^{j+1})] - D(\boldsymbol{\alpha}^*) \leq \eta \left(E[D(\boldsymbol{\alpha}^j)] - D(\boldsymbol{\alpha}^*)\right),$$

where $\boldsymbol{\alpha}^*$ is the optimal solution.

Unfortunately, Lemma 1 suggests that the convergence of *PASSCoDe-Atomic* is guaranteed **only if the delay τ is sufficiently small**, and we have observed that on some real datasets the algorithm will eventually diverge (see Figure 1). In the next section, we propose an algorithm, *PASSCoDe-fix*, to fix the convergence problems of *PASSCoDe-Atomic*.

IV. Proposed Algorithm

In *PASSCoDe-fix*, we periodically check and fix the updates to ensure the convergence. The framework is presented in Algorithm 3. Step 2 is the original *PASSCoDe* algorithm; after running *PASSCoDe* for N coordinate updates, we propose a new procedure to adjust the solution in Step 3 to avoid divergence. This framework can be conceptually viewed as a family of **semi-asynchronous** algorithms—instead of allowing all the threads conduct asynchronous updates wildly, we add periodic check points to the algorithm to ensure the convergence. When N is large and the checkpoints (step 3) are computational efficient, the run time of the semi-asynchronous algorithm will be almost identical to the fully asynchronous version.

Now we propose an *efficient* and *simple* fixing procedure for step 3 so that (1) the algorithm is guaranteed to converge and (2) the fixing procedure is quick and parallelizable.

A. How to adjust the solution in Step 3?

We define the global dual and primal updates in Step 2 of Algorithm 3 by:

$$\Delta\boldsymbol{\alpha} := \boldsymbol{\alpha} - \boldsymbol{\alpha}^{\text{old}},\ \Delta\boldsymbol{w} := \boldsymbol{w} - \boldsymbol{w}^{\text{old}},$$

Algorithm 3 Our proposed framework—An Overview

Require: Initial $\boldsymbol{\alpha}$ and $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i$

For outer_iter $= 1, 2, \cdots$

1. $\boldsymbol{\alpha}^{\text{old}} \leftarrow \boldsymbol{\alpha}$, $\boldsymbol{w}^{\text{old}} \leftarrow \boldsymbol{w}$
2. Run *PASSCoDe* with totally N updates using all the threads to obtain $\boldsymbol{\alpha}$, $\boldsymbol{w}$
3. Adjust solution $\boldsymbol{\alpha}, \boldsymbol{w}$ based on $\boldsymbol{\alpha}^{\text{old}}, \boldsymbol{w}^{\text{old}}$

and we adjust the dual and primal solutions by

$$\boldsymbol{\alpha} := \boldsymbol{\alpha}^{\text{old}} + \beta \Delta \boldsymbol{\alpha}, \quad \boldsymbol{w} := \boldsymbol{w}^{\text{old}} + \beta \Delta \boldsymbol{w}. \tag{5}$$

If $\beta = 1$ then there will be no adjustment made by this procedure, so the algorithm is identical to *PASSCoDe*. However, for the cases where *PASSCoDe* diverges, a smaller $\beta \in [0, 1)$ can be used to keep the dual objective function value from increasing. Therefore, we want to develop an *efficient* way to find a β such that

$$D(\boldsymbol{\alpha}^{\text{old}} + \beta \Delta \boldsymbol{\alpha}) \text{ is sufficiently smaller than } D(\boldsymbol{\alpha}^{\text{old}}).$$

We consider two cases of the loss functions $\ell(\cdot)$. In the first case, the dual loss $\ell^*(\cdot)$ can be written as a quadratic function with bounded constraints, and in the second case the dual loss can be any general function. The first case include Support Vector Machine (SVM) with hinge loss and L2-SVM (with squared hinge loss), and we show there is a *closed form solution* for the optimal β. The second case include logistic regression, and we propose an efficient line-search to find a good β value.

Case I: we assume the dual loss can be written as

$$\ell_i^*(\alpha_i) = a_i \alpha_i + b_i \alpha_i^2 + I(\alpha_i \in [c_i, d_i]), \quad \forall i \tag{6}$$

where a_i, c_i, d_i are arbitrary real numbers, $b_i \geq 0$ since $\ell_i^*(\alpha_i)$ is convex, and $I(\alpha_i \in [c_i, d_i]) = 0$ if $\alpha_i \in [c_i, d_i]$, $I(\alpha_i \in [c_i, d_i]) = \infty$ if $\alpha_i \notin [c_i, d_i]$. SVM and L2-SVM belongs to this category:

$$\begin{aligned} \text{SVM: } \ell_i(\boldsymbol{w}^T \boldsymbol{x}_i) &= \max(1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i, 0), \\ \ell_i^*(\alpha_i) &= y_i \alpha_i + I(y_i \alpha_i \in [-1, 0]) \\ \text{L2-SVM: } \ell_i(\boldsymbol{w}^T \boldsymbol{x}_i) &= \max(1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i, 0)^2, \\ \ell_i^*(\alpha_i) &= \alpha_i^2/4 + y_i \alpha_i + I(y_i \alpha_i \in [-\infty, 0]). \end{aligned}$$

In this case, the dual objective function can be written as:

$$\begin{aligned} D(\boldsymbol{\alpha}^{\text{old}} + \beta \Delta \boldsymbol{\alpha}) &= \frac{1}{2} \|\boldsymbol{w}^{\text{old}} + \beta \Delta \boldsymbol{w}\|^2 + \sum_{i=1}^{n} \ell_i^*(-\alpha_i^{\text{old}} - \beta \Delta \alpha_i) \\ &= \left(\frac{1}{2} \Delta \boldsymbol{w}^T \Delta \boldsymbol{w} + \sum_{i=1}^{n} b_i \Delta \alpha_i^2 \right) \beta^2 + \left(\Delta \boldsymbol{w}^T \boldsymbol{w}^{\text{old}} - \sum_{i=1}^{n} a_i \Delta \alpha_i \right. \\ &\quad \left. + \sum_{i=1}^{n} 2 b_i \alpha_i^{\text{old}} \Delta \alpha_i \right) \beta + \sum_{i=1}^{n} I(-\alpha_i^{\text{old}} - \beta \Delta \alpha_i \in [c_i, d_i]). \end{aligned}$$

Both α_i^{old} (the previous iterate) and the current iterate $\alpha_i^{\text{old}} + \Delta \alpha_i$ belong to $[c_i, d_i]$ due to the *DCD* update rule, so the constraints will hold for all $\beta \in [0, 1]$. We then propose to find the optimal β in this range:

$$\begin{aligned} \beta^* &= \arg \min_{\beta \in [0,\ 1]} D(\boldsymbol{\alpha}^{\text{old}} + \beta \Delta \boldsymbol{\alpha}) \\ &= \arg \min_{\beta \in [0,\ 1]} A\beta^2 + B\beta \\ &= \max(\min(-B/(2A), 1), 0) \end{aligned} \tag{7}$$

where

$$\begin{aligned} A &= \frac{1}{2} \Delta \boldsymbol{w}^T \Delta \boldsymbol{w} + \sum_{i=1}^{n} b_i \Delta \alpha_i^2 \geq 0 \\ B &= \Delta \boldsymbol{w}^T \boldsymbol{w}^{\text{old}} - \sum_{i=1}^{n} a_i \Delta \alpha_i + \sum_{i=1}^{n} 2 b_i \alpha_i^{\text{old}} \Delta \alpha_i. \end{aligned} \tag{8}$$

Both A and B can be computed in $O(n + d)$ time, so the time complexity for computing β^* is only $O(n + d)$. When we set $N = n$ in Algorithm 3, *PASSCoDe* in step 2 will require $O(\text{nnz})$ time, so the amortized cost is very small. Furthermore, computing A and B can be embarrassingly parallelized. Finally, we need to discuss the case when $\beta^* = 0$. From (7) we can see

$$\beta^* = 0 \ \Leftrightarrow \ B \geq 0 \ \Leftrightarrow \ \partial_\beta D(\boldsymbol{\alpha}^{\text{old}} + \beta \Delta \boldsymbol{\alpha}) \mid_{\beta = 0} \geq 0.$$

This means $\Delta \boldsymbol{\alpha}$ computed by *PASSCoDe* is not even a descent direction. In this case, $\boldsymbol{\alpha}$ remains unchanged at this iteration, and we will discuss how to handle this in the next subsection (Section IV-B).

Case II: in the second case we consider a general loss function where the dual loss cannot be written as (6). In this case, the optimal β may not have a closed form solution, so we conduct a typical Armijo-rule backtracking line search to search for a good β. More specifically, we try $\beta = \{1, \frac{1}{2}, \frac{1}{4}, \cdots\}$ until it satisfies the following sufficient decrease condition:

$$D(\boldsymbol{\alpha}^{\text{old}} + \beta \Delta \boldsymbol{\alpha}) \leq D(\boldsymbol{\alpha}^{\text{old}}) - \sigma \beta \left| \partial_\beta D(\boldsymbol{\alpha}^{\text{old}} + \beta \Delta \boldsymbol{\alpha}) \mid_{\beta = 0} \right|$$

with some constant $\sigma \in (0, 1)$. Computing

$$\partial_\beta D(\boldsymbol{\alpha}^{\text{old}} + \beta \Delta \boldsymbol{\alpha}) \mid_{\beta = 0} = \Delta \boldsymbol{w}^T \boldsymbol{w}^{\text{old}} - \sum_{i=1}^{n} (\Delta \alpha_i)(\partial \ell_i^*(-\alpha_i^{\text{old}}))$$

requires $O(n + d)$ time complexity and is just a one-time cost. For each β, the dual objective function can be computed efficiently by exploiting the primal-dual structure:

$$\begin{aligned} D(\boldsymbol{\alpha}^{\text{old}} + \beta \Delta \boldsymbol{\alpha}) = &\frac{1}{2} \|\boldsymbol{w}^{\text{old}}\|^2 + (\Delta \boldsymbol{w}^T \boldsymbol{w}^{\text{old}}) \beta + \frac{\|\Delta \boldsymbol{w}\|^2}{2} \beta^2 \\ &+ \sum_{i=1}^{n} \ell_i^*(-\alpha_i^{\text{old}} - \beta \Delta \alpha_i). \end{aligned} \tag{9}$$

$\Delta \boldsymbol{w}^T \boldsymbol{w}^{\text{old}}$ and $\|\Delta \boldsymbol{w}\|^2$ can be pre-computed and used for any β, so each function value evaluation of (9) only requires $O(n)$ time complexity.

In summary, if the dual loss cannot be written as (6), we use a line search procedure to find a β. If there are T line search steps, the overall procedure (Step 3 of Algorithm 3) requires

$O(d + Tn)$ time complexity. This can also be parallelized, and the amortized cost per coordinate descent update is not too high. However, the same problem in case I may also arise here: if $\Delta\boldsymbol{\alpha}$ is not a descent direction (subgradient $\partial_\beta D(\boldsymbol{\alpha}^{\text{old}} + \beta\Delta\boldsymbol{\alpha}) \mid_{\beta=0}$ is non-negative), the step size β will be 0, and in this iteration we cannot make any progress.

B. How to handle the case when $\Delta\boldsymbol{\alpha}$ is not a descent direction?

In the previous section we have discussed how to adjust the updates when $\Delta\boldsymbol{\alpha}$ computed in step 2 of Algorithm 3 *is a descent direction* ($\nabla_\beta D(\boldsymbol{\alpha} + \beta\boldsymbol{\alpha}) \mid_{\beta=0} < 0$). This looks like a reasonable assumption to make, but unfortunately in some of the datasets we do observe the situation that $\Delta\boldsymbol{\alpha}$ is not a descent direction. Therefore, to ensure the convergence of the whole procedure, we have to further fix this case.

To overcome this issue, we first introduce a generalized version of the *DCD* update with a step size. The new *DCD* update rule can be written as:

$$\begin{aligned}\delta^* &= \arg\min_\delta \frac{1}{2\gamma}(\delta + \frac{\gamma \boldsymbol{w}^T \boldsymbol{x}_i}{\|\boldsymbol{x}_i\|^2})^2 + \frac{1}{\|\boldsymbol{x}_i\|^2}\ell_i^*(-(\alpha_i + \delta)).\\ &= \arg\min_\delta \frac{1}{2\gamma}\delta^2 + \frac{\boldsymbol{w}^T \boldsymbol{x}_i}{\|\boldsymbol{x}_i\|^2}\delta + \frac{1}{\|\boldsymbol{x}_i\|^2}\ell_i^*(-(\alpha_i + \delta)) \quad (10)\end{aligned}$$

In (10), we add a weight $\frac{1}{\gamma}$ to the quadratic term where $\gamma \in (0, 1)$ so each coordinate descent update is more conservative. This is similar to using a smaller step size in proximal gradient method. The modified *PASSCoDe* algorithm is in Algorithm 4.

Algorithm 4 *PASSCoDe-γ($\boldsymbol{\alpha}$, $\boldsymbol{w}$)*

Require: Initial $\boldsymbol{\alpha}$ and $\boldsymbol{w} = \sum_{i=1}^n \alpha_i \boldsymbol{x}_i$

Each thread repeatedly performs the following updates:

step 1: Randomly pick i

step 2: Update $\alpha_i \leftarrow \alpha_i + \Delta\alpha_i$, where

$$\delta^* \leftarrow \arg\min_\delta \frac{1}{2\gamma}\delta^2 + \frac{\boldsymbol{w}^T \boldsymbol{x}_i}{\|\boldsymbol{x}_i\|^2}\delta + \frac{1}{\|\boldsymbol{x}_i\|^2}\ell_i^*(-(\alpha_i + \delta))$$

step 3: For each $j \in N(i)$

Update $w_j \leftarrow w_j + \delta^*(\boldsymbol{x}_i)_j$ **atomically**

Intuitively, when we use a smaller γ, the updates are more conservative, so the error caused by delayed parameter access will be milder and *PASSCoDe-γ* will be more likely to converge. To formally prove this property, we define the following notations. In *PASSCoDe-γ*, we can assign a global counter j for the updates, and $i(j)$ is the coordinate index chosen by the j-th update. The sequence $\{\boldsymbol{\alpha}^1, \boldsymbol{\alpha}^2, \cdots\}$ is the sequence of dual solutions, and we define

$$\delta^j = \alpha_{i(j)}^{j+1} - \alpha_{i(j)}^j.$$

Since δ^j is computed by *PASSCoDe-γ*, we have

$$\delta^j = \arg\min_\delta \frac{1}{2\gamma}(\delta + \frac{\gamma \boldsymbol{x}_{i(j)}^T \hat{\boldsymbol{w}}^j}{\|\boldsymbol{x}_{i(j)}\|^2})^2 + \frac{1}{\|\boldsymbol{x}_{i(j)}\|^2}\ell_i^*(-(\alpha_{i(j)}^j + \delta)),$$

where $\hat{\boldsymbol{w}}^j$ is the current $\boldsymbol{w}$ stored in the memory.

Since multiple threads conduct updates parallel asynchronously, $\hat{\boldsymbol{w}}^j \neq \boldsymbol{w}(\boldsymbol{\alpha}^j)$. However, following [7] we assume that for each iteration j, all the "writes" to $\boldsymbol{w}$ before τ iterations are done. This can be formulated as:

$$\hat{\boldsymbol{w}}^j = \boldsymbol{w}(\boldsymbol{\alpha}^{j-\tau}) + \sum_{t=j-\tau+1}^{j-1} \sum_{k \in N(i(t))} \xi_{j,t,k} \delta^t (\boldsymbol{x}_{i(t)})_k \boldsymbol{e}_k. \quad (11)$$

where each $\xi_{j,t,k}$ can be either 0 or 1, indicating whether the update has been written into $\boldsymbol{w}$.

Similar to [7], we define the following constants:

$$M_i = \max_{S \subseteq [d]} \| \sum_{t \in S} \bar{X}_{:,t} X_{i,t} \|, \quad M = \max_i M_i,$$

where $[d] := \{1, \ldots, d\}$ is the set of all the feature indices, and $\bar{X}_{:,t}$ is the t-th column of the normalized data matrix $\bar{X}$ (each row of $\bar{X}$ is $\boldsymbol{x}_i / \|\boldsymbol{x}_i\|$). Assume $R_{min} = \min_i \|\boldsymbol{x}_i\|^2 > 0$ (no $\boldsymbol{x}_i = \boldsymbol{0}$) and without loss of generality, we assume $R_{max} = \max_i \|\boldsymbol{x}_i\|_2 = 1$. We can then prove the following lemma:

Lemma 2. *Let ρ be a constant with $\rho^{-1} < 1 - 4/\sqrt{n}$, and $\theta = \sum_{t=1}^\tau \rho^{t/2}$. If γ is small enough such that*

$$\gamma \leq (\sqrt{n}(1 - \rho^{-1}) - 4)/(4(1 + \theta)M) \quad (12)$$

then the sequence $\{\boldsymbol{\alpha}^j\}$ generated by PASSCoDe-γ satisfies the following inequality:

$$E(\|\boldsymbol{\alpha}^{j-1} - \boldsymbol{\alpha}^j\|^2) \leq \rho E(\|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j+1}\|^2), \quad (13)$$

where $E(\cdot)$ is the expectation with respect to the indices selected in stochastic dual coordinate descent.

The proof is in the Appendix. Note that ρ can be any number satisfies $\rho^{-1} < 1 - 4/\sqrt{n}$, and this ensures the right hand side of (12) is a positive number. As a result, as we select γ small enough, (12) will be satisfied. In contrast, the condition in Lemma 1 of [7] may not be satisfied, which is the reason that *PASSCoDe* diverges on some datasets.

We can then derive the following main theorem:

Theorem 1. *Assume the objective function* (2) *admits a global error bound with a constant κ from the beginning (see Definition 1 in [7]) and the Lipschitz constant L_{max} is finite in the level set. If γ is small enough such that* (12) *holds and*

$$\left(1 + \frac{e\tau M\gamma}{\sqrt{n}}\right)\left(\frac{\tau^2\gamma^2 M^2 e^2}{n}\right) \leq \frac{R_{min}}{2L_{max}} \leq 1$$

where e is the natural logarithm base, then PASSCoDe-γ has a global linear convergence rate in expectation, that is,

$$E[D(\boldsymbol{\alpha}^{j+1})] - D(\boldsymbol{\alpha}^*) \leq \eta \left(E[D(\boldsymbol{\alpha}^j)] - D(\boldsymbol{\alpha}^*)\right), \quad (14)$$

where $\boldsymbol{\alpha}^*$ is the optimal solution and

$$\eta = 1 - \frac{\kappa}{L_{max}}\left(1 - \frac{2L_{max}}{R_{min}}\left(1 + \frac{e\gamma\tau M}{\sqrt{n}}\right)\left(\frac{\tau^2 M^2 e^2 \gamma^2}{n}\right)\right).$$

Note that the global error bound assumption is a weaker version of strong convexity, and it has been shown in [7] that

this assumption covers linear SVM, L2-SVM, and logistic regression. Compared with the proof in *PASSCoDe* [7], we have a γ factor in our update rule to ensure the convergence of the algorithm, while they only consider the special case $\gamma = 1$ so the convergence will not hold when τ (delay) is large. Compared with the analysis of general asynchronous coordinate descent [11], we consider the dual coordinate descent where the inconsistent read can happen when maintaining the $\boldsymbol{w}$ vector, where they do not have inconsistent read in $\boldsymbol{w}$ (see Section 4 in [7] for the detailed discussion about this). Also, the update rule is different from [11].

Based on Theorem 1, we can see $\Delta\boldsymbol{\alpha}$ will decrease the objective function value with a linear rate if γ is small enough. Therefore, in *PASSCoDe-fix*, if $\beta^* = 0$ (which means $\Delta\boldsymbol{\alpha}$ is not a descent direction), we decrease γ by half in the next iteration. As a result, our algorithm can keep decreasing γ until it is small enough to ensure the global convergence.

In summary, our *PASSCoDe-fix* algorithm is presented in Algorithm 5. Here we only present the case for case I (dual loss satisfies (6)); the algorithm for case II is similar but the closed form solution β^* is replaced by line search.

Algorithm 5 Our proposed algorithm: *PASSCoDe-fix*

Require: Initial $\boldsymbol{\alpha}$ and $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i$, $\gamma = 1$

For outer_iter $= 1, 2, \cdots$

1. $\boldsymbol{\alpha}^{\text{old}} \leftarrow \boldsymbol{\alpha}$, $\boldsymbol{w}^{\text{old}} \leftarrow \boldsymbol{w}$
2. Run *PASSCoDe-*γ (Algorithm 4) with totally N updates using all the threads to obtain $\boldsymbol{\alpha}$, $\boldsymbol{w}$
3. Compute A, B by (8).
4. If $B \geq 0$:
   $\beta^* \leftarrow 0, \gamma \leftarrow \gamma/2$
   Else:
   $\beta^* \leftarrow \max(\min(-B/(2A), 1), 0)$
5. $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}^{\text{old}} + \beta^*(\boldsymbol{\alpha} - \boldsymbol{\alpha}^{\text{old}}), \boldsymbol{w} \leftarrow \boldsymbol{w}^{\text{old}} + \beta^*(\boldsymbol{w} - \boldsymbol{w}^{\text{old}})$

V. Experimental Results

A. Implementation

Our implementation is based on the publicly available C++ code of *PASSCoDe* [2]. It was developed under the code base of LIBLINEAR with OpenMP parallelization. We implement our Algorithm 5 for both L1-SVM and L2-SVM. The major computation added in *PASSCoDe-fix* is (7) and (8), and we parallelize them as well to avoid unnecessary serial bottleneck. Our implementation is publicly available [3].

In practice, in *PASSCoDe* sometimes $\boldsymbol{\alpha}$ can go to $\pm\infty$ (exceeds the maximum floating point number) due to the divergence of asynchronous updates. In those cases, the calculation of our fixing procedure in Eq (7) will get NaN according to IEEE 754-1985. Therefore, in our code we will set $\boldsymbol{\alpha} = \boldsymbol{\alpha}^{\text{old}}$, $\boldsymbol{w} = \boldsymbol{w}^{\text{old}}$ and $\gamma = \frac{1}{2}\gamma$ when we detect β^* is NaN.

[2] http://www.cs.utexas.edu/~rofuyu/exp-codes/passcode-icml15-exp/
[3] http://huanzhang12.github.io/passcode-fix/

TABLE I: Dataset statistics

Dataset	# train samples	# test samples	# features	nnz%
a9a	32,561	16,281	123	11.3 %
covtype	464,810	116,202	54	22.12 %
news20	16,000	3996	1,355,191	0.034 %
webspam	280,000	70,000	254	33.52 %
rcv1	677,399	20,242	47,236	0.155 %
music	463,715	51,630	91	1
epsilon	400,000	100,000	2,000	1

B. Experiment Setup and Datasets

We performed our experiments on a dual-socket E5-2680 v2 machine with 20 physical cores and hyperthreading disabled. We explicitly set thread affinities so that the worker threads are bind to as few CPU socket as possible. We conduct our experiments on solving the L2-SVM (with squared hinge loss) problem, but our algorithm can also be applied to other objective functions. For simplicity, we set the regularization parameter $C = 1$. In all our experiments, we use the atomic version of *PASSCoDe* and *PASSCoDe-fix*. The wild version of *PASSCoDe* cannot converge to the same solution due to the conflicting writes to $\boldsymbol{w}$. Note that all the algorithms (including our method) do not have any additonal parameter to set (except C), so the experiments can be easily reproduced.

We run our experiments on a wide range of datasets with different characteristics: a9a, covtype, music, rcv1, news20, epsilon, webspam (unigram). Among them, music is downloaded from UCI [4] and transformed into binary class, other datasets are downloaded from LIBSVM website [5]. For music, a9a, epsilon and rcv1 we use the standard training/testing partition suggested by their source, and for other datasets we split them randomly into 80% for training and 20% for testing.

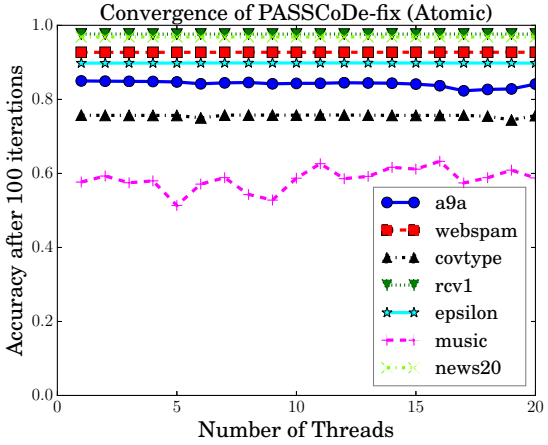


Fig. 2: Convergence of *PASSCoDe-fix*

a) Convergence of PASSCoDe-fix: Similar to Figure 1, we run *PASSCoDe-fix* with up to 20 threads and show the impact of more threads on classification accuracy after 100 iterations in Figure 2. *PASSCoDe-fix* exhibits excellent stability: it converges on all datasets, regardless of the number of threads used (a9a and music will get more stable accuracy if we run more iterations, but for a fair comparison with Figure 1

[4] http://archive.ics.uci.edu/ml/datasets/YearPredictionMSD
[5] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

we run all datasets with 100 iterations). Also, unlike Figure 1, accuracy does not jump when the number of thread changes from 10 to 11; *PASSCoDe-fix* is not affected when the second CPU socket starts being used and shows predictable behavior.

b) Efficiency of PASSCoDe-fix: In this experiment, we run *PASSCoDe-fix* and *PASSCoDe* with 20 threads, and compare their primal objective function values and test accuracy in terms of training time in Figure 3, 4, 5, 6, 7 and 8. We also add a single thread reference of *PASSCoDe* in these figures, showing the best accuracy and objective value that *PASSCoDe* can achieve. Usually, single thread *PASSCoDe* converges better in terms of number of iterations because there is no delay in updating $\boldsymbol{\alpha}$ and $\boldsymbol{w}$, but it needs more time for each iteration.

We can made the following observations from these results:

- *PASSCoDe-fix* is able to converge on all the datasets. In comparison, *PASSCoDe* diverges (out of range in figures) or oscillates badly on a9a, covtype and webspam.
- *PASSCoDe-fix* converges to the same (or very close) primal objective as the single-thread version does. This verifies our theoretical guarantee that *PASSCoDe-fix* converges to the optimal solution.
- on the dataset where *PASSCoDe* does converge well, *PASSCoDe-fix* shows minimal overhead. For epsilon and rcv1, *PASSCoDe-fix* is almost as fast as *PASSCoDe*. Since *PASSCoDe-fix* needs $O(n+d)$ time in the fixing step and in news20 $O(\text{nnz})$ is comparable to the large $d = 1,355,191$. *PASSCoDe-fix* is slightly slower than *PASSCoDe*.
- In a9a (the smallest dataset), we can see that single thread *PASSCoDe* converges faster than using 20 threads. However, our profiling results show that this slow down is not related to the fixing step we proposed. Instead, it comes from the cache coherence overhead of accessing a shared $\boldsymbol{w}$ among multiple cores during the coordinate updates. Improving *PASSCoDe*'s performance in highly parallel situations using techniques proposed in [21], [22] is our future work.

VI. Conclusions

Parallel Asynchronous Stochastic Dual Coordinate Descent (*PASSCoDe*) is the most promising algorithm to solve linear SVM and logistic regression in shared memory multi-core systems. Unfortunately, the convergence is not always guaranteed and it has been observed that the algorithm diverges on several cases In this paper, we propose a simple and efficient way to fix this convergence problem. Instead of allowing all the threads conduct updates wildly, we add periodic check points to enforce the convergence. The resulting algorithm, *PASSCoDe-fix*, is guaranteed to converge for any datasets and has minimal overhead when *PASSCoDe* also converges. The results are verified theoretically and empirically on real-world datasets. This idea of "semi-asynchronous" updates could be potentially applied to other asynchronous algorithms.

Acknowledgments. The authors are grateful to the XSEDE startup resources.

References

[1] Haim. Avron, Alex. Druinsky, and Anshul Gupta. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. In *IEEE International Parallel and Distributed Processing Symposium*, 2014.

[2] Joseph K. Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *International Conference on Machine Learning (ICML)*, pages 321–328, 2011.

[3] Corina Cortes and Vladimir Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

[4] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[5] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

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

[7] Cho-Jui. Hsieh, Hsiang-Fu. Yu, and Inderjit. S. Dhillon. PASSCoDe: Parallel ASynchronous Stochastic dual Coordinate Descent. In *International Conference on Machine Learning (ICML),*, 2015.

[8] S. S. Keerthi, S. Sundararajan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A sequential dual method for large scale multi-class linear SVMs. In *KDD*, 2008.

[9] Mu Li, Dave Andersen, Alex Smola, Junwoo Park, Amr Ahmed, Vanja Josifovski, James Long, Eugene Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[10] Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton method for large-scale logistic regression. In *ICML*, 2007.

[11] Ji. Liu and Stephen. J. Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *arXiv:1403.3862*, 2014.

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

[13] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J. Wright. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701, 2011.

[14] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 2012. Under revision.

[15] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: primal estimated sub-gradient solver for SVM. In *Proceedings of the Twenty Fourth International Conference on Machine Learning (ICML)*, 2007.

[16] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599, 2013.

[17] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: a new platform for distributed machine learning on big data. *Big Data, IEEE Transactions on*, 1(2):49–67, 2015.

[18] H.-F. Yu, C.-J. Hsieh, H. Yun, S. Vishwanathan, and I. S. Dhillon. A scalable asynchronous distributed algorithm for topic modeling. In *WWW*, 2015.

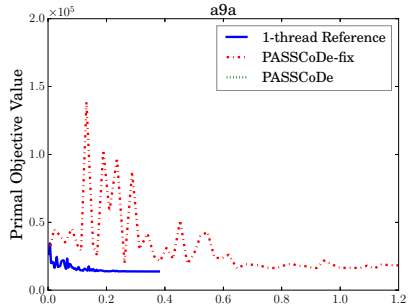
[19] Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, October 2011.

[20] Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, S.V.N. Vishwanathan, and Inderjit S. Dhillon. NOMAD: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. In *International Conference on Very Large Data Bases (VLDB)*, 2014.

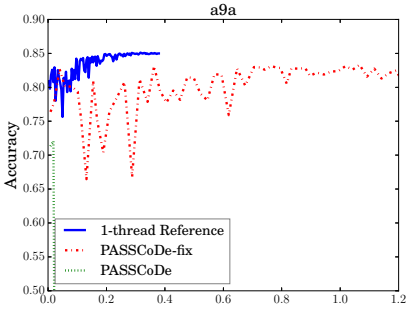
[21] Ce Zhang and Christopher Ré. Dimmwitted: A study of main-memory statistical analytics. *Proceedings of the VLDB Endowment*, 7(12):1283–1294, 2014.

[22] Huan Zhang, Cho-Jui Hsieh, and Venkatesh Akella. Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent. In *ICDM*, 2016.

[23] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

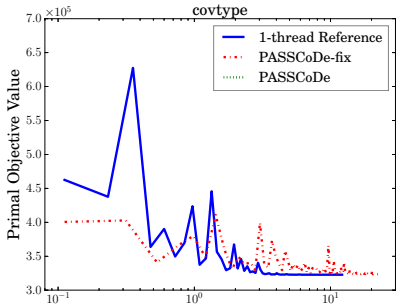


(a) Objective

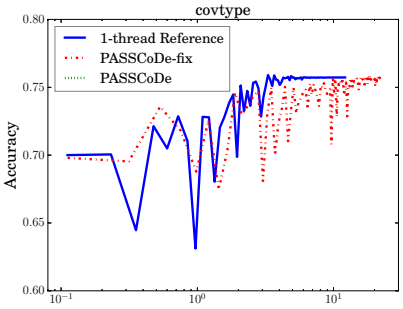


(b) Accuracy

Fig. 3: Convergence of a9a dataset

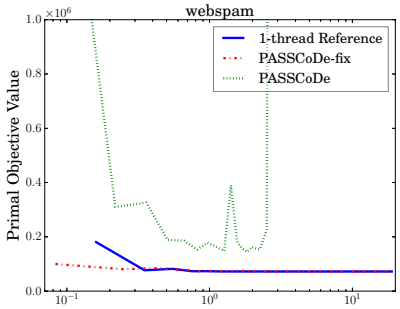


(a) Objective

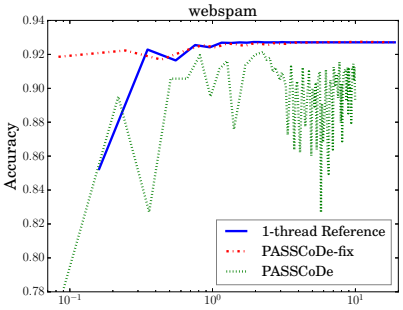


(b) Accuracy

Fig. 4: Convergence of covtype dataset

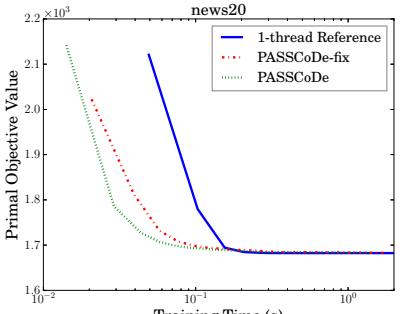


(a) Objective

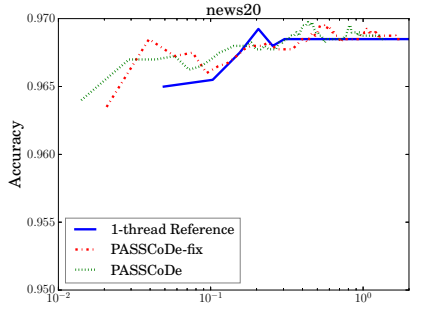


(b) Accuracy

Fig. 5: Convergence of webspam dataset

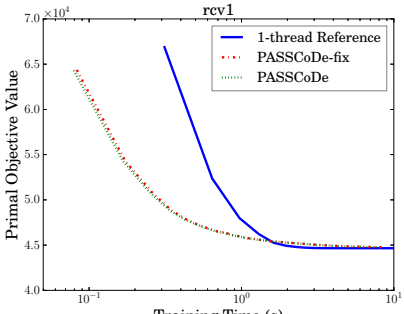


(a) Objective

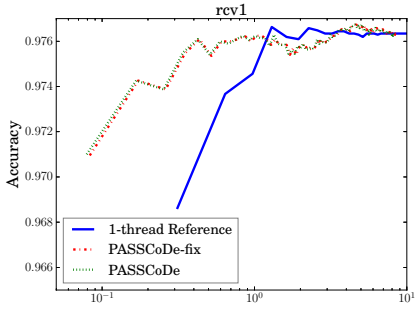


(b) Accuracy

Fig. 6: Convergence of news20 dataset

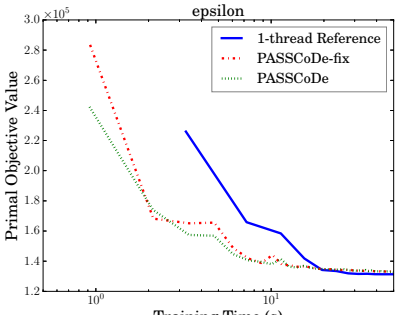


(a) Objective

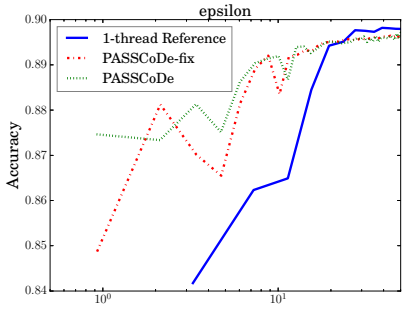


(b) Accuracy

Fig. 7: Convergence of rcv1 dataset



(a) Objective



(b) Accuracy

Fig. 8: Convergence of epsilon dataset

Note: if a line is not shown in a figure, that line is out of range due to divergence.

VII. Appendix

We prove the convergence theorems and lemmas in this section. For convenience, we rewrite the update rule of *PASSCoDe-fix* (Eq (10)) by

$$\alpha_t^{j+1} \leftarrow T_t^{\gamma}(\hat{\boldsymbol{w}}^j, \alpha_t^j),$$

where $\hat{\boldsymbol{w}}^j$ is the maintained $\boldsymbol{w}$ used in the j-th iteration, and

$$\begin{aligned} T_t^{\gamma}(\boldsymbol{w}, s) :=& \arg\min_u \ \frac{1}{2\gamma}\left[u - \left(s - \frac{\gamma \boldsymbol{w}^T \boldsymbol{x}_t}{\|\boldsymbol{x}_t\|^2}\right)\right]^2 + h_t(u), \\ =& \operatorname{prox}_t(s - \gamma \boldsymbol{w}^T \boldsymbol{x}_t / \|\boldsymbol{x}_t\|^2). \\ \operatorname{prox}_t(s) :=& \arg\min_u \ (u-s)^2/2 + h_t(u)/\gamma \\ h_t(u) :=& \ell_t^*(-u)/\|\boldsymbol{x}_i\|^2. \end{aligned}$$

Here are some notations used in the proof:

- The $\{\boldsymbol{\alpha}^j\}$ and $\{\hat{\boldsymbol{w}}^j\}$ sequence are generated from *PASSCoDe-γ* (Algorithm 5) and satisfies

$$\alpha_t^{j+1} = \begin{cases} T_t(\hat{\boldsymbol{w}}^j, \alpha_t^j) & \text{if } t = i(j), \\ \alpha_t^j & \text{if } t \neq i(j), \end{cases}$$

  where $i(j)$ is the index selected at j-th iteration.
- We define $\tilde{\boldsymbol{\alpha}}^j$ for each j by

$$\tilde{\boldsymbol{\alpha}}_t^j = T_t^{\gamma}(\hat{\boldsymbol{w}}^j, \alpha_t^j) \ \ \forall t = 1, \cdots, n.$$

  For convenience, we define $T^{\gamma}(\hat{\boldsymbol{w}}^j, \boldsymbol{\alpha}^j)$ to be a vector where each component is $T_t^{\gamma}(\hat{\boldsymbol{w}}^j, \alpha_t^j)$, and $\operatorname{prox}(\boldsymbol{s})$ is a vector where each element is $\operatorname{prox}_t(s_t)$. We have

$$\tilde{\boldsymbol{\alpha}}^j = T^{\gamma}(\hat{\boldsymbol{w}}^j, \boldsymbol{\alpha}^j) = \operatorname{prox}(\boldsymbol{\alpha}^j - \gamma \hat{\boldsymbol{w}}^T \boldsymbol{x}_i / \|\boldsymbol{x}_i\|^2).$$

- $\bar{\boldsymbol{w}} = w(\boldsymbol{\alpha}^j) = \sum_{i=1}^n \alpha_i^j \boldsymbol{x}_i$ is the true primal variables.
- The sequence $\{\boldsymbol{\beta}^j\}$ and $\{\tilde{\boldsymbol{\beta}}^j\}$ are defined by

$$\begin{aligned} \beta_t^{j+1} &= \begin{cases} T_t^{\gamma}(\bar{\boldsymbol{w}}^j, \alpha_t^j) & \text{if } t = i(j), \\ \alpha_t^j & \text{if } t \neq i(j), \end{cases} \\ \tilde{\beta}_t^{j+1} &= T_t^{\gamma}(\bar{\boldsymbol{w}}^j, \alpha_t^j) \quad \forall t = 1, \ldots, n. \end{aligned}$$

- Let $g^j(u)$ be the univariate function considered at the j-th iteration:

$$g^j(u) := \frac{1}{2\gamma}\left[u - \left(s - \frac{\gamma(\bar{\boldsymbol{w}}^j)^T \boldsymbol{x}_{i(j)}}{\|\boldsymbol{x}_{i(j)}\|^2}\right)\right]^2 + h_{i(j)}(u).$$

  Thus, $\beta_{i(j)}^{j+1} = T_{i(j)}(\bar{\boldsymbol{w}}^j, \alpha_{i(j)}^j)$ is the minimizer for $g^j(u)$.

[7, Proposition 1, 2, 3] can be directly applied here. We do not need to use [7, Proposition 4] in our proof, and [7, Proposition 5] can be proved for *PASSCoDe-γ*:

Proposition 5. *For all $j > 0$, we have*

$$D(\boldsymbol{\alpha}^j) \geq D(\boldsymbol{\beta}^{j+1}) + \frac{\|\boldsymbol{x}_{i(j)}\|^2}{2}\|\boldsymbol{\alpha}^j - \boldsymbol{\beta}^{j+1}\|^2 \qquad (15)$$

$$D(\boldsymbol{\alpha}^{j+1}) \leq D(\boldsymbol{\beta}^{j+1}) + \frac{L_{max}}{2}\|\boldsymbol{\alpha}^{j+1} - \boldsymbol{\beta}^{j+1}\|^2 \qquad (16)$$

Proof. First, from the definition of $g^j(u)$ it is $\|\boldsymbol{x}_{i(j)}\|^2$-strongly convex and L_{max} Lipschitz continuous (since $D(\cdot)$ is $L_{\max}$ Lipschitz continuous). We define $u^* = \arg\min_u g^j(u)$ to be the minimizer of the uni-variate function, and this implies $\nabla g^j(u^*) = 0$. From the strong convexity of $g^j(\cdot)$,

$$\begin{aligned} g^j(\alpha_{i(j)}^j) \geq& g^j(\beta_{i(j)}^{j+1}) + \nabla g^j(\beta_{i(j)}^{j+1})(\alpha_{i(j)}^j - \beta_{i(j)}^{j+1}) + \\ & \frac{\|\boldsymbol{x}_{i(j)}\|^2}{2}\|\alpha_{i(j)}^j - \beta_{i(j)}^{j+1}\|^2. \end{aligned}$$

By definition, $\beta_{i(j)}^{j+1}$ is between $\alpha_{i(j)}^j$ and u^*, so either $\alpha_{i(j)}^j \leq \beta_{i(j)}^{j+1} \leq u^*$ or $u^* \leq \beta_{i(j)}^{j+1} \leq \alpha_{i(j)}^j$. Since the function is strongly convex, the gradient $\nabla g^j(\cdot)$ is increasing, so for the first case $\nabla g^j(\beta_{i(j)}^{j+1}) \leq 0$ and $\alpha_{i(j)}^j - \beta_{i(j)}^{j+1} \leq 0$. For the second case both terms ≥ 0. These imply

$$g^j(\alpha_{i(j)}^j) \geq g^j(\beta_{i(j)}^{j+1}) + \frac{\|\boldsymbol{x}_{i(j)}\|^2}{2}\|\alpha_{i(j)}^j - \beta_{i(j)}^{j+1}\|^2. \qquad (17)$$

From the Lipschitz continuity, we also have

$$g^j(\alpha_{i(j)}^{j+1}) \leq g^j(\beta_{i(j)}^{j+1}) + \frac{L_{max}}{2}\|\alpha_{i(j)}^{j+1} - \beta_{i(j)}^{j+1}\|^2 \qquad (18)$$

By the definitions of g^j, $\boldsymbol{\alpha}^j$, $\boldsymbol{\alpha}^{j+1}$, and $\boldsymbol{\beta}^{j+1}$, (17) and (18) imply (15) and (16). □

A. Proof of Lemma 2

Similar to [11], we prove Eq. (13) by induction. First, for all j , we have

$$\begin{aligned} &\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|^2 - \|\boldsymbol{\alpha}^j - \tilde{\boldsymbol{\alpha}}^{j+1}\|^2 \\ &\leq 2\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|\|\boldsymbol{\alpha}^j - \tilde{\boldsymbol{\alpha}}^{j+1} - \boldsymbol{\alpha}^{j-1} + \tilde{\boldsymbol{\alpha}}^j\|. \end{aligned} \qquad (19)$$

See [11] for a proof for the above inequality. We can further bound this term by

$$\begin{aligned} &\|\boldsymbol{\alpha}^j - \tilde{\boldsymbol{\alpha}}^{j+1} - \boldsymbol{\alpha}^{j-1} + \tilde{\boldsymbol{\alpha}}^j\| \\ \leq& \|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j-1}\| + \|\operatorname{prox}(\boldsymbol{\alpha}^j - \gamma\bar{X}\hat{\boldsymbol{w}}^j) - \operatorname{prox}(\boldsymbol{\alpha}^{j-1} - \gamma\bar{X}\hat{\boldsymbol{w}}^{j-1})\| \\ \leq& \|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j-1}\| + \|(\boldsymbol{\alpha}^j - \gamma\bar{X}\hat{\boldsymbol{w}}^j) - (\boldsymbol{\alpha}^{j-1} - \gamma\bar{X}\hat{\boldsymbol{w}}^{j-1})\| \\ \leq& \|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j-1}\| + \|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j-1}\| + \gamma\|\bar{X}\hat{\boldsymbol{w}}^j - \bar{X}\hat{\boldsymbol{w}}^{j-1}\| \\ \leq& 2\|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j-1}\| + \gamma\|\bar{X}\bar{\boldsymbol{w}}^j - \bar{X}\bar{\boldsymbol{w}}^{j-1}\| \\ &+ \gamma\|\bar{X}\hat{\boldsymbol{w}}^j - \bar{X}\bar{\boldsymbol{w}}^j\| + \gamma\|\bar{X}\bar{\boldsymbol{w}}^{j-1} - \bar{X}\hat{\boldsymbol{w}}^{j-1}\| \\ \leq& (2+\gamma M)\|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j-1}\| + \sum_{t=j-\tau}^{j-1}\|\delta^t\|\gamma M + \sum_{t=j-\tau-1}^{j-2}\|\delta^t\|\gamma M \\ =& (2+2\gamma M)\|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j-1}\| + 2\gamma M\sum_{t=j-\tau-1}^{j-2}\|\delta_t\| \end{aligned} \qquad (20)$$

Now we prove (13) by induction.

Induction Hypothesis. Based on [7, Proposition 1], it suffices to prove the following inequality: For all j,

$$E(\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|^2) \leq \rho E(\|\boldsymbol{\alpha}^j - \tilde{\boldsymbol{\alpha}}^{j+1}\|^2), \qquad (21)$$

Induction Basis. When $j = 1$,

$$\|\boldsymbol{\alpha}^1 - \tilde{\boldsymbol{\alpha}}^2 + \boldsymbol{\alpha}^0 - \tilde{\boldsymbol{\alpha}}^1\| \leq (2 + 2\gamma M)\|\boldsymbol{\alpha}^1 - \boldsymbol{\alpha}^0\|.$$

Taking the expectation of (19), we have

$$\begin{aligned} &E[\|\boldsymbol{\alpha}^0 - \tilde{\boldsymbol{\alpha}}^1\|^2] - E[\|\boldsymbol{\alpha}^1 - \tilde{\boldsymbol{\alpha}}^2\|^2] \\ &\leq (4 + 4\gamma M)E(\|\boldsymbol{\alpha}^0 - \tilde{\boldsymbol{\alpha}}^1\|\|\boldsymbol{\alpha}^0 - \boldsymbol{\alpha}^1\|). \end{aligned}$$

From [7, Proposition 1] we have $E[\|\boldsymbol{\alpha}^0 - \boldsymbol{\alpha}^1\|^2] = \frac{1}{n}\|\boldsymbol{\alpha}^0 - \tilde{\boldsymbol{\alpha}}^1\|^2$. Also, using the inequality of arithmetic and geometric means, for any $\mu_1, \mu_2, c > 0$ we have $\mu_1\mu_2 \leq \frac{1}{2}(c\mu_1^2 + c^{-1}\mu_2^2)$. Therefore, we have

$$\begin{aligned}
&E[\|\boldsymbol{\alpha}^0 - \tilde{\boldsymbol{\alpha}}^1\|\|\boldsymbol{\alpha}^0 - \boldsymbol{\alpha}^1\|] \\
&\leq \frac{1}{2}E\Big[n^{1/2}\|\boldsymbol{\alpha}^0 - \boldsymbol{\alpha}^1\|^2 + n^{-1/2}\|\tilde{\boldsymbol{\alpha}}^1 - \boldsymbol{\alpha}^0\|^2\Big] \\
&= \frac{1}{2}E\Big[n^{-1/2}\|\boldsymbol{\alpha}^0 - \tilde{\boldsymbol{\alpha}}^1\|^2 + n^{-1/2}\|\tilde{\boldsymbol{\alpha}}^1 - \boldsymbol{\alpha}^0\|^2\Big] \\
&= n^{-1/2}E[\|\boldsymbol{\alpha}^0 - \tilde{\boldsymbol{\alpha}}^1\|^2].
\end{aligned}$$

Therefore,

$$E[\|\boldsymbol{\alpha}^0 - \tilde{\boldsymbol{\alpha}}^1\|^2] - E[\|\boldsymbol{\alpha}^1 - \tilde{\boldsymbol{\alpha}}^2\|^2] \leq \frac{4 + 4\gamma M}{\sqrt{n}}E[\|\boldsymbol{\alpha}^0 - \tilde{\boldsymbol{\alpha}}^1\|^2],$$

which implies

$$E[\|\boldsymbol{\alpha}^0 - \boldsymbol{\alpha}^1\|^2] \leq \frac{1}{1 - \frac{4+4\gamma M}{\sqrt{n}}}E[\|\boldsymbol{\alpha}^1 - \tilde{\boldsymbol{\alpha}}^2\|^2]. \tag{22}$$

From (12) we have

$$\rho^{-1} \leq 1 - \frac{4 + 4\gamma M(1+\theta)}{\sqrt{n}} \leq 1 - \frac{4 + 4\gamma M}{\sqrt{n}}. \tag{23}$$

Combining with (22) we have

$$E[\|\boldsymbol{\alpha}^0 - \boldsymbol{\alpha}^1\|^2] \leq \rho E[\|\boldsymbol{\alpha}^1 - \tilde{\boldsymbol{\alpha}}^2\|^2].$$

Induction Step. By the induction hypothesis, we assume

$$E[\|\boldsymbol{\alpha}^{t-1} - \tilde{\boldsymbol{\alpha}}^t\|^2] \leq \rho E[\|\boldsymbol{\alpha}^t - \tilde{\boldsymbol{\alpha}}^{t+1}\|^2] \quad \forall t \leq j-1. \tag{24}$$

The goal is to show

$$E[\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|^2] \leq \rho E[\|\boldsymbol{\alpha}^j - \tilde{\boldsymbol{\alpha}}^{j+1}\|^2].$$

Using the derivations in the inductive step of [7], for all $t < j$ we have

$$E\Big[\|\boldsymbol{\alpha}^t - \boldsymbol{\alpha}^{t+1}\|\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|\Big] \leq \frac{\rho^{(j-1-t)/2}}{\sqrt{n}}E\Big[\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|^2\Big] \tag{25}$$

Furthermore,

$$\begin{aligned}
&E[\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|^2] - E[\|\boldsymbol{\alpha}^j - \tilde{\boldsymbol{\alpha}}^{j+1}\|^2] \\
&\leq (4 + 4\gamma M)E(\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|\|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j-1}\|) \\
&\quad + 4\gamma M \sum_{t=j-\tau-1}^{j-1} E\Big[\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|\|\boldsymbol{\alpha}^t - \boldsymbol{\alpha}^{t-1}\|\Big] \quad \text{by (19), (20)} \\
&\leq (4 + 4\gamma M)n^{-1/2}E[\|\tilde{\boldsymbol{\alpha}}^j - \boldsymbol{\alpha}^{j-1}\|^2] \\
&\quad + 4\gamma M n^{-1/2}E[\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|^2] \sum_{t=j-1-\tau}^{j-2} \rho^{(j-1-t)/2} \quad \text{by (24)} \\
&\leq \frac{4 + 4\gamma M + 4\gamma M\theta}{\sqrt{n}}E[\|\boldsymbol{\alpha}^{j-1} - \tilde{\boldsymbol{\alpha}}^j\|^2].
\end{aligned}$$

Combining with (23) we have proved the induction step.

B. Proof of Theorem 1

We bound the expected distance by

$$\begin{aligned}
&E\left[\|\tilde{\boldsymbol{\beta}}^{j+1} - \tilde{\boldsymbol{\alpha}}^{j+1}\|^2\right] = E\left[\sum_{t=1}^n \left(T_t^\gamma(\bar{\boldsymbol{w}}^j, \alpha_t^j) - T_t^\gamma(\hat{\boldsymbol{w}}^j, \alpha_t^j)\right)^2\right] \\
&\leq \gamma^2 E\left[\sum_{t=1}^n \left(\left(\bar{\boldsymbol{w}}^j - \hat{\boldsymbol{w}}^j\right)^T \boldsymbol{x}_t/\|\boldsymbol{x}_t\|^2\right)^2\right] \quad \text{By [7, Proposition 3]} \\
&= \gamma^2 E\left[\|\bar{X}(\bar{\boldsymbol{w}}^j - \hat{\boldsymbol{w}}^j)\|^2\right] \\
&\leq \gamma^2 M^2 E\left[\left(\sum_{t=j-\tau}^{j-1} \|\boldsymbol{\alpha}^{t+1} - \boldsymbol{\alpha}^t\|\right)^2\right] \quad \text{By [7, Proposition 2]} \\
&\leq \gamma^2 \tau M^2 E\left[\left(\sum_{t=1}^{\tau} \rho^t \|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j+1}\|^2\right)\right] \quad \text{By Lemma 2} \\
&\leq \frac{\gamma^2\tau^2M^2}{n}\rho^\tau E\left[\|\tilde{\boldsymbol{\alpha}}^{j+1} - \boldsymbol{\alpha}^j\|^2\right] \\
&\leq \frac{\gamma^2\tau^2M^2e^2}{n}E\left[\|\tilde{\boldsymbol{\alpha}}^{j+1} - \boldsymbol{\alpha}^j\|^2\right], \qquad (26)
\end{aligned}$$

where the last inequality uses the fact that $\rho^{(\tau+1)/2} \leq e$, which further implies $\rho^\tau \leq e^2$ because $\rho \geq 1$. Applying Cauchy-Schwarz Inequality, we obtain

$$\begin{aligned}
&E\left[\|\tilde{\boldsymbol{\beta}}^{j+1} - \boldsymbol{\alpha}^j\|^2\right] = E\left[\|\tilde{\boldsymbol{\beta}}^{j+1} - \tilde{\boldsymbol{\alpha}}^{j+1} + \tilde{\boldsymbol{\alpha}}^{j+1} - \boldsymbol{\alpha}^j\|^2\right] \\
&\leq 2\left(1 + \frac{\gamma^2e^2\tau^2M^2}{n}\right)E\left[\|\tilde{\boldsymbol{\alpha}}^{j+1} - \boldsymbol{\alpha}^j\|^2\right]. \qquad (27)
\end{aligned}$$

Next, we bound the decrease of objective function value by

$$\begin{aligned}
&D(\boldsymbol{\alpha}^j) - D(\boldsymbol{\alpha}^{j+1}) \\
&= \left(D(\boldsymbol{\alpha}^j) - D(\boldsymbol{\beta}^{j+1})\right) + \left(D(\boldsymbol{\beta}^{j+1}) - D(\boldsymbol{\alpha}^{j+1})\right) \\
&\geq \left(\frac{R_{min}}{2}\|\boldsymbol{\alpha}^j - \boldsymbol{\beta}^{j+1}\|^2\right) - \left(\frac{L_{max}}{2}\|\boldsymbol{\beta}^{j+1} - \boldsymbol{\alpha}^{j+1}\|^2\right)
\end{aligned}$$

Where the inequality is from Proposition 5. So

$$\begin{aligned}
&E[D(\boldsymbol{\alpha}^j)] - E[D(\boldsymbol{\alpha}^{j+1})] \\
&\geq \frac{R_{min}}{2n}E[\|\tilde{\boldsymbol{\beta}}^{j+1} - \boldsymbol{\alpha}^j\|^2] - \frac{L_{max}}{2n}E[\|\tilde{\boldsymbol{\beta}}^{j+1} - \tilde{\boldsymbol{\alpha}}^{j+1}\|^2] \\
&\geq \frac{R_{min}}{2n}E[\|\tilde{\boldsymbol{\beta}}^{j+1} - \boldsymbol{\alpha}^j\|^2] - \frac{L_{max}\tau^2\gamma^2M^2e^2}{2n^2}E[\|\tilde{\boldsymbol{\alpha}}^{j+1} - \boldsymbol{\alpha}^j\|^2] \text{ by (26)} \\
&\geq \frac{R_{min}}{2n}\left(1 - \frac{2L_{max}}{R_{min}}\left(1 + \frac{e\gamma\tau M}{\sqrt{n}}\right)\left(\frac{\tau^2\gamma^2M^2e^2}{n}\right)\right) \times \\
&\quad E[\|\tilde{\boldsymbol{\beta}}^{j+1} - \boldsymbol{\alpha}^j\|^2] \quad \text{by (27)}
\end{aligned}$$

Let $b = \left(1 - \frac{2L_{max}}{R_{min}}\left(1 + \frac{e\tau\gamma M}{\sqrt{n}}\right)\left(\frac{\tau^2\gamma^2M^2e^2}{n}\right)\right)$ and combining the above inequality with the definition of Global Error Bound (Definition 1 in [7]) we have

$$E[D(\boldsymbol{\alpha}^j)] - E[D(\boldsymbol{\alpha}^{j+1})] \geq \frac{b\kappa}{L_{max}}E[D(\boldsymbol{\alpha}^j) - D^*].$$

Therefore, we have

$$\begin{aligned}
&E[D(\boldsymbol{\alpha}^{j+1})] - D^* \\
&= E[D(\boldsymbol{\alpha}^j)] - \left(E[D(\boldsymbol{\alpha}^j)] - E[D(\boldsymbol{\alpha}^{j+1})]\right) - D^* \\
&\leq \eta\left(E[D(\boldsymbol{\alpha}^j)] - D^*\right),
\end{aligned}$$

where $\eta = 1 - \frac{b\kappa}{L_{max}}$.