

Accelerating Gossip-Based Deep Learning in Heterogeneous Edge Computing Platforms

Han, Rui; Li, Shilin; Wang, Xiangwei; Liu, Chi Harold; Xin, Gaofeng; Chen, Lydia Y.

DOI

[10.1109/TPDS.2020.3046440](https://doi.org/10.1109/TPDS.2020.3046440)

Publication date

2021

Document Version

Final published version

Published in

IEEE Transactions on Parallel and Distributed Systems

Citation (APA)

Han, R., Li, S., Wang, X., Liu, C. H., Xin, G., & Chen, L. Y. (2021). Accelerating Gossip-Based Deep Learning in Heterogeneous Edge Computing Platforms. *IEEE Transactions on Parallel and Distributed Systems*, 32(7), 1591-1602. Article 9303468. <https://doi.org/10.1109/TPDS.2020.3046440>

Important note

To cite this publication, please use the final published version (if applicable). Please check the document version above.





Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights. We will remove access to the work immediately and investigate your claim.

Accelerating Gossip-Based Deep Learning in Heterogeneous Edge Computing Platforms

Rui Han , Shilin Li , Xiangwei Wang, Chi Harold Liu , *Senior Member, IEEE*, Gaofeng Xin, and Lydia Y. Chen 

Abstract—With the exponential growth of data created at the network edge, decentralized and Gossip-based training of deep learning (DL) models on edge computing (EC) gains tremendous research momentum, owing to its capability to learn from resource-strenuous edge nodes with limited network connectivity. Today's edge devices are extremely heterogeneous, e.g., hardware and software stacks, and result in high performance variation of training time and inducing extra delay to synchronize and converge. The large body of prior art accelerates DL, being data or model parallelization, via a centralized server, e.g., parameter server scheme, which may easily turn into the system bottleneck or single point of failure. In this article, we propose EdgeGossip, a framework specifically designed to accelerate the training process of decentralized and Gossip-based DL training for heterogeneous EC platforms. EdgeGossip features on: (i) low performance variation among multiple EC platforms during iterative training, and (ii) accuracy-aware training to fastly obtain best possible model accuracy. We implement EdgeGossip based on popular Gossip algorithms and demonstrate its effectiveness using real-world DL workloads, i.e., considerably reducing model training time by an average of 2.70 times while only incurring accuracy losses of 0.78 percent.

Index Terms—Deep learning, decentralized training, gossip, edge computing

1 INTRODUCTION

THE fast development Edge computing (EC) platform in recent years extends cloud service capability to the edge ends of the network [1], [2], e.g., mobile and sensing devices. The prosperity of EC platforms, either in the industry (e.g., AWSGreengrass and Azure IoT Edge) or in the open-source community (e.g., KubeEdge and Apache Edgent), promotes the adaptation of deep learning (DL) models at edge devices [3]. Today's EC platforms however are highly heterogeneous, i.e., roughly 2 million types of devices configurations [4], due to following two reasons. First, EC platforms have versatile types of hardware (e.g., accelerators such as TITAN Xp or RTX 2080 Ti Graphics Cards, or different CPU types), and different resource capacities (e.g., number of processor cores, and amount of memory and I/O bandwidths). Second, platform resources are shared by both training tasks and its co-running workloads which are often short-running and cause intermittent inferences. Such a complex heterogeneity increases the difficulty of DL training by multiple folds.

Example Scenario. Fig. 1 illustrates an example of DL training on heterogeneous EC platforms. Three participants work

together to learn a DL model and each participant performs her/his own training task in a separate EC platform. Consequently, each task has a distinct execution environment: platform 1 has a GPU accelerator, platforms 2 and 3 have different CPU types and resource capacities, and the workloads co-run with the training task continuously change. At each epoch, each platform first processes its own local training data, updates model parameters, and synchronizes information (gradients or parameters) either using a central parameter server or in a peer-to-peer fashion via a gossiping protocol.

To enable DL algorithms on distributed environment, the prior art leverages data parallelism [5], [6], [7], [8] and model parallelism [9], [10], or develops asynchronous training algorithms [11], [12] for unreliable networks. Federated learning paradigm takes a step further to enable training on datasets distributed among multiple participants [13], [14]. Most of aforementioned approaches, however, rely on a central scheduling node which easily become the system bottleneck and a single point of failure.

Decentralized and Gossip-based DL¹ training algorithms are emerging alternatives that offer synchronisation free solution without the central server [15], [16], [17], [18]. The prevailing underlying assumption here is that participant nodes are homogeneous – strongly invalidated by the reality. Indeed, the tasks running in slow platforms can significantly delay the overall decentralized training process. Such a severe yet over-looked performance discrepancy among EC platforms calls for a new solution that can accelerate the Gossip-based training process while producing the the best possible model accuracy.

1. We interchangeably use the term of decentralized and Gossip-based learning in this paper.

• Rui Han, Shilin Li, Xiangwei Wang, Chi Harold Liu, and Gaofeng Xin are with the Beijing Institute of Technology, Beijing 100811, P. R. China. E-mail: hanrui@bit.edu.cn, byuego@163.com, {wangxw-cn, 2361417120}@qq.com, liuchi02@gmail.com.

• Lydia Y. Chen is with the Delft University of Technology. Mekelweg 5, 2628 CD Delft, The Netherlands. E-mail: lydiaychen@ieee.org.

Manuscript received 30 June 2020; revised 16 Oct. 2020; accepted 30 Nov. 2020. Date of publication 22 Dec. 2020; date of current version 11 Feb. 2021. (Corresponding author: Chi Harold Liu.)

Recommended for acceptance by P. Balaji, J. Zhai, and M. Si. Digital Object Identifier no. 10.1109/TPDS.2020.3046440

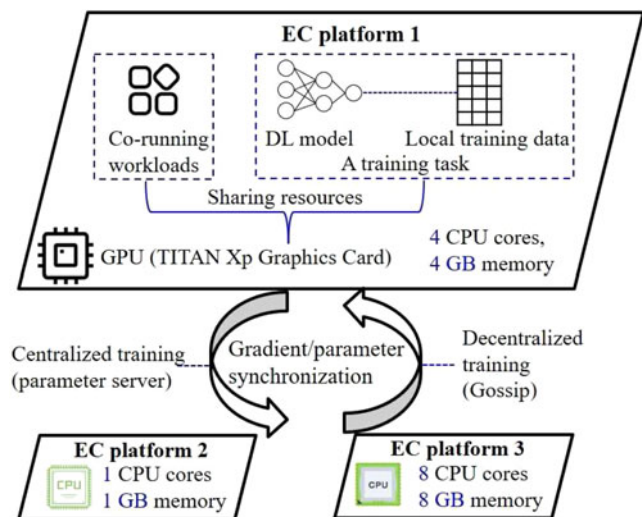


Fig. 1. Example heterogeneous EC platforms.

Motivated by the challenge of heterogeneity, we propose EdgeGossip, a framework to accelerate the Gossip-based DL training on different EC platforms. The basic approach taken by EdgeGossip is to maintain low performance heterogeneity, i.e., differences in training times, among multiple EC platforms by dynamically determining local data computation per training epoch. At the same time, EdgeGossip minimizes result accuracy losses by first processing the most accuracy-related parts of input data in each platform through the novel concept of aggregated data points. In detail, we make the following technical contributions:

- *EdgeGossip Framework on Heterogeneous EC Platforms.* A first of kind heterogeneity-aware Gossip training framework that can ensure the performance homogeneity.
- *Predictive Gossip Training Balancer.* The ultra lightweight balancer dynamically sets the amounts of data processed in different EC platforms according to their predicted performances, thus balancing their training times to accelerate the completion of each training epoch.
- *Accuracy-Aware Trainer.* In each platform, the trainer quickly estimates the effects between different parts of the input data and model accuracy, and first processes the most-accuracy related parts to minimize accuracy losses.
- *Implementation and Evaluation.* We implement EdgeGossip on KubeEdge, an emerging EC platform in the Kubernetes ecosystem and incorporate our approach with the deep ML algorithms in Intel BigDL [7] and PyTorch [19]. By applying EdgeGossip in the typical Gossip-based training algorithm and heterogeneous platforms of CPU and GPU nodes, the evaluation results show our approach considerably reduces model training time by 2.70x with small accuracy losses of 0.78 percent.

The remainder of this paper is organized as follows: Section 2 introduces the background of this work and Section 3 presents our approach. Section 4 evaluates the proposed approach, and finally, Section 5 summarizes the work.

2 BACKGROUND AND RELATED WORK

In this section, we first introduce background of decentralized and Gossip-based training (Section 2.1), following by a discussion of existing techniques on improving distributed mode training and challenges of running Gossip-based DL training in heterogeneous EC platforms (Section 2.2).

2.1 Decentralized and Gossip-Based Training

Parameter server is a major category of *centralized* training techniques that can be divided into three categories: (1) bulk synchronous parallel model that requires all workers to wait until parameter server completes gradient synchronization and model updating at each iteration [20]; (2) stale synchronous parallel model that only requires a worker to wait for other workers once a bounded stale iteration is reached [21]; (3) asynchronous parallel model that allows all workers to be fully asynchronous during training [22].

The *ring AllReduce* approach improves the scalability of centralized training by organizing all n workers in a ring topology and dividing each worker's parameters into n parts. At each iteration, this approach has two steps: (1) at the scatter-reduce step, each worker uses one communication to obtain one part of parameters from another worker. After $n - 1$ communications, the worker gets its final set of parameters; (2) at the allgather step, the worker sends its final set of parameters to other workers using $n - 1$ communications, where each communication transmits one part of parameters to the subsequent worker in the ring.

Compared to centralized training, the main advantage of *decentralized* training lies on addressing the limitation of communication traffic and computing capacity of the central node, thus avoiding the failure of the entire system due to the synchronization problems with the node's critical resources [23]. The basic idea of this training paradigm is to eliminate the use of the central node and allow training participants to exchange information with some of the other participants in a peer-to-peer fashion [11], [16]. Some improvement techniques have been proposed to improve this training paradigm by introducing backup workers and bounded staleness [24] or joint synchronization [25].

Gossip is a dominant algorithm used in decentralized DL model training [16], [17], [18], [26]. This algorithm is initially developed for distributed averaging problems [15], [27], which iteratively calculate the average vector across all multiple nodes in a peer-to-peer network topology. When applying Gossip in DL training, all the training participants (EC platforms) communicate information according to a mixed matrix and finally converge to the same minimum. For example, the process of the typical GoSGD algorithm [18], [26] consists of three steps in a training epoch. First, for each participant and iteration, the algorithm calculates gradients using the participant's local input data and model, and updates the parameters using the gradients. The mini-batch gradient descent method is usually used in this step. Second, it sends weights to other participants according to the matrix. Finally, it receives weights from other participants, and merges them with the local weights to update the model.

Relationship to Federated Learning. After first proposed by Google in 2016, federated learning [13], [14] enables multiple

TABLE 1
Existing Performance Improvement Techniques
in Large-Scale ML Applications

Category	Introduction
Data parallelization	Stragglers [28] I/O bottlenecks [29] Data locality and task dependency in MapReduce jobs [30] Hybrid parallelization strategy [31]
Specific ML operations or algorithms	Join operations [33] Matrix computations [34] Graph mining [35] Ensemble learning [36] Deep learning [37]
Approximate computation	Approximate programs [38] Lower precision [39]

participants (data owners) to build a common, robust machine learning model without making public their own data. To this end, federated learning is designed to solve various issues related to training data, such as data privacy, data security, and data heterogeneity. The general principle of federated learning is to train local models using local data, while exchanging parameters among different participants, typically through a central scheduling node like parameter server [9]. The central node acts as a reference clock for distributed working nodes, and orchestrates the different steps of the algorithm by controlling data and computing resource allocation. The decentralized training studied in this work naturally applies to federated learning, which enables using decentralized data for training models. Decentralized training is prospering topic in federated learning, in particular for training conducted in an EC environment [14].

2.2 Techniques on Improving Distributed Training

Improving performance in distributed ML and DL has been the subject of extensive research in recent years, as summarized in Table 1.

First, *data parallelization* is the most widely used technique that accelerates the training process by processing data points in a parallel and distributed manner [5], [6], [7]. Many of these techniques are developed to handle challenges in data parallelization, such as stragglers [28], I/O bottlenecks [29], data locality [30], and algorithm-level parallelization [31]. MapReduce [5] is a mainstream technique in this area, some other popular ML engines include TensorFlow [32], PyTorch [19], and MLbase [6].

Some other techniques focus on optimizing the performance of *specific computations* such as join operations [33] or matrix computations [34] in model training, or they are designed for *specific applications* such as graph mining [35], decision tree ensembles [36], and DL [37].

In addition, *approximate computation* techniques either use approximate programs or codes over large datasets [38], or apply lossy compression schemas to model parameters to reduce computation and communication overheads in training [39].

The above performance improvement techniques cover different aspects of distributed model training, and most of

them (either data parallelization or model parallelization [9], [10]) are designed for training built upon a central scheduling node. This work is complementary to existing techniques that it focuses on improving decentralized training deployed in heterogeneous EC platforms. We summarise the two key characteristics of such training as follows:

- Real EC nodes, e.g., mobile devices or server nodes, usually have different resource capacities, and slow nodes may delay the overall training process in the Gossip framework. The *first challenge*, therefore, is how to coordinate the model training among these nodes with consideration of their discrepancy of resources and performances, and try to produce the best possible results.
- Moreover, even given a resource capacity in a node, the actual performance of the training task may degrade, or vary, with time, because it shares devices resources such as memory, and processing cycles, and communication bandwidth with other co-running workloads. The *second challenge* is to control how the training algorithm adapts its processing of data in response to such changes.

3 EDGEGOSSIP

In this section, we first present an overview of EdgeGossip in Section 3.1, followed by an explanation of its modules in Sections 3.2, 3.3, and 3.4.

3.1 Overview

For a DL training task, EdgeGossip is presented to accelerate its decentralized model training using modules developed for both the pre-training stage and the the iterative training stage, as shown in Fig. 2.

The Pre-Training Stage. The module at this stage transforms the input dataset X into multiple aggregated data points using three steps. Step 1 transforms X into a low-dimensional dataset X' that can be processed efficiently by the following data division step. Step 2 divides the data points X' into multiple subsets and preserves data similarity in the division process. Note that at steps 1 and 2, we selected two and three techniques respectively. These techniques have both good effectiveness and acceptable overheads. Finally, step 3 aggregates the information of *original* input data point in each subset to generate an *aggregated* data point. Note that the above creation process is only applied once before model training.

The Iterative Training Stage. At this stage, the three modules are designed to accelerate the model training process. Before each training epoch, the *performance predictor* in each platform reports its estimated performance to the *Gossip training balancer*. The balancer reduces the discrepancy among different platforms' training times by setting different ratios of input data for the coming epoch. In the setting, platforms of higher performances or smaller input data sizes are set to process larger ratios of input data. Moreover, the *Accuracy-aware trainer* module minimizes the accuracy losses due to the removed input data. This is achieved by identifying and first processing the most accuracy-related input data based on the aggregated data points.

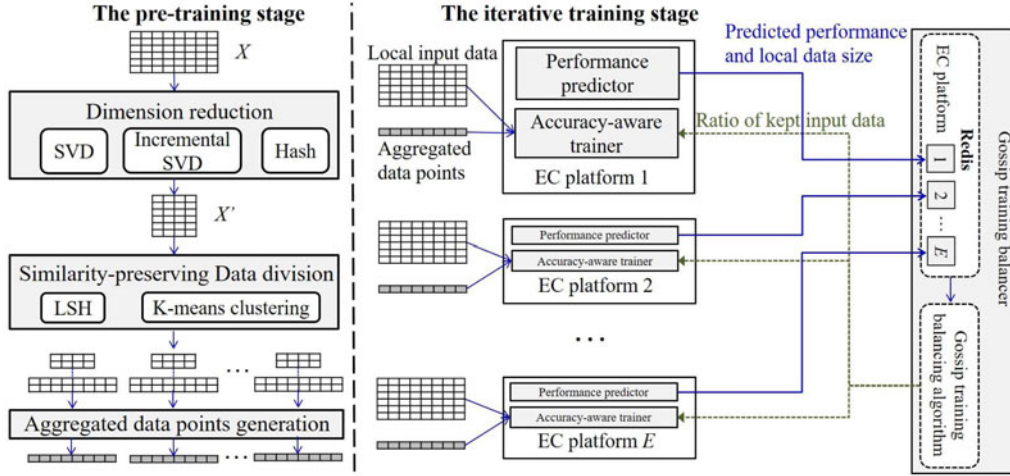


Fig. 2. The overview of EdgeGossip.

3.2 Input Data Aggregator

The input data aggregator generates aggregated data points with two basic purposes: preserving data similarity when grouping input data points and low generation overheads. To this end, we design an end-to-end process with three steps (Table 2 lists the symbols used here):

- *Dimensionality Reduction.* This step is designed to reduce *generation overheads* when dealing with high-dimensional datasets. It employs a dimensionality reduction technique to transform the original $N \times d$ dataset X into a reduced $N \times d'$ ($d' \ll d$) dataset X' .
- *Similarity-Preserving Data Division.* This step divides the $N \times d'$ dataset X' into M subsets $\{X_1, X_2, \dots, X_M\}$ while *preserving data similarity*. That is, each subset consists of multiple data points of similar feature values.
- *Aggregated Data Points Generation.* Using the first two steps, the *Input Data Aggregator* module divides the reduced dataset X' (and the original input dataset X) into M subsets. For each subset, it generates an aggregated data point using the *original* input data points in X in the final step. Specifically, this step calculates the j th feature of the aggregated data point of subset X_i ($1 \leq i \leq M$ and $1 \leq j \leq d$) as

$$a_j = \frac{\sum_{k=1}^{|X_i|} x_j^{(k)}}{|X_i|}, \quad (1)$$

TABLE 2
A Summary of Symbols

Symbol	Symbol meaning
d	Data dimensionality of input data
\vec{x}	The attribute vector (x_1, x_2, \dots, x_d) of an input data point
X	Input dataset
y	The class label of input data point
N	The number of points in X
\vec{a}	An aggregated data point
M	The number of aggregated data points corresponding to the original data points in X

where $x_j^{(k)}$ denotes the j th feature of the k th point in X_i . In generation, each subsets consists of data points belonging to the same class.

Selection of Aggregation Techniques. We note that there exists a variety of dimensionality reduction and data vision techniques. Prevalent techniques of the first type include singular value decomposition (SVD) based on single data points or the entire dataset, incremental SVD [40], Hash, Principal components analysis (PCA), and histogram. Prevalent techniques of the second type include k-means clustering, Locality Sensitive Hash (LSH), and Gaussian mixture model (GMM). In EdgeGossip, we implemented eight representative combinations of the above techniques as listed in Fig. 3, and compare their them with both overhead and accuracy metrics. The *overhead* metric is each combination's time consumption, including the execution times of the three aggregation steps and their total time. The *similarity-preserving* metric is denoted by the top-1 classification accuracy of the

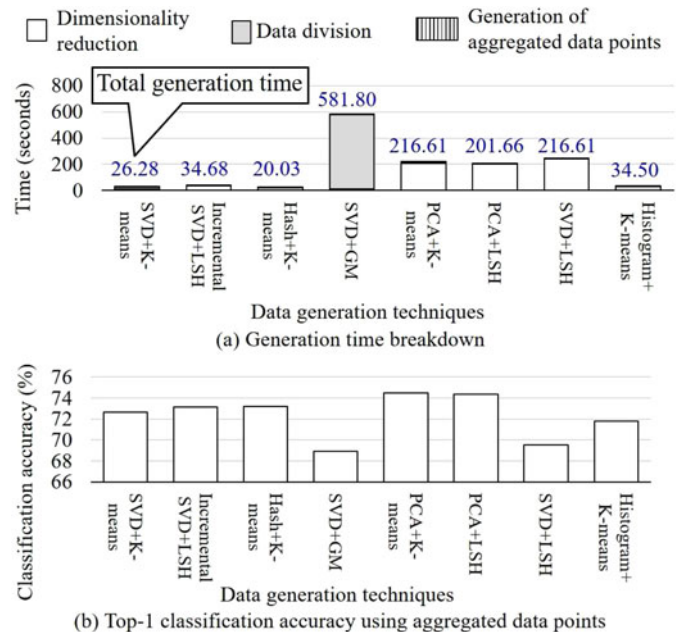


Fig. 3. Comparative evaluations of eight combinations of dimensionality reduction and data vision techniques.

TABLE 3
Time Complexities of Data Aggregation Techniques

Step	Technique	Time complexity
Dimensionality reduction	SVD	$O(N \times d^{3/2})$
	Incremental SVD	$O(d' \times e \times v \times N)$
	Hash	$O(N \times d)$
Data division	LSH	$O(d \times N \times \log N)$
	K-means	$O(d \times M \times e \times N)$

1. d' represent dimensionality after reduction.
2. e represents the number of iterations.
3. v represents the number of attributes used in each iteration.

DL model trained using the generated aggregated data points. This accuracy represents these points' approximation level to the input data. In evaluation, AlexNet and Cifar10 dataset [41] are tested and one aggregated data point corresponds to five original data points. We can observe in 3(a) that the generation times of four combinations (SVD+GM, PCA+K-means, PCA+LSH, SVD (based on the entire dataset)+LSH) are 10 times longer than those of other combinations. In addition, although the last combination (histogram +K-means) has a similar generation time with the first three combinations, it has the lowest classification accuracy among all combinations (Fig. 3b). In conclusion, we select the first three combinations that achieve both low overheads and

high accuracy. Table 3.2 summarizes the time complexities of the selected techniques.

Example. Fig. 4 shows an example of generating aggregated data points. *Step 1* transforms a 12×5 input dataset X into a 12×2 dataset X' using three data aggregation techniques. We can see that in all techniques, the data points with similar feature values (e.g., points \bar{x}_1 and \bar{x}_2) still have similar features in X' . In dimensionality reduction, the incremental SVD technique uses the longest time (12 time units), and the other two techniques (SVD and hash) use much shorter times. Based on the reduced dataset X' , *step 2* divides it into four subsets using either k-means clustering or LSH. We can see that although the two techniques have different data division process, similar data points in X' are still divided into the same category. Finally, *step 3* creates aggregated data points according to the divided subsets. Each aggregated data point corresponds to one to seven original data points.

3.3 Predictive Gossip Training Balancing

In Gossip-based training, predicting performances of training tasks deployed in different EC platforms is the key step to accelerate unbalanced training in different platforms. This prediction is based on the contention information of shared resources collected by monitors. The proposed performance predictor comprehensively considers both processor cores

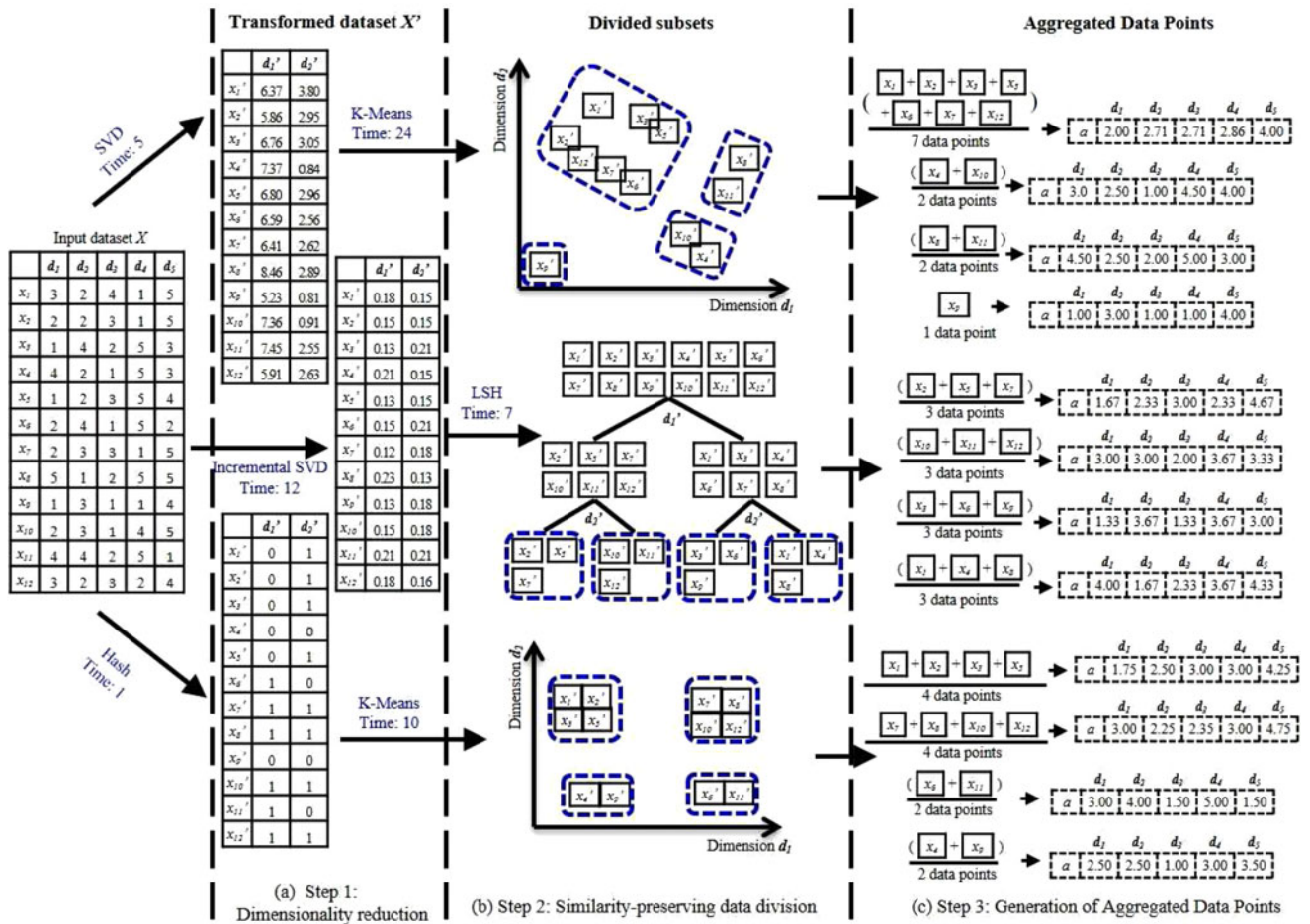


Fig. 4. An example of generating aggregated data points with three steps.

TABLE 4
Contention Information of Shared Resources

Shared resources	Contention information
Floating point and vector processing units, pipelines, and data prefetchers	U^{core} =core usage
Network bandwidth	$U^{networkBW}$ =the amount of send/receive data per second
Disk bandwidth	U^{diskBW} =the amount of read/write data per second

(e.g., shared processing units and caches) and I/O resources (disk and network bandwidths) contended by different programs in the same node. Table 4 lists the contention information of these shared resources, in which core usage represents the ratio of time running instructions on the cores (including private cache hits). Note that the *contention* of these resources comes from a training task's co-running programs within the same service or across other applications, and node's hardware/software activities.

The performance predictor consists of two parts. First, in the presence of fine-grained heterogeneity of resource contentions in each node of the platform, the *basic* performance predictor is responsible for collecting the information of resource sharing and contention, and predicting the impact on individual node's performance. Second, the *extended* performance estimator further calculates the overall performance (throughput) of the platform based on the training implementation topology. With these two parts, the performance predictor finally exposes the platform performance to the Gossip training balancer.

Performance Prediction for a Node. Given a training task hosted in a node, the *basic* performance predictor is developed to capture the impact of resource sharing and contention on task's performance and estimate its processing time t per unit of input data. The predictor employs a regression model to describe the relationship between contention information and processing time t . Specifically, the training of the regression model takes a set of V samples $\{(\vec{U}_1, t_1), (\vec{U}_2, t_2), \dots, (\vec{U}_V, t_V)\}$ as input and outputs a model $RG(\vec{U})$, where $\vec{U}_i = (U_i^{core}, U_i^{networkBW}, U_i^{diskBW})$ ($i = 1, \dots, V$) and the training samples can be obtained from profiling runs or historical running logs. In model $RG(\vec{U})$, $\vec{U} = (U^{core}, U^{networkBW}, U^{diskBW})$ represents the CPU, network, and disk utilizations of the task's co-running jobs in the same node. The task's process time is predicted as: $t_0 \times RG(\vec{U})$, where t_0 denotes the task's original processing time without performance inference and $RG(\vec{U})$ denotes the performance deprecation ratio ($RG(\vec{U}) > 1$). We note that EC nodes may have random slowdown, which influences t_0 in the basic performance predictor. Hence the prediction mechanism can handle such slowdown by dynamically changing t_0 according to the machine status.

Performance Estimation for an EC Platform. Support the training task is parallelized across I executors (nodes) of the platform. Using the basic performance predictor, the i th executor's processing time t_i per unit of input data is calculated according to its hosted node. The *extended* performance estimator then calculates the overall *throughput* p of the platform by taking a summarization of all executors: $p = \sum_{i=1}^I \frac{1}{t_i}$, where $\frac{1}{t_i}$ represents the number of processed input data points per unit of time (e.g., 100 points per second).

The estimation of p is based on the assumption that the I parallel executors have the same processing algorithm.

Algorithm 1. Gossip Training Balancing Algorithm

Require: E : the total number of EC platforms;

- p_i : the predicted throughput (number of processed data points per second) in platform i ($1 \leq i \leq E$);
 $|X_i|$: the number of data points in dataset X_i ;
 $T[i]$: platform i 's estimated processing time of completing its epoch;
 r_i : the ratio of platform i 's kept training data at the next epoch;
 $F[i]$: the failure status of platform i .
1. Obtains performances p_1 to p_E ;
 2. **for** ($i=1; i \leq E; i++$) **do**
 3. **if** ($p_i < p^{threshold}$) **then**
 4. $F[i] = 1$; // platform i is failed;
 5. $T[i] = \infty$;
 6. **else**
 7. $F[i] = 0$;
 8. $T[i] = \frac{|X_i|}{p_i}$;
 9. **end if**
 10. **if** ($\frac{\sum_{j=1}^E F[j]}{E} > f^{threshold}$) **then**
 11. Halt the balancing process and trigger an exceptional handling;
 12. **end if**
 13. **end for**
 14. Sort the E platforms in ascending order according to their estimated processing times, where $T[i] \leq T[i+1]$ ($i=1, \dots, E-1$);
 15. **for** ($i=1; i \leq E^{successful}; i++$) **do**
 16. $t'_i = \frac{T[i]}{T[1]}$;
 17. $r_i = \frac{1}{t'_i}$;
 18. **end for**
 19. Return $\{r_1, r_2, \dots, r_E\}$.

Gossip Training Balancer. Given a set of E EC platforms and their predicted performances, the steps of the Gossip training balancer are detailed in Algorithm 1. This algorithm is executed before each epoch of model training. It first obtains the platforms of all E platforms (line 1). If platform i 's performance p_i is smaller than a threshold (e.g., 0.001), this platform is judged as a failure one (lines 3 to 5) and its estimated processing time $T[i]$ is set as infinity; otherwise the processing time is calculated as its input data size $|X_i|$ divided by its predicted throughput p_i (lines 6 to 8). The calculation is based on the assumption that the E platforms have the same training algorithm and their local input datasets have the same feature space. Hence a platform's estimated training time depends on its performance and input data size. Note that previous studies show portion of devices that drop out due to computation or network errors ranges between 6 to 10 percent [16]. In the Gossip-based training scenario, a small portion of failure platforms does not have an immediate negative impact on the training convergency time, because remaining platforms will continue to make progress. If this portion is larger than a threshold $f^{threshold}$ (e.g., 10 percent), the algorithm triggers an exception handling that can execute the training tasks of the failure platforms in other nodes (lines 10 to 12). Subsequently, the algorithm ranks the E platforms in ascending order according to their estimated processing times (line 14). Finally, the algorithm sequentially calculates each platform's

ratio of kept input data (line 15 to 18). A platform of shorter estimated time is set to process a higher ratio of input data, and vice versa. Such setting balances the training time of these heterogeneous platforms.

Overhead Analysis. In our predictive Gossip training balancer, the overhead of the *performance predictor* consists of two parts. The first part comes from collecting different resource usages (processor cores, disk and network I/O bandwidths) by accessing the information stored in the /proc filesystem in Linux based systems. The predictor collects the information once every second and periodically (e.g., 1 minute) writes the mean resource utilizations to a Redis data store. Hence the second part is the communication cost between the predictor and the balancer. In addition, the time complexity of the *Gossip training balancing algorithm* is $O(E \log E)$, where E is the number of EC platforms.

3.4 Accuracy-Aware Trainer

In each EC platform, the steps of accuracy-aware training in an epoch are detailed in Algorithm 2. The algorithm first estimates the M aggregated data points' effects on model accuracy (line 1). In a multi-class problem, the effect of an aggregated point a_i is calculated as the value of the loss function to be minimized in training, similar to current importance sampling techniques [42]. Note that importance sampling employs input data points' loss values to skew the sampling towards important ones and select a subset of input data to improve the convergence speed. This subset is *kept unchanged* during the iterative training process. In contrast, our approach dynamically changes the ratios of processed input data at each iteration according to the predicted performances in all EC platforms. Subsequently, the algorithm first ranks the aggregated data points in descending order according to their effect values (line 2), it then uses each point's ranking order to decide the ranking order of its corresponding subset of original data points (line 3). That is, a higher value of e_i means a_i and its corresponding subset X_1^a has a higher correlation to model parameter updating, thus having a higher probability of improve model accuracy.

Based on the ranked subsets of input data, the algorithm performs the training iterations in an epoch (line 6 to 16). Specifically, the algorithm sequentially adds the ranked sets to $X_{iteration}$, which includes at least b data points to be used in the next training iteration (line 8 to 12). This addition is based on the idea that the data points with higher ranks, namely higher correlations to model accuracy, are first used in model training, because processing a proportion of the top ranked points determines most of the model accuracy.

4 EVALUATION

Our evaluation of EdgeGossip on real-life DL applications has three objectives. First we show the effectiveness of EdgeGossip against the typical Gossip algorithm in reducing model training time (Section 4.2). Then, we present the timing and accuracy advantages achieve in two key modules of EdgeGossip: the generation and processing overheads of aggregated data points (Section 4.3), and the accuracy and overheads of the proposed performance predictor (Section 4.4). Finally, we discuss the factors that influence EdgeGossip (Section 4.5).

Algorithm 2. Accuracy-Aware Training in Platform i

Require: X_i : the training data in platform i ;
 $\{a_1, a_2, \dots, a_M\}$: the M aggregated data points in platform i ;
 e_i : a_i 's effect on model accuracy ($1 \leq i \leq m$);
 X_i^a : the set of original data points represented by a_i ;
 r_i : the ratio of kept data;
 b : the batch size per training iteration;

1. Process the M aggregated data points to estimate their effects e_1 to e_M ;
2. Rank the M points in descending order according to their effect values;
3. Obtain the ranked sets $\{X_1^a, X_2^a, \dots, X_M^a\}$ according to the ranking orders of aggregated data points;
4. $b_{processed}=0$; // the number of processed input data
5. $i=1$; // the index of processed aggregated data point
6. **while** ($b_{processed} \leq |X_i| \times r_i$) **do**
7. $X_{iteration}=\phi$;
8. **for** ($i \leq M$; $i++$) **do**
9. $X_{iteration}=X_{iteration} \cup X_i^a$
10. **if** ($|X_{iteration}| \geq b$) **then**
11. Break; // stop the for loop
12. **end if**
13. **end for**
14. Perform an iteration of model training using $X_{iteration}$;
15. $b_{processed}=b_{processed} + |X_{iteration}|$;
16. **end while**

4.1 Evaluation Settings

Experimental Environment. The experiments are performed in three types of heterogeneous EC platforms, including one CPU platform and two GPU platforms. In the CPU platform, each node is equipped with 18-core Intel E5-2695 v4 processors, and 256 GB of DRAM. The first GPU platform has 12-core Intel(R) Core(TM) i7-8700K processors, 12-GB TITAN Xp Graphics Card, and 64 GB memory. The second GPU platform has 88-core Intel(R) Xeon(R) Gold 6238 processors, 11-GB GeForce RTX 2080 Ti Graphics Card, and 512 GB memory. All nodes run Linux Ubuntu 18.04. In the Spark cluster, the JDK, Spark, and KubeEdge versions are 1.8.0.181, 2.4.3, and v1.15.3 respectively. In the PyTorch cluster, the python, pytorch, cuda, cudnn, and redis versions are 3.7.7, 1.6.0, 10.1, 7.6.3, and 4.0.9, respectively.

Tested Workloads and Datasets. We consider two popular DL models (LeNet-5 [43] and AlexNet [44]) based on the implementation of EdgeGossip in PyTorch. LeNet-5 has 60k parameters and it is tested using the MNIST dataset [45]. AlexNet has 61.5 million parameters and it is tested using the Cifar10 dataset [41] and the downsampled ImageNet8 × 8 dataset [46]. Using Cifar10, we also tested two typical DL models widely used in EC and mobile devices: Squeezenet [47] and MobileNets-v2 [48]. Both models achieve similar levels of accuracy with AlexNet but have much smaller numbers of parameters and thus have lower inference time. Both MNIST and Cifar10 datasets have 60k data points and the numbers of training and testing points are 50k and 10k, respectively. ImageNet dataset has 1.28 million data points from 1000 classes and 50k testing points (50 ones per class) [46].

Model Training Settings. The mini-batch gradient descent method [49] is used in model training. For all models, the learning rate, the momentum, and the batch size are set to

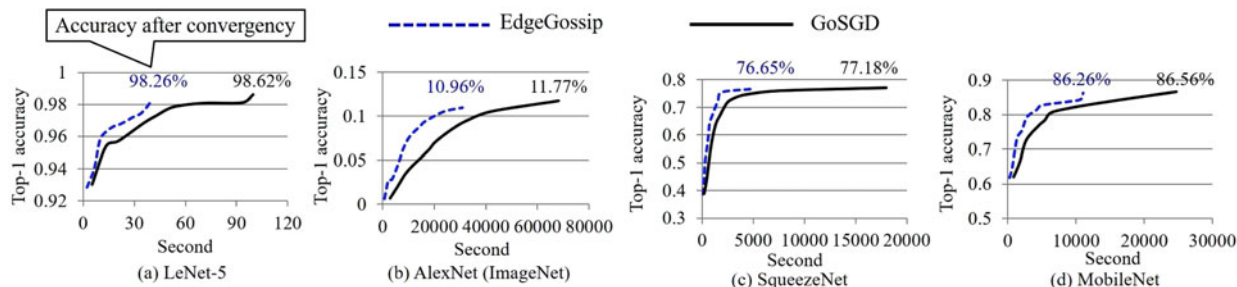


Fig. 5. Comparison of training time and model accuracy with GoSGD.

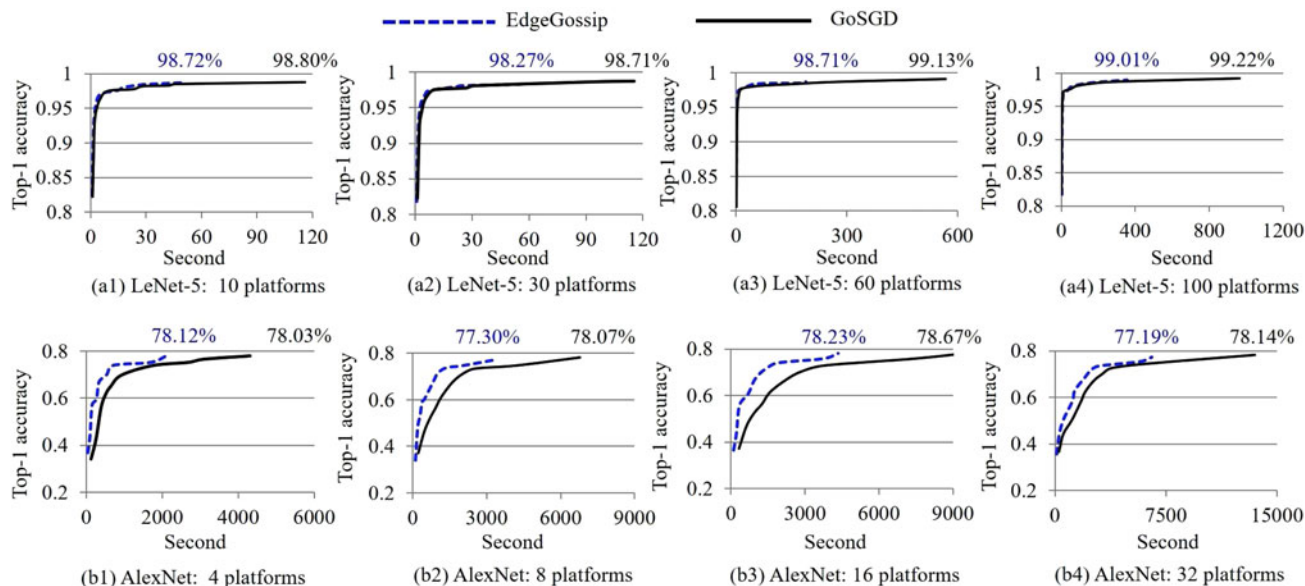


Fig. 6. Comparison of training time and model accuracy under different numbers of EC platforms.

0.01, 0.9, and 64, respectively. Note that in all comparative evaluations, we use the same hyperparameters and initial model parameters.

Evaluation Metrics. For a fair comparison, we report both *performance* and *accuracy* metrics in model training. The performance is the model training time across different epochs. In our approach, this time also includes the generation and processing times of aggregated data points. The *accuracy* metric is top-1 classification accuracy on the test set: the top 1 predicted class (the one having the highest probability) is the same as the actual class label.

4.2 Effectiveness of Training Acceleration

Here, we evaluate the effectiveness of EdgeGossip in accelerating model training. We implement our approach in the representative GoSGD algorithm for deep learning [18], [26], and compare the standard GoSGD algorithm with EdgeGossip. We test different cases of Gossip-based training from three aspects: (1) four DL models (LeNet-5, AlexNet, SqueezeNet, and MobileNets); (2) three input data generation methods, in which the dimension reduction techniques are SVD, Incremental SVD, and Hash respectively, and the data division techniques are K-means, LSH, and K-means respectively. In all methods, the *aggregation ratio* (the number of original input data points divided by the number of aggregated data points) is 10. (3) Five training deployments. The first deployment has four EC platforms (two CPU ones and

two GPU ones), and each platform has 25 percent of training data. The following four deployments correspond to four different numbers of platforms for LeNet-5 and AlexNet. In each deployment, training tasks co-run with MapReduce workloads (WordCount and Sort, whose input data size ranges from 1 MB to 10 GB) in the executor nodes.

Fig. 5 shows the comparative results in terms of training time (x axis) and top-1 accuracy (y axis) for the first deployment. We can observe that during the iterative training process, applying GossipEdge to balance the training times across different EC platforms considerably accelerates the training speed. In addition, both approaches obtain similar accuracies after convergence. These results verify that using aggregated data points (generated by any of the three methods), GossipEdge correctly identifies and removes less accuracy-related input data and the retained input data in slow platforms contributes to large parts of model accuracy.

Fig. 6 further shows the comparative results under different scales of deployments. For both models, the training takes longer time when the scale becomes larger and the communications among different EC platforms increase. Figs. 6a1 to 6d1 show that in LeNet-5, the acceleration of training time is smaller than other models. This is because when the deployment scale increases, the accuracy quickly increases to 0.95 in this simple model and hence the training times in EdgeGossip and GoSGD are similar. In addition, the results in Figs. 5c, 5d and 6(b1) show that SqueezeNet and

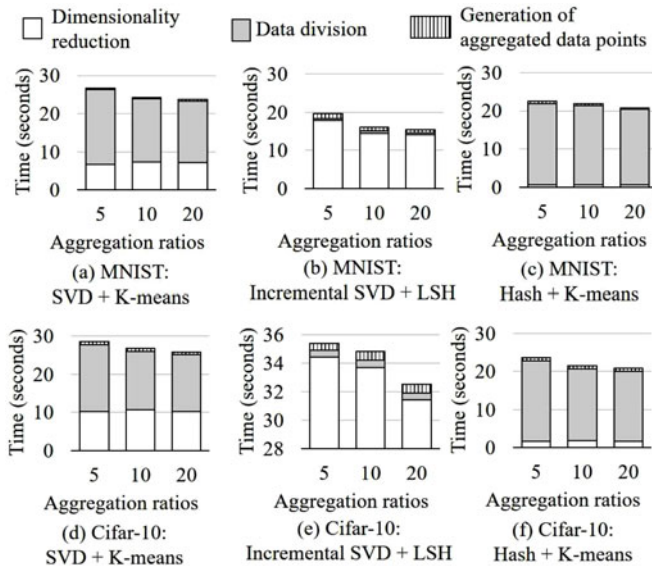


Fig. 7. Generation time breakdown under three aggregation ratios.

MobileNets-v2 have longer training time than AlexNet when processing the same dataset (Cifar10). This is because both models reduce parameters by increasing the numbers of feature maps using 3×3 and 1×1 convolutional layer, and thus have much more expensive computations of convolutional layers in model training. In conclusion, EdgeGossip is applicable to models with high training complexity (e.g., Squeezenet and MobileNets-v2), which causes a large discrepancy of training times among different EC platforms and hence EdgeGossip accelerates model training by reducing such performance discrepancy. The effectiveness of EdgeGossip is also influenced by the scale of training deployment for a similar reason, because the increase of scale may bring a higher heterogeneity in EC platforms and thus increases of discrepancy of training times among these platforms.

4.3 Overheads of Aggregated Data Points

In this section, we evaluate the overheads of aggregated data points in terms of their generation and processing times. At the pre-training stage, the generation overhead comes from the three operations on the input data: dimension reduction, data division, and generation of aggregated data points. We report the *execution time of each part*. At the iterative training stage, the processing overhead comes from calculating aggregated data points' losses. This calculation is performed before each training epoch, and we report its *percentage computation time*, which denotes the execution time of this calculation divided by the total execution time in an epoch.

Evaluation Settings. We test the same generation methods as the previous section. In SVD, the number of singular value is 10, that is: the reduced dimensionality $d'=10$. In the optimization process of Incremental SVD, 1 percent of original feature values are used, the learning rate is 0.001, the constant value $l=0.015$, and the number of iterations is 20. The dimensionality after reduction is $d' = \log_2 \frac{0.1 \times d}{\text{aggregation ratio}}$. In Hash, the "fingerprint" length, namely the reduced dimensionality d' , is 10. In K-means clustering, the number of iterations is 20 and the number of cluster centroids is $\frac{N}{\text{aggregation ratio}}$. Similarity, in LSH, the number of buckets is $\frac{N}{\text{aggregation ratio}}$. For each technique, we test three cases of

input data aggregation, whose aggregation ratios are 5, 10, and 20, respectively. The evaluation is performed in a Spark cluster with one master node and one executor node (both nodes belong to the CPU platform).

Generation Time. Fig. 7 shows the generation times of three operations in each technique. We can see that in technique 1 (Figs. 7a and 7d) and technique 3 (Figs. 7c and 7f), data division (using K-means) take most of the generation time. In contrast, dimensionality reduction (using incremental SVD) takes the longest time in technique 2 (Figs. 7b and 7e). These results verify the analysis of time complexity in Table 3. In addition, a larger aggregation ratio (e.g., 20) means a smaller number of aggregated data points, and these points also need less time to generate. In conclusion, the generation time of all three techniques is much shorter than that of model training time.

Processing Time. Fig. 8 further shows the training time breakdown of two parts: processing aggregated data points and the training time using the retained input data. We can see that the execution time of processing aggregated data points is inversely proportional to the aggregation ratio. That is, a larger ratio (less aggregated data points) means shorter processing time. When considering all evaluation cases, processing aggregated data points only takes an average of 3.0 and 8.78 percent of the training time in LeNet-5 and AlexNet, respectively.

4.4 Effectiveness of Performance Predictor

In decentralized DL training, the accuracy of the performance predictor decides the ratio of kept input data in each EC platform, and thus the training time of platforms in an epoch. In this section, we first evaluate this accuracy with consideration of both *different resource capacities* and *changing performance interferences*. The evaluation is conducted in CPU platforms, and each platform has a VM of 4 CPU cores and 4 GB in its executor node. In the VM, we run MapReduce workload (WordCount or Sort) as the co-running job. We tested 13 different input data sizes (between 1 MB to 10 GB) for either workload to reflect the different performance interferences to the training task.

Prediction Accuracy. Fig. 9 shows the mean and variance of prediction errors of our performance predictor under different co-running workloads. We can see that when the input data size of the co-running workload increases, the prediction error slightly increases. This is because the workload with larger input data consumes more resources and thus may cause more interferences. To demonstrate this, Fig. 10 shows the utilizations of three shared resources (CPU cores, network and disk bandwidths) under different input data sizes. For each resource type, the distribution of its utilizations (mean \pm 2 standard deviations (SDs)) is reported. We can observe that with larger input data sizes, the CPU utilization of both workloads increases, and the disk utilization of the Sort workload increases because this workload is more I/O intensive than WordCount. Also, both workloads have larger fluctuations in network and disk bandwidths with larger input data sizes, thus may cause larger performance interferences. Overall, the average prediction error is 4.30 percent, which means our performance predictor precisely estimates the platform performances under different resource contentions.

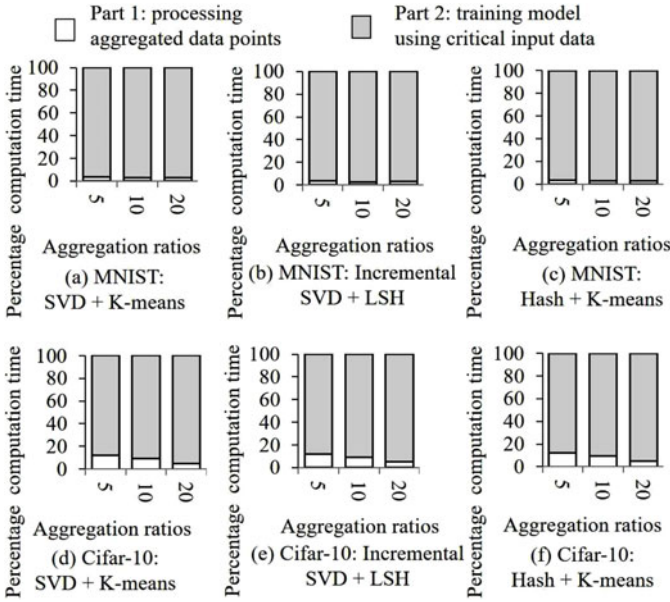


Fig. 8. Percentage training time breakdown under three aggregation ratios.

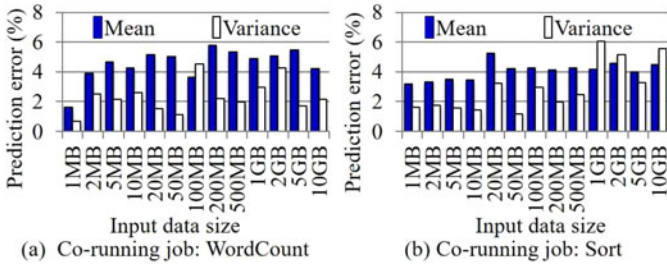


Fig. 9. Prediction errors of the performance predictor with different co-running jobs.

Prediction Overheads. Fig. 11 shows the two major overheads of the performance predictor: collecting resource usages and communication with Redis. Each test was repeated 10 times for consistency and the average is reported. The evaluation results show: (i) both overheads are two or three orders of magnitude shorter than 1 second, thus having negligible impact on the model training; (ii) the overhead increases when the input data size of the co-running workload increases. The overhead of the Sort workload is larger because this workload incurs larger fluctuations in resource usages.

4.5 Discussions

EdgeGossip provides the notion of trade-off between training time and model accuracy in decentralized DL training. In this section, we take the LeNet-5 workload as an example and design experiments to discuss other practical factors that influence such a trade-off. The following experiments are conducted under the same evaluation settings as previous sections.

Discussion of Unbalanced Platform Resources. In this evaluation, we test two larger unbalanced situations: the resource capacity of fast CPU platforms are 3 and 5 times larger than that of slow platforms. This increased unbalance has twofold effects, as shown in Fig. 12. First of all, the GoSGD algorithm needs longer time to complete the training process, because slow platforms cause a longer delay in each epoch's

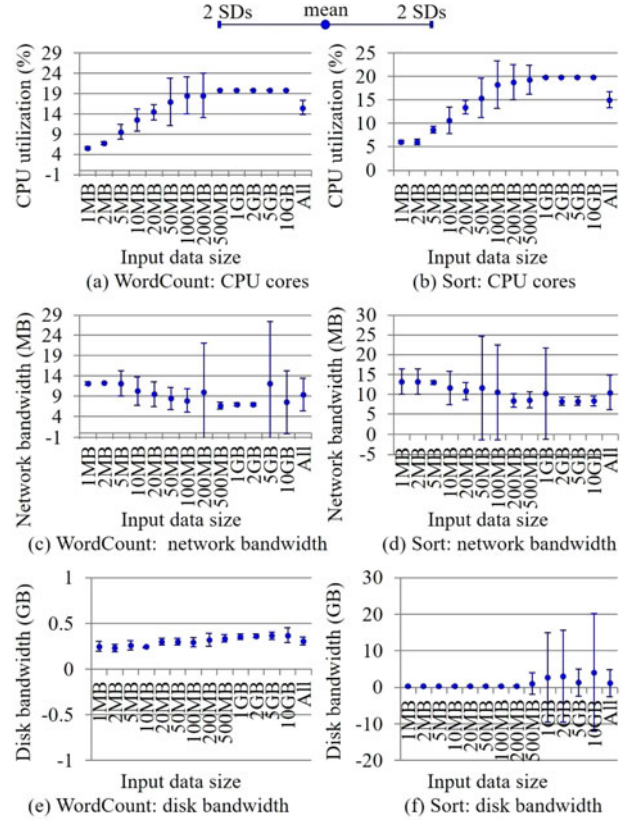


Fig. 10. Utilizations of three shared resources under different workloads.

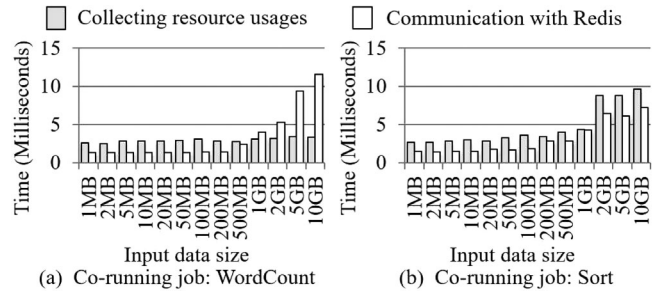


Fig. 11. Overheads of the performance predictor under different workloads.

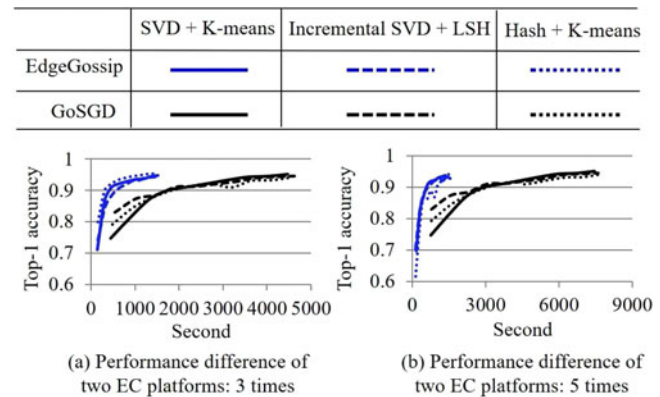


Fig. 12. Comparison of training time and model accuracy under different unbalanced resources.

synchronization. Second, GossipEdge can still maintain similar model training times but also incurs slightly larger accuracy losses, because less input data is processed in slow platforms.

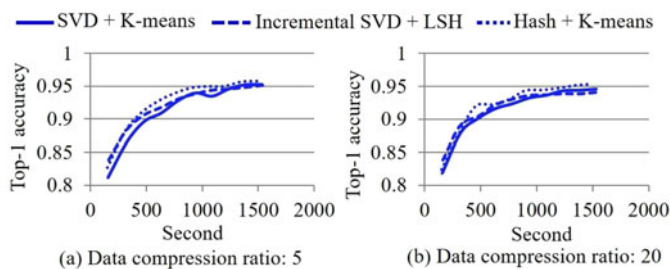


Fig. 13. Comparison of training time and model accuracy under different aggregation ratios.

Discussion of Aggregation Ratios. In the accuracy-aware trainer, the aggregation ratio is a key parameter that determines aggregated data points' granularity of approximating the input data. In addition to the previous ratio (10), we test two new aggregation ratios (5 and 20) in this experiment. The results in Fig. 13 show that selecting a larger ratio (20) results in a smaller number of aggregated data points, thus requiring shorter processing time on aggregated data points but also cause slightly lower model accuracies. Note that the reduced processing time has a very small influence on the total model training time, because this time takes less than 3 percent of training time at each epoch.

Results. When considering different model and deployment settings, EdgeGossip accelerates the standard Gossip training process by an average of 2.70 times with small accuracy losses of 0.78 percent.

5 CONCLUSION

In this paper, we presented EdgeGossip to accelerate the training process of decentralized DL in heterogeneous EC platforms, and demonstrated its effectiveness using popular DL algorithms. EdgeGossip is based on two key ideas: (1) it balances the training times among different EC platforms according to their predicted performances; (2) using aggregated data points, it quickly identifies the most accuracy-related parts of input data and first process them to improve model accuracy. Evaluation results using real workloads demonstrate the effectiveness of EdgeGossip in bringing considerable reductions in model training times while only causing small accuracy losses.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for careful review of their article. This work was supported in part by the National Key Research and Development Plan of China (Grant No. 2018YFB1003701 and 2018YFB1003700), in part by the National Natural Science Foundation of China (Grant No. 61872337), and in part by the Swiss National Science Foundation NRP75 project 407540_167266.

REFERENCES

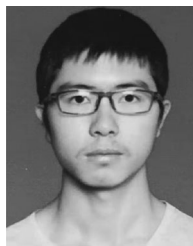
- [1] M. Satyanarayanan, "The emergence of edge computing," *IEEE Comput.*, vol. 50, no. 1, pp. 30–39, Jan. 2017.
- [2] D. Balouekthomert, E. G. Renart, A. R. Zamani, A. Simonet, and M. Parashar, "Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows," *Int. J. High Perform. Comput. Appl.*, vol. 33, no. 6, pp. 1159–1174, 2019.
- [3] M. Ali *et al.*, "Edge enhanced deep learning system for large-scale video stream analytics," in *Proc. IEEE 2nd Int. Conf. Fog Edge Comput.*, 2018, pp. 1–10.

- [4] C. Wu *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 331–344.
- [5] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [6] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "MLbase: A distributed machine-learning system," in *Proc. 6th Biennial Conf. Innovative Data Syst. Res.*, vol. 1, 2013, pp. 2–8.
- [7] J. J. Dai *et al.*, "BigDL: A distributed deep learning framework for big data," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 50–60.
- [8] R. Han *et al.*, "SlimML: Removing non-critical input data in large-scale iterative machine learning," in *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2019.2951388.
- [9] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Conf. Operating Syst. Des. Implementation*, vol. 14, 2014, pp. 583–598.
- [10] A. Harlap, A. Tumanov, A. Chung, G. R. Ganger, and P. B. Gibbons, "Proteus: Agile ML elasticity through tiered reliability in dynamic resource markets," in *Proc. 12th Eur. Conf. Comput. Syst.*, 2017, pp. 589–604.
- [11] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," 2017, *arXiv:1710.06952*.
- [12] C. Yu, H. Tang, C. Renggli, S. Kassing, A. Singla, D. Alistarh, C. Zhang, and J. Liu, "Distributed learning over unreliable networks," 2018, *arXiv:1810.07766*, 826.
- [13] J. Konecny, H. B. McMahan, D. Ramage, and P. Richtarik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016, *arXiv:1610.02527*.
- [14] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, 12, pp. 1–19, 2019.
- [15] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Trans. Inf. Theory*, vol. 14, no. 6, pp. 2508–2530, Jun. 2006.
- [16] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
- [17] J. Dailly, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, "GossipGraD: Scalable deep learning using gossip communication based asynchronous gradient descent," 2018, *arXiv:1803.05880*.
- [18] M. Blot, D. Picard, N. Thome, and M. Cord, "Distributed optimization for deep learning with gossip exchange," *Neurocomputing*, vol. 330, pp. 287–296, 2019.
- [19] "Pytorch," 2020. [Online]. Available: <https://pytorch.org/>
- [20] S. Ghadimi, G. Lan, and H. Zhang, "Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization," *Math. Program.*, vol. 155, pp. 267–305, 2016.
- [21] Q. Ho, J. Cipar, H. Cui, J. K. Kim, and E. P. Xing, "More effective distributed ML via a stale synchronous parallel parameter server," *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 1223–1231.
- [22] H. R. Feyzmahdavian, A. Aytakin, and M. Johansson, "An asynchronous mini-batch algorithm for regularized stochastic optimization," *IEEE Trans. Autom. Control*, vol. 61, no. 12, pp. 3740–3754, Dec. 2016.
- [23] H. Wang, D. Niu, and B. Li, "Turbo: Dynamic and decentralized global analytics via machine learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1372–1386, Jun. 2020.
- [24] Q. Luo, J. Lin, Y. Zhuo, and X. Qian, "Hop: Heterogeneity-aware decentralized training," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2019, pp. 893–907.
- [25] Q. Luo, J. He, Y. Zhuo, and X. Qian, "Prague: High-performance heterogeneity-aware asynchronous decentralized training," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2020, pp. 401–416.
- [26] M. Blot, D. Picard, M. Cord, and N. Thome, "Gossip training for deep learning," 2016, *arXiv:1611.09726*.
- [27] I. Colin, A. Bellet, J. Salmon, and S. Clemencon, "Gossip dual averaging for decentralized optimization of pairwise functions," in *Proc. 33rd Int. Conf. Int. Conf. Mach. Learn.*, 2016, pp. 1388–1396.
- [28] A. Harlap *et al.*, "Addressing the straggler problem for iterative convergent parallel ML," in *Proc. 7th ACM Symp. Cloud Comput.*, 2016, pp. 98–111.
- [29] S. Pumma, M. Si, W. Feng, and P. A. Balaji, "Scalable deep learning via I/O analysis and optimization," *ACM Trans. Parallel Comput.*, vol. 6, no. 2, pp. 1–34, 2019.

- [30] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.
- [31] M. Boehm *et al.*, "Hybrid parallelization strategies for large-scale machine learning in systemML," in *Proc. VLDB Endowment*, vol. 7, no. 7, pp. 553–564, 2014.
- [32] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 265–283.
- [33] A. Kumar, J. Naughton, and J. M. Patel, "Learning generalized linear models over normalized data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1969–1984.
- [34] A. Elgohary, M. Boehm, P. J. Haas, F. R. Reiss, and B. Reinwald, "Compressed linear algebra for large-scale machine learning," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 960–971, 2016.
- [35] C. H. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulhaja, "Arabesque: A system for distributed graph mining," in *Proc. 25th Symp. Operating Syst. Princ.*, 2015, pp. 425–440.
- [36] M. Owaida, H. Zhang, C. Zhang, and G. Alonso, "Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms," in *Proc. IEEE 27th Int. Conf. Field Programmable Logic Appl.*, 2017, pp. 1–8.
- [37] P. Watcharapichat, V. L. Morales, R. Fernandez, and P. R. Pietzuch, "Ako: Decentralised deep learning with partial gradient exchange," in *Proc. 7th ACM Symp. Cloud Comput.*, 2016, pp. 84–97.
- [38] I. Gouri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen, "Approxhadoop: Bringing approximations to mapreduce frameworks," in *Proc. Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2015, pp. 383–397.
- [39] I. Hubara, M. Courbariaux, D. Soudry, R. Elyaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 187, pp. 1–30, 2016.
- [40] G. Gorrell, "Generalized hebbian algorithm for incremental singular value decomposition in natural language processing," in *Proc. EACL*, 2006, vol. 6, pp. 97–104.
- [41] "Cifar-10 database," 2009. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [42] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," *Proc. Int. Conf. Mach. Learn.*, pp. 2525–2534, 2018.
- [43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
- [44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
- [45] "Mnist handwritten digit database," 1998. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [46] "Downsampled imageNet datasets," 2016. [Online]. Available: <http://image-net.org/download-images>
- [47] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size," 2016, *arXiv:1602.07360*.
- [48] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," in *Proc. Comput. Vis. Pattern Recognit.*, 2017, pp. 1–9.
- [49] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 437–478.



Rui Han received the MSc degree with honor from Tsinghua University, China in 2010, and the PhD degree from the Department of Computing, Imperial College London, UK, in 2014. He is currently an associate professor with the School of Computer Science and Technology, Beijing Institute of Technology, China. His research interests are system optimization for deep learning workloads. He has more than 40 publications in these areas, including papers at *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Computers*, *INFOCOM*, and *ICDCS*.



Shilin Li is currently working toward the graduate degree with the School of Computer Science and Technology, Beijing Institute of Technology. His work focuses on federated learning, optimization of big data system for machine learning, and deep learning workloads.



Xiangwei Wang is currently working toward the graduate degree with the School of Computer Science and Technology, Beijing Institute of Technology. His work focuses on decentralized training of deep learning models in edge computing platforms.



Chi Harold Liu (Senior Member, IEEE) received the BEng degree from Tsinghua University, Beijing, China, and the PhD degree from the Imperial College London, London, UK. He is currently a full professor and the vice dean with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing. Before that, he worked for IBM Research - China and Deutsche Telekom Laboratories, Berlin, Germany, and IBM T. J. Watson Research Center, USA. He is currently an associate editor for *IEEE Trans. Network Science and Engineering*. His current research interests include the big data analytics, mobile computing, and deep learning. He is a fellow of IET and a fellow of Royal Society of the Arts.



Gaofeng Xin is currently working toward the graduate degree with the School of Computer Science and Technology, Beijing Institute of Technology. His work focuses on decentralized machine learning and deep learning workloads in edge computing platforms.



Lydia Y. Chen received the BA degree from National Taiwan University, and the PhD degree from the Pennsylvania State University. She is currently an associate professor with the Department of Computer Science at the Technology University Delft. Prior to joining TU Delft, she was a research staff member at IBM Zurich Research Lab from 2007 to 2018. Her research interests center around dependability management, resource allocation and privacy enhancement for large scale data processing systems and services. She has published more than 80 papers in journals, e.g., *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Services Computing*, and conference proceedings, e.g., *INFOCOM*, *Sigmetrics*, *DSN*, and *Eurosys*.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.