Lighter-Communication Distributed Machine Learning via Sufficient Factor Broadcasting

Pengtao Xie, Jin Kyu Kim, Yi Zhou*, Qirong Ho†, Abhimanu Kumar§, Yaoliang Yu, Eric Xing
School of Computer Science, Carnegie Mellon University;
*Department of EECS, Syracuse University;
†Institute for Infocomm Research, A*STAR, Singapore; §Groupon

Abstract

Matrix-parametrized models (MPMs) are widely used in machine learning (ML) applications. In large-scale ML problems, the parameter matrix of a MPM can grow at an unexpected rate, resulting in high communication and parameter synchronization costs. To address this issue, we offer two contributions: first, we develop a computation model for a large family of MPMs, which share the following property: the parameter update computed on each data sample is a rank-1 matrix, *i.e*. the outer product of two "sufficient factors" (SFs). Second, we implement a decentralized, peer-to-peer system, Sufficient Factor Broadcasting (SFB), which broadcasts the SFs among worker machines, and reconstructs the update matrices locally at each worker. SFB takes advantage of small rank-1 matrix updates and efficient partial broadcasting strategies to dramatically improve communication efficiency. We propose a graph optimization based partial broadcasting scheme, which minimizes the delay of information dissemination under the constraint that each machine only communicates with a subset rather than all of machines. Furthermore, we provide theoretical analysis to show that SFB guarantees convergence of algorithms (under full broadcasting) without requiring a centralized synchronization mechanism. Experiments corroborate SFB's efficiency on four MPMs.

1 INTRODUCTION

Machine Learning (ML) provides a principled and effective mechanism for extracting latent structure and patterns from raw data and making automatic predictions and decisions. The growing prevalence of *big data*, such as billions of text pages in the web, hundreds of hours of video uploaded to video-sharing sites every minute[1], accompanied by an increasing need of *big model*, such as neural networks (Dean et al., 2012) and topic models (Yuan et al., 2015) with billions of parameters, has inspired the design and development of distributed machine learning systems (Dean and Ghemawat, 2008; Gonzalez et al., 2012; Zaharia et al., 2012; Li et al., 2014; Xing et al., 2015) running on research clusters, data center and cloud platforms with 10s-1000s machines.

For many machine learning (ML) models, such as multiclass logistic regression (MLR), neural networks (NN) (Chilimbi et al., 2014), distance metric learning (DML) (Xing et al., 2002) and sparse coding (SC) (Olshausen and Field, 1997), their parameters can be represented by a matrix $\mathbf{W}$. For example, in MLR, rows of $\mathbf{W}$ represent the classification coefficient vectors corresponding to different classes; whereas in SC rows of $\mathbf{W}$ correspond to the basis vectors used for reconstructing the observed data. A learning algorithm, such as stochastic gradient descent (SGD), would iteratively compute an update $\Delta\mathbf{W}$ from data, to be aggregated with the current version of $\mathbf{W}$. We call such models *matrix-parameterized models* (MPMs).

Learning MPMs in large scale ML problems is challenging: ML application scales have risen dramatically, a good example being the ImageNet (Deng et al., 2009) compendium with millions of images grouped into tens of thousands of classes. To ensure fast running times when scaling up MPMs to such large problems, it is desirable to turn to distributed computation; however, a unique challenge to MPMs is that the parameter matrix grows rapidly with problem size, causing straightforward parallelization strategies to perform less ideally. Consider a data-parallel algorithm, in which every worker uses a subset of the data to update the parameters — a common paradigm is to synchronize the full parameter matrix and update matrices amongst all workers (Dean and Ghemawat, 2008; Dean et al., 2012; Li et al., 2015; Chilimbi et al., 2014; Sindhwani and Ghoting, 2012; Gopal and Yang, 2013). However, this synchronization can quickly become a bottle-

[1] https://www.youtube.com/yt/press/statistics.html

neck: take MLR for example, in which the parameter matrix $\mathbf{W}$ is of size $J \times D$, where J is the number of classes and D is the feature dimensionality. In one application of MLR to Wikipedia (Partalas et al., 2015), $J = 325$k and $D > 10,000$, thus $\mathbf{W}$ contains several billion entries (tens of GBs of memory). Because typical computer cluster networks can only transfer a few GBs per second at the most, inter-machine synchronization of $\mathbf{W}$ can dominate and bottleneck the actual algorithmic computation. In recent years, many distributed frameworks have been developed for large scale machine learning, including Bulk Synchronous Parallel (BSP) systems such as Hadoop (Dean and Ghemawat, 2008) and Spark (Zaharia et al., 2012), graph computation frameworks such as GraphLab (Gonzalez et al., 2012), and bounded-asynchronous key-value stores such as DistBelief(Dean et al., 2012), Petuum-PS (Ho et al., 2013), Project Adam (Chilimbi et al., 2014) and (Li et al., 2014). When using these systems to learn MPMs, it is common to transmit the full parameter matrices $\mathbf{W}$ and/or matrix updates $\Delta\mathbf{W}$ between machines, usually in a server-client style (Dean and Ghemawat, 2008; Dean et al., 2012; Sindhwani and Ghoting, 2012; Gopal and Yang, 2013; Chilimbi et al., 2014; Li et al., 2015). As the matrices become larger due to increasing problem sizes, so do communication costs and synchronization delays — hence, reducing such costs is a key priority when using these frameworks.

We begin by investigating the structure of matrix parameterized models, in order to design efficient communication strategies. We focus on models with a common property: when the parameter matrix $\mathbf{W}$ of these models is optimized with stochastic gradient descent (SGD) (Dean et al., 2012; Ho et al., 2013; Chilimbi et al., 2014) or stochastic dual coordinate ascent (SDCA) (Hsieh et al., 2008; Shalev-Shwartz and Zhang, 2013), the update $\triangle\mathbf{W}$ computed over one (or a few) data sample(s) is of low-rank, *e.g.* it can be written as the outer product of two vectors $\mathbf{u}$ and $\mathbf{v}$: $\triangle\mathbf{W} = \mathbf{u}\mathbf{v}^\top$. The vectors $\mathbf{u}$ and $\mathbf{v}$ are *sufficient factors* (SF, meaning that they are sufficient to reconstruct the update matrix $\triangle\mathbf{W}$). A rich set of models (Olshausen and Field, 1997; Lee and Seung, 1999; Xing et al., 2002; Chilimbi et al., 2014) fall into this family: for instance, when solving an MLR problem using SGD, the stochastic gradient is $\triangle\mathbf{W} = \mathbf{u}\mathbf{v}^\top$, where $\mathbf{u}$ is the prediction probability vector and $\mathbf{v}$ is the feature vector. Similarly, when solving an ℓ_2 regularized MLR problem using SDCA, the update matrix $\triangle\mathbf{W}$ also admits such as a structure, where $\mathbf{u}$ is the update vector of a dual variable and $\mathbf{v}$ is the feature vector. Other models include neural networks (Chilimbi et al., 2014), distance metric learning (Xing et al., 2002), sparse coding (Olshausen and Field, 1997), non-negative matrix factorization (Lee and Seung, 1999) and principal component analysis, to name a few.

Leveraging this property, we propose a system called Sufficient Factor Broadcasting (SFB), whose basic idea is to send sufficient factors (SFs) between workers, which then reconstruct matrix updates $\triangle\mathbf{W}$ locally, thus greatly reducing inter-machine parameter communication. This stands in contrast to the well-established parameter server idiom (Chilimbi et al., 2014; Li et al., 2014), a centralized design where workers maintain a "local" image of the parameters $\mathbf{W}$, which are synchronized with a central parameter image $\mathbf{W}$ (stored on the "parameter servers"). In existing parameter server designs, the (small, low-rank) updates $\triangle\mathbf{W}$ are accumulated into the central parameter server's $\mathbf{W}$, and the low-rank structure of each update $\triangle\mathbf{W}$ is lost in the process. Thus, the parameter server can only transmit the (large, full-rank) matrix $\mathbf{W}$ to the workers, inducing extra communication that could be avoided. We address this issue by designing SFB as a *decentralized, peer-to-peer system*, where each worker keeps its own image of the parameters $\mathbf{W}$ (either in memory or on local disk), and sends sufficient factors to only a subset of other workers, via "partial broadcasting" strategies that avoid the usual $O(P^2)$ peer-to-peer broadcast communication over P machines. SFB also exploits ML algorithm tolerance to bounded-asynchronous execution (Ho et al., 2013), as supported by both our experiments and a theoretical proof. SFB is highly communication-efficient; transmission costs are linear in the dimensions of the parameter matrix, and the resulting faster communication greatly reduces waiting time in synchronous systems (e.g. Hadoop and Spark), or improves parameter freshness in (bounded) asynchronous systems (e.g. GraphLab, Petuum-PS and (Li et al., 2014)). SFs have been used to speed up some (but not all) network communication in deep learning (Chilimbi et al., 2014); our work differs primarily in that we always transmit SFs, never full matrices.

The major contributions of this paper are as follows:

- We identify the *sufficient factor property* of a large family of matrix-parametrized models when solved with two popular algorithms: stochastic gradient descent and stochastic dual coordinate ascent.
- In light of the sufficient factor property, we propose a *sufficient factor broadcasting* (SFB) model of computation. Through a decentralized, peer-to-peer architecture with bounded-asynchronous partial broadcasting, SFB greatly reduces communication complexity while maintaining excellent empirical performance.
- To further reduce communication cost, we investigate a partial broadcasting scheme and propose a graph-optimization based approach to determine the topology of the communication network.
- We analyze the communication and computation costs of SFB and provide a convergence guarantee of SFB based minibatch SGD algorithm, under bulk synchronous and bounded asynchronous executions.

- We empirically evaluate SFB on four popular models, and confirm the efficiency and low communication complexity of SFB.

The rest of the paper is organized as follows. In Section 2 and 3, we introduce the sufficient factor property of matrix-parametrized models and propose the sufficient factor broadcasting computation model, respectively. Section 4 analyzes the costs and convergence behavior of SFB. Section 5 gives experimental results. Section 6 reviews related works and Section 7 concludes the paper.

2 SUFFICIENT FACTOR PROPERTY OF MATRIX-PARAMETRIZED MODELS

The core goal of Sufficient Factor Broadcasting (SFB) is to reduce network communication costs for matrix-parametrized models; specifically, those that follow an optimization formulation

$$\textbf{(P)} \quad \min_{\mathbf{W}} \quad \tfrac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{W}\mathbf{a}_i) + h(\mathbf{W}) \tag{1}$$

where the model is parametrized by a matrix $\mathbf{W} \in \mathbb{R}^{J\times D}$. The loss function $f_i(\cdot)$ is typically defined over a set of training samples $\{(\mathbf{a}_i, \mathbf{b}_i)\}_{i=1}^N$, with the dependence on $\mathbf{b}_i$ being suppressed. We allow $f_i(\cdot)$ to be either convex or nonconvex, smooth or nonsmooth (with subgradient everywhere); examples include ℓ_2 loss and multiclass logistic loss, amongst others. The regularizer $h(\mathbf{W})$ is assumed to admit an efficient proximal operator $\text{prox}_h(\cdot)$. For example, $h(\cdot)$ could be an indicator function of convex constraints, ℓ_1-, ℓ_2-, trace-norm, to name a few. The vectors $\mathbf{a}_i$ and $\mathbf{b}_i$ can represent observed features, supervised information (e.g., class labels in classification, response values in regression), or even unobserved auxiliary information (such as sparse codes in sparse coding (Olshausen and Field, 1997)) associated with data sample i. The key property we exploit below ranges from the matrix-vector multiplication $\mathbf{W}\mathbf{a}_i$. This optimization problem **(P)** can be used to represent a rich set of ML models (Olshausen and Field, 1997; Lee and Seung, 1999; Xing et al., 2002; Chilimbi et al., 2014), such as the following:

Distance Metric Learning (DML) (Xing et al., 2002) improves the performance of other ML algorithms, by learning a new distance function that correctly represents similar and dissimilar pairs of data samples; this distance function is a matrix $\mathbf{W}$ that can have billions of parameters or more, depending on the data sample dimensionality. The vector $\mathbf{a}_i$ is the difference of the feature vectors in the ith data pair and $f_i(\cdot)$ can be either a quadratic function or a hinge loss function, depending on the similarity/dissimilarity label $\mathbf{b}_i$ of the data pair. In both cases, $h(\cdot)$ can be an ℓ_1-, ℓ_2-, trace-norm regularizer or simply $h(\cdot) = 0$ (no regularization).

Sparse Coding (SC) (Olshausen and Field, 1997) learns a dictionary of basis from data, so that the data can be represented sparsely (and thus efficiently) in terms of the dictionary. In SC, $\mathbf{W}$ is the dictionary matrix, $\mathbf{a}_i$ are the sparse codes, $\mathbf{b}_i$ is the input feature vector and $f_i(\cdot)$ is a quadratic function (Olshausen and Field, 1997). To prevent the entries in $\mathbf{W}$ from becoming too large, each column $\mathbf{W}_k$ must satisfy $\|\mathbf{W}_k\|_2 \leq 1$. In this case, $h(\mathbf{W})$ is an indicator function which equals 0 if $\mathbf{W}$ satisfies the constraints and equals ∞ otherwise.

2.1 OPTIMIZATION VIA PROXIMAL SGD, SDCA

To solve the optimization problem **(P)**, it is common to employ either (proximal) stochastic gradient descent (SGD) (Dean et al., 2012; Ho et al., 2013; Chilimbi et al., 2014; Li et al., 2015) or stochastic dual coordinate ascent (SDCA) (Hsieh et al., 2008; Shalev-Shwartz and Zhang, 2013), both of which are popular and well-established parallel optimization techniques.

Proximal SGD: In proximal SGD, a stochastic estimate of the gradient, $\triangle\mathbf{W}$, is first computed over one data sample (or a mini-batch of samples), in order to update $\mathbf{W}$ via $\mathbf{W} \leftarrow \mathbf{W} - \eta \triangle \mathbf{W}$ (where η is the learning rate). Following this, the proximal operator $\text{prox}_{\eta h}(\cdot)$ is applied to $\mathbf{W}$. Notably, the stochastic gradient $\triangle\mathbf{W}$ in **(P)** can be written as the outer product of two vectors $\triangle\mathbf{W} = \mathbf{u}\mathbf{v}^\top$, where $\mathbf{u} = \frac{\partial f(\mathbf{W}\mathbf{a}_i, \mathbf{b}_i)}{\partial(\mathbf{W}\mathbf{a}_i)}$, $\mathbf{v} = \mathbf{a}_i$, according to the chain rule. Later, we will show that this low rank structure of $\triangle\mathbf{W}$ can greatly reduce inter-worker communication.

Stochastic DCA: SDCA applies to problems **(P)** where $f_i(\cdot)$ is convex and $h(\cdot)$ is strongly convex (*e.g.* when $h(\cdot)$ contains the squared ℓ_2 norm); it solves the dual problem of **(P)**, via stochastic coordinate ascent on the dual variables. Introducing the dual matrix $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_N] \in \mathbb{R}^{J\times N}$ and the data matrix $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_N] \in \mathbb{R}^{D\times N}$, the dual problem of **(P)** can be written as

$$\textbf{(D)} \quad \min_{\mathbf{U}} \quad \tfrac{1}{N}\sum_{i=1}^{N} f_i^*(-\mathbf{u}_i) + h^*(\tfrac{1}{N}\mathbf{U}\mathbf{A}^\top) \tag{2}$$

where $f_i^*(\cdot)$ and $h^*(\cdot)$ are the Fenchel conjugate functions of $f_i(\cdot)$ and $h(\cdot)$, respectively. The primal-dual matrices $\mathbf{W}$ and $\mathbf{U}$ are connected by[2] $\mathbf{W} = \nabla h^*(\mathbf{Z})$, where the auxiliary matrix $\mathbf{Z} := \frac{1}{N}\mathbf{U}\mathbf{A}^\top$. Algorithmically, we need to update the dual matrix $\mathbf{U}$, the primal matrix $\mathbf{W}$, and the auxiliary matrix $\mathbf{Z}$: every iteration, we pick a random data sample i, and compute the stochastic update $\triangle\mathbf{u}_i$ by minimizing **(D)** while holding $\{\mathbf{u}_j\}_{j\neq i}$ fixed. The dual variable is updated via $\mathbf{u}_i \leftarrow \mathbf{u}_i - \triangle\mathbf{u}_i$, the auxiliary variable via $\mathbf{Z} \leftarrow \mathbf{Z} - \triangle\mathbf{u}_i\mathbf{a}_i^\top$, and the primal variable via $\mathbf{W} \leftarrow \nabla h^*(\mathbf{Z})$. Similar to SGD, the update of $\mathbf{Z}$ is also the outer product of two vectors: $\triangle\mathbf{u}_i$ and $\mathbf{a}_i$, which can be exploited to reduce communication cost.

[2]The strong convexity of h is equivalent to the smoothness of the conjugate function h^*.

Sufficient Factor Property in SGD and SDCA: In both SGD and SDCA, the parameter matrix update can be computed as the outer product of two vectors — we call these sufficient factors (SFs). This property can be leveraged to improve the communication efficiency of distributed ML systems: instead of communicating parameter/update matrices among machines, we can communicate the SFs and reconstruct the update matrices locally at each machine. Because the SFs are much smaller in size, synchronization costs can be dramatically reduced. See Section 4 below for a detailed analysis.

Low-rank Extensions: More generally, the update matrix $\triangle\mathbf{W}$ may not be exactly rank-1, but still of very low rank. For example, when each machine uses a mini-batch of size K, $\triangle\mathbf{W}$ is of rank at most K; in Restricted Boltzmann Machines, the update of the weight matrix is computed from four vectors $\mathbf{u}_1, \mathbf{v}_1, \mathbf{u}_2, \mathbf{v}_2$ as $\mathbf{u}_1\mathbf{v}_1^\top - \mathbf{u}_2\mathbf{v}_2^\top$, *i.e.* rank-2; for the BFGS algorithm (Bertsekas, 1999), the update of the inverse Hessian is computed from two vectors $\mathbf{u}, \mathbf{v}$ as $\alpha\mathbf{u}\mathbf{u}^\top - \beta(\mathbf{u}\mathbf{v}^\top + \mathbf{v}\mathbf{u}^\top)$, *i.e.* rank-3. Even when the update matrix $\triangle\mathbf{W}$ is not genuinely low-rank, to reduce communication cost, it might still make sense to send only a certain low-rank *approximation*. We intend to investigate these possibilities in future work.

3 SUFFICIENT FACTOR BROADCASTING

Leveraging the SF property of the update matrix in problems **(P)** and **(D)**, we propose a *Sufficient Factor Broadcasting* (SFB) system that supports efficient (low-communication) distributed learning of the parameter matrix $\mathbf{W}$. We assume a setting with P workers, each of which holds a data shard and a copy of the parameter matrix[3] $\mathbf{W}$. Stochastic updates to $\mathbf{W}$ are generated via proximal SGD or SDCA, and communicated between machines to ensure parameter consistency. In proximal SGD, on every iteration, each worker p computes SFs $(\mathbf{u}_p, \mathbf{v}_p)$, based on one data sample $\mathbf{x}_i = (\mathbf{a}_i, \mathbf{b}_i)$ in the worker's data shard. The worker then broadcasts $(\mathbf{u}_p, \mathbf{v}_p)$ to all other workers; once all P workers have performed their broadcast (and have thus received all SFs), they re-construct the P update matrices (one per data sample) from the P SFs, and apply them to update their local copy of $\mathbf{W}$. Finally, each worker applies the proximal operator $\text{prox}_h(\cdot)$. When using SDCA, the above procedure is instead used to broadcast SFs for the auxiliary matrix $\mathbf{Z}$, which is then used to obtain the primal matrix $\mathbf{W} = \nabla h^*(\mathbf{Z})$. Figure 1 illustrates SFB operation: 4 workers compute their respective SFs $(\mathbf{u}_1, \mathbf{v}_1)$, ..., $(\mathbf{u}_4, \mathbf{v}_4)$, which are then broadcast to the other 3 workers. Each worker p uses all 4 SFs $(\mathbf{u}_1, \mathbf{v}_1), \ldots, (\mathbf{u}_4, \mathbf{v}_4)$ to exactly reconstruct the update matrices $\triangle\mathbf{W}_p = \mathbf{u}_p\mathbf{v}_p^\top$, and update their local copy of the parameter matrix: $\mathbf{W}_p \leftarrow \mathbf{W}_p - \sum_{q=1}^4 \mathbf{u}_q\mathbf{v}_q^\top$. While the above description reflects synchronous execution, it is easy to extend to (bounded) asynchronous execution.

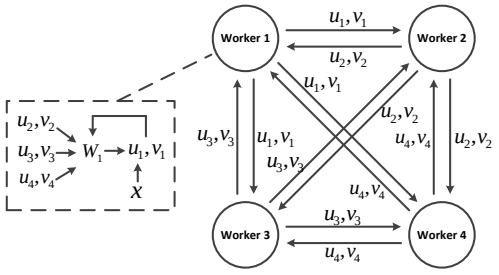

Figure 1: Sufficient Factor Broadcasting (SFB).

SFB vs Client-Server Architectures: The SFB peer-to-peer topology can be contrasted with a "full-matrix" client-server architecture for parameter synchronization, e.g. as used by Project Adam (Chilimbi et al., 2014) to learn neural networks: there, a centralized server maintains the global parameter matrix, and each client keeps a local copy. Clients compute sufficient factors and send them to the server, which uses the SFs to update the global parameter matrix; the server then sends the full, updated parameter matrix back to clients. Although client-to-server costs are reduced (by sending SFs), server-to-client costs are still expensive because full parameter matrices need to be sent. In contrast, the peer-to-peer SFB topology never sends full matrices; only SFs are sent over the network. We also note that under SFB, the update matrices are reconstructed at each of the P machines, rather than once at a central server (for full-matrix architectures). Our experiments show that the time taken for update reconstruction is empirically negligible compared to communication and SF computation.

Partial Broadcasting A naive peer-to-peer topology incurs a communication cost of $O(P^2)$ (where P is the number of worker machines), which inhibits scalability to data center scale clusters where P can reach several thousand (Dean et al., 2012; Li et al., 2014). Hence, SFB adopts an (optional) partial broadcasting scheme where each machine connects with and sends messages to a subset of Q machines (rather than all other machines), thus reducing communication costs from $O(P^2)$ to $O(PQ)$. Figure 2 presents an example. In partial broadcasting, an update U_p^t generated by machine p at iteration t is sent only to machines that are directly connected with p (and the update U_p^t takes effect at iteration $t+1$). The effect of U_p^t is *indirectly and eventually* transmitted to every other machine q, via the updates generated by machines sitting between p and q in the topology. This happens at iteration $t+\tau$, for some delay $\tau > 1$ that depends on Q and the location of p and q in the network topology. Consequently, the

[3]For simplicity, we assume each worker has enough memory to hold a full copy of the parameter matrix $\mathbf{W}$. If $\mathbf{W}$ is too large, one can either partition it across multiple machines (Dean et al., 2012; Li et al., 2014), or use local disk storage (*i.e.* out of core operation). We plan to investigate these strategies as future work.

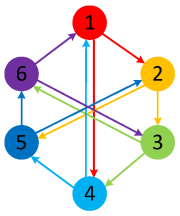

Figure 2: An Example of Partial Broadcasting.

P machines will not have the exact same parameter image $\mathbf{W}$, even under bulk synchronous parallel execution — yet surprisingly, this does not empirically (Section 5) compromise algorithm accuracy as long as Q is not too small. We hypothesize that this property is related to the tolerance of ML algorithms to bounded-asynchronous execution and random error, and we defer a formal proof to future work.

Determining the "best" topology of partial broadcasting, i.e., which subset of machines each machine should send message to, is a challenging issue. Previous studies (Li et al., 2015) are mostly based on heuristics. While empirically effective, these heuristics lack a mathematically formalizable objective. To address this issue, we propose an optimization-oriented solution, with the goal to achieve fast dissemination of information: the effect of updates generated at each machine should be "seen" by all other machines as quickly as possible. Formally, we aim to reduce the delay τ. Consider a directed network with P nodes, where $e_{pq} = 1$ denotes that node p sends message to node q and $e_{pq} = 0$ otherwise. Let c_{pq} denote the shortest directed path from node p to q. It is easy to see that $\tau = |c_{pq}|$, where $|c_{pq}|$ is the length of c_{pq}. Letting $E = \{e_{pq}\}_{p=1,q\neq p}^{P}$ we can find a network topology that minimizes τ by solving this optimization problem:

$$\begin{array}{ll} \min_E & \sum_{p=1}^{P} \sum_{q\neq p}^{P} |c_{pq}| \\ s.t. & \sum_{q\neq p}^{P} e_{pq} = Q, \forall p \end{array} \quad (3)$$

which can be efficiently solved by using a Branch and Bound (Land and Doig, 1960) searching of the graph structure, in conjunction with an algorithm (Williams, 2014) finding all-pairs shortest path in directed graphs.

Mini-batch Proximal SGD/SDCA: SFB can also be used in mini-batch proximal SGD/SDCA; every iteration, each worker samples a mini-batch of K data points, and computes K pairs of sufficient factors $\{(\mathbf{u}_i, \mathbf{v}_i)\}_{i=1}^{K}$. These K pairs are broadcast to all other workers, which reconstruct the originating worker's update matrix as $\triangle\mathbf{W} = \frac{1}{K}\sum_{i=1}^{K} \mathbf{u}_i\mathbf{v}_i^{\mathsf{T}}$.

Consistency Models SFB supports two consistency models: Bulk Synchronous Parallel (BSP-SFB) and Stale Synchronous Parallel (SSP-SFB), and we provide theoretical convergence guarantees in the next section.

- **BSP-SFB:** Under BSP (Dean and Ghemawat, 2008; Zaharia et al., 2012), an end-of-iteration global barrier ensures all workers have completed their work, and synchronized their parameter copies, before proceeding to the next iteration. BSP is a strong consistency model, that guarantees the same computational outcome (and thus algorithm convergence) each time.

```
sfb_app mlr ( int J, int D, int staleness )
//SF computation function
function compute_sv ( sfb_app mlr ):
  while ( ! converged ):
    X = sample_minibatch ()
    foreach x_i in X:
      //sufficient factor u_i
      pred = predict ( mlr.para_mat, x_i )
      mlr.sv_list[i].write_u ( pred )
      //sufficient factor v_i
      mlr.sv_list[i].write_v ( x_i )
    commit()
```

Figure 3: Multiclass LR Pseudocode.

- **SSP-SFB:** BSP can be sensitive to stragglers (slow workers) (Ho et al., 2013), limiting the distributed system to the speed of the slowest worker. Stale Synchronous Parallel (SSP) (Bertsekas and Tsitsiklis, 1989; Ho et al., 2013) communication model addresses this issue, by allowing workers to advance at different rates, provided that the difference in iteration number between the slowest and fastest workers is no more than a user-provided staleness s. SSP alleviates the straggler issue while guaranteeing algorithm convergence (Ho et al., 2013). Under SSP-SFB, each worker p tracks the number of SF pairs computed by itself, t_p, versus the number $\tau_p^q(t_p)$ of SF pairs received from each worker q. If there exists a worker q such that $t_p - \tau_p^q(t_p) > s$ (*i.e.* some worker q is likely more than s iterations behind worker p), then worker p pauses until q is no longer s iterations or more behind.

Programming Interface The SFB programming interface is simple; users need to provide a SF computation function to specify how to compute the sufficient factors. To send out SF pairs $(\mathbf{u}, \mathbf{v})$, the user adds them to a buffer object sv_list, via: $write\_u(vec\_u)$, $write\_v(vec\_v)$, which set i-th SF $\mathbf{u}$ or $\mathbf{v}$ to vec_u or vec_v. All SF pairs are sent out at the end of an iteration, which is signaled by $commit()$. Finally, in order to choose between BSP and SSP consistency, users simply set $staleness$ to an appropriate value (0 for BSP, > 0 for SSP). SFB automatically updates workers' local parameter matrix using all SF pairs — including both locally computed SF pairs added to sv_list, as well as SF pairs received from other workers. Figure 3 shows SFB pseudocode for multiclass logis-

tic regression. For proximal SGD/SDCA algorithms, SFB requires users to write an additional function, $prox(mat)$, which applies the proximal operator $\text{prox}_h(\cdot)$ (or the SDCA dual operator $h^*(\cdot)$) to the parameter matrix mat.

4 COST ANALYSIS AND THEORY

We now examine the costs and convergence behavior of SFB under synchronous and bounded-async (e.g. SSP (Bertsekas and Tsitsiklis, 1989; Ho et al., 2013)) consistency, and show that SFB can be preferable to full-matrix synchronization/communication schemes.

4.1 COST ANALYSIS

Figure 4 compares the communications, space and time (to apply updates to $\mathbf{W}$) costs of peer-to-peer SFB, against full matrix synchronization (FMS) under a client-server architecture (Chilimbi et al., 2014). For SFB with a full broadcasting scheme, in each minibatch, every worker broadcasts K SF pairs $(\mathbf{u}, \mathbf{v})$ to $P-1$ other workers, i.e. $O(P^2K(J+D))$ values are sent per iteration — linear in matrix dimensions J, D, and quadratic in P. For SFB with a partial broadcasting scheme, every worker communicates SF pairs with $Q < P$ peers, hence the communication cost is reduced to $O(PQK(J+D))$. Because SF pairs cannot be aggregated before transmission, the cost has a dependency on K. In contrast, the communication cost in FMS is $O(PJD)$, linear in P, quadratic in matrix dimensions, and independent of K. For both SFB and FMS, the cost of storing $\mathbf{W}$ is $O(JD)$ on every machine. As for the time taken to update $\mathbf{W}$ per iteration, FMS costs $O(PJD)$ at the server (to aggregate P client update matrices) and $O(PKJD)$ at the P clients (to aggregate K updates into one update matrix). By comparison, SFB bears a cost of $O(P^2KJD)$ under full broadcasting and $O(PQKJD)$ under partial broadcasting due to the additional overhead of reconstructing each update matrix P or Q times.

Compared with FMS, SFB achieves communication savings by paying an extra computation cost. In a number of practical scenarios, such a tradeoff is worthwhile. Consider large problem scales where $\min(J, D) \geq 10000$, and moderate minibatch sizes $1 \leq K \leq 100$ (as studied in this paper); when using a moderate number of machines (around 10-100), the $O(P^2K(J+D))$ communications cost of SFB is lower than the $O(PJD)$ cost for FMS, and the relative benefit of SFB improves as the dimensions J, D of $\mathbf{W}$ grow. In data center scale computing environments with thousands of machines, we can adopt the partial broadcasting scheme. As for the time needed to apply updates to $\mathbf{W}$, it turns out that the additional cost of reconstructing each update matrix P or Q times in SFB is negligible in practice — we have observed in our experiments that the time spent computing SFs, as well as communicating SFs over the network, greatly dominates the cost of reconstructing update matrices using SFs. Overall, the communication savings dominate the added computational overhead, which we validated in experiments.

4.2 CONVERGENCE ANALYSIS

We study the convergence of minibatch SGD under full broadcasting SFB (with extensions to proximal-SGD, SDCA being a topic for future study). Since SFB is a peer-to-peer decentralized computation model, we need to show that parameter copies on different workers converge to the same limiting point without a centralized coordination, even under delays in communication due to bounded asynchronous execution. In this respect, we differ from analyses of centralized parameter server systems (Ho et al., 2013), which instead show convergence of global parameters on the central server.

We wish to solve the optimization problem $\min_{\mathbf{W}} \sum_{m=1}^{M} f_m(\mathbf{W})$, where M is the number of training data minibatches, and f_m corresponds to the loss function on the m-th minibatch. Assume the training data minibatches $\{1, ..., M\}$ are divided into P disjoint subsets $\{S_1, ..., S_P\}$ with $|S_p|$ denoting the number of minibatches in S_p. Denote $F = \sum_{m=1}^{M} f_m$ as the total loss, and for $p = 1, \ldots, P$, $F_p := \sum_{j \in S_p} f_j$ is the loss on S_p (on the p-th machine).

Consider a distributed system with P machines. Each machine p keeps a local variable $\mathbf{W}_p$ and the training data in S_p. At each iteration, machine p draws one minibatch I_p uniformly at random from partition S_p, and computes the partial gradient $\sum_{j \in I_p} \nabla f_j(\mathbf{W}_p)$. Each machine updates its local variable by accumulating partial updates from all machines. Denote η_c as the learning rate at c-th iteration on every machine. The partial update generated by machine p at its c-th iteration is denoted as $U_p(\mathbf{W}_p^c, I_p^c) = -\eta_c |S_p| \sum_{j \in I_p^c} \nabla f_j(\mathbf{W}_p^c)$. Note that I_p^c is random and the factor $|S_p|$ is to restore unbiasedness in expectation. Then the local update rule of machine p is

$$\begin{array}{l} \mathbf{W}_p^c = \mathbf{W^0} + \sum_{q=1}^{P} \sum_{t=0}^{\tau_p^q(c)} U_q(\mathbf{W}_q^t, I_q^t) \\ 0 \leq (c-1) - \tau_p^q(c) \leq s \end{array} \tag{4}$$

where $\mathbf{W}^0$ is the common initializer for all P machines, and $\tau_p^q(c)$ is the number of iterations machine q has transmitted to machine p when machine p conducts its c-th iteration. Clearly, $\tau_p^p(c) = c$. Note that we also require $\tau_p^q(c) \leq c-1$, *i.e.*, machine p will not use any partial updates of machine q that are too fast forward. This is to avoid correlation in the theoretical analysis. Hence, machine p (at its c-th iteration) accumulates updates generated by machine q up to iteration $\tau_p^q(c)$, which is restricted to be at most s iterations behind. This formulation, in which s is the maximum "staleness" allowed between any update and any worker, covers bulk synchronous parallel (BSP) full broadcasting ($s = 0$) and bounded-asynchronous full broadcasting ($s > 0$). The following standard assumptions are needed for our analysis:

Computational Model	Total comms, per iter	$\mathbf{W}$ storage per machine	$\mathbf{W}$ update time, per iter
SFB (peer-to-peer, full broadcasting)	$O(P^2K(J+D))$	$O(JD)$	$O(P^2KJD)$
SFB (peer-to-peer, partial broadcasting)	$O(PQK(J+D))$	$O(JD)$	$O(PQKJD)$
FMS (client-server (Chilimbi et al., 2014))	$O(PJD)$	$O(JD)$	$O(PJD)$ at server, $O(PKJD)$ at clients

Figure 4: Cost of using SFB versus FMS. K is minibatch size, J, D are dimensions of $\mathbf{W}$, and P is the number of workers.

Assumption 1. *(1) For all j, f_j is continuously differentiable and F is bounded from below; (2) ∇F, ∇F_p are Lipschitz continuous with constants L_F and L_p, respectively, and let $L = \sum_{p=1}^{P} L_p$; (3) There exists B, σ^2 such that for all p and c, we have (almost surely) $\|\mathbf{W}_p^c\| \leq B$ and $\mathbb{E}\| \, |S_p| \sum_{j \in I_p} \nabla f_j(\mathbf{W}) - \nabla F_p(\mathbf{W}) \, \|_2^2 \leq \sigma^2$.*

Our analysis is based on the following auxiliary update

$$\mathbf{W}^c = \mathbf{W}^0 + \sum_{q=1}^{P} \sum_{t=0}^{c-1} U_q(\mathbf{W}_q^t, I_q^t), \qquad (5)$$

Compare to the local update (4) on machine p, essentially this auxiliary update accumulates all $c-1$ updates generated by all machines, instead of the $\tau_p^q(c)$ updates that machine p has access to. We show that all local machine parameter sequences are asymptotically consistent with this auxiliary sequence:

Theorem 1. *Let $\{\mathbf{W}_p^c\}$, $p = 1, \ldots, P$, and $\{\mathbf{W}^c\}$ be the local sequences and the auxiliary sequence generated by SFB for problem **(P)** (with $h \equiv 0$), respectively. Under Assumption 1 and set the learning rate $\eta_c = O(\sqrt{\frac{1}{L\sigma^2 Psc}})$, then we have*

- $\liminf_{c \to \infty} \mathbb{E}\|\nabla F(\mathbf{W}^c)\| = 0$, *hence there exists a subsequence of $\nabla F(\mathbf{W}^c)$ that almost surely vanishes;*
- $\lim_{c \to \infty} \max_p \|\mathbf{W}^c - \mathbf{W}_p^c\| = 0$, i.e. *the maximal disagreement between all local sequences and the auxiliary sequence converges to 0 (almost surely);*
- *There exists a common subsequence of $\{\mathbf{W}_p^c\}$ and $\{\mathbf{W}^c\}$ that converges almost surely to a stationary point of F, with the rate $\min_{c \leq C} \mathbb{E}\| \sum_{p=1}^{P} \nabla F_p(\mathbf{W}_p^c)\|_2^2 \leq O\left(\sqrt{\frac{L\sigma^2 Ps}{C}}\right)$*

Intuitively, Theorem 1 says that, given a properly-chosen learning rate, all local worker parameters $\{\mathbf{W}_p^c\}$ eventually converge to stationary points (i.e. local minima) of the objective function F, despite the fact that SF transmission can be delayed by up to s iterations. Thus, SFB learning is robust even under bounded-asynchronous communication (such as SSP). Our analysis differs from (Bertsekas and Tsitsiklis, 1989) in two ways: (1) Bertsekas and Tsitsiklis (1989) explicitly maintains a consensus model which would require transmitting the parameter matrix among worker machines — a communication bottleneck that we were able to avoid; (2) we allow subsampling in each worker machine. Accordingly, our theoretical guarantee is probabilistic, instead of the deterministic one in (Bertsekas and Tsitsiklis, 1989). In future work, we intend to extend the analysis to partial broadcasting under BSP and bounded-asynchronous execution. Partial broadcasting presents additional challenges, because updates are only sent to a subset of machines (rather than every machine).

5 EXPERIMENTS

We demonstrate how four popular models can be efficiently learnt using SFB: (1) multiclass logistic regression (MLR) and distance metric learning (DML)[4] based on SGD; (2) sparse coding (SC) based on proximal SGD; (3) ℓ_2 regularized multiclass logistic regression (L2-MLR) based on SDCA. For baselines, we compared with (a) Spark (Zaharia et al., 2012) for MLR and L2-MLR, and (b) full matrix synchronization (FMS) implemented on open-source parameter servers (Ho et al., 2013; Li et al., 2014) for all four models. In certain experiments, we made a comparison with the distributed (L2-)MLR system proposed in (Gopal and Yang, 2013). In FMS, workers send update matrices to the central server, which then sends up-to-date parameter matrices to workers[5]. Due to data sparsity, both the update matrices and sufficient factors are sparse; we use this fact to reduce communication and computation costs. The experiments were performed on a cluster where each machine has 64 2.1GHz AMD cores, 128G memory, and a 10Gbps network interface. Unless otherwise noted, 12 machines were used. Some experiments were conducted on 28 machines.

Datasets and Experimental Setup We used two datasets for our experiments: (1) ImageNet (Deng et al., 2009) ILSFRC2012 dataset, which contains 1.2 million images from 1000 categories; the images are represented with LLC features (Wang et al., 2010), whose dimensionality is 172k. (2) Wikipedia (Partalas et al., 2015) dataset, which contains 2.4 million documents from 325k categories; documents are represented with *term frequency, inverse document frequency* (tf-idf), with a dimensionality of 20k. We ran MLR, DML, SC, L2-MLR on the Wikipedia, Ima-

[4]For DML, we use the parametrization proposed in (Weinberger et al., 2005), which is a linear projection matrix $\mathbf{L} \in \mathrm{R}^{d \times k}$, where d is the feature dimension and k is the latent dimension.

[5]This has the same communication complexity as (Chilimbi et al., 2014), which sends SFs from workers to servers, but sends matrices from servers to workers; the latter matrix transmission dominates the total cost.

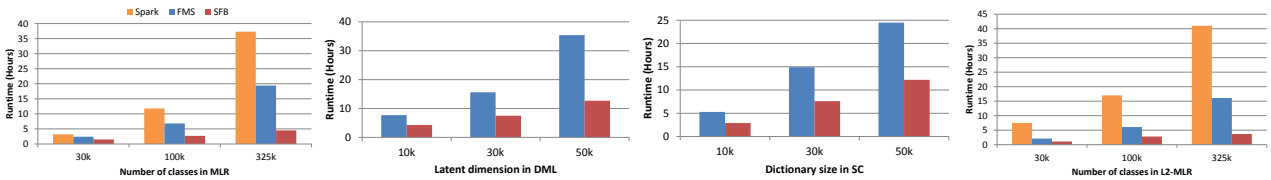

Figure 5: Convergence time versus model size for MLR, DML, SC, L2-MLR (left to right), under BSP.

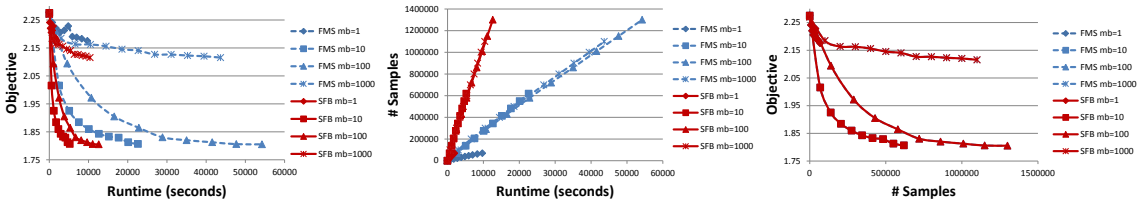

Figure 6: MLR objective vs runtime (left), #samples vs runtime (middle), objective vs #samples (right).

geNet, ImageNet, Wikipedia datasets respectively, and the parameter matrices contained up to 6.5b, 8.6b, 8.6b, 6.5b entries (the largest latent dimension for DML and largest dictionary size for SC were both 50k). The tradeoff parameters in SC and L2-MLR were set to 0.001 and 0.1. We tuned the minibatch size, and found that $K = 100$ was near-ideal for all experiments. All experiments used the same constant learning rate (tuned in the range $[10^{-5}, 1]$). Unless otherwise stated, a full broadcasting scheme is used for most experiments.

Convergence Speed and Quality Figure 5 shows the time taken to reach a fixed objective value, for different model sizes, using BSP consistency (SSP results are in the supplement), on 12 machines. SFB converges faster than FMS, as well as Spark v1.3.1[6]. This is because SFB has lower communication costs, hence a greater proportion of running time gets spent on computation rather than network waiting. This is shown in Figure 6, which plots data samples processed per second[7] (throughput) and algorithm progress per sample for MLR, under BSP consistency (SSP results are in the supplement) and varying minibatch sizes. The middle graph shows that SFB processes far more samples per second than FMS, while the rightmost graph shows that SFB and FMS produce exactly the same algorithm progress per sample under BSP. For this experiment, minibatch sizes between $K = 10$ and 100 performed the best as indicated by the leftmost graph. We point out that larger model sizes should further improve SFB's advantage over FMS, because SFB has linear communications cost in the matrix dimensions, whereas FMS has quadratic costs. Under a large model size (e.g., 325k classes in MLR), the communication cost becomes the bottleneck in FMS and causes prolonged network waiting time and parameter synchronization delays, while the cost is moderate in SFB.

We also evaluated SFB on 28 machines, under BSP and full broadcasting (Q=27). On MLR (325k classes), SFB took 2.46 hours to converge while FMS took 10.77 hours. On L2-MLR (325k classes), the convergence time of SFB is 2.14 hours while that of FMS is 9.31 hours.

We made a comparison with the distributed (L2-)MLR system proposed by (Gopal and Yang, 2013) on 12 machines. On MLR (325k classes), Gopal and Yang (2013) took 31.7 hours to converge while SFB took 4.5 hours. To converge on L2-MLR (325k classes), Gopal and Yang (2013) took 28.3 hours while SFB took 3.7 hours.

Scalability In all experiments that follow, we set the number of (L2-)MLR classes, DML latent dimension, SC dictionary size to 325k, 50k, 50k. Figure 7 shows SFB scalability with varying machines under BSP (SSP results are in the supplement), for MLR, DML, SC, L2-MLR, on 12 machines. In general, we observed close to linear (ideal) speedup, with a slight drop at 12 machines. On 28 machines, for MLR and L2-MLR, SFB achieved 17.4x and 15.7x speedup over one machine respectively.

Computation Time vs Network Waiting Time Figure 8 shows the *total* computation and network time required for SFB and FMS to converge, across a range of SSP staleness values[8] — in general, higher communication cost and lower staleness induce more network waiting. For all staleness values, SFB requires far less network waiting (because SFs are much smaller than full matrices in FMS). Computation time for SFB is slightly longer than FMS because (1) update matrices must be reconstructed on each SFB worker, and (2) SFB requires a few more iterations for convergence, because peer-to-peer communication causes a slightly more parameter inconsistency under staleness. Overall, the SFB reduction in network waiting time remains far greater than the added computation time, and outperforms FMS in total time. For both FMS and SFB,

[6]Spark is about 2x slower than PS (Ho et al., 2013) based C++ implementation of FMS, due to JVM and RDD overheads.

[7]We use samples per second instead of iterations, so different minibatch sizes can be compared.

[8]The Spark implementation does not easily permit this time breakdown, so we omit it.

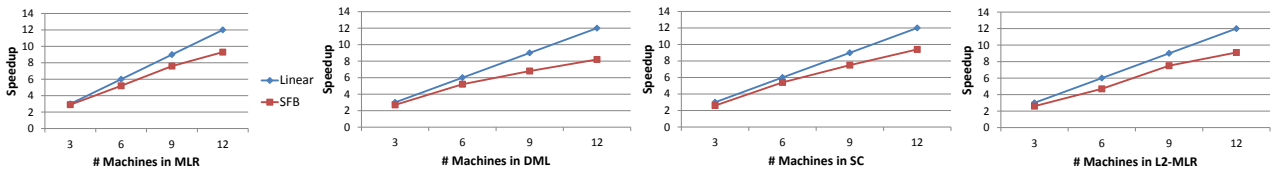

Figure 7: SFB scalability with varying machines under BSP, for MLR, DML, SC, L2-MLR (left to right).

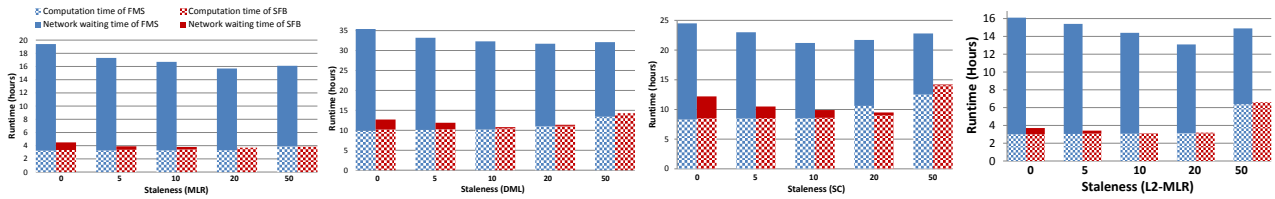

Figure 8: Computation vs network waiting time for MLR, DML, SC, L2-MLR (left to right).

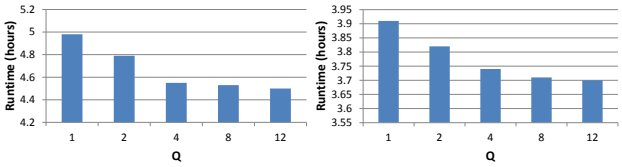

Figure 9: Convergence time versus Q in partial broadcasting for MLR (left) and L2-MLR (right), under BSP.

the shortest convergence times are achieved at moderate staleness values, confirming the importance of bounded-asynchronous communication.

Partial Broadcasting On 12 machines, we studied in partial broadcasting how the parameter Q, which is the number of peers to which each machine sends messages, affects the convergence speed of SFB. Figure 9 shows the convergence time of SFB on MLR and L2-MLR versus varying Q, under BSP (SSP results are given in supplements). First, we observed that SFB under partial broadcasting (PB) with $Q < 11$ converges to the same objective value as under full broadcasting (FB) where $Q = 11$. This provides empirical justification that PB preserves correctness and convergence of algorithms. Second, we noted that the convergence time of PB is affected by Q. As observed in this figure, a smaller Q incurs longer convergence time. This is because a smaller Q is more likely to cause the parameter copies on different workers to be out of synchronization and degrade iteration quality. However, as long as Q is not too small, the convergence speed of PB is comparable with FB. As shown in the figure, for $Q \geq 4$, the convergence time of PB is very close to FB. This demonstrates that using PB, we can reduce the communication cost from $O(P^2)$ to $O(PQ)$ with slight sacrifice of the convergence speed.

6 RELATED WORKS

A number of system and algorithmic solutions have been proposed to reduce communication cost in distributed ML. On the system side, Dean et al. (2012) proposed to reduce communication overhead by reducing the frequency of parameter/gradient exchanges between workers and the server. Li et al. (2014) used filters to select "important" parameters/updates for transmission to reduce the number of data entries to be communicated. On the algorithm side, Tsianos et al. (2012) studied the tradeoffs between communication and computation in distributed dual averaging and distributed stochastic dual coordinate ascent respectively. Shamir et al. (2014) proposed an approximate Newton-type method to achieve communication efficiency in distributed optimization. SFB is orthogonal to these existing approaches and be potentially combined with them to further reduce communication cost.

Peer-to-peer, decentralized architectures have been investigated in other distributed ML frameworks (Li et al., 2015). Our SFB system also adopts such an architecture, but with the specific purpose of supporting the SFB computation model, which is not explored by existing peer-to-peer ML frameworks.

7 CONCLUSIONS

In this paper, we identify the sufficient factor property of a large set of matrix-parametrized models: when these models are optimized with stochastic gradient descent or stochastic dual coordinate ascent, the update matrices are of low-rank. Leveraging this property, we propose a sufficient factor broadcasting strategy to efficiently handle the learning of these models with low communication cost. A partial broadcasting scheme is investigated to alleviate the overhead of full broadcasting. We analyze the cost and convergence property of SFB, whose communication efficiency is demonstrated in empirical evaluations.

Acknowledgements

P.X and E.X are supported by ONR N00014140684, DARPA XDATA FA8701220324, NSF IIS1447676.

References

Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: building an efficient and scalable deep learning training system. In *USENIX Symposium on Operating Systems Design and Implementation*, 2014.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *CACM*, 2008.

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 *NIPS*, 2012.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: distributed graph-parallel computation on natural graphs. In *USENIX Symposium on Operating Systems Design and Implementation*, 2012.

Siddharth Gopal and Yiming Yang. Distributed training of large-scale logistic models. In *ICML*, 2013.

Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *NIPS*, 2013.

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.

Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, 1960.

Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 1999.

Hao Li, Asim Kadav, Erik Kruus, and Cristian Ungureanu. Malt: distributed data-parallelism for existing ml applications. In *Proceedings of the Tenth European Conference on Computer Systems*, 2015.

Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *USENIX Symposium on Operating Systems Design and Implementation*, 2014.

Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 1997.

Ioannis Partalas, Aris Kosmopoulos, Nicolas Baskiotis, Thierry Artieres, George Paliouras, Eric Gaussier, Ion Androutsopoulos, Massih-Reza Amini, and Patrick Galinari. Lshtc: A benchmark for large-scale text classification. *arXiv:1503.08581 [cs.IR]*, 2015.

Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *JMLR*, 2013.

Ohad Shamir, Nathan Srebro, and Tong Zhang. Communication efficient distributed optimization using an approximate newton-type method. In *ICML*, 2014.

Vikas Sindhwani and Amol Ghoting. Large-scale distributed non-negative sparse coding and sparse dictionary learning. In *KDD*, 2012.

Konstantinos Tsianos, Sean Lawlor, and Michael G Rabbat. Communication/computation tradeoffs in consensus-based distributed optimization. In *NIPS*, 2012.

Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *CVPR*, 2010.

Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2005.

Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, 2014.

Eric P Xing, Michael I Jordan, Stuart Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *NIPS*, 2002.

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. In *KDD*, 2015.

Jinhui Yuan, Fei Gao, Qirong Ho, Wei Dai, Jinliang Wei, Xun Zheng, Eric Po Xing, Tie-Yan Liu, and Wei-Ying Ma. Lightlda: Big topic models on modest computer clusters. In *Proceedings of the 24th International Conference on World Wide Web*, 2015.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX Symposium on Networked Systems Design and Implementation*, 2012.