Towards Geo-Distributed Machine Learning

Ignacio Cano[w*] Markus Weimer[m] Dhruv Mahajan[f†] Carlo Curino[m]
Giovanni Matteo Fumarola[m] Arvind Krishnamurthy[w]

[w]University of Washington
{icano,arvind}@cs.washington.edu
[m]Microsoft
{mweimer,ccurino,gifuma}@microsoft.com
[f]Facebook
dhruvm@fb.com

Abstract

Latency to end-users and regulatory requirements push large companies to build data centers all around the world. The resulting data is "born" geographically distributed. On the other hand, many Machine Learning applications require a global view of such data in order to achieve the best results. These types of applications form a new class of learning problems, which we call Geo-Distributed Machine Learning (GDML). Such applications need to cope with: 1) scarce and expensive cross-data center bandwidth, and 2) growing privacy concerns that are pushing for stricter data sovereignty regulations.

Current solutions to learning from geo-distributed data sources revolve around the idea of first centralizing the data in one data center, and then training locally. As Machine Learning algorithms are communication-intensive, the cost of centralizing the data is thought to be offset by the lower cost of intra-data center communication during training.

In this work, we show that the current centralized practice can be far from optimal, and propose a system architecture for doing geo-distributed training. Furthermore, we argue that the geo-distributed approach is structurally more amenable to dealing with regulatory constraints, as raw data never leaves the source data center. Our empirical evaluation on three real datasets confirms the general validity of our approach, and shows that GDML is not only possible but also advisable in many scenarios.

1 Introduction

Modern organizations have a planetary footprint. Data is created where users and systems are located, *all around the globe*. The reason for this is two-fold: 1) minimizing latency between serving infrastructure and end-users, and 2) respecting regulatory constraints, that might require data about citizens of a nation to reside within the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[*]Most of this work was done while interning at Microsoft.
[†]Most of this work was done while at Microsoft.

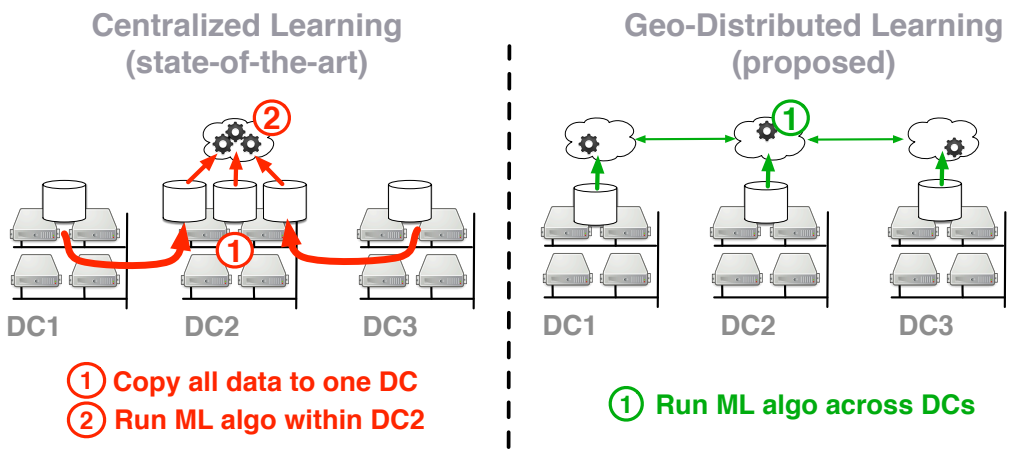


Figure 1: Centralized vs Geo-distributed Learning.

nation's borders. On the other hand, many machine learning applications require access to all that data at once to build accurate models. For example, fraud prevention systems benefit tremendously from the global picture in both finance and communication networks, recommendation systems rely on the maximum breadth of data to overcome cold start problems, and the predictive maintenance revolution is only possible because of data from all markets. These types of applications that deal with geo-distributed datasets belong to a new class of learning problems, which we call Geo-Distributed Machine Learning (GDML).

The state-of-the-art approach to Machine Learning from decentralized datasets is to centralize them. As shown in the left-side of Figure 1, this involves a two-step process: 1) the various partitions of data are copied into a single data center (DC)—thus recreating the overall dataset in a central location, and 2) learning takes place there, using existing intra-data center technologies. Based on conversations with practitioners at Microsoft, we gather that this *centralized* approach is predominant in most practical settings. This is consistent with reports on the infrastructures of other large organizations, such as Facebook [1], Twitter [2], and LinkedIn [3]. The reason for its popularity is two-fold, on the one hand, centralizing the data is the easiest way to reuse existing ML frameworks [4–6]), and on the other hand, Machine Learning algorithms are notoriously communication-intensive, and thus assumed to be non-amenable to cross-data center execution—as we will see, in many practical settings, we can challenge this assumption.

The centralized approach has two key shortcomings:

1. It consumes large amounts of cross-data center (X-DC) bandwidth (in order to copy the raw data to a single location). X-DC bandwidth has been shown to be scarce, expensive, and growing at a slower pace than most other intra-data center (in-DC) resources [7–10].

2. It requires raw data to be copied across data centers, thus potentially across national borders. While international regulations are quickly evolving, the authors of this paper speculate that the growing concerns regarding privacy and data sovereignty [11, 12] might become a key limiting factor to the applicability of centralized learning approaches.

We hypothesize that both challenges will persist or grow in the future [13, 14].

In this paper, we propose the geo-distributed learning approach (right-side of Figure 1), where raw data is kept in place, and learning tasks are executed in a X-DC fashion. We show that by leveraging communication-efficient algorithmic solutions [15] together with a distributed resource management fabric, this approach can achieve orders of magnitude lower X-DC bandwidth consumption in practical settings. Moreover, as the geo-distributed learning approach does not require to copy raw data outside their native data center (only statistics

and estimates are copied), it is structurally better positioned to deal with evolving regulatory constraints. A detailed study of this legal aspect of the geo-distributed approach is beyond the scope of this paper.

The solution we propose serves as an initial stand-in for a new generation of geo-distributed learning systems and algorithms, and allows us to present the first study on the relative efficiency of centralized vs. distributed approaches. In this paper, we concentrate on two key metrics: X-DC bandwidth consumption and learning runtime. We show experimentally that properly designed centralized solutions can achieve faster learning times (when the data copy latency is hidden by streaming data as it becomes available), but that distributed solutions can achieve much lower X-DC bandwidth utilization, and thus substantially lower cost for large-scale learning.

Note that while the above two metrics are of great practical relevance, many other dimensions (e.g., resilience to catastrophic DC failures) are worth considering when comparing alternative approaches. In §6, we briefly list these and other open problems that emerge when considering this new class of learning tasks: Geo-Distributed Machine Learning (GDML).

Summarizing, our main contributions are:

- We introduce GDML, an important new class of learning system problems that deals with geo-distributed datasets, and provide a study of the relative merits of state-of-the-art centralized solutions vs. novel geo-distributed alternatives.
- We propose a system that builds upon Apache Hadoop YARN [16] and Apache REEF [17], and extends their functionality to support multi-data center ML applications. We adopt a communication-sparse learning algorithm [15], originally designed to accelerate learning, and leverage it to lower the bandwidth consumption (i.e., cost) for geo-distributed learning.
- We present empirical results from both simulations and a real deployment across continents, which show that under common conditions distributed approaches can trade manageable penalties in training latency (less than $5\times$) for massive bandwidth reductions (multiple orders of magnitude).
- Finally, we highlight that GDML is a tough new challenge, and that many problems, such as WAN fault-tolerance, regulatory compliance, privacy preservation, X-DC scheduling coordination, latency minimization, and support for broader learning tasks remain open.

The remainder of this paper presents these findings as follows: §2 formalizes the problem setting, §3 introduces our approach, §3.1 introduces the algorithmic solution, and §3.2 describes our system implementation. We explain the evaluation setup and show the experimental results in §4. Finally, we discuss related work in §5, open problems in §6 and conclusions in §7.

2 Problem Formulation

In order to facilitate a study of the state-of-the-art centralized approach in contrast to potential geo-distributed alternatives, we formalize the problem below in two dimensions: 1) we specify assumptions about the data, its size and partitioning, and 2) we restrict the set of learning problems to the well known Statistical Query Model (SQM [18]) class.

2.1 Data distribution

We assume the dataset D of N examples (x_i, y_i), where $x_i \in \mathbb{R}^d$ denotes the feature vector and $y_i \in \{-1, 1\}$ denotes the label of example i, to exist in $p \in \{1, \ldots, P\}$ partitions D_p, each of which is generated in one of P data centers. Those P partitions consist of n_p examples each, with $N = \sum_p n_p$. Let d be the dimensionality of the feature vectors and $\bar{d}$ the average sparsity (number of non-zeros) per example. The total size of each partition

can be (roughly) estimated as $s_p = n_p \cdot \bar{d}$. Although this approximation serves our purposes, in order to have a more precise estimate of the partition sizes, we should not rely on a single $\bar{d}$ value (average instance sparsity across the entire data). This is because the sparsity of instance vectors may depend on the data center location. For example, in the case of a recommendation application, the US data center might have more dense feature vectors (user profiles) than those in a data center in South America.

Further, let p^* be the index of the largest partition, i.e., $p^* = \arg\max_p n_p$. Then, the total X-DC transfer needed to centralize the dataset is:

$$T_C = (N - n_{p^*})\, \bar{d} \tag{1}$$

The goal here is to transfer all instances to the data center that holds the largest subset of instances. Data compression is commonly applied to reduce this size, but only by a constant factor.

2.2 Learning Task

For any meaningful discussion of the relative merits of the centralized approach when compared to alternatives, we need to restrict the set of learning problems we consider. Here, we choose those that fit the Statistical Query Model (SQM) [18]. This model covers a wide variety of important practical machine learning models, including K-Means clustering, linear and logistic regression, and support vector machines.

The *beauty* of algorithms that fit into the SQM model is that they can be written in a summation form, which allows them to be easily parallelized [19]. In SQM, the learning algorithm is allowed to obtain estimates of statistical properties of the examples (e.g., sufficient statistics, gradients) but cannot see the examples themselves [20]. In other words, the algorithm can be phrased purely in terms of statistical queries over the data, and those statistical queries decompose into the sum of a function applied to each example [19].

Let that function be denoted by $f_q \in \{f_1, f_2, \ldots f_Q\}$. A query result F_q is then computed as $F_q = \sum_{i=1}^{N} f_q(x_i, y_i)$. With the data partitioning, this can be rephrased as

$$F_q = \sum_{p=1}^{P} \sum_{i=1}^{n_p} f_q(x_i, y_i) \tag{2}$$

In other words, the X-DC transfer per statistical query is the size of the output of its query function f_q, which we denote as s_q. The total X-DC transfer then depends on the queries and the number of such queries issued, n_q, as part of the learning task, both of which depend on the algorithm executed and the dataset. Let us assume we know these for a given algorithm and dataset combination. Then, we can estimate the total X-DC transfer cost of a fully distributed execution as:

$$T_D = (P - 1) \sum_{q=1}^{Q} n_q\, s_q \tag{3}$$

Note that this relies on the cumulative and associative properties of the query aggregation: we only need to communicate one result of size s_q per data center. The data center that aggregates the results does not need to communicate any data over the X-DC network, thus the $P - 1$ term in (3).

With this formalization, the current state-of-the-art approach of centralizing the data relies on the assumption that $T_C \ll T_D$. However, it is not obvious why this should always be the case: the X-DC transfer of the centralized approach T_C grows linearly with the dataset size, whereas the X-DC transfer of a distributed approach T_D grows linearly with the size and number of the queries. Additionally, relatively large partitions per DC typically yield more meaningful statistics per DC. This in turn means that the learning algorithm needs *fewer* queries to converge, lowering T_D as the dataset size grows given a fixed number of partitions. Hence, it is apparent that the assumption that $T_C \ll T_D$ holds for some, but not all regimes. All things being equal, it seems

that larger datasets would favor the distributed approach. Similarly, all things being equal, larger query results and algorithms issuing more queries seem to favor the centralized approach.

In order to study this more precisely, we need to restrict the discussion to a concrete learning problem for which the queries q, their functions f_q and their output sizes s_q are known. Further, the number of such queries can be bounded by invoking the convergence theorems for the chosen learning algorithm. Here, we choose linear modeling to be the learning problem for its simplicity and rich theory. In particular, we consider the l_2 regularized linear learning problem.

Let $l(w\, x_i, y_i)$ be a continuously differentiable loss function with Lipschitz continuous gradient, where $w \in \mathbb{R}^d$ is the weight vector. Let $L_p(w) = \sum_{i \in D_p} l(w\, x_i, y_i)$ be the loss associated with data center p, and $L(w) = \sum_p L_p(w)$ be the total loss over all data centers. Our goal is to find w that minimizes the following objective function, which decomposes per data center:

$$f(w) = \frac{\lambda}{2}||w||^2 + L(w) = \frac{\lambda}{2}||w||^2 + \sum_p L_p(w) \tag{4}$$

where $\lambda > 0$ is the regularization constant. Depending on the loss chosen, this objective function covers important cases such as linear support vector machines (hinge loss) and logistic regression (logistic loss). Learning such model amounts to optimizing (4). Many optimization algorithms are available for the task, and in §3.1 we describe the one we choose.

It is important to note that one common statistical query of all those algorithms is the computation of the gradient of the model in (4) with respect to w. The size of that gradient (per partition and globally) is d. Hence, s_q for this class of models can be approximated by d. This allows us to rephrase the trade-off mentioned above. All things being equal, datasets with more examples (x_i, y_i) would tend to favor the distributed approach. Similarly, all things being equal, datasets with higher dimensionality d would generally lean towards the centralized setting.

3 Approach

In order to study the problem described above, we need two major artifacts: 1) a communication-efficient algorithm to optimize (4), and 2) an actual geo-distributed implementation of that algorithm. In this section, we describe both of these items in detail.

3.1 Algorithm

We focus on the X-DC communication costs. Hence, we need a communication-efficient algorithm capable of minimizing the communication between the data centers. It is clear from (3) that such an algorithm should try to minimize the number of queries whose output size is very large. In the case of (4), this means that the number of X-DC gradient computations should be reduced.

Recently, many communication-efficient algorithms have been proposed that trade-off local computation with communication [18, 21–24]. Here, we use the algorithm proposed by Mahajan et al. [25] to optimize (4), shown in Algorithm 1. We choose this algorithm because experiments show that it performs better than the aforementioned algorithms, both in terms of communication and running time [25]. The algorithm was initially designed for running in a traditional distributed Machine Learning setting, i.e., single data center.[1] In this work, we adapt it for X-DC training, a novel application that was not originally intended for.

The main idea of the algorithm is to trade-off in-DC computation and communication with X-DC communication. The minimization of the objective function $f(w)$ is performed using an iterative descent method in

[1] We confirmed this with the first author as per 11/2017.

which the r-th iteration starts from a point w^r, computes a direction d^r, and then performs a line search along that direction to find the next point $w^{r+1} = w^r + t\, d^r$.

We adapt the algorithm to support GDML in the following manner. Each node in the algorithm now becomes a data center. All the local computations like gradients and loss function on local data now involves both computation as well as in-DC communication among the nodes in the same data center. On the other hand, all global computations like gradient aggregation involves X-DC communication. This introduces the need for two levels of communication and control, described in more detail in §3.2.

In a departure from communication-heavy methods, this algorithm uses distributed computation for generating a good search direction d^r in addition to the gradient g^r. At iteration r, each data center has the current global weight vector w^r and the gradient g^r. Using its local data D_p, each data center can form an approximation $\hat{f}_p$ of f. To ensure convergence, $\hat{f}_p$ should satisfy a gradient consistency condition, $\nabla \hat{f}_p(w^r) = g^r$. The function $\hat{f}_p$ is approximately[2] optimized using a method M to get the local weight vector w_p, which enables the computation of the local direction $d_p = w_p - w^r$. The global update direction is chosen to be $d^r = \frac{1}{P}\sum_p d_p$, followed by a line search to find w^{r+1}.

In each iteration, the computation of the gradient g^r and the direction d^r requires communication across data centers. Since each data center has the global approximate view of the full objective function, the number of iterations required are significantly less than traditional methods, resulting in orders of magnitude improvements in terms of X-DC communication.

The algorithm offers great flexibility in choosing $\hat{f}_p$ and the method M used to optimize it. A general form of $\hat{f}_p$ for (4) is given by:

$$\hat{f}_p(w) = \frac{\lambda}{2}||w||^2 + \tilde{L}_p(w) + \hat{L}_p(w) \tag{5}$$

where $\tilde{L}_p$ is an approximation of the total loss L_p associated with data center p, and $\hat{L}_p(w)$ is an approximation of the loss across all data centers except p, i.e., $L(w) - L_p(w) = \sum_{q \neq p} L_q(w)$. Among the possible choices suggested in [25], we consider the following quadratic approximations[3] in this work:

$$\tilde{L}_p(w) = \nabla L_p(w^r)(w - w^r) + \frac{1}{2}(w - w^r)^T H_p^r (w - w^r) \tag{6}$$

$$\hat{L}_p(w) = (\nabla L(w^r) - \nabla L_p(w^r))(w - w^r) + \frac{P-1}{2}(w - w^r)^T H_p^r (w - w^r) \tag{7}$$

where H_p^r is the Hessian of L_p at w^r. Replacing (6) + (7) in (5) we have the following objective function:

$$\hat{f}_p(w) = \frac{\lambda}{2}||w||^2 + g^r \cdot (w - w^r) + \frac{P}{2}(w - w^r)^T H_p^r (w - w^r) \tag{8}$$

We use the conjugate gradient (CG) algorithm [26] to optimize (8). Note that each iteration of CG involves a statistical query with output size d to do a hessian-vector computation. However, this query involves only in-DC communication and computation, whereas for traditional second order methods like TRON [27], it will involve X-DC communication.

Discussion Let T_{outer} be the number of iterations required by the algorithm to converge. Each iteration requires two queries with output size $s_q = d$ for the gradient and direction computation, and few queries of output

[2]Mahajan et al. [25] proved linear convergence of the algorithm even when the local problems are optimized approximately.

[3]One can simply use $\tilde{L}_p = L_p$, i.e., the exact loss function for data center p. However, Mahajan et. al [25] showed better results if the local loss function is also approximated.

Algorithm 1 Functional Approximation based Distributed Learning Algorithm (FADL)

Choose w^0
for $r = 0, 1...$ **do**
  Compute g^r (X-DC communication)
  Exit if $||g^r|| \leq \epsilon_g ||g^0||$
  for $p = 1, ..., P$ (in parallel) **do**
    Construct $\hat{f}_p(w)$ ((8))
    $w_p \leftarrow$ Optimize $\hat{f}_p(w)$ (in-DC communication)
  end for
  $d^r \leftarrow \frac{1}{P} \sum_p w_p - w^r$ (X-DC communication)
  Line Search to find t (negligible X-DC communication)
  $w^{r+1} \leftarrow w^r + t\, d^r$
end for

size $s_q = 1$ for the objective function computation in the line search. Since $d \gg 1$, we can ignore the X-DC communication cost for the objective function computation. Hence, we can rewrite (3) as $T_D = 2\,(P-1)\,d\,T_{outer}$. Hence, for T_D to be less than T_C the following must hold:

$$2\,(P-1)\,d\,T_{outer} \quad < \quad (N - n_{p^*})\,\bar{d} \tag{9}$$

In practice, the typical value of P (data centers) is relatively small (in the 10s). Since there are few data centers (i.e., few partitions of the data), the above algorithm will take only few (5-20) outer iterations to converge. In fact, in all our experiments in §4, the algorithm converges in less than 7 iterations. This means that as long as the total size of the data is roughly more than $2-3$ orders of magnitude greater than the dimensionality d,[4] doing geo-distributed learning would reduce the X-DC transfers compared to the centralized approach.

3.2 Distributed Implementation

We need a flexible system that can run in two regimes, i.e., in-DC and X-DC, without requiring two separate implementations. Here, we describe such system. Note that our system is *not tied* to the specific algorithm described above, rather, it exposes a generic framework suitable for geo-distributed and centralized implementations of, at least, the algorithms expressible in the Statistical Query Model.

SQM can be implemented using only Broadcast and Reduce operators (including the algorithm described in §3.1). As part of this work, we add X-DC versions of those to Apache REEF, which provides the basic control flow for our implementation. Moreover, our system needs to obtain resources (CPU cores, RAM) across different data centers in a uniformly basis. We manage those resources using Apache Hadoop YARN. Finally, our system leverages YARN's new federation feature (released as part of Apache Hadoop YARN 2.9) to *view* multiple data centers as a single one. We extend Apache REEF with support for that. Below, we provide more details on our three-layer architecture, from bottom to top.[5]

3.2.1 Resource Manager: Apache Hadoop YARN

A resource manager (RM) is a platform that dynamically leases resources, known as containers, to various competing applications in a cluster. It acts as a central authority and negotiates with potentially many Application Masters (AM) the access to those containers [17]. The most well known implementations are Apache Hadoop

[4] Note that for large datasets this is typically the case.

[5] All changes to Apache REEF and Apache Hadoop YARN have been contributed back to the projects where appropriate.

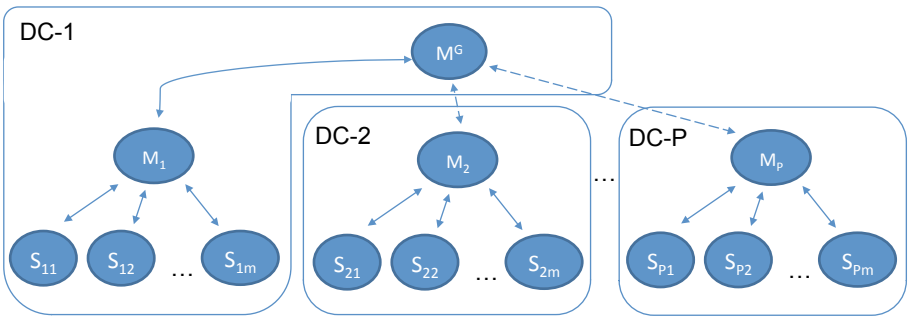


Figure 2: Multi-Level Master/Slave Communication Tree with P data centers, each with its own data center master (M_i) and slaves (S_{ij}). The global master M^G is physically located in DC-1. The solid and dashed lines refer to in-DC and X-DC links respectively. Our current implementation supports recovery from slave failures. Fault-tolerance at the master levels is left for future work.

YARN [16], Apache Mesos [28] and Google Omega [29]. All of these systems are designed to operate *within* one DC and multiplex applications on a collection of shared machines.

In our setting, we need a similar abstraction, but it must *span* multiple DCs. We build our solution on top of Apache Hadoop YARN. As part of our effort to scale-out YARN to Microsoft-scale clusters (tens of thousands of nodes), we have been contributing to Apache a new architecture that allows to *federate* multiple clusters [30]. This effort was not originally intended to operate in a X-DC setting, and as such, was focused on *hiding* from the application layer the different sub-clusters. It is worth mentioning that single DC federation is deployed in production at scale at Microsoft. As part of this work, we have been experimenting and extending this system, leveraging its transparency yet providing enough visibility of the network topology to our application layer (Apache REEF). As a result, we can run a *single* application that spans different data centers in an efficient manner.

3.2.2 Control Flow: Apache REEF

Apache REEF [17] provides a generalized control plane to ease the development of applications on resource managers. REEF provides a control flow master called Driver to applications, and an execution environment for tasks on containers called Evaluator. Applications are expressed as event handlers for the Driver to perform task scheduling (including fault handling) and the task code to be executed in the Evaluators. As part of this work, we extend REEF to support geo-federated YARN, including scheduling of resources to particular data centers.

REEF provides a group communications library that exposes Broadcast and Reduce operators similar to Message Passing Interface (MPI) [31]. Like MPI, REEF's group communications library is designed for the single data center case. We expand it to cover the X-DC case we study here.

3.2.3 GDML Framework

Statistical Query Model algorithms, such as the one introduced in §3.1, can be implemented using nothing more than Broadcast and Reduce operators [19], where data partitions reside in each machine, and the statistical query is Broadcast to those, while its result is computed on each machine and then aggregated via Reduce.

Both Broadcast and Reduce operations are usually implemented via communication trees in order to maximize the overall throughput of the operation. Traditional systems, such as MPI [31] implementations, derive much of their efficiency from intelligent (and fast) ways to establish these trees. Different from the in-DC environment where those are typically used, our system needs to work with network links of vastly different characteristics. X-DC links have higher latency than in-DC ones, whereas the latter have usually higher band-

widths [32]. Further, X-DC links are much more expensive than in-DC links, as they are frequently rented or charged-for separately, in the case of the public cloud.

In our system, we address these challenges with a heterogeneous, multi-level communication tree.

Figure 2 shows an example of the multi-level communication tree we use. A global Broadcast originates from M^G to the data center masters M_i, which in turn do a local Broadcast to the slave nodes S_{ij} in their own data centers. Conversely, local Reduce operations originate on those slave nodes, while the data center masters M_i aggregate the data prior to sending it to M^G for global aggregation.

To make this happen, the underlying implementation creates multiple communication groups. The global master M^G together with the data centers masters M_i, form the global communication group (GCG), where the global Broadcast / Reduce operations are performed, and used in the outer loop of Algorithm 1. Likewise, the slave nodes within each data center and their masters M_i form the local communication groups (LCG), where the local Broadcast / Reduce operations execute, and are used to optimize the local approximations $\hat{f}_p$ of (8).

Summary Overall, our system enables popular resource managing frameworks (Apache Hadoop YARN and Apache REEF) to be used effectively in the management of algorithms running in a geo-distributed fashion. Note that our framework *can work* with any algorithm that can be implemented using Broadcast and Reduce primitives. The specific changes our system embraces, perhaps surprising to software engineers but not to algorithm developers, make the geo-distribution *less* transparent in order for the algorithm author to explicitly control what to run in-DC and what to run across data centers.

4 Evaluation

The algorithm and system presented above allow us to evaluate the state-of-the-art of centralizing the data before learning in comparison with truly distributed approaches. In this section, we describe our findings, starting with the setup and definition of the different approaches used, followed by experimental results from both simulated and real deployments.

4.1 Experimental Setup

We report experiments on two deployments: a distributed deployment on Microsoft Azure across two data centers, and a large centralized cluster on which we simulate a multi-data center setup (2, 4, and 8 data centers). This simulated environment is our main test bench, and we mainly use it for multi-terabyte scale experiments, which are not cost-effective on public clouds. We use 256 slave nodes divided into 2-8 simulated datacenters in all our simulations. Further, all the experiments are done with the logistic loss function.

We ground and validate the findings from the simulations on a real cross-continental deployment on Microsoft Azure. We establish two clusters, one in a data center in Europe and the other on the U.S. west coast. We deploy two DS12 VMs into each of these clusters. Each of those VMs has 4 CPU cores and 28GB of RAM. We establish the site-to-site connectivity through a VPN tunnel using a High Performance VPN Gateway.[6]

4.2 Data

We use three datasets of user behavior data in web sites for our evaluation, two of which are publicly available. All of them are derived from click logs. Table 1 summarizes their statistics. CRITEO and KAGGLE are publicly available [33, 34]. The latter is a small subset of the former, and we use it for the smaller scale experiments in Azure. WBCTR is an internal Microsoft dataset. We vary the number of features in our experiments using hashing kernels as suggested in [35].

[6] `https://azure.microsoft.com/en-us/documentation/articles/vpn-gateway-about-vpngateways/`

Dataset	Examples (N)	Features (d)	Size	
			Model	Dataset
CRITEO	4B	5M	20MB	1.5TB
		10M	40MB	1.5TB
		50M	200MB	1.6TB
		100M	400MB	1.7TB
WBCTR	730M	8M	32MB	347GB
		16M	64MB	362GB
		80M	320MB	364GB
		160M	640MB	472GB
KAGGLE	46M	0.5M	2MB	8.5GB
		1M	4MB	8.5GB
		5M	20MB	9GB

Table 1: Datasets statistics. Dataset sizes reported are *after* compression. Weights in the models are represented in single-precision floating-point format (32 bits) with no further compression. Note that the average sparsity (number of non-zeros) of each of the dataset versions are very similar, thus we do not observe a significant size change after increasing the number of features.

The dataset sizes reported in Table 1 refer to compressed data. The compression/decompression is done using Snappy,[7] which enables high-speed compression and decompression with reasonable compression size. In particular, we achieve compression ratios of around 62-65% for the CRITEO and WBCTR datasets, and 50% for KAGGLE. Following current practice in large scale Machine Learning, our system performs all computations using double precision arithmetic, but communicates single precision floats. Hence, model sizes in Table 1 are reported based on single-precision floating point numbers.

In our experiments, we assume the datasets are randomly partitioned across the data centers, i.e., we assume each data center keeps an approximately equal number of instances. Note that although this is a strong assumption, as data in different data centers can be distributed differently, it holds true in some important production use cases we observe. In such cases, load balancing across data centers forces data to be "randomly" spread across them. However, this is not fully general, as other important GDML workloads require data to be close to the users (to achieve low latency interactions), thus strong geographically biases emerge. Besides the dataset sizes, in practice, data centers can also vary significantly in terms of their bandwidth and computational resources. We plan on addressing all these issues in future work.

4.3 Methods

We contrast the state-of-the-art approach of centralizing the data prior to learning with several alternatives, both within the regime requiring data copies and truly distributed:

centralized denotes the current state-of-the art, where we copy the data to one data center prior to training. Based on the data shipping model used, two variants of this approach arise:

> ***centralized-stream*** refers to a streaming copy model where the data is replicated as it arrives. When the learning job is triggered in a particular data center, the data has already been transferred there, therefore, no copy time is included in the job running time, and

[7] `https://github.com/xerial/snappy-java`

centralized-bulk refers to a batch replication scheme where the data still needs to be copied by the time the learning process starts, therefore, the copy time has an impact on the job running time, i.e., the job needs to wait until the transfer is made to begin the optimization.

We observe both flavors occur in practice. We simply refer to *centralized* when no distinction between its variants is required. This approach (and its variants) only performs *compressed* data transfers, and uses the algorithm described in §3.1 for solving the l_2 regularized linear classification problem mentioned in §2.

distributed builds the multi-level master/slave tree for X-DC learning, but does not use the communication-efficient algorithm in §3.1 to optimize (4), instead, it optimizes using TRON [27].

distributed-fadl uses the algorithm introduced in §3.1 to optimize (4), and similarly to *distributed*, it performs the optimization in a geo-distributed fashion, i.e., it leaves the data in place and runs a single job that spans training across data centers.

Both *distributed* and *distributed-fadl* methods represent the furthest departure from the current state-of-the-art as their execution is truly geo-distributed. Studying results from both allows us to draw conclusions about the relative merits of the system enabling truly geo-distributed training (*distributed*) as well as the system together with a communication-sparse algorithm (*distributed-fadl*).

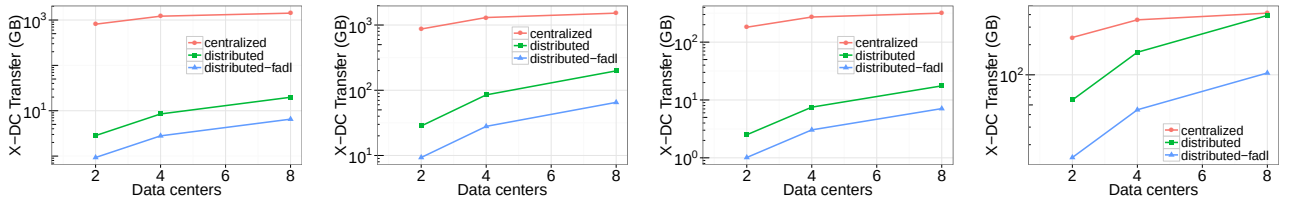
4.4 Results

In this section we present measurements from the methods introduced above, using the datasets described in §4.2. We focus on two key metrics: 1) total X-DC transfer size, and 2) latency to model.

4.4.1 Simulation

X-DC Transfer Figure 3 illustrates the total X-DC transfer of the different methods for different numbers of data centers. We only show two versions of CRITEO and WBCTR for space limitations, though the others follow the same patterns. In general, X-DC transfers increase with the number of data centers as there are more X-DC communication paths. As expected, increasing the model dimensionality also impacts the transfers in the distributed versions. In Figure 3b, the efficient distributed approach (*distributed-fadl*) performs at least one order of magnitude better than *centralized* in every scenario, achieving the biggest difference (2 orders of magnitude) for 2 data centers. In this setting, *centralized* (any variant) transfers half of the compressed data (870 GB) through the X-DC link before training, whereas *distributed-fadl* just needs 9 GBs worth of transfers to train the model. Likewise, in the WBCTR dataset (Figure 3d), we see the biggest difference in the 2 data centers scenario (1 order of magnitude). When the data is spread across 8 data centers, *centralized* transfers almost the same as *distributed*. In general, even the non communication-efficient *distributed* baseline also outperforms the current practice, *centralized*, on both datasets.

Objective / X-DC Transfer Trade-off Commercial deployments of Machine Learning systems impose deadlines and resource boundaries on the training process. This can make it impossible to run the algorithm till convergence. Hence, it is interesting to study the performance of the centralized and distributed approaches in relationship to their resource consumption. Figure 4 shows the relative objective function over time as a function of X-DC transfers for 2 and 8 data centers on the CRITEO and WBCTR datasets. We use the relative difference to the optimal function value, calculated as $(f - f^*)/f^*$, where f^* is the minimum value obtained across methods. X-DC transfers remain constant in the *centralized* (any variant) method as it starts the optimization after the data is copied, i.e., no X-DC transfers are made while training. In general, *distributed-fadl* achieves lower objective values much sooner in terms of X-DC transfers, which means that this method can get some meaningful



(a) CRITEO 10M (b) CRITEO 100M (c) WBCTR 16M (d) WBCTR 160M

Figure 3: X-DC transfer (in GB) versus number of data centers for two versions of CRITEO and WBCTR datasets (y-axis is in log scale). The method *distributed-fadl* consumes orders of magnitude less X-DC bandwidth than any variant (*stream* or *bulk*) of the compressed *centralized* approach. Moreover, a naive algorithm that does not economize X-DC communication, as is the case of the *distributed* method, also reduces transfers with respect to the current *centralized* state-of-the-art.

results faster. If an accurate model is not needed (e.g., 10^{-2} relative objective function value), *distributed-fadl* gives a quicker response. As we increase the number of data centers, X-DC communication naturally increases, which explains the right shift trend in the plots (e.g., Figures 4a and 4b).

Storage As the number of data centers increases, *centralized* (any variant) requires more space on disk. In particular, assuming the data is randomly partitioned across data centers, *centralized* stores at least $1.5\times$ more data than the distributed versions, with a maximum difference of almost $2\times$ when considering 8 data centers. On the other hand, both *distributed* and *distributed-fadl* only need to store the original dataset ($1\times$) throughout the different configurations.

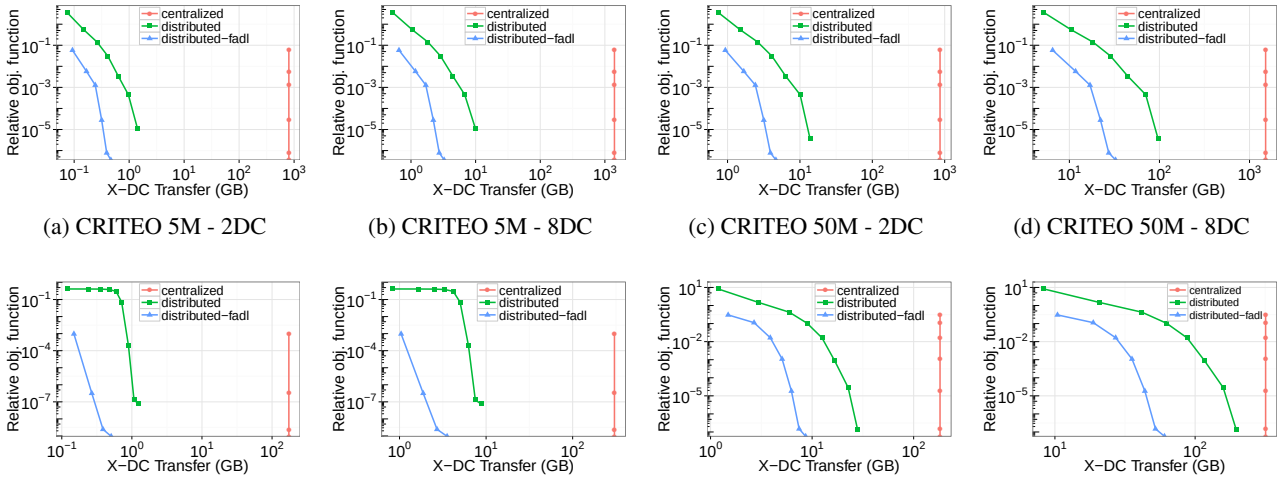
4.4.2 Real Deployment

X-DC Transfer To validate our simulation, we include Figure 5, which shows the relative objective function with respect to the X-DC bandwidth for the KAGGLE dataset in 2 Azure DCs (Western US and Western Europe). These experiments completely match our findings in the simulated environment. For the *centralized* approach, we transfer the data from EU to US, and run the optimization in the latter data center. Similar to Figure 4, the increase in the number of features expectedly causes more X-DC transfers (right shift trend in the plots). The efficient geo-distributed method *distributed-fadl* still communicates the least amount of data, almost 2 orders of magnitude less than the *centralized* (any variant) approach for the 500K model (Figure 5a).

Runtime Figure 6 shows the relative objective function over time for the 2 Azure data centers using the KAGGLE dataset. We normalize the time to the *centralized-stream* approach, calculated as t/t^*, where t^* is the overall time taken by *centralized-stream*. This method performs the fastest in every version of the dataset (500K, 1M, and 5M features) as the data has already been copied by the time it starts, i.e., no copy time overhead is added, and represents the lower bound in terms of running time.

Although the *centralized* approach always transfers compressed data, we do not take into account the compression/decompression time for computing the *centralized-bulk* runtime, which would have otherwise tied the results to the choice of the compression library. Figures 6a, 6b, and 6c show that *centralized-bulk* pays a high penalty for copying the data, it runs in approximately $8\times$ or more of its *stream* counterpart.

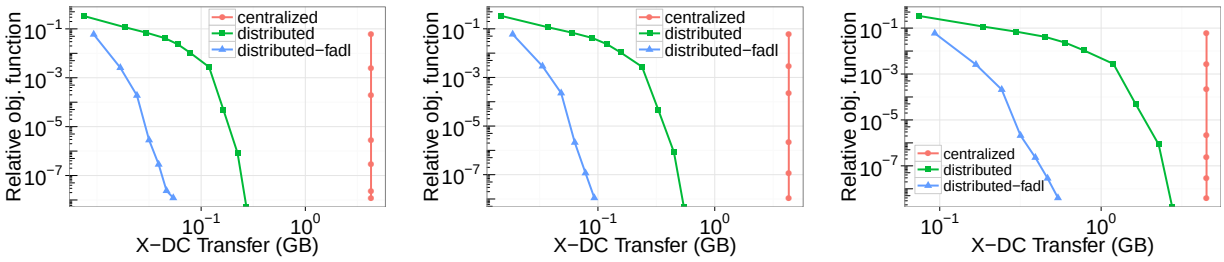
The communication-efficient *distributed-fadl* approach executes in $1.3\times$, $2.4\times$, and $7.4\times$ of the *centralized-stream* baseline for 500K, 1M, and 5M models respectively, which is a remarkable result given that it transfers orders of magnitude less data (Figure 5), and executes in a truly geo-distributed manner, respecting potentially strict regulatory constraints. Moreover, if we consider the relative objective function values commonly used



(a) CRITEO 5M - 2DC (b) CRITEO 5M - 8DC (c) CRITEO 50M - 2DC (d) CRITEO 50M - 8DC

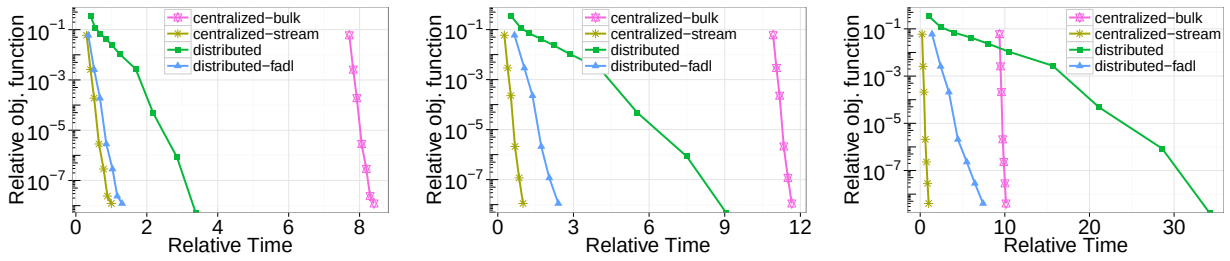
(e) WBCTR 8M - 2DC (f) WBCTR 8M - 8DC (g) WBCTR 80M - 2DC (h) WBCTR 80M - 8DC

Figure 4: Relative objective function (compared to the best) versus X-DC transfer (in GB) for 2 and 8 DCs for two versions of CRITEO and WBCTR datasets (both axis are in log scale). The method *distributed-fadl* achieves lower objective values much sooner in terms of X-DC transfers than the other methods. The *centralized* objective remains constant with respect to X-DC transfers throughout the optimization as it starts once the data has been transferred. The *distributed* method does incur in more transfers than *distributed-fadl*, although it also reduces the overhead of the *centralized* approach. Increasing the models dimensionality, naturally increases the X-DC transfers. Note that *centralized* refers to both of its variants (*stream* and *bulk*), and we only report compressed data transfers for this method.



(a) KAGGLE 500K (b) KAGGLE 1M (c) KAGGLE 5M

Figure 5: Relative objective function (compared to the best) versus X-DC transfer (in GB) for the KAGGLE dataset in 2 Azure data centers (both axis are in log scale). The increase in the model size explains the right shift trend in the plots. The method *distributed-fadl* consumes the least amount of X-DC bandwidth, at least 1 order less in every scenario, and 2 when using the 500K model. The objective/transfer pattern is similar to Figure 4. Both distributed methods transfer much less X-DC data than the *centralized* state-of-the-art. Note that *centralized* refers to both of its flavors (*stream* and *bulk*), and only transfers compressed data.



(a) KAGGLE 500K (b) KAGGLE 1M (c) KAGGLE 5M

Figure 6: Relative objective function (compared to the best) over time (relative to the *centralized-stream* method) for the KAGGLE dataset in 2 Azure data centers (y-axis is in log scale). The method *distributed-fadl* beats every approach but *centralized-stream*. This latter method is the best case scenario, where the data has already been copied and is available in a single data center when the job is executed. The *distributed-fadl* method lies very close to the optimum (*centralized-stream*), especially in low-dimensional models and when considering commonly accepted objective function values (10^{-4}, 10^{-5}). Both *distributed* and *distributed-fadl* performance degrades when the model size increases (as expected), but *distributed* does so much worse (6c), which further shows that in order to do Geo-Distributed Machine Learning, a communication-efficient algorithm is needed.

in practice to achieve accurate models (10^{-4}, 10^{-5}), this method's convergence time lies in the same ballpark as the lower bound *centralized-stream* in terms of running time. Still, *distributed-fadl* is way ahead in terms of X-DC transfers (orders of magnitude of savings in X-DC bandwidth), while at the same time it potentially complies with data sovereignty regulations.

Figure 6a shows that *distributed-fadl* performs very close to the best scenario, matching the intuition built in §2 that this method does very well on tasks with (relatively) small models and (relatively) large number of examples. Furthermore, this efficient method also runs faster than *distributed* in every setting, which further highlights the importance and benefits of the algorithm introduced in §3.1.

Finally, *distributed* performance degrades considerably as the model size increases. In particular, this method does a poor job when running with 5M features (Figure 6c), which concurs with the intuition behind the state-of-the-art *centralized* approach: copying the data offsets the communication-intensive nature of (naive) Machine Learning algorithms. We see that this intuition does not hold true for the efficient algorithm described in §3.1.

5 Related Work

Prior work on systems that deal with geographically distributed datasets exists in the literature.

The work done by Vulimiri et al. [8, 13] poses the thesis that increasing global data and scarce WAN bandwidth, coupled with regulatory concerns, will derail large companies from executing centralized analytics processes. They propose a system that supports SQL queries for doing X-DC analytics. Unlike our work, they do not target iterative machine learning workflows, neither do they focus on jobs latency. They mainly discuss reducing X-DC data transfer volume.

Pu et al. [36] proposes a low-latency distributed analytics system called Iridium. Similar to Vulimiri et al., they focus on pure data analytics and not on Machine Learning tasks. Another key difference is that Iridium optimizes task and data placement across sites to minimize query response time, while our system respects stricter sovereignty constraints and does not move raw data around.

JetStream [7] is a system for wide-area streaming data analysis that performs efficient queries on data stored "near the edge". They provide different approximation techniques to reduce the data size transfers at the ex-

pense of accuracy. One of such techniques is dropping some fraction of the data via sampling. Similar to our system, they only move *important* data to a centralized location for global aggregation (in our case, we only move statistics and models), and they compute local aggregations per site prior to sending (in our case, we perform local optimizations per data center using the algorithm described in §3.1). Another streaming application is distributed monitoring, which has focused on continuous tracking of complex queries over collections of physically distributed data streams. Effective solutions in their setting also need to guarantee communication efficiency over the underlying network [37].

Another line of research has focused on multi-site distributed search engines [38, 39]. Such work has shown to reduce the resource consumption in query processing as well as user perceived latency when compared to single-site centralized search engines [40, 41]. It bears some resemblance to our work but in the context of information retrieval.

Other existing Big Data processing systems, such as Parameter Server, Graphlab, or Spark [4–6, 42], efficiently process data in the context of a single data center, which typically employs a high-bandwidth, relatively low-cost network. To the best of our knowledge, they have not been tested in multi-data center deployments (and were not designed for it), where scarce WAN bandwidth makes it impractical to naively communicate parameters between locations. Instead, our system was specifically optimized to perform well on this X-DC setting.

Since our initial work on GDML systems [43, 44], other studies have emerged in the area. Among the most prominent ones we find Gaia [45], which also focuses on leveraging intelligent communication mechanisms, with more emphasis on reducing training times rather than X-DC transfers. Further, the work by Konečný et al. [46–48] introduces the concept of Federated Optimization, where the idea is to train a global model with data residing in users' mobile devices, instead of DCs. Their setting is very similar to GDML in the sense that communication efficiency is of utmost importance, but the cardinality is quite different (millions of devices as opposed to tens of DCs). This poses other research questions, e.g., what sample of devices to choose at a given point in time, how to alleviate the fact that mobile devices are frequently offline, etc.

Besides the systems solutions, the design of efficient distributed Machine Learning algorithms has also been the topic of a broad research agenda [21–25, 49–52]. In general, all these algorithms perform more complex computations to decrease the overall number of communication rounds. Some recent work uses model quantization (i.e., reduce the number of bits of the model parameters at the expense of potentially losing some accuracy) to reduce the communication cost [53]. The Terascale method [21] might be the best representative method from the Statistical Query Model class and is considered a state-of-the-art solver. CoCoA [24] represents the class of distributed dual methods that, in each outer iteration, solve (in parallel) several local dual optimization problems. Alternating Direction Method of Multipliers (ADMM) [22, 23] is a dual method different from the primal method we use here, however, it also solves approximate problems in the nodes and iteratively reaches the full batch solution. Recent follow up work [25] shows that the algorithm described in §3.1 performs better than the aforementioned ones, both in terms of communication rounds and running time.

6 Discussion and Future Work

GDML is an interesting, challenging and open area of research. Although we have proposed an initial and novel geo-distributed approach that shows substantial gains over the centralized state-of-the-art in many practical settings, many open questions remain, both from a systems and a Machine Learning perspective.

Perhaps, the most crucial aspect is fault tolerance. With data centers distributed across continents, consistent network connectivity is harder to ensure than within a single data center, and network partitioning is more likely to occur. On the other hand, a DC-level failure might completely compromise the centralized approach (if the primary DC is down), while the geo-distributed solution might continue to operate on the remaining data partitions. There has been some initial work [54] to make ML algorithms tolerant to missing data (e.g., machine

failures). This work assumes randomly distributed data across partitions. Hence, a failure removes an unbiased fraction of the data. In production settings, this is the case when multi-DC deployments are created for load-balancing (e.g., within a region)—we are aware of multiple such scenarios within Microsoft's infrastructure. However, cross-region deployments are often dictated by latency-to-end-user considerations. In such settings, losing a DC means losing a heavily biased portion of the population (e.g., all users residing in Western US). Coping with faults and tolerating transient or persistent data unavailability, as well as understanding the impact of different data distributions in convergence speeds are still open problems that will likely require both systems and ML contributions.

In this work we have restricted ourselves to linear models with l_2 regularization, and shown results on logistic regression models. It would be interesting to validate similar observations in other regularizers (e.g., l_1). More broadly, studying geo-distributed solutions that can minimize X-DC transfers for other complex learning problems such as trees, deep neural networks, etc., is still an open area of research.

Further, a truly geo-distributed approach surely does no worse than a centralized method when analyzed from regulatory and data sovereignty angles. Questions in this area arise not only at the global scale, where different jurisdictions might not allow raw data sharing, but also at the very small scale, e.g., between data stored in a private cluster and data shared in the cloud. We believe that studying the setup presented here from a privacy-preserving and regulatory-compliance angle will yield important improvements, and potentially inform regulators.

One aspect we did not cover is related to the work-cycle of these global data repositories and its impact in the efficacy of geo-distributed learning. If the data gets crunched by X algorithms once it is gathered into a single DC, including, perhaps, by algorithms that depend on each others inputs and/or encompass interactive workflows, the centralized methodology might be more effective than the geo-distributed one. This latter approach would increase communication by X-fold, whereas the centralized method would not incur in any extra communication. We consider that a more in-depth study of which approach (centralized or geo-distributed) is more adequate for different problem settings is still missing. Even more, we have not yet addressed the issue of whether a hybrid method that combines both centralized and geo-distributed learning could be more suitable under certain circumstances.

Besides presenting early results in this area, this paper is intended as an open invitation to researchers and practitioners from both systems and ML communities. We foresee the need for substantial advances in theory, algorithms and system design, as well as the engineering of a whole new practice of Geo-Distributed Machine Learning (GDML). To that end, we contributed back all the changes done to Apache Hadoop YARN and Apache REEF as part of this work.

7 Conclusions

Large organizations have a planetary footprint with users scattered in all continents. Latency considerations and regulatory requirements motivate them to build data centers all around the world. From this, a new class of learning problems emerge, where global learning tasks need to be performed on data "born" in geographically disparate data centers, which we call Geo-Distributed Machine Learning (GDML).

To the best of our knowledge, this aspect of Machine Learning has not been studied in great detail before, despite being faced by practitioners on a daily basis.

In this work, we introduce and formalize this problem, and challenge common assumptions and practices. Our empirical results show that a geo-distributed system, combined with communication-parsimonious algorithms, can deliver a substantial reduction in costly and scarce cross data center bandwidth. Further, we speculate distributed solutions are structurally better positioned to cope with the quickly evolving regulatory frameworks.

To conclude, we acknowledge this work is just a first step of a long journey, which will require significant advancements in theory, algorithms and systems.

Acknowledgments

We are grateful to our reviewers and colleagues for their help and comments on earlier versions of this paper. One of the authors was supported by the Argentine Ministry of Science, Technology and Productive Innovation with the program BEC.AR. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors nor the authors' affiliations.

References

[1] Ashish Thusoo et al. Data Warehousing and Analytics Infrastructure at Facebook. SIGMOD, 2010.

[2] George Lee, Jimmy Lin, Chuang Liu, Andrew Lorek, and Dmitriy Ryaboy. The Unified Logging Infrastructure for Data Analytics at Twitter. *PVLDB*, 2012.

[3] Aditya Auradkar et al. Data Infrastructure at LinkedIn. In *ICDE*, 2012.

[4] 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 *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2. USENIX Association, 2012.

[5] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph Hellerstein. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *PVLDB*, 2012.

[6] 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 *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO, October 2014. USENIX Association.

[7] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman. Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 275–288. USENIX Association, 2014.

[8] Ashish Vulimiri, Carlo Curino, P. Brighten Godfrey, Thomas Jungblut, Jitu Padhye, and George Varghese. Global Analytics in the Face of Bandwidth and Regulatory Constraints. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 323–336, Oakland, CA, 2015. USENIX Association.

[9] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. Inter-datacenter Bulk Transfers with Netstitcher. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 74–85. ACM, 2011.

[10] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[11] Martin Rost and Kirsten Bock. Privacy by Design and the New Protection Goals. *DuD, January*, 2011.

[12] European Commission press release. Commission to pursue role as honest broker in future global negotiations on Internet Governance. `http://europa.eu/rapid/press-release_IP-14-142_en.htm`.

[13] Ashish Vulimiri, Carlo Curino, Brighten Godfrey, Konstantinos Karanasos, and George Varghese. WANalytics: Analytics for a Geo-Distributed Data-Intensive World. *CIDR 2015*, January 2015.

[14] Court of Justice of the European Union Press Release No 117/15. The Court of Justice declares that the Commission's US Safe Harbour Decision is invalid, 2015. `http://g8fip1kplyr33r3krz5b97d1.wpengine.netdna-cdn.com/wp-content/uploads/2015/10/schrems-judgment.pdf`. Accessed 2015-10-17.

[15] Dhruv Mahajan, S. Sathiya Keerthi, S. Sundararajan, and Léon Bottou. A Functional Approximation Based Distributed Learning Algorithm. *CoRR*, abs/1310.8418, 2013.

[16] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16. ACM, 2013.

[17] Markus Weimer, Yingda Chen, Byung-Gon Chun, Tyson Condie, Carlo Curino, Chris Douglas, Yunseong Lee, Tony Majestro, Dahlia Malkhi, Sergiy Matusevych, Brandon Myers, Shravan Narayanamurthy, Raghu Ramakrishnan, Sriram Rao, Russel Sears, Beysim Sezgin, and Julia Wang. REEF: Retainable Evaluator Execution Framework. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1343–1355. ACM, 2015.

[18] Michael Kearns. Efficient Noise-tolerant Learning from Statistical Queries. *J. ACM*, 45(6):983–1006, nov 1998.

[19] Cheng tao Chu, Sang K. Kim, Yi an Lin, Yuanyuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y. Ng. Map-Reduce for Machine Learning on Multicore. In B. Schölkopf, J.C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 281–288. MIT Press, 2007.

[20] Vitaly Feldman. A complete characterization of statistical query learning with applications to evolvability. *J. Comput. Syst. Sci.*, 78(5):1444–1459, 2012.

[21] Alekh Agarwal, Oliveier Chapelle, Miroslav Dudík, and John Langford. A Reliable Effective Terascale Linear Learning System. *Journal of Machine Learning Research*, 15:1111–1133, 2014.

[22] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, January 2011.

[23] Caoxie Zhang, Honglak Lee, and Kang G. Shin. Efficient Distributed Linear Classification Algorithms via the Alternating Direction Method of Multipliers. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, April 21-23, 2012*, pages 1398–1406, 2012.

[24] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I. Jordan. Communication-Efficient Distributed Dual Coordinate Ascent. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3068–3076, 2014.

[25] Dhruv Mahajan, Nikunj Agrawal, S. Sathiya Keerthi, S. Sundararajan, and Léon Bottou. An efficient distributed learning algorithm based on effective local functional approximations. *Journal of Machine Learning Research*, 2015.

[26] Jonathan R Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, Pittsburgh, PA, USA, 1994.

[27] Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust Region Newton Method for Logistic Regression. *J. Mach. Learn. Res.*, 9:627–650, 2008.

[28] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308. USENIX Association, 2011.

[29] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)*, pages 351–364, Prague, Czech Republic, 2013.

[30] Apache Hadoop community. Scaling out YARN via Federation. `https://issues.apache.org/jira/browse/YARN-2915`, 2016.

[31] William D. Gropp, Steven Huss-Lederman, Andrew Lumsdaine, and Inc netLibrary. *MPI : The Complete Reference. Vol. 2. , The MPI-2 extensions*. Scientific and engineering computation series. Cambridge, Mass. MIT Press, 1998.

[32] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards Predictable Datacenter Networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 242–253. ACM, 2011.

[33] Criteo Labs. Terabyte Click Logs, 2015. `http://labs.criteo.com/downloads/download-terabyte-click-logs/`. Accessed 2016-02-03.

[34] Kaggle Criteo Labs. Display Advertising Challenge, 2014. `https://www.kaggle.com/c/criteo-display-ad-challenge`. Accessed 2015-09-30.

[35] K.Q. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature Hashing for Large Scale Multitask Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.

[36] Qifan Pu, Ganesh Anantharayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low Latency, Geo-distributed Data Analytics. In *ACM SIGCOMM*, London, UK, 2015.

[37] Arnon Lazerson, Izchak Sharfman, Daniel Keren, Assaf Schuster, Minos Garofalakis, and Vasilis Samoladas. Monitoring Distributed Streams Using Convex Decompositions. *Proc. VLDB Endow.*, 8(5):545–556, January 2015.

[38] Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vassilis Plachouras, and Luca Telloli. On the feasibility of multi-site web search engines. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 425–434. ACM, 2009.

[39] Guillem Francès, Xiao Bai, B. Barla Cambazoglu, and Ricardo Baeza-Yates. Improving the Efficiency of Multi-site Web Search Engines. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 3–12. ACM, 2014.

[40] Berkant Barla Cambazoglu and Ricardo Baeza-Yates. Scalability Challenges in Web Search Engines. pages 27–50, 2011.

[41] E. Kayaaslan, B. B. Cambazoglu, and C. Aykanat. Document replication strategies for geographically distributed web search engines. 2013.

[42] Mu Li, David G. Andersen, Alexander Smola, and Kai Yu. Communication Efficient Distributed Machine Learning with the Parameter Server. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 19–27, Cambridge, MA, USA, 2014. MIT Press.

[43] Ignacio Cano, Markus Weimer, Dhruv Mahajan, Carlo Curino, and Giovanni Matteo Fumarola. Towards Geo-Distributed Machine Learning. In *Learning Systems Workshop at NIPS 2015*, 2015.

[44] Ignacio Cano, Markus Weimer, Dhruv Mahajan, Carlo Curino, and Giovanni Matteo Fumarola. Towards geo-distributed machine learning. *CoRR*, abs/1603.09035, 2016.

[45] Hsieh Kevin, Harlap Aaron, Vijaykumar Nandita, Konomis Dimitris, Ganger Gregory R., Gibbons Phillip B., and Mutlu Onur. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *NSDI*, 2017.

[46] Jakub Konecný, Brendan McMahan, and Daniel Ramage. Federated Optimization: Distributed Optimization Beyond the Datacenter. *CoRR*, abs/1511.03575, 2015.

[47] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated Learning: Strategies for Improving Communication Efficiency. *CoRR*, abs/1610.05492, 2016.

[48] Jakub Konecný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *CoRR*, abs/1610.02527, 2016.

[49] Yuchen Zhang, John C. Duchi, and Martin J. Wainwright. Communication-efficient algorithms for statistical optimization. *J. Mach. Learn. Res.*, 14(1):3321–3363, January 2013.

[50] McMahan H. Brendan, Moore Eider, Ramage Daniel, Hampson Seth, and Agüera y Arcas Blaise. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[51] Maria-Florina Balcan, Avrim Blum, Shai Fine, and Yishay Mansour. Distributed Learning, Communication Complexity and Privacy. *Journal of Machine Learning Research*, 2012.

[52] A. Bar-Or, D. Keren, A. Schuster, and R. Wolff. Hierarchical Decision Tree Induction in Distributed Genomic Databases. *IEEE Transactions on Knowledge and Data Engineering – Special Issue on Mining Biological Data*, 17(8), August 2005.

[53] Ananda Theertha Suresh, Felix X. Yu, H. Brendan McMahan, and Sanjiv Kumar. Distributed Mean Estimation with Limited Communication. *CoRR*, abs/1611.00429, 2016.

[54] Shravan Narayanamurthy, Markus Weimer, Dhruv Mahajan, Sundararajan Sellamanickam, Tyson Condie, and Keerthi Selvaraj. Towards Resource-Elastic Machine Learning. In *NIPS 2013, Workshop on Big Learning*, NIPS 2013, pages 1–6, 2013.