Scaling Up Stochastic Dual Coordinate Ascent

Kenneth Tran
Microsoft
one@kentran.net

Saghar Hosseini
University of Washington
saghar@uw.edu

Lin Xiao
Microsoft
lin.xiao@microsoft.com

Thomas Finley
Microsoft
tfinley@microsoft.com

Mikhail Bilenko
Microsoft
mbilenko@microsoft.com

ABSTRACT

Stochastic Dual Coordinate Ascent (SDCA) has recently emerged as a state-of-the-art method for solving large-scale supervised learning problems formulated as minimization of convex loss functions. It performs iterative, random-coordinate updates to maximize the dual objective. Due to the sequential nature of the iterations, it is typically implemented as a single-threaded algorithm limited to in-memory datasets. In this paper, we introduce an asynchronous parallel version of the algorithm, analyze its convergence properties, and propose a solution for primal-dual synchronization required to achieve convergence in practice. In addition, we describe a method for scaling the algorithm to out-of-memory datasets via multi-threaded deserialization of block-compressed data. This approach yields sufficient pseudo-randomness to provide the same convergence rate as random-order in-memory access. Empirical evaluation demonstrates the efficiency of the proposed methods and their ability to fully utilize computational resources and scale to out-of-memory datasets.

1. INTRODUCTION

Efficient linear learning techniques are essential for training accurate prediction models in big-data business-critical applications. Examples of such applications include text classification, click probability estimation in online advertising, and malware detection. In these domains, dimensionality of representation induced by key predictive features is very high: for word n-grams, user IPs, advertisement IDs, and file signatures, many millions and billions of possible values exist.

Despite high overall dimensionality, examples in such domains are typically very sparse with few non-zero features encoded directly or via feature hashing. This results in computationally cheap prediction, making linear models a popular choice for high-throughput applications. For additional accuracy gains, linear models can be extended via polynomial expansions explicitly or implicitly [2].

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.

DOI: http://dx.doi.org/10.1145/2783258.2783412.

While training linear models has been a well-studied area of machine learning and optimization for decades, in recent years, a number of advances in stochastic gradient methods have considerably advanced the state-of-the-art. In particular, Stochastic Dual Coordinate Ascent (SDCA) has emerged as a highly competitive algorithm due to its combination of excellent performance on benchmarks, lack of learning rate to tune, and strong convergence guarantees [19, 9, 22].

SDCA is a primal-dual optimization algorithm that requires sequential random-order access to training examples. The per-example iterative nature of the coordinate updates effectively results in SDCA being a single-threaded in-memory algorithm, which limits its scalability to very large training sets, and underutilizes modern hardware with commonly available multiple CPU cores.

This paper addresses the problem of scaling SDCA to modern multi-core hardware and large out-of-memory datasets. First, we introduce a basic asynchronous parallelization scheme for SDCA, A-SDCA, and prove that it retains the fast linear convergence guarantees of single-threaded SDCA to a certain suboptimality level. We observe that a naive implementation of the algorithm routinely fails to achieve asymptotic convergence to optimal values due to lack of synchronization between primal and dual updates, and propose a modified version of the algorithm, Semi-asynchronous SDCA (SA-SDCA), which periodically enforces primal-dual synchronization in a separate thread, which empirically results in convergence.

Our second contribution is a method for scaling SDCA to out-of-memory datasets. Our approach departs from previous algorithms for out-of-memory learning that rely either on repeated same-order streaming through examples, or repeated iterations through individual blocks, neither of which is suitable for SDCA. Instead, we propose a block-compressed binary deserialization scheme that includes indexing for random block access, while supporting random-order within-block iteration. By offloading decompression and disk I/O to separate threads, the proposed method provides efficient access to data in pseudo-random order, which empirically is shown to provide similar convergence behavior as truly random-order access.

We empirically demonstrate that proposed techniques result in significant speedups and full hardware utilization, as well as the ability to train on out-of-memory datasets effectively. Extensive comparisons on multi-gigabyte datasets demonstrate strong gains over existing state-of-the-art implementations, setting a new high bar for large-scale supervised learning.

2. PRELIMINARIES

We consider regularized empirical loss minimization of linear predictors. Let $x_1, \ldots, x_n \in \mathbb{R}^d$ be the feature vectors of n training examples, and $w \in \mathbb{R}^d$ be a weight vector which generates linear predictions $w^T x_i$ for $i = 1, \ldots, n$. In addition, let $\phi_i : \mathbb{R} \to \mathbb{R}$ be a convex loss function associated with linear prediction $w^T x_i$. Our goal is to minimize the following regularized empirical loss

$$P(w) = \frac{1}{n}\sum_{i=1}^{n} \phi_i(w^T x_i) + \frac{\lambda}{2}||w||_2^2, \tag{1}$$

where $\lambda > 0$ is a regularization parameter. This formulation is used in many well-known classification and regression problems. For binary classification, each feature vector x_i is associated with a label $y_i \in \{\pm 1\}$. We obtain linear SVM (without the bias term) by using the hinge loss $\phi_i(a) = \max\{0, 1 - y_i a\}$, and regularized logistic regression is obtained by setting $\phi_i(a) = \log(1 + \exp(-y_i a))$. For linear regression problems, each x_i is associated with a response $y_i \in \mathbb{R}$, and we use $\phi_i(a) = (a - y_i)^2$.

Our methods described in this paper can be readily extended to work with a more general formulation, that is, we can replace $(\lambda/2)\|w\|_2^2$ with a general convex regularizer $g(w)$. For example, $g(w) = \lambda\|w\|_1$ or $g(w) = \lambda_1\|w\|_1 + (\lambda_2/2)\|w\|_2^2$. Details for handling such more general regularizations can be found in [17]. Here we focus on the ℓ_2 regularization for clarity and simplicity.

The Stochastic Dual Coordinate Ascent (SDCA) solves a dual problem of (1). More specifically, let $\phi_i^* : \mathbb{R} \to \mathbb{R}$ be the convex conjugate of ϕ_i, i.e., $\phi_i^*(u) = \max_a(u \cdot a - \phi_i(a))$. The dual problem is to maximize the dual objective

$$D(\alpha) = \frac{1}{n}\sum_{i=1}^{n} -\phi_i^*(-\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i\right\|_2^2. \tag{2}$$

Notice that $\alpha \in \mathbb{R}^n$ and each dual variable α_i is associated with a different example in the training set. At each iteration of SDCA, a dual coordinate is chosen uniformly at random and $D(\alpha)$ is maximized with respect to that coordinate while the rest of the dual variables are not modified.

Let $w^\star = \arg\min P(w)$ and $\alpha^\star = \arg\min D(\alpha)$ be the primal and dual optimal solutions respectively. If we define

$$w(\alpha) = \frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i, \tag{3}$$

then $w^\star = w(\alpha^\star)$ and $P(w^\star) = D(\alpha^\star)$. We say the solution $w \in \mathbb{R}^d$ is ε_P-sub-optimal if the primal sub-optimality $P(w) - P(w^*)$ is less than ε_P.

2.1 Smoothness Assumptions

In this section, we introduce some typical smoothness assumptions on the convex losses ϕ_i, and explain its implications for the dual function defined in (2). Throughout this paper, we use $\|\cdot\|$ to denote the Euclidean norm $\|\cdot\|_2$.

Definition 1. A function ϕ_i is called L-Lipschitz continuous if there exists a positive constant L such that for all $a, b \in \mathbb{R}$,

$$|\phi_i(a) - \phi_i(b)| \le L\|a - b\|. \tag{4}$$

Definition 2. A function ϕ_i is $(1/\gamma)$-smooth if it is differentiable and its derivative is $(1/\gamma)$-Lipschitz continuous, i.e., for all $a, b \in \mathbb{R}$, we have

$$|\nabla\phi_i(a) - \nabla\phi_i(b)| \le \frac{1}{\gamma}\|a - b\|. \tag{5}$$

For convex functions, this is equivalent to

$$\phi_i(a) \le \phi_i(b) + (a - b)^T\nabla\phi_i(b) + \frac{1}{2\gamma}||a - b||^2. \tag{6}$$

If ϕ_i is $(1/\gamma)$-smooth, then its convex conjugate ϕ_i^* is γ-strongly convex (see, e.g., [8]). That is, for all $u, v \in \mathbb{R}$ and $s \in [0, 1]$, we have

$$\begin{aligned} -\phi_i^*(su + (1-s)v) \quad \ge \quad & -s\phi_i^*(u) - (1-s)\phi_i^*(v) \\ & + \frac{\gamma_i s(1-s)}{2}||u - v||^2. \end{aligned} \tag{7}$$

If ϕ_i is $(1/\gamma)$-smooth, then we define the condition number

$$\kappa := \frac{1}{\lambda\gamma}, \tag{8}$$

which is a key quantity in characterizing the complexity of optimization algorithms. For example, it is shown in [19] that the number of iterations of the SDCA algorithm to find a $w \in \mathbb{R}^d$ such that $P(w) - P(w^\star) \le \epsilon$ (with high probability) is $O((n + \kappa)\log(1/\epsilon))$.

2.2 Characterizing Sparse Datasets

Many popular machine learning problems are represented by very sparse datasets. Specifically, in datasets where the number of examples n and dimensionality d in problem (1) can be very large, the number of non-zero elements in the feature vectors $x_i \in \mathbb{R}^d$ can be relatively small. Here we give a formal characterization of sparse datasets, following the model used in Hogwild! [13].

We can construct a *hypergraph* $G = (V, E)$ representing the sparsity patterns in the dataset. The hypergraph's vertex set is $V = \{1, \ldots, d\}$, with each vertex $v \in V$ denoting an individual coordinate in the space of feature vectors ($\mathbb{R}^d$). Each hyper edge $e \in E$ represents a training example, which covers a subset of vertices indicating its nonzero coordinates.Since $|E| = n$, we can also label the hyper edges by $i = 1, \ldots, n$.

Several statistics can be defined for the hyper graphs G. The first one characterizes the maximum size of the hyper edges, or number of non-zero elements in an example:

$$\Omega := \max_{e \in E}|e|. \tag{9}$$

The next one bounds the maximum fraction of edges that covers any given vertex:

$$\tilde{\Delta} := \frac{\max_{1 \le v \le n}|\{e \in E : v \in e\}|}{|E|}, \tag{10}$$

which translate into the maximum frequency of appearance of any feature in the training examples. We also define

$$\rho := \frac{\max_{e \in E}|\{\hat{e} \in E : \hat{e} \cap e \neq \emptyset\}|}{|E|}, \tag{11}$$

which is the maximum fraction of edges that intersect any given edge, and can serve as a measure of sparsity of the hypergraph.

Algorithm 1: A-SDCA (on each processor)

1 **repeat**
2 | sample $i \in \{1, \ldots, n\}$ uniformly at random and let e denote the corresponding hyper edge
3 | read current state w and α_i
4 | $\Delta\alpha_i = \arg\max_{\Delta\alpha_i} \{-\phi_i^*(-\alpha_i - \Delta\alpha_i) - \frac{\lambda n}{2}||w + \frac{1}{\lambda n}\Delta\alpha_i x_i||^2\}$
5 | $\alpha_i \leftarrow \alpha_i + \Delta\alpha_i$
6 | **for** $v \in e$ **do**
7 | | $w_v \leftarrow w_v + \frac{1}{\lambda n}\Delta\alpha_i x_{i,v}$
8 | **end**
9 **until** *stop*

3. PARALLELIZING SDCA

In this section, we first describe an asynchronous parallel SDCA algorithm (A-SDCA) based on a shared memory model with multiple processors (threads). Convergence analysis of A-SDCA shows that it exhibits fast linear convergence before reaching a dataset-dependent suboptimality level, beyond which it may not converge asymptotically to the optimal parameter values. A study of the algorithm's empirical performance revealed that asynchronous updates of primal and dual are problematic, leading us to derive a semi-asynchronous SDCA technique (SA-SDCA), in which periodic synchronization of the primal and dual variables allow satisfying equation (3), and empirically result in convergence to optimal values demonstrated in Section 5.

Suppose we have a shared-memory computer with m processors (threads). Each processor has read and write access to a state vector $w \in \mathbb{R}^d$ stored in shared memory. In the A-SDCA algorithm, each processor follows the procedure shown in Algorithm 1. In line 7, w_v and $x_{i,v}$ denote the components of weight vector w and training example x_i, respectively, indexed by $v \in \{1, \ldots, d\}$.

When there is only one processor (thread), Algorithm 1 is equivalent to the sequential SDCA algorithm in [19]. For multiple processors, each operation in Algorithm 1 can be considered a local event that occurs asynchronously across processors. In particular, the dual update computation in line 4 of Algorithm 1 takes the bulk of computation time during each loop execution, and may take each processor a different amount of time to complete. As a result, when a particular processor updates components of w in the shared memory (lines 6-8), the component w_v on the right-hand side of line 7 may be different from the one read in line 3 (which was used to compute the update $\Delta\alpha_i$). Despite this asynchronicity, we assume the component-wise addition in line 7 is *atomic*.

In order to analyze the performance of Algorithm 1, we define a global iteration counter $t = 0, 1, 2, \ldots$. We increase t by 1 whenever some component of w in the shared memory is updated by a processor. Thus, line 7 of Algorithm 1 can be written as:

$$w_v^{(t+1)} = \begin{cases} w_v^{(t)} + \frac{1}{\lambda n}\Delta\alpha_i^{k(t)} x_{i,v} & \text{if } v \in e \\ w_v^{(t)} & \text{otherwise} \end{cases} \quad (12)$$

where $k(t)$ denotes the time at which line 3 of Algorithm 1 was executed (with $k(t) \le t$). The formula (12) assumes the operations in lines 6-8 of Algorithm 1 are indivisible (or simultaneous), when the global event counter t is incremented. If this cannot be guaranteed in the implementation, we can still analyze a modified version where the for loop in lines 6-8 is replaced by updating a single $v \in e$, picked randomly from the set e (of nonzero feature coordinates).

In terms of the global counter t, computation of dual update $\Delta\alpha_i^{k(t)}$ in line 4 can be written as:

$$\Delta\alpha_i^{k(t)} = \arg\max_{\Delta\alpha_i} \Big\{-\phi_i^*(-\alpha_i^{k(t)} - \Delta\alpha_i) - \frac{\lambda n}{2}||w^{k(t)} + \frac{1}{\lambda n}\Delta\alpha_i x_i||^2\Big\}.$$

We assume that dual variables remain fixed:

$$\alpha_i^{(t)} = \alpha_i^{k(t)}, \quad \text{for all } i \in \{1, \ldots, n\} \text{ and all } t \ge 0.$$

allowing the dual update in line 5 to be consistent:

$$\alpha_i^{(t+1)} \leftarrow \alpha_i^{(t)} + \Delta\alpha_i^{k(t)} = \alpha_i^{k(t)} + \Delta\alpha_i^{k(t)}.$$

This requires that no more than one processor can work on the same example i. It can be easily guaranteed by partitioning the datasets into m subsets $S_1, \ldots, S_m \subset \{1, \ldots, n\}$, and each processor p only works on random samples from the local subset S_p, for $p = 1, \ldots, m$.

We assume that the lag between the read and write operations at each processor is bounded, i.e., there is a constant τ such that

$$t - k(t) \le \tau, \quad \text{for all } t \ge 0. \quad (13)$$

Another assumption we make is that the updates $\Delta\alpha_i$ are always bounded, i.e., there is a constant $M > 0$ such that

$$|\Delta\alpha_i^{(t)}| \le M, \quad \text{for all } i \in \{1, \ldots, n\} \text{ and all } t \ge 0. \quad (14)$$

Based on the above assumptions, the following theorem describes the behavior of the A-SDCA algorithm (see proof in Appendix A):

Theorem 3. *Suppose each loss function ϕ_i is convex and $(1/\gamma)$-smooth, and we initialize the shared state by $\alpha^{(0)} = 0$ and $w^{(0)} = w(\alpha^{(0)})$. Let the sequence of $w^{(t)}$ and $\alpha^{(t)}$ be generated by Algorithm 1, indexed by the global iteration counter t. If*

$$\varepsilon_P > K(n + \kappa)(2 + n + \kappa),$$

where $\kappa = 1/(\lambda\gamma)$ and

$$K = \frac{\Omega M^2 \rho\tau}{n}\left(2\Omega\rho\tau + \frac{1 + 3\lambda n\gamma}{\lambda n(1 + \lambda n\gamma)}\right),$$

then we have $\mathbb{E}[P(w^{(T)}) - D(\alpha^{(T)})] \le \varepsilon_P$ whenever

$$T \ge (n + \kappa)\log\left(\frac{(n + \kappa)(1 - K(n + \kappa))}{\varepsilon_P - K(n + \kappa)(2 + n + \kappa)}\right).$$

Here τ and M are the constants in equations (13) *and* (14) *respectively, and Ω and ρ are the statistics of hypergraph as defined in* (9) *and* (11) *respectively.*

The theorem proves that the A-SDCA algorithm enjoys a fast linear convergence up to suboptimality level $K(n + \kappa)(2 + n + \kappa)$. This suboptimality level depends on the sparsity parameters Ω and ρ of the dataset, as well as the lag τ, which usually grows with the number of processors m.

With a single processor, when Algorithm 1 reduces to the sequential SDCA method, there is no the lag between the iteration counter: $\tau = 0$. Consequently, $K = 0$ and theorem yields the rate previously proven for sequential SDCA in [19].

In the multiple processor case, consider the typical setting with $\lambda \sim 1/\sqrt{n}$. Since the smoothness parameter γ can be

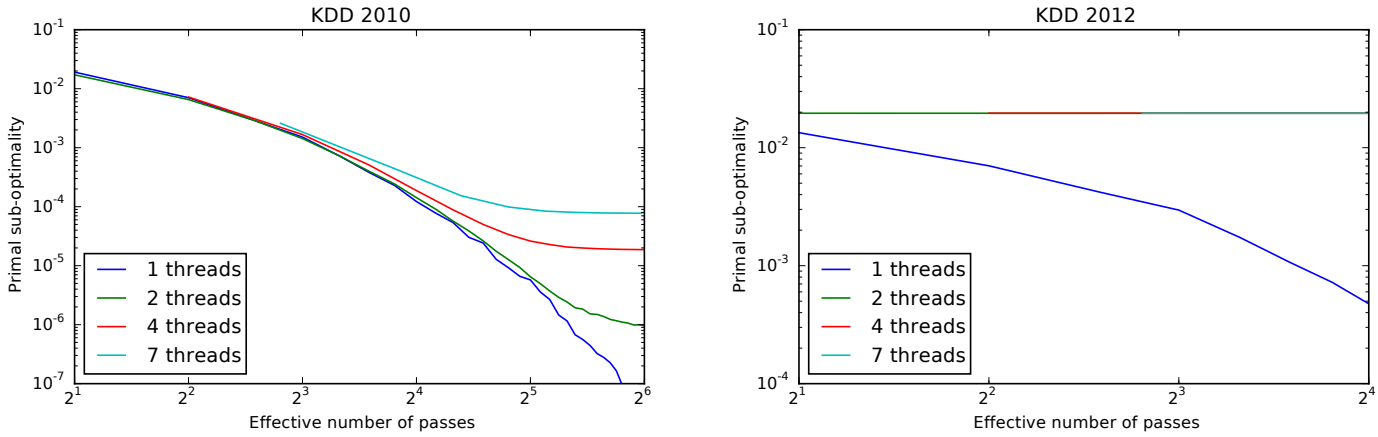

Figure 1: Performance of A-SDCA on two different datasets. Left: KDD 2010 data. Right: KDD 2012 data. The datasets are summarized in Table 1. Log-log scale is used to illustrate the super-polynomial convergence of sequential SDCA.

regarded as a constant, we have $\kappa = 1/(\lambda\gamma) \sim \sqrt{n}$. In this case, the suboptimality level scales as

$$K(n+\kappa)(2+n+\kappa) \sim (\Omega M \rho\tau)^2 n + \Omega M^2 \rho\tau\sqrt{n}$$

To make the result in Theorem 3 meaningful, we need the suboptimality level be a small constant, which requires

$$\Omega\rho\tau = O(1/\sqrt{n}).$$

This condition can be satisfied by many sparse datasets. When this condition is not satisfied, A-SDCA algorithm may fail to converge to desired suboptimality gap, degrading generalization accuracy.

This is illustrated in Figure 1 that shows the performance of A-SDCA on two large datasets. For the KDD 2010 dataset, convergence suboptimality gap degrades gracefully when the number of threads (hence the lag τ) increases. For the KDD 2012 dataset, increasing the number of threads beyond one leads to convergence at a high suboptimal gap, failing to improve after the first epoch.

Our analysis of empirical results revealed that the primary reason that A-SDCA does not converge asymptotically to the optimal solution is that, due to the asynchronous updates, the following primal-dual relation does not hold in general:

$$w^{(t)} = \frac{1}{\lambda n}\sum_{i=1}^{n} \alpha_i^{(t)} x_i, \qquad (15)$$

which, by contrast, always holds in the sequential (1 thread) case. As a result, we observe the update $\left\|w^{(t)} - \frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i^{(t)}x_i\right\|$ not converging in the asynchronous case.

The above observation motivates us to propose a semi-asynchronous SDCA method, SA-SDCA, described in Algorithm 2, which is the primary contribution of this paper. We solve the above problem by periodically forcing the synchronization of the primal and dual variables to enforce their correspondence in Eq. (15).

Note that in Algorithm 2, the synchronization thread that computes w_{sync} does not block the SDCA threads. Instead, it consumes a dynamically-updated most-recent version of α during the computation of w_{sync}, allowing full utilization of CPU at all times. In experiments described in Section 5, we observe nearly linear speed-ups and convergence in both suboptimality gap and holdout-set error accuracy, empirically demonstrating the effectiveness of SA-SDCA.

Algorithm 2: Semi-asynchronous SDCA (SA-SDCA)

1. Run m SDCA threads to update weights and dual vectors in shared memory per Algorithm 1.
2. Every k parallel epochs ($k \times m$ effective epochs), recompute $w_{\text{sync}} \leftarrow \frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i$ in a separate thread, and replace w in shared memory with w_{sync}.

4. OUT-OF-MEMORY SCALING

While the previous section has proposed an attractive asynchronous parallelization of SDCA with strong theoretical guarantees, the assumption of random access to examples implies that the dataset is sufficiently small to reside in memory. The growth of modern industrial datasets to tens of gigabytes and higher, however, invites a technique for efficiently providing high-speed random-order access to disk-based datasets. This section introduces such a technique based on decoupling the data input interfaces, and implementing them for disk-based data with block-wise compression and indexing on top of multi-threaded, buffered I/O.

The basis for the proposed method is a block-compressed binary format with indexing that provides random-order access to blocks. Examples in the dataset are partitioned into equal-sized blocks. Random-order block access is provided by an offset table, with within-block random access provided by upfront decompression of the block upon access.

The algorithm consumes data via an abstraction of an iterator over shuffled examples. This shuffling is not truly uniform, as it involves two dependent levels on randomness: blocks are read from disk in uniformly random order, fol-

lowed by random-order iteration over examples in the block. Threading is orchestrated to coordinate reading compressed blocks from disk with simultaneous decompression and with consumption of examples by the learning algorithm.

Without compression this process is heavily I/O bound, hence compression provides better balancing of available CPU cores and disk throughput. Because I/O and decompression threads do not perform floating-point computations, on modern hyper-threaded hardware these threads do not interfere with the training threads described in the previous section.

Zlib compression [6] works well as it minimizes CPU costs while achieving high compression rates for typical datasets. Furthermore, we note that reduction in data size with compression may result in file size that effectively leads to in-memory reading due to disk caching.

The user chooses the count of examples per block when writing the file. For performant shuffling, this choice should ideally balance some practical considerations: blocks should be large enough that seeks do not dominate I/O and compute time, but small enough that decompressed blocks fit within L3 cache, so that each access of an example in a block is a cache hit.

We note that this approach of a block-partitioned dataset shares motivation with earlier work on out-of-memory SVM training [24]. Despite some similarities, there are two key distinctions between approaches. First, the approach above performs complete streaming pass over the data, whereas [24] loads makes multiple passes over each block loaded into memory. The second key difference that we are plugging our shuffling example iterator into an existing SDCA learner with a general data access interface, not proposing a new learner coupled to a particular storage format. In contrast, earlier work was centered around devising a novel block minimization framework that could perform SVM training when only a subset of the dataset was available in memory at any given time.

5. EXPERIMENTAL RESULTS

We present an experimental study of the proposed methods covering three areas:

- Effects of parallelizing SDCA on convergence
- Performance and convergence for out-of-memory training and impact of pseudo-shuffling
- Comparisons with leading linear learners.

All experimental results were obtained by optimizing the logistic loss. For convergence analysis experiments, we used L_2 parameters (λ) that give best generalization accuracy as measured on held-out test sets. Optimum loss values were obtained by running single-threaded SDCA sufficiently long for between-iterations improvement to be within single floating point precision. For each setting, experiments were repeated 5 times using different random seeds.

Datasets used in this section are summarized in Table 1. We note that all datasets are multi-gigabyte in size and very sparse. For KDD 2010 [21], we used the featurized version on LibSVM website. For KDD 2012 [14] and Criteo [5], we performed a random 90/10 train-test split on the publicly available train sets (hosted by Kaggle), preprocessing categorical features by hashing using 25 hash bits.

Dataset	*#Examples*	*Dimension*	*#Features*
KDD 2010	19.3×10^6	29.9×10^6	0.6×10^9
KDD 2012	33.6×10^6	33.6×10^6	1.4×10^9
Criteo	41.8×10^6	5.6×10^6	1.5×10^9

Table 1: Datasets summary. The *#Features* column denotes the total number of non-zero features.

5.1 Convergence of SA-SDCA

In this section, we analyze empirical convergence and scaling properties, as well as accuracy, of SA-SDCA. Experiments were performed on a hyper-threaded machine with 8 physical cores, hence we investigated parallelization up to 7 threads, reserving one thread for loss computation and periodic primal-dual syncing.

Figure 2 demonstrates that on sparse datasets, SA-SDCA algorithm converges as quickly as the baseline sequential SDCA for a given effective number of passes over data. AUC curves on bottom-most sub-figures show that results with respect to hold-out error mirror those for suboptimality, with near-linear scaling for both with respect to the number of threads.

5.2 Out-of-memory training

In next set of experiments, we investigate the effectiveness of the technique proposed in Section 4 for out-of-memory training. Figure 3 contains results for no shuffling (other than once before training), uniform-random, and pseudo-random shuffling, yielding several interesting observations. First, we note that while pseudo-random shuffling's convergence rate lags that of true random shuffling, it significantly outperforms not shuffling while still allowing disk-based training.

More importantly, per iteration, out-of-memory training is actually faster computationally than standard in-memory training. This is due to two reasons: first, for some datasets, the block-compressed data reduced physical dataset footprint enough to be at least partially cached in memory by the operating system, which reduces disk-access penalties after the initial iteration. Second, block-based shuffling strategy has better higher-level cache efficiency than the uniformly random shuffling scheme, resulting in faster wall-clock performance.

5.3 Comparison with Alternatives

In this subsection, we compare SA-SDCA with leading linear learning implementations detailed below. It is important to emphasize that comparing different software implementations of learning algorithms is inherently difficult, and we tried our best to ensure fairness. To this end, we ran a random hyper-parameter search [3] for all competing algorithm over 50 trials on a homogeneous cluster of nodes with 6-core 2.5GHz CPUs and 48GB of RAM (except for LibLinear as noted below). The following learners were compared:

- LBFGS: a highly tuned implementation of limited-memory BFGS [11], a batch quasi-Newton method, parallelized over 5 threads, swept for memory size and convergence tolerance;
- SGD1: an implementation of Stochastic Gradient Descent, highly tuned following [4], swept for initial learning rate and convergence tolerance;

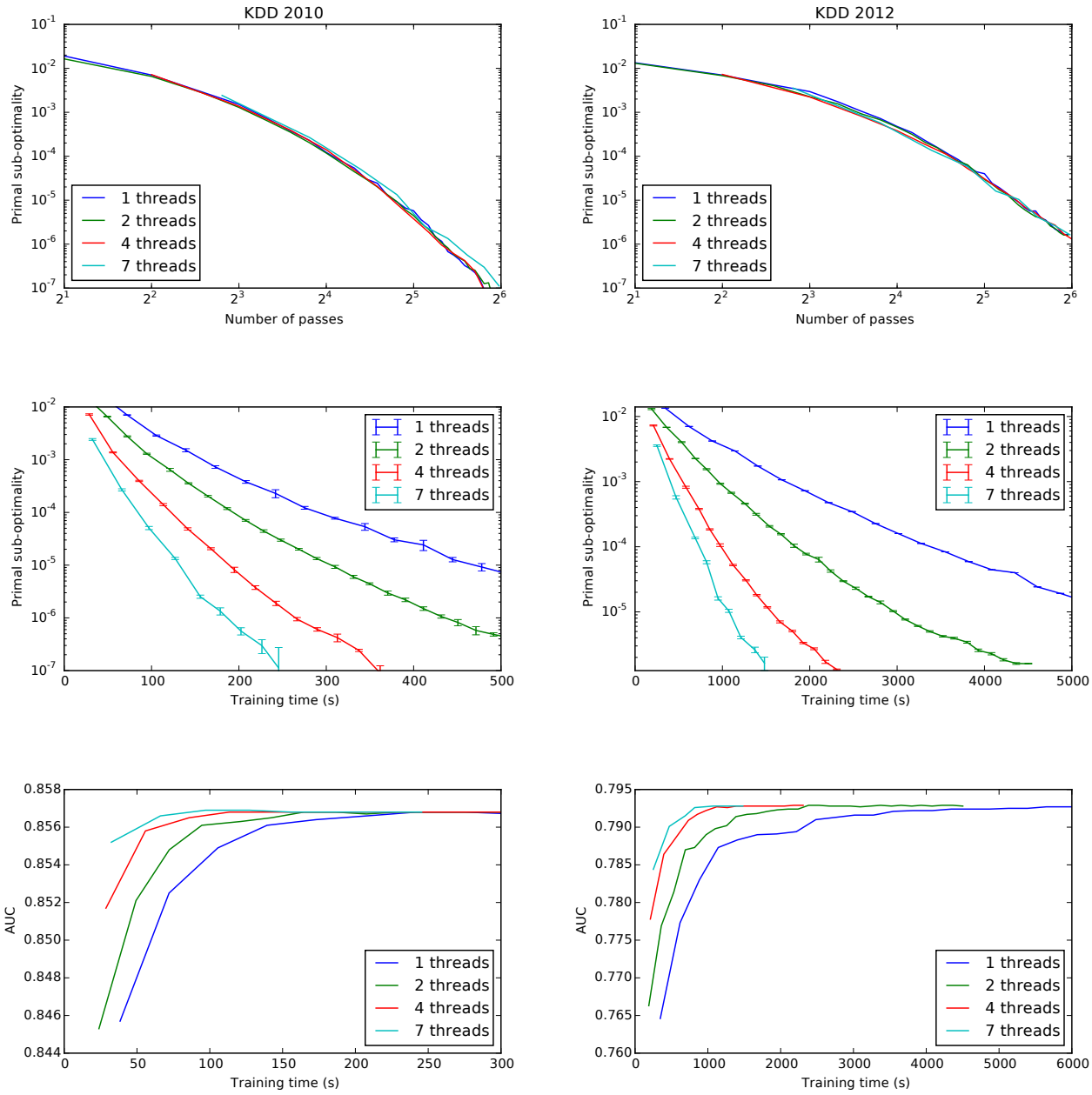

Figure 2: Convergence of SA-SDCA for different number of threads. Left: results on KDD 2010 data. Right: results on KDD 2012 data. Top and middle: primal sub-optimality vs. effective number of passes and training-time, bottom: area under ROC curve for holdout set. Log-log scale for top plots illustrates the super-polynomial convergence rate of the SA-SDCA

- SGD5: asynchronous parallel SGD (Hogwild) with 5 threads, swept as SGD1;
- SDCA1: our sequential SDCA implementation, swept for convergence tolerance;
- SDCA5: SA-SDCA with 5 threads, swept as SDCA1;
- LIBLIN: LibLinear toolkit [25] implementation of SDCA1, swept by one of its authors on different hardware with comparable characteristics;
- VW: Vowpal Wabbit [1] toolkit (SGD and L-BFGS), swept per authors' suggestions for number of passes, initial learning rate (for SGD), and learning rate adaptation power.

For all methods, the same loss function (log-loss) and L_2 regularization parameters were used. LBFGS and LIBLIN required loading datasets into memory, while the rest were streaming, with SGD1, SGD5, SDCA1, and SDCA5 using the out-of-memory pseudo-shuffling described in section 4. For each learner, we select top 20 AUC results, shown in Fig-

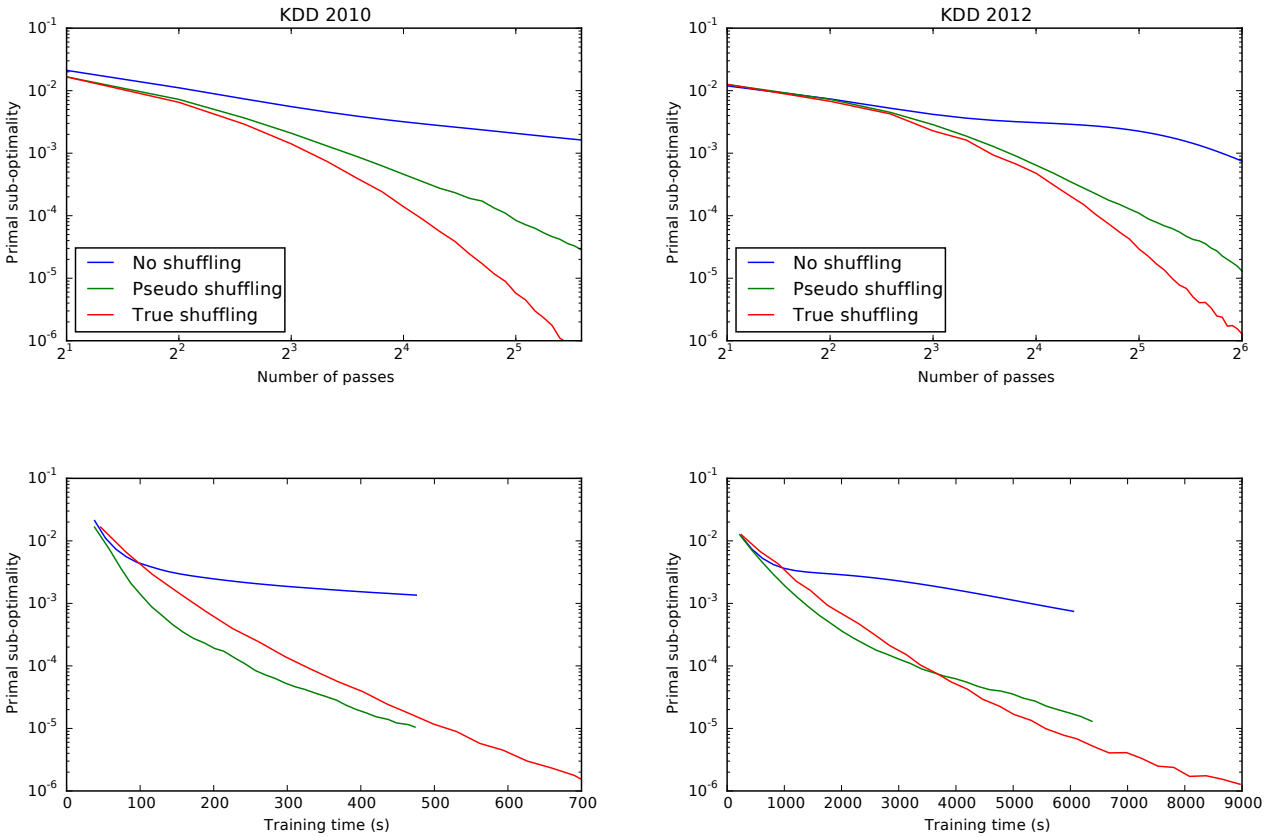

Figure 3: Convergence of SDCA for different shuffling options.

ure 4, with the right panel zooming into the top-performance quadrant indicated by dotted lines in the left panel.

These results demonstrate that our baseline sequential SDCA implementation is competitive with LibLinear, while both outperform VW and L-BFGS. The difference confirms that SDCA demonstrates faster convergence than primal methods, and that dataset reshuffling between iterations is essential for learning optimal parameters. The results also show that SA-SDCA effectively speeds up sequential SDCA by fully utilizing computing resources of modern multi-core processors.

6. RELATED WORK

Recent attention to dual coordinate descent methods was brought by [9], who have shown that they allow achieving linear convergence rates for large-scale linear SVM problems. More generally, [19] proposed and analyzed SDCA method for regularized risk minimization in which a significantly better convergence rate than the commonly used Stochastic Gradient Descent (SGD) methods was proven. In related work, [16] proposed Stochastic Average Gradient (SAG) method for smoothed convex loss functions, which also achieves linear convergence while preserving the iteration complexity of SGD methods, but also requires careful selection of learning step size. In a more general setting, an accelerated variant of SDCA was proposed in [20],with superior performance achieved for a sufficient condition number.

In order to addresses the problem of scaling these methods to modern multi-core hardware systems, a number of synchronous parallel algorithms were introduced in recent years, which assume distributed computation across multiple nodes [15, 26, 26, 23]. In [7, 13, 12], the sparsity has been utilized to develop asynchronous parallel coordinate descent and stochastic gradient type algorithms. In particular, the Hogwild! framework of [13] provided inspiration for the A-SDCA algorithm in Section 3, from which our SA-SDCA method is derived.

In both [13] and [12], it is assumed that there is a bound on the lag between when a processor reads w and the time when this processor makes its update to a single element of w. Moreover, it was shown that a near-linear speedup on a multi-core system can be achieved if the number of processors is $O(n^{1/4})$. Despite this attention, very little work exists on scaling up dual coordinate ascent. [18] have considered the mini-batch approach, where updates are computed on example subsets and aggregated collectively. Experimental evaluation has shown that mini-batches slow down convergence, inviting the use of either Nesterov acceleration or approximate Newton step.

In more recent work, [10] have considered a data-distributed variant of SDCA, named CoCoA, where a master node aggregates updates computed by multiple worker machines on local examples. Results reported in [10] on relatively small datasets do not appear competitive, however. For RCV1, a common text classification benchmark, CoCoA is reported to take 300 seconds on an 8-node cluster to reach the pri-

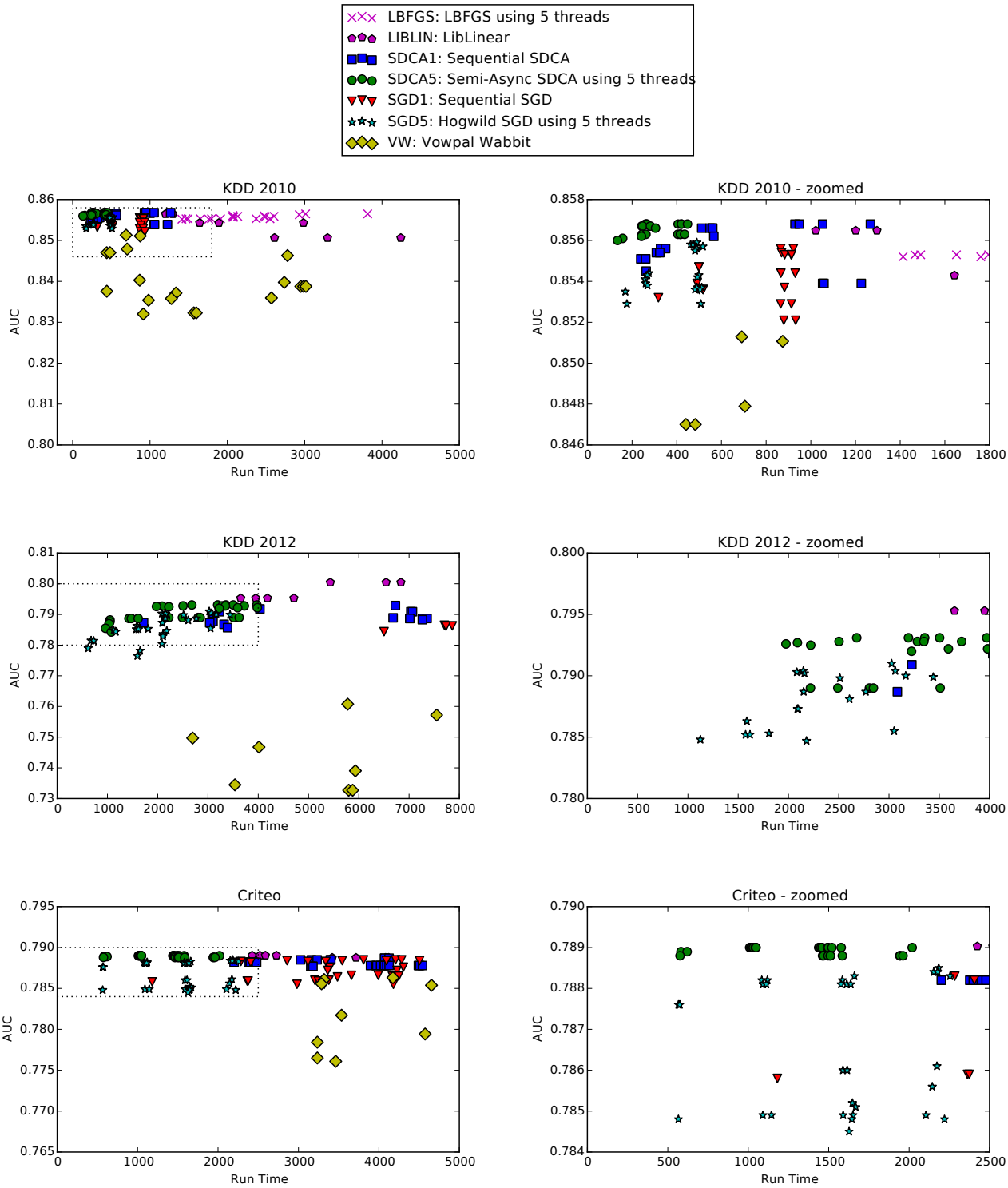

Figure 4: Comparison results. Left: Total run time (in seconds) and AUC of top 20 candidates from each learner. Right: larger display of the dotted region in the left plots. Top: results on KDD 2010 data. Middle: results on KDD 2012 data. Bottom: results on CRITEO data.

mal sub-optimality of 10^{-4}. In contrast, it takes the single-threaded SDCA implementation that is our baseline approximately 5 seconds to reach the same suboptimality level.

7. CONCLUSIONS AND FUTURE WORK

We have described, analyzed and evaluated two techniques for scaling up Stochastic Dual Coordinate Ascent (SDCA) to large datasets: asynchronous updates with primal-dual synchronization, and pseudo-random iteration via indexed, block-compressed serialization. Empirical results demonstrate strong performance in comparison to existing state-of-the-art software for linear learning. This work yields a new state-of-the-art baseline for single-node linear learning, and invites an investigation of combining the method with distributed learning approaches. Further investigation into pseudo-random access is another interesting direction for further research, calling for theoretical analysis of convergence implications of imperfect randomness, and investigating alternative designs, such as the use of quasi-random (low-discrepancy) sequences, that could yield random-quality convergence with even higher throughput.

8. ACKNOWLEDGMENTS

Authors wish to thank Wei-Sheng Chin for assistance with computing LibLinear baseline results, and John Langford and Paul Mineiro for Vowpal Wabbit hyper-parameter suggestions.

9. REFERENCES

[1] A. Agarwal, A. Beygelzimer, D. J. Hsu, J. Langford, and M. J. Telgarsky. Scalable non-linear learning with adaptive polynomial expansions. In *NIPS*, 2014.

[2] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *The Journal of Machine Learning Research*, 15(1):1111–1133, 2014.

[3] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[4] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[5] O. Chapelle et al. http://labs.criteo.com/downloads/2014-kaggle-display-advertising-challenge-dataset.

[6] P. Deutsch and J.-L. Gailly. Zlib compressed data format specification version 3.3. Technical report, RFC 1950, May, 1996.

[7] J. Duchi, M. Jordan, and B. McMahan. Estimation, optimization, and parallelism when data is sparse. *NIPS*, pages 1–9, 2013.

[8] J.-B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Springer, Berlin, 2001.

[9] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th ICML*, pages 408–415, 2008.

[10] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan. Communication-efficient distributed dual coordinate ascent. In *NIPS*, pages 3068–3076, 2014.

[11] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[12] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. In *Proceedings of the 31st ICML*, 2014.

[13] F. Niu, B. Recht, C. Ré, and S. J. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *In NIPS*, 2011.

[14] Y. Niu, Y. Wang, G. Sun, A. Yue, B. Dalessandro, C. Perlich, and B. Hamner. The tencent dataset and kdd-cup'12. In *KDD-Cup Workshop*, 2012.

[15] P. Richtárik and M. Takáč. Distributed coordinate descent method for learning with big data. *arXiv preprint arXiv:1310.2059*, 2013.

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

[17] S. Shalev-Shwartz and T. Zhang. Proximal stochatic dual coordinate ascent. arXiv:1211.2772, November 2012.

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

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

[20] S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. In *Proceedings of the 31st ICML*, pages 1–41, 2014.

[21] Stamper, Niculescu-Mizil, Ritter, Gordon, and Koedinger. Bridge to algebra 2006-2007 - challenge data set from kdd cup 2010 educational data mining challenge, 2010.

[22] T. Suzuki. Stochastic dual coordinate ascent with alternating direction method of multipliers. In *Proceedings of the 31st ICML*, pages 736–744, 2014.

[23] T. Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *NIPS*, pages 629–637, 2013.

[24] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. *ACM Transactions on Knowledge Discovery From Data*, pages 1–23, 2012.

[25] H.-F. Yu, F.-L. Huang, and C.-J. Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, 2011.

[26] Y. Zhang, M. J. Wainwright, and J. C. Duchi. Communication-efficient algorithms for statistical optimization. In *NIPS*, pages 1502–1510, 2012.

10. APPENDIX

In this appendix, we sketch the proof of Theorem 3. The proof mainly follows the framework in [19, 17], combined with additional techniques for handling asynchronicity in [13]. First, we need the following key lemmas, which we prove in a longer report.

Lemma 4. *Assume that each ϕ_i^* is γ-strongly convex and $s \in [0,1]$. Then the sequence of $w^{(t)}$ and $\alpha^{(t)}$ generated by* Algorithm 1 *with $\alpha^{(0)} = 0$ satisfy*

$$\begin{aligned}\mathbb{E}[D(\alpha^{(t+1)}) - D(\alpha^{(t)})] \;\geq\; & \frac{s}{n}\mathbb{E}\left[P(w^{(t)}) - D(\alpha^{(t)})\right] \\ & -\frac{1}{2\lambda}(\frac{s}{n})^2 H^{t+1} \\ & -\frac{\Omega M^2\rho\tau}{n}(2\Omega\rho\tau + \frac{1+2s}{\lambda n}),\end{aligned}$$

where

$$H^{(t+1)} = \frac{1}{n}\sum_{i=1}^{n}\left(||x_i||^2 - \frac{\gamma(1-s)\lambda n}{s}\right) E\left[(u_i^{(t)} - \alpha_i^{(t)})^2\right],$$

and $-u_i^{(t)} \in \partial\phi_i(x_i^T w^{(t)})$.

Moreover, the following observation is presented in [19] and thus, it is presented here without proof.

Lemma 5. *For all α, $D(\alpha) \leq P(w^*) \leq P(0) \leq 1$ and $D(0) \geq 0$.*

The proof of our basic results stated in Theorem 3 relies on the boundedness of the expected increase in dual objective from below by the duality gap. Lemma 4 implies that the duality gap can be further lower bounded using dual suboptimality and solved to obtain the convergence of dual objective based on recursion. Note that since, ϕ_i is $(1/\gamma)$ - smooth, we can assume that there exist $M \in \mathbb{R}$ such that ϕ_i is locally $\frac{M}{2}$-Lipshitz continuous and subsequently we have $||\Delta\alpha||_2 \leq M$. Moreover, ϕ_i^* is γ - strongly convex and from Lemma 4 we have

$$\begin{aligned}E[D(\alpha^{t+1}) - D(\alpha^t)] \;\geq\; & \frac{s}{n}E\left[P(w^t) - D(\alpha^t)\right] - \\ & \frac{1}{2\lambda}(\frac{s}{n})^2 H^{t+1} - \\ & \frac{\Omega M^2\rho\tau}{n}(2\Omega\rho\tau + \frac{1+2s}{\lambda n}),\end{aligned}$$

where $H^{t+1} = \frac{1}{n}\sum_{i=1}^{n}\left(||x_i||^2 - \frac{\gamma(1-s)\lambda n}{s}\right) E\left[(u_i^t - \alpha_i^t)^2\right]$. By choosing $s = \frac{\lambda n\gamma}{1+\lambda n\gamma} \in [0,1]$, we have $H^{t+1} \leq 0$ and subsequently

$$\begin{aligned}E[D(\alpha^{t+1}) - D(\alpha^t)] \;\geq\; & \frac{s}{n}E\left[P(w^t) - D(\alpha^t)\right] - \\ & \frac{\Omega M^2\rho\tau}{n}(2\Omega\rho\tau + \frac{1+2s}{\lambda n}). \quad (16)\end{aligned}$$

Let $\varepsilon_D^t \coloneqq D(\alpha^*) - D(\alpha^t) \leq P(w^t) - D(\alpha^t)$ and thus we have $\varepsilon_D^t - \varepsilon_D^{t+1} = D(\alpha^{t+1}) - D(\alpha^t)$. Therefore, by using recursion on (16), we obtain

$$\begin{aligned}E[\varepsilon_D^t] \;\leq\; & (1-\frac{s}{n})^t E[\varepsilon_D^0] + \\ & \frac{\Omega M^2\rho\tau}{n}(2\Omega\rho\tau + \frac{1+2s}{\lambda n})\sum_{i=0}^{t}(1-\frac{s}{n})^i,\end{aligned}$$

and $(1-\frac{s}{n})^t \leq e^{-\frac{s}{n}t}$ implies

$$E[\varepsilon_D^t] \leq e^{-\frac{s}{n}t} + \frac{\Omega M^2\rho\tau}{n}(2\Omega\rho\tau + \frac{1+2s}{\lambda n})\sum_{i=0}^{t} e^{-\frac{s}{n}i}. \quad (17)$$

In addition, the last term o the right hand side of (17) can be bounded using integral test as

$$\sum_{i=0}^{t} e^{-\frac{s}{n}i} \leq 1 + \int_0^t e^{-\frac{s}{n}x}dx = 1 + \frac{n}{s} - \frac{n}{s}e^{-\frac{s}{n}t},$$

which implies

$$\begin{aligned}E[\varepsilon_D^t] \;\leq\; & \frac{\Omega M^2\rho\tau}{n}(2\Omega\rho\tau + \frac{1+2s}{\lambda n})\left(1 + \frac{n}{s} - \frac{n}{s}e^{-\frac{s}{n}t}\right) + \\ & e^{-\frac{s}{n}t}. \quad (18)\end{aligned}$$

If it is desired to have $E[\varepsilon_D^t] \leq \epsilon_D$ then we need

$$e^{-\frac{s}{n}t} + \frac{\Omega M^2\rho\tau}{n}(2\Omega\rho\tau + \frac{1+2s}{\lambda n})\left(1 + \frac{n}{s} - \frac{n}{s}e^{-\frac{s}{n}t}\right) \leq \varepsilon_D$$

or equivalently

$$K\left(1 + \frac{1+\lambda\gamma n}{\lambda\gamma}\right) + \left(1 - K\frac{1+\lambda\gamma n}{\lambda\gamma}\right)e^{-\frac{\lambda\gamma}{1+\lambda\gamma n}t} \leq \varepsilon_D,$$

where $s = \frac{\lambda n\gamma}{1+\lambda n\gamma}$ and

$$K = \frac{\Omega M^2\rho\tau}{n}(2\Omega\rho\tau + \frac{1+3\lambda n\gamma}{\lambda n(1+\lambda n\gamma)}).$$

Therefore, the dual problem sub-optimality is bounded by ε_D if

$$t \geq (n + \frac{1}{\lambda\gamma})\log(\frac{1 - K(n + \frac{1}{\lambda\gamma})}{\varepsilon_D - K\left(1 + n + \frac{1}{\lambda\gamma}\right)}).$$

Moreover, the duality gap can be presented as

$$\begin{aligned}E\left[P(w^t) - D(\alpha^t)\right] \;\leq\; & \frac{n}{s}E[\varepsilon_D^t - \varepsilon_D^{t+1}] + \\ & \frac{M^2}{s}\left[2\left(\Omega\rho\tau\right)^2 + \frac{1+2s}{\lambda n}\left(\Omega\rho\tau\right)\right] \\ \leq\; & \frac{n}{s}E[\varepsilon_D^t] + \\ & \frac{M^2}{s}\left[2\left(\Omega\rho\tau\right)^2 + \frac{1+2s}{\lambda n}\left(\Omega\rho\tau\right)\right] \quad (19)\end{aligned}$$

Based on (18) and (19) we have

$$\begin{aligned}E\left[P(w^t) - D(\alpha^t)\right] \;\leq\; & (n+\kappa)(e^{-\frac{t}{\kappa+n}}(1 - K(n+\kappa)) + \\ & K(2+n+\kappa)),\end{aligned}$$

where $\kappa = 1/(\lambda\gamma)$. Moreover, based on $1 - K(n+\kappa) < 0$, if

$$K(n+\kappa)(2+n+\kappa) \leq \varepsilon_P,$$

then we have

$$(n+\kappa)(e^{-\frac{t}{\kappa+n}}(1 - K(n+\kappa)) + K(2+n+\kappa)) < \varepsilon_P,$$

which implies we obtain a duality gap of at most ε_P whenever

$$t \geq (n+\kappa)\log(\frac{(n+\kappa)(1 - K(n+\kappa))}{\varepsilon_P - K(n+\kappa)\left(2+n+\kappa\right)}).$$