通过减少冗余通信，为大规模模型提供高效的分布式机器学习

Efficient Distributed Machine Learning for Large-scale Models by Reducing Redundant Communication

分布式机器学习用于在适当的时间内训练大型模型。为了加速训练，所有节点必须频繁地交换计算结果。但是，更新参数的通信开销很大，影响了总执行时间。在这篇论文中，提出了一种在不改变算法语义的情况下减少参数冗余传输的通信方法。在训练之前，我们确定可以省略的集体通信的部分，并用请求中间结果的节点之间的直接通信来替换它们。我们实现了这个算法，并在一个由5个节点组成的集群上对其进行了评估。通过对实际数据集的评价表明，该方法减少了节点间交换的元素数量，缩短了通信时间。

Harumichi Yokoyama and Takuya Araki
NEC Corporation
Email: h-yokoyama@jf.jp.nec.com, t-araki@dc.jp.nec.com

***Abstract*—Distributed machine learning is used to train large-scale models within a moderate amount of time. To accelerate the training, all nodes have to exchange calculation results frequently. However, the communication of the updated parameters is a large overhead affecting the total execution time. This paper proposes a communication method to decrease redundant transmissions of parameters without changing the semantics of the algorithm. Before the training, we identify the parts of the collective communication that can be omitted and replace them with direct communication between the nodes requesting the intermediate results. We implemented this algorithm and evaluated it on a cluster of five nodes connected with 10-Gbps Ethernet. The evaluation using a real dataset showed that our method reduced the number of elements exchanged between nodes and shortened the communication time.**

I. Introduction

Many organizations analyze big data with the aim of extracting knowledge from it to improve the quality of their services. Machine learning, which is a basic tool of data analysis, aims to build models based on the past data in order to predict answers in uncertain situations. In the last several years, the number of dimensions of these models has increased considerably. In particular, some of these prediction tasks, including document analysis, item recommendation, and click-through rate prediction, use models with a huge number of features in the form of, e.g., the vocabulary of a language or URLs on the web. Building large-scale models imposes a heavy load on both the CPU and memory, and hence, it is difficult to complete such tasks (called jobs) with a single computing node in a reasonable time.用单个节点来完成是有很大的负担的

分布式计算是缩短大规模数据分析执行时间的一种很有前途的方法。要分析的整个数据被分割并分配给单独的机器(分布式计算网络的节点)。这将分配计算统计数据的负载。机器学习作业主要由顺序操作组成，所以每台机器(称为工作节点)在进行下一次计算之前必须通过网络交换中间结果。这些反复的计算和通信是使模型收敛到最佳状态的关键。

Distributed computing is a promising way to shorten the execution time for such large-scale data analyses. The whole data to be analyzed is divided up and assigned to individual machines (nodes of the distributed computing network). This distributes the load of calculating the statistics. Machine learning jobs consist mainly of sequential operations, so the individual machines (called workers) must exchange intermediate results through the network before they proceed to the next calculation. These repeated computations and communications are essential to make the model converge to the optimal one.

在多个节点之间分配机器学习任务时，通信常常成为性能瓶颈。

Communication often becomes a performance bottleneck when distributing machine learning jobs among multiple nodes. In a major optimization method in machine learning, the nodes iterate a mathematical calculation many times by fetching small subsets of the whole data (these subsets are called *mini-batches*). However, frequent calculations tend to make the communication time increase relative to the entire execution time. The overhead of communication is heavy particularly when the model has a large number of dimensions. Although multiple processes running in a single machine can exchange their results quickly via local memory, distributed nodes have difficulty in sharing local updates because of the limited bandwidth of the network. This is why we were motivated to develop a strategy to efficiently communicate the intermediate results with separate computing nodes.

然而，频繁的计算往往会使通信时间相对于整个执行时间有所增加。通信开销非常大，尤其是当模型具有大量的维度时。虽然在一台机器上运行的多个进程可以通过本地内存快速交换结果，但由于网络带宽有限，分布式节点在共享本地更新方面存在困难。这就是为什么我们要开发一种策略来有效地将中间结果与独立的计算节点进行通信。

There are many studies on scaling out machine learning to multiple nodes. Typically, they adopt a certain communication method: computing nodes send the locally updated parameters to a management node, and the management node then aggregates the received values and pushes the necessary parts of them to each of the computing nodes. This method is straightforward and suitable for most cases. However, it sometimes produces redundant, unnecessary communications between nodes. This situation occurs when only single node updates specific parameters. In this case, the computing nodes do not need to send the values via the management nodes and can simply deliver them directly to the requesting nodes instead.

通常，它们采用某种通信方法:计算节点将本地更新的参数发送给管理节点，然后管理节点聚合接收到的值，并将它们的必要部分推送到每个计算节点。这种方法简单明了，适用于大多数情况。然而，它有时会在节点之间产生冗余的、不必要的通信。这种情况发生在只有单个节点更新特定参数时。在这种情况下，计算节点不需要通过管理节点发送值，而可以直接将值发送到请求节点。

Redundant communications often appear when dealing with large-scale, sparse data. Sparseness means that the features representing the data are enormous, but most of their components are empty. The individual computing nodes fetch a mini-batch from the local data, and they update sets of parameters that appear in each mini-batch. The content of the updated parameters depends on each node, and many of the parameters are likely to be updated only by a single node. Consequently, training for sparse models tends to involve a large amount of the unnecessary communication.

在处理大规模、稀疏的数据时，经常会出现冗余通信。稀疏意味着表示数据的特性是巨大的，但是它们的大多数组件是空的。单个计算节点从本地数据获取一个小批处理，它们更新出现在每个小批处理中的参数集。更新参数的内容取决于每个节点，许多参数可能只由一个节点更新。因此，稀疏模型的训练往往涉及大量不必要的通信。

In this paper, we propose a new, efficient communication scheme for avoiding redundant transmissions in distributed machine learning. The first step of our method is to divide up the data into distributed mini-batches and to extract the parameters updated with each mini-batch. The second step is to identify the redundant part of the collective communications via the management nodes. The third step is to replace the redundant communications with ones directly between the nodes that demand each other ’ s content. We evaluated our

我们的方法的第一步是将数据划分为分布式的小批，提取每个小批更新的参数。第二步是通过管理节点识别集体通信的冗余部分。第三步是用的替换冗余通信节点之间直接相互需求的内容

algorithm on five node clusters connected with 10 Gbps Ethernet. The evaluation using a real dataset showed that the algorithm reduced the total number of elements exchanged between nodes and shortened the communication time.

II. Background

A. Basis of Machine Learning

机器学习是一种揭示数据背后的模式并预测输入输出的技术。

Machine learning is a technique to reveal patterns behind data and predict the output from the input. It does so by finding a function $\hat{y}(\boldsymbol{x})$ that correctly maps $\boldsymbol{x}$ to the given output y. In practice, $\hat{y}$ is assumed to take a specific form that is parametrized by a set of variables. That is, we use a function $\hat{y}(\boldsymbol{x};\boldsymbol{w})$ of the model parameters $\boldsymbol{w}$.

The proper parameters $\boldsymbol{w}$ are selected by minimizing an objective function that is calculated from the data. The objective function includes an empirical loss $L(\hat{y}(\boldsymbol{x};\boldsymbol{w}),y)$, which represents the difference between the predicted output and the desired one,

$$V(\boldsymbol{w}) = \frac{1}{N}\sum_{i\in S} L(\hat{y}(\boldsymbol{x}^{(i)};\boldsymbol{w}),\, y^{(i)})$$

Here, $(\boldsymbol{x}^{(i)}, y^{(i)})$ is the i-th pair of the input and the output in a set of data samples S, and the number of pairs in S is N.

An iterative algorithm called stochastic gradient descent (SGD) is frequently used for finding the parameters minimizing the objective function. In SGD, we take the gradient of the objective function that is approximately calculated using a subset of the whole data,

$$V_m(\boldsymbol{w}) = \frac{1}{N_m}\sum_{i\in S_m} L(\hat{y}(\boldsymbol{x}^{(i)};\boldsymbol{w}),\, y^{(i)})$$

where S_m is a subset of the data samples, which is called a *mini-batch*, and whose elements number N_m. Then, we subtract the gradient from $\boldsymbol{w}$ while multiplyingwith learning rate α_t,

$$w_j \leftarrow w_j - \alpha_t \frac{\partial}{\partial w_j} V_m(\boldsymbol{w})$$

α_t decreases with the iteration time t. We repeat this calculation each time we select a different mini-batch. By iterating the calculation, the parameters get closer to the minimum and eventually reach it.

The size of a mini-batch affects the number of updates required for convergence [1]. Regarding SGD for convex objective functions [2], the convergence rate per iteration using one sample is $\mathcal{O}(1/\sqrt{T})$, while the rate when using mini-batches of size b is $\mathcal{O}(1/\sqrt{bT}+1/T)$. Using mini-batches improves the rate by $\sqrt{b}$, yet the total number of samples processed is b times larger. Hence, in practice, the convergence becomes faster as the size of the mini-batch decreases.

B. Distributed Training

分布式处理是解决具有大量数据的繁重训练任务的有效方法。

Distributed processing is an effective solution for heavy training tasks with large amounts of data. The training data is divided up and placed on separate nodes connected to a network, and each node calculates the statistical values

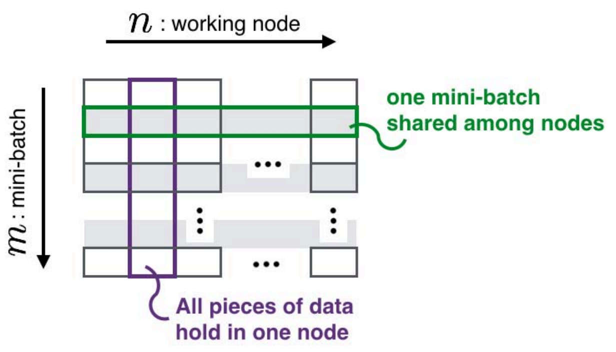
n : working node
m : mini-batch
one mini-batch shared among nodes
All pieces of data hold in one node

Fig. 1. Sketch of partitioning the bulk data into distributed mini-batches

based on the local shard. Distributed schemes of machine learning are widely studied [3]–[5]. Among them, the basic algorithm, called the bulk synchronous parallel method [6], is a straightforward application of mini-batch SGD in distributed environments.

Before explaining the parallelizing method, we show the typical way of changing the original data into mini-batches used in each iteration on local nodes. First, the whole data is partitioned into a collection of mini-batches. Then, each mini-batch is further divided up and distributed among the working nodes. Fig.1 sketches the idea. Each node calculates the gradient from the part of the mini-batch it has.

In the bulk synchronous parallel method, a group of working nodes iteratively updates the parameters by using mini-batches in a synchronous way. The details are shown in Algorithm 1, in which M denotes the number of mini-batches and N the number of nodes, as in Fig.1. One mini-batch is divided up into parts, and these parts are sent to all the nodes. The nodes then calculate the gradients of these parts in parallel. The nodes wait for the other nodes to finish the calculation. After that, they gather the calculated gradients and globally update the shared parameters.

节点等待其他节点完成计算。

The communication cost for mini-batch training is potentially large in a distributed system. As we mentioned in the previous subsection, making the mini-batch small speeds up convergence in SGD. Therefore, repeated updates with tiny mini-batches may substantially reduce the total time of training. This is true in the case of a single node. However, frequent synchronization of the computed values increases the communication overhead relative to the computation time. The bandwidth and the latency of networks such as Ethernet are several orders of magnitude worse than that of the main memory, so the issue now becomes how to reduce the overhead.

因此，小批量的重复更新可以大大减少训练的总时间。

C. Sparse Features

Some data mining tasks exploit datasets with a huge number of dimensions. An example is document analysis, which deals with a text composed of multiple words. We make a dictionary that translates all possible words into unique integer values, which are called features. In addition, we encode each text in

一些数据挖掘任务利用具有大量维数的数据集。文档分析就是一个例子，它处理由多个单词组成的文本。我们制作了一个字典，将所有可能的单词转换成唯一的整数值，称为特征。

Algorithm 1 Bulk synchronous parallel algorithm

```
1: for $t = 1$ to the fixed times do
2:     for batch $m = 1$ to $M$ do
3:         for node $n = 1$ to $N$ do                ▷ do in parallel
4:             $g_m^{(n)} \leftarrow \nabla_w V_m^{(n)}(w)$
5:         end for
6:         $w \leftarrow w - \alpha_t \sum_n g_m^{(n)}$          ▷ gather gradients
7:     end for
8: end for
```

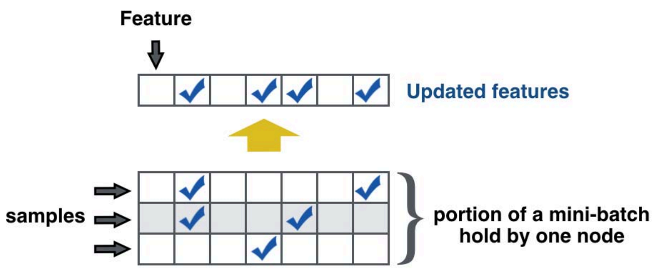

Fig. 2. Updated feature dimensions by using a portion of a mini-batch at one node.

文本中出现的单词与整个词汇表相比非常少，因此向量中有大量的零或零元素，如(0,0，···，0,1,0，···，0);我们称之为稀疏特征.

the data into a collection of integer indices and represent the text as a vector of large dimension. Words that appears in a text are extremely few compared with the whole vocabulary, so the vector has a huge number of zero or null elements, like $(0, 0, \cdots, 0, 1, 0, \cdots, 0)$; we call such elements sparse features.

用大规模数据训练的模型通常具有相当数量的参数。特别是，一个称为线性模型的流行模型计算数据样本x和参数w之间的内积的输出。

A model trained with large-scale data usually has a comparable number of parameters. In particular, a popular model named the *linear model* calculates the output from the inner product between the data sample $\boldsymbol{x}$ and the parameters $\boldsymbol{w}$.

$$\boldsymbol{w} \cdot \boldsymbol{x} = w_0 x_0 + w_1 x_1 + \cdots + w_d x_d$$

参数w的维数与x相同。然而，并不是所有的元素在每次更新SGD时都使用一个小批更新。需要更新的参数仅限于一个小批的参数，如图2所示。

The parameters $\boldsymbol{w}$ have the same number of dimensions as $\boldsymbol{x}$. However, not all elements in $\boldsymbol{w}$ are changed in each update of SGD using a mini-batch. The parameters to be updated are limited to those in a mini-batch, as shown in Fig.2.

在分布式设置中，每个计算节点在特定的迷你批处理中更新的特性是不同的。这是因为正如上面提到的，每个节点都有自己独立的小批处理部分。节点只需要交换改变的参数的值。使用稀疏数据的分布式训练使得通信模式比使用密集数据的训练更加复杂。

In a distributed setting, the features updated in a specific mini-batch are different for each computing node. This is because each node has its own separate portion of the mini-batch, as mentioned above. The nodes are required to exchange only the values of the parameters that changed. Distributed training with sparse data makes the pattern of communication more complicated than training with dense data.

D. Parameter Server Architecture

有许多框架可以将机器学习扩展到分布式计算节点，例如，Project Adam，Petuum,Tensorflow，MXNet。大多数框架都采用参数服务器体系结构的作者对此进行了系统的讨论。下面我们将简要介绍参数服务器体系结构

There are many frameworks to scale out machine learning to distributed computing nodes, e.g., Project Adam [7], Petuum [5], Tensorflow [8], and MXNet [9]. Most frameworks adopt the *parameter server* architecture, which is systematically discussed by the authors of [10]. We will briefly explain the parameter server architecture below.

The parameter server architecture has two kinds of nodes; *worker* nodes that compute statistical values based on the data, and a *server* that manages the global parameters. The workers compute statistics such as the gradients by using the local shard of the data. They push the calculated gradients to the server, and the server aggregates the gathered values and updates the parameters it manages. The server function can be realized on multiple nodes; at that time, each node maintains part of the whole parameters. The server responds to the workers' requests and gives the latest parameters to the workers before they perform the next gradient computation. Each worker talks only to the server, not to other workers.

工作节点使用数据的局部碎片来计算统计数据，比如梯度。它们将计算出的梯度推入服务器，服务器聚合收集到的值并更新它管理的参数。服务器功能可在多个节点上实现;此时，每个节点维护整个参数的一部分。服务器响应工作节点的要求，给工作节点最新的参数。

The parameter server architecture supports communication of sparse vectors. The sparse vectors are transmitted using the key-values structure.

参数服务器架构支持稀疏向量通信。利用键值结构传递稀疏向量。

III. Related Work

The authors of the parameter server architecture [10]–[12] introduced several techniques to reduce the traffic involved in transmitting sparse vectors. The server uses filters that let pass only the significantly modified parameters. Because of the large number of dimensions of the model, there are many features whose values rarely change in one computation. Filtering out the parameters with little differences results in a large decrease in total traffic. However, this filtering makes the model's convergence worse, and the degree of degradation can depend on the training data or the model. Our method does not change the semantics from that of an algorithm executed in an single process, so it is a more general way to make communications efficient.

服务器使用过滤器，只允许传递经过显著修改的参数。由于模型的维数很大，在一次计算中有很多特征的值很少改变。过滤掉几乎没有差别的参数会导致总通信量的大幅减少。然而，这种滤波使得模型的收敛性更差，退化程度可能取决于训练数据或模型。我们的方法不会改变在单个进程中执行的算法的语义，因此它是一种更通用的通信效率方法。

Some parameter server systems support asynchronous communication [5], [8], [13]. This means that a worker pushes and pulls the parameters to the server without waiting for the other workers to finish a single computation. This reduces the communication overhead when there are load imbalances among the workers. Nonetheless, a recent study [14] showed that asynchronous communication impaired the training of the model; there was a discrepancy between the parameters used to compute gradients and the ones the server updated. Thus, to ensure that the training would not be impaired, we chose synchronous communication for our system.

当工作人员之间存在负载不平衡时，这就减少了通信开销。（但是现在基本上都是负载不平衡的）然而，[14]最近的一项研究表明，异步通信损害了模型的训练，用于计算梯度的参数与服务器更新的参数之间存在差异。因此，为了确保培训不会受到影响，我们选择了系统的同步通信。（所以在本论文中采用的是同步通信）

Collective communication has been extensively studied and optimized to reduce the communication cost. The optimization is reflected in MPI software such as OpenMPI [15] and MPICH [16]. Furthermore, an approach for sparse vectors has been proposed in Kylix [17]. This approach combines different kinds of collective communication, i.e., pairwise exchange and recursive trees, and it adjusts their combination according to the properties of the sparse data. Our approach is to replace the redundant collective communications by the less number of direct communications. This method is orthogonal to Kylix and not competing, so they can be applied at the same time.

此外，本文还提出了一种求解Kylix中稀疏向量的方法。这种方法结合了不同类型的集体交流，即，并根据稀疏数据的性质，调整它们的组合。我们的方法是用较少的直接通信来取代冗余的集体通信。这种方法与Kylix正交，不竞争，所以它们可以同时应用。

IV. Proposed Algorithm

A. Key Observation

A characteristic pattern of communication is observed in distributed machine learning systems using the bulk synchronous parallel method in mini-batch SGD. After distributed

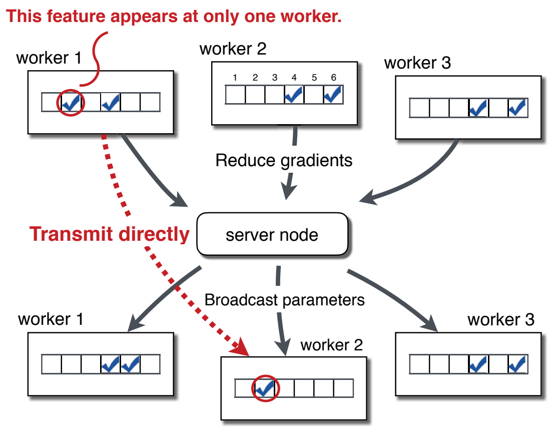

Fig. 3. Illustration of inefficient communication in the parameter server architecture and alternative more efficient operation

nodes calculate the statistics such as gradients, they exchange local values to update the global shared parameters. In the case of the parameter server architecture, the server aggregates and reduces the gradients sent by the workers. Then it broadcasts the updated parameters to the workers that need them in the next calculation. Here, the system requires two collective communications, reducing and broadcasting, in each computation round with a mini-batch.

在参数服务器架构的情况下，服务器聚合并减少工人发送的梯度。

In the communication pattern explained above, there is room for optimization of the throughput. There are unnecessary transmissions between the server and the workers when training with sparse data and the associated model. In the parameter server system, all nodes send their values to the server and the server then sends updated parameters to the other nodes. However, as the parameters updated at only one computing node at a time, transmitting them via the server is redundant and inefficient; because it is enough for the relevant computing nodes to exchange the parameters directly.. Fig.3 is a diagram showing the node-server-node communication and the more efficient process. By modifying the communication paths properly, the amount of transmitted data can be reduced by as much as half. This direct communication method is expected to reduce the total amount of the communication.

在上面解释的通信模式中，有优化吞吐量的空间。在使用稀疏数据和相关模型进行培训时，服务器和工作节点之间存在不必要的传输。

但是，由于每次只更新一个计算节点上的参数，因此通过服务器传输这些参数是冗余和低效的。因为相关计算节点直接交换参数就足够了。

通过适当地修改通信路径，传输的数据量可以减少一半。（但是怎样知道什么时候能直接交换？）

Inefficient communications occur more frequently as the training data gets sparser. In sparse data, most of the features rarely appear in a mini-batch, and even when they appear, there is a high probability that only one node has the partition containing the features and the other nodes do not. This tendency is especially strong when the mini-batches are small. Collective communications for rare features can be replaced by direct communications in most cases.

由于训练数据越来越少，效率低下的通信更加频繁。在稀疏数据中，大多数特性很少出现在小型批处理中，即使它们出现，也很可能只有一个节点具有包含这些特性的分区，而其他节点没有。

B. Notation and Definitions

In order to explain the algorithm clearly, we give the notation of variables in advance. We assume the system consists of N computing nodes, each of which has pieces of the mini-batches. All the nodes are assumed to have the same number

TABLE I
DEFINITIONS OF SYMBOLS

N	# of computing nodes.
M	# of mini-batches in a node (the maximum of rounds).
$D_{n,m}$	a set of unique features updated with worker n, round m.
$F^R_{n,m}$	a set of features reduced from worker n, after computing at round m.
$F^B_{n,m}$	a set of features broadcasted to worker n, before computing at round m.

of the pieces. The period when all nodes are computing is called a *round*. Each piece of data in a node is utilized for the computation in each round. The total number of rounds is M, which is same as the number of mini-batches. We extract the set of unique features updated at node n, round m, from the part of the mini-batch; we denote the set as $D_{n,m}$.

所有节点都在计算的周期称为一轮

Regarding the features communicated between the server and the workers, the server broadcasts to each node the parameters that are required in round m. The parameters that node n gets in round m are denoted as $F^B_{n,m}$. In a similar fashion, $F^R_{n,m}$ is the set of parameters from node n that the server reduces. Generally speaking, $F^B_{n,m}$ and $F^R_{n,m}$ may not be the same. Table I summarizes the definitions explained above.

C. Identification of Redundant Transmissions

Some of the transmissions between the server and the workers are redundant. Here, we explain the procedure to identify them. The procedure is executed once at the beginning of the training. Therefore, this identification is performed before the data is distributed to the nodes.

First, we identify the features that can be transmitted directly between the workers. Such a feature is updated at only one node during a given round. These features can be identified by inspecting the features that were updated across the nodes. We iterate the search for these features and the nodes where they are updated over the rounds (Algorithm 2).

首先，我们确定可以在工作人员之间直接传输的特性。这样的特性在给定的一轮中只在一个节点上更新。通过检查跨节点更新的特性，可以识别这些特性。我们遍历搜索这些特性和节点，这些特性和节点是在轮询期间更新的。

Algorithm 2 Identification of features transmittable directly

Input: $D_{n,m}$, features appeared in node n, m-th round
Output: S, set of (feature, node, round)

```
1: $S \leftarrow \emptyset$
2: for $m = 1$ to $M$ do
3:     extract features updated at only one node from $D_{\cdot,m}$ (set to $F$)
4:     if $f$ in $F$ then
5:         assign $n$ to the node updating $f$
6:         $S \leftarrow S \cup \{(f, n, m)\}$
7:     end if
8: end for
9: return $S$
```

怎样指定直接通信？

After specifying all of the features that are directly transmittable, we have to discover the related communications that can be replaced, as described in Algorithm 3. Here, we should look at the rounds before and after the feature was updated (we call these rounds neighbor rounds). The communication of a feature that is directly transmittable in a neighbor round is redundant. Note that the neighbor rounds are searched for either in the forward or backward time direction. This means that, when a feature f is updated at node n on round m, its forward (backward) neighbor is the nearest round when any of the nodes deals with feature f after (before) round m. Algorithm 3 is for all features that were identified by Algorithm 2.

Algorithm 3 Search of neighbor rounds

```
Input: target feature $f$ appearing at node $n$, round $m$
Forward:
1: $i \leftarrow m$
2: repeat
3:     $i \leftarrow i + 1$
4:     if $f$ appears in $D_{n,m}$ at any $n$ then
5:         return round $i$, all nodes updating $f$
6:     end if
7: until arrive the last round $(i = M)$
Backward:
1: $i \leftarrow m$
2: repeat
3:     $i \leftarrow i - 1$
4:     if $f$ appears in $D_{n,m}$ at any $n$ then
5:         return round $i$, all nodes updating $f$
6:     end if
7: until arrive the first round $(i = 0)$
```

D. Replacement with Direct Communication between Workers

In the previous subsection, we identified the features updated only by a single node in one round and the neighbor ones. If we use the existing communication method, there are four kinds of transmission related to the features that can be directly transmitted. Fig.4 depicts the communication for one of these features. First, in the preceding neighbor round, the nodes treating the same feature send the computed values to the server (This is illustrated by "reduce" at the top in Fig.4). Second, in the current round, the server sends the values to the node that needs them, and the node returns the gradient to the server after computing it. Finally, in the subsequent neighbor round, the server sends the values to the nodes that need them (shown as "broadcast" at the bottom of the figure). These send operations are redundant and can be omitted.

We replace the redundant transmission with a direct one between workers. The situation in which we introduce the direct communication is shown in Fig.5. We define the variable $F^D(n_s, n_d, m)$ to be a set of parameters transmitted directly from node n_s to n_d at round m. Here, we denote the feature that can be replaced and the node, and round related to it as f_0, n_0, and m_0, and the backward and forward neighbor rounds as m'_0 and m''_0, respectively. The neighbor rounds are identified

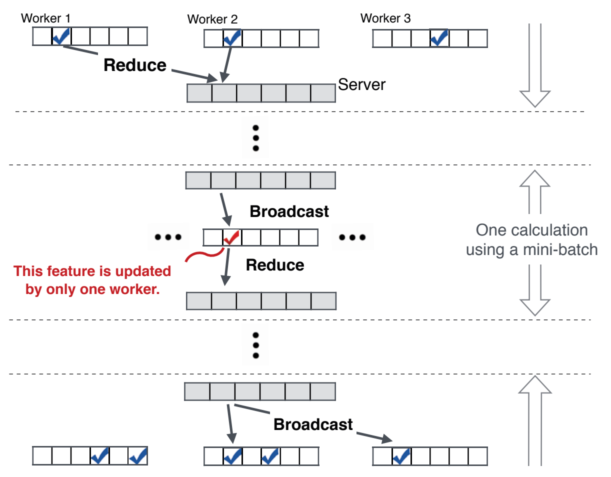
Worker 1
Worker 2
Worker 3
Reduce
Server
Broadcast
Reduce
This feature is updated by only one worker.
One calculation using a mini-batch
Broadcast

Fig. 4. Schematic diagram of redundant communication of features in the existing method. The red check mark indicates the redundantly transmitted feature in a particular round. Blue check marks mean the features updated by each worker. 红色复选标记表示在特定的一轮中冗余传输特征。蓝色检查标记表示每个工人更新的功能。

by using Algorithm 3. As shown in Fig.4 f_0 belongs to the set of reduced parameters F^R_{n,m'_0} such that node n updates f_0. We remove the feature f_0 from F^R_{n,m'_0}. In the same way, the feature f_0 in included in F^B_{n,m_0}, F^R_{n,m_0} and F^B_{n,m''_0}. We remove f_0 from these sets as well.

After removing the target features from the sets for collective communication, we add them to the sets for direct communication. f_0 is transmitted to the relevant working nodes just after the calculating of the gradient. This means that we push f_0 to $F^D(n, n_0, m'_0)$ such that node n updates f_0 and round m'_0. As well, we push f_0 to $F^D(n_0, n, m_0)$ such that node n updates f_0 and round m_0.

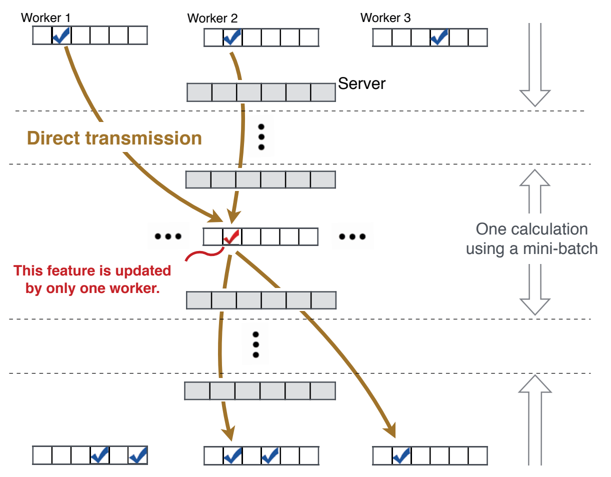
Worker 1
Worker 2
Worker 3
Server
Direct transmission
This feature is updated by only one worker.
One calculation using a mini-batch

Fig. 5. Schematic diagram of communication for directly transmittable feature in the proposed method. The redundant messaging is replaced with direct ones shown by the brown arrows.

V. Implementation

Here, we explain the implementation of the distributed training system with the direct communication scheme. The system uses a symmetric configuration, which is different from the parameter server architecture. The message passing interface (MPI) is used for highly efficient communication among collectively working nodes. In addition, the traffic volume is reduced by omitting the keys of the sparse vectors, which will be explained later.

A. Symmetric Node Composition

The system is composed of symmetric nodes for load balancing. Physical machines have both the worker function and the server function. This implementation is different from what was originally intended in the parameter server architecture. The good aspect of the symmetric composition is that it allows us to balance the communication load and thereby effectively utilize the resources. In particular, as the communications increase, the parameter server architecture has more trouble transferring the parameters.

We use MPI [18] to enable the nodes to communicate efficiently with each other. MPI is the standard communication API for parallel processing. MPI is favorable for the symmetric system because collective communications like reduce, broadcast, alltoall are well optimized in its major implementations, such as OpenMPI [15] and MPICH [16]. The communications between nodes are synchronous. The nodes communicate synchronously after their calculations are finished. We assume that there is little disturbance brought to homogeneous machines when we scale machine learning jobs.

Sparse data is represented by matrices in the compressed sparse row (CSR) format. The CSR format contains a list of values and column indices with the same length and a list of numbers called the offset, which shows where to place the row break. CSR format works well when matrix-to-vector multiplications frequently appear in machine learning computations.

The original data are divided into partitions of the same size so that the load of computation is equally balanced. The original matrix is split in the row direction of a CSR matrix such that the separate partitions have almost the same amount of non-zero elements. This is done when the original data is split into mini-batches and divided into local partitionss. The number of non-zero elements varies with each row, so the local data are divided into mini-batches with different row lengths. In this way, we can carry out a mini-batch computation with nearly the same number of operations on all nodes.

It also matters to determine the ranges of the reduced parameters assigned to each node. Each node manages a specific part of the parameters reduced from all nodes after computation. If the assignment is biased to certain nodes, the communication may be unbalanced, resulting in a longer training time. To avoid the biased configuration, we adjust the ranges according to the number of the features appearing in the mini-batches. We identify the unique features in a batch ($D_{n,m}$ in Table I) and count the number of mini-batches in which each feature appears. We partition an array of parameters into as many sections as nodes, in order that the sums of the counts are nearly equal to each other.

B. All-to-All Communication with Omission of Keys

In symmetric machine configuration, after computing the gradients using a local mini-batch, all nodes send the values that are separated by the predefined ranges to the corresponding node. This operation is implemented by the *MPI-Alltoallv* function in MPI. Only the parts of the whole features are treated in each mini-batch, so the size of the sent data varies from round to round. After reducing the gradients, the server aggregates them to make the latest parameters. Then, the server broadcasts the set of parameters to the worker nodes that need them in the round that needs them.

We developed a communication method that omits the keys by monitoring the mini-batches. The values to be sent online are not filtered. Thus, the contents of the transmissions are completely identified before the training. At the given round, each node has the keys of the features it is dealing with. The array of keys is split on the basis of the range defined before, and all pieces of the split arrays are sent to all nodes once at the beginning of the training. By referring to this information, all nodes are able to identify the features from the received values without the corresponding keys. Therefore, the nodes exchange only the vector of values every after batch computation, which leads to an improvement in the throughput.

VI. Evaluation

A. Experimental Setup

The distributed system we evaluated consisted of five computing nodes, each of which had two processor sockets of Intel Xeon CPU E2-2630. Each socket had 64GB of memory attached. The MPI processes were launched at each processor, so the training consisted of ten processes. The nodes were connected by 10 Gbps Ethernet.

B. Model and Dataset

As a specific model of machine learning, we chose factorization machine [19] for this evaluation. Factorization machine is often used for tasks such as click-through rate item prediction or recommendation. The characteristic is to estimate the interaction between features by using factorized parameters within a reasonable amount of time. The combination of features seems to cost the square of the time to compute the statistical values, but it actually requires only linear time; this can be understood by doing the simple arithmetic shown in the above mentioned literature.

The detail of factorization machine is explained here. The A model equation of degree 2 is defined as

$$\hat{y}(\boldsymbol{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=i+1}^{d} \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle x_i x_j,$$

where $\boldsymbol{x}$ is a sample of data, w_i the coefficient of the linear term and bias, and $\boldsymbol{v_i}$ the factorized parameter of dimension k, The dot product $\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$ means

$$\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f}$$

The product $\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$ models the interaction between the i-th and j-th features. In binary classification, the empirical loss for the label y and the prediction $\hat{y}(\boldsymbol{x})$ is calculated as $l(y, \hat{y}(\boldsymbol{x})) = -\ln \sigma(y \cdot \hat{y}(\boldsymbol{x}))$, where the sigmoid function $\sigma(x)$ is defined as $\sigma(x) = 1/(1 + e^{-x})$.

We performed evaluations on three datasets that reveal high sparsity: news20 [20]: a collection of news documents categorized into twenty different news topics, a URL dataset [21]: information about malicious URL (spam, phishing, exploits, and so on) and criteo: click-through rate logs of advertisements displayed on the web.

C. Number of Transmitted Features

We confirm that the proposed communication method can decrease the number of transmitted elements. Table II shows that the number of elements for reduce communication (or broadcast communication, same) and that of direct transmission when we apply the proposed method to the system. When we set the former number to n_m, and the latter number to n_d, total number of parameters transmitted is $2n_r + 2n_d$ in the existing method, $2n_r + n_d$ in the proposed method. In the proposed method, the number is less by n_d, and the ratio of two numbers are put on the leftmost column in Table. II.

We see that the traffic decreased more as the size of mini-batch gets smaller. This is because smaller size of mini-batches have less overlap of features updated at the same round. Therefore, the effectiveness of the proposed method is limited when the size of the mini-batch is large. It is not a problem because our targeted situation is that we iterate short computation using mini-batch with small size.

D. Decrease in Total Training Time

We evaluated the decrease in total execution time when applying the proposed method. Table III display the breakdown to total execution times in one epoch, both for the existing method and the proposed method. For each method, the total time are decomposed into the reduce plus direct time, broadcast time, and time for computing gradients. We could confirm that in all situations, the total execution time have decreased. Fig.6 shows the result of the most improved situation of news20 dataset with batch size 1000.

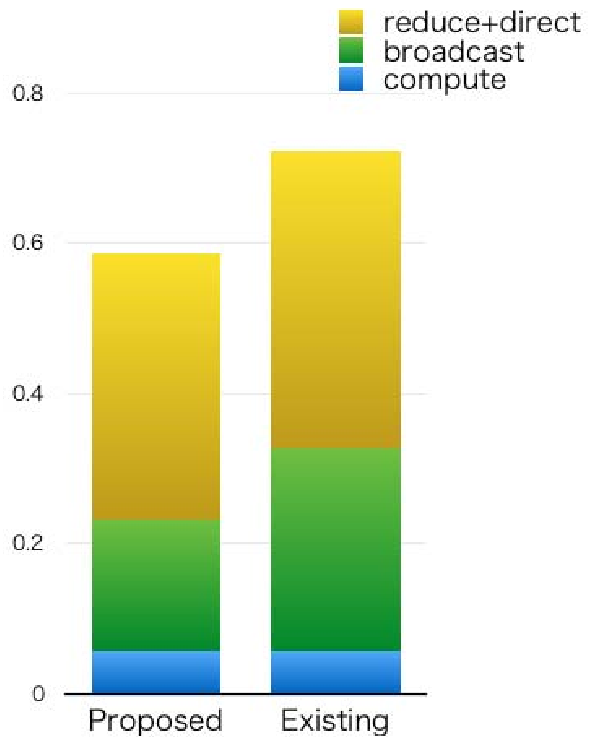

Fig. 6. Improvement in execution time for news20 with batch size 1000. The yellow part represent the time spent for reduce and direct communication (component of direct one is nothing in existing method).

VII. Conclusion

We propose an efficient communication method for distributed machine learning. When we execute machine learning jobs in separate nodes, communication often becomes the large bottleneck in the total processing time. Our method can decrease redundant transmission of parameters without approximating the algorithm. The key procedure is to identify a part of collective communication that can be omitted, and replace them with direct communication between the two computing nodes. The evaluation on 10 computing clusters revealed that the traffic have been reduced, so we confirm that our method is effective.

References

[1] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 661–670.

[2] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu, "Sample size selection in optimization methods for machine learning," *Mathematical programming*, vol. 134, no. 1, pp. 127–155, 2012.

[3] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 69–77.

[4] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011, pp. 693–701.

[5] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[6] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin, "Parallel coordinate descent for l1-regularized loss minimization," *arXiv preprint arXiv:1105.5379*, 2011.

[7] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system." in *OSDI*, vol. 14, 2014, pp. 571–582.

TABLE II
NUMBER OF TRANSMITTED FEATURES

Dataset	Batch size	# Reduce/Broadcast elements	# Directly transmitted elements	Decreasing rate
news20	500	3.14×10^6	2.68×10^6	0.780
	1000	3.33×10^6	2.01×10^6	0.812
	2000	3.18×10^6	1.39×10^6	0.850
url	5000	1.77×10^7	7.97×10^6	0.845
	10000	1.50×10^7	6.35×10^6	0.851
	30000	1.18×10^7	4.41×10^6	0.864
criteo	1000	1.09×10^7	5.40×10^6	0.834
	3000	9.35×10^6	3.43×10^6	0.865
	10000	7.67×10^6	1.87×10^6	0.902

TABLE III
BREAKDOWN OF THE TOTAL EXECUTION TIME IN A EPOCH

Dataset	Batch size	Proposed method [sec]			Existing method [sec]			Decreasing Rate
		Reduce+Direct	Broadcast	Compute	Reduce+Direct	Broadcast	Compute	
news20	500	0.517	0.210	0.0575	0.536	0.312	0.055	**0.857**
	1000	0.356	0.174	0.0569	0.397	0.269	0.0567	**0.795**
	2000	0.316	0.156	0.056	0.342	0.232	0.057	**0.823**
url	5000	4.02	1.15	1.15	4.46	1.42	1.14	**0.880**
	10000	2.44	0.98	1.14	2.68	1.20	1.12	**0.879**
	30000	1.24	0.54	1.11	1.12	0.73	1.10	**0.961**
criteo	1000	5.41	1.50	0.27	5.37	1.64	0.26	**0.985**
	3000	1.97	0.70	0.23	2.00	0.79	0.23	**0.960**
	10000	0.84	0.40	0.22	0.84	0.50	0.21	**0.928**

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning,” in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.

[9] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems,” *arXiv preprint arXiv:1512.01274*, 2015.

[10] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server.” in *OSDI*, vol. 14, 2014, pp. 583–598.

[11] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, “Communication efficient distributed machine learning with the parameter server,” in *Advances in Neural Information Processing Systems*, 2014, pp. 19–27.

[12] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, “Parameter server for distributed machine learning,” in *Big Learning NIPS Workshop*, vol. 6, 2013, p. 2.

[13] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, “Large scale distributed deep networks,” in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[14] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, “Revisiting distributed synchronous sgd,” *arXiv preprint arXiv:1604.00981*, 2016.

[15] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, “Open mpi: Goals, concept, and design of a next generation mpi implementation,” in *European Parallel Virtual Machine/Message Passing Interface Users’ Group Meeting*. Springer, 2004, pp. 97–104.

[16] R. Thakur, R. Rabenseifner, and W. Gropp, “Optimization of collective communication operations in mpich,” *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[17] H. Zhao and J. Canny, “Kylix: A sparse allreduce for commodity clusters,” in *Parallel Processing (ICPP), 2014 43rd International Conference on*. IEEE, 2014, pp. 273–282.

[18] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, “A high-performance, portable implementation of the mpi message passing interface standard,” *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.

[19] S. Rendle, “Factorization machines,” in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 995–1000.

[20] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “Liblinear: A library for large linear classification,” *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[21] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Identifying suspicious urls: an application of large-scale online learning,” in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 681–688.