electronics

MDPI

Article

BACombo—Bandwidth-Aware Decentralized Federated Learning

Jingyan Jiang [1], Liang Hu [1], Chenghao Hu [2], Jiate Liu [2] and Zhi Wang [2,3]*

[1] College of Computer Science and Technology, Jilin University, Changchun 130012, China; jiangjy14@mails.jlu.edu.cn (J.J.); hul@jlu.edu.cn (L.H.)
[2] Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China; huch16@mails.tsinghua.edu.cn (C.H.); liujt19@mails.tsinghua.edu.cn (J.L.)
[3] Peng Cheng Laboratory, Shenzhen 518066, China
* Correspondence: wangzhi@sz.tsinghua.edu.cn

Received: 2 February 2020; Accepted: 29 February 2020; Published: 5 March 2020

Abstract: The emerging concern about data privacy and security has motivated the proposal of federated learning. Federated learning allows computing nodes to only synchronize the locally-trained models instead of their original data in distributed training. Conventional federated learning architecture, inherited from the parameter server design, relies on highly centralized typologies and large nodes-to-server bandwidths. However, in real-world federated learning scenarios, the network capacities between nodes are highly uniformly distributed and smaller than that in data centers. As a result, how to efficiently utilize network capacities between computing nodes is crucial for conventional federated learning. In this paper, we propose Bandwidth Aware Combo (BACombo), a model segment level decentralized federated learning, to tackle this problem. In BACombo, we propose a segmented gossip aggregation mechanism that makes full use of node-to-node bandwidth for speeding up the communication time. Besides, a bandwidth-aware worker selection model further reduces the transmission delay by greedily choosing the bandwidth-sufficient worker. The convergence guarantees are provided for BACombo. The experimental results on various datasets demonstrate that the training time is reduced by up to 18 times that of baselines without accuracy degrade.

Keywords: decentralized machine learning; federated learning; system for machine learning

1. Introduction

Recent years have witnessed a rapid growth of deep learning algorithms which achieve and even transcend the human-level performance on nature language processing and computer vision [1,2]. Deep learning models often requires larger datasets to achieve better performance. While with the popularity of the IoT devices and edge computing, data is often collected outside the datacenter, and processed by distributed devices, such as sensors, smart phones. Thus, federated learning (FL) [3] is proposed to address key challenges such as data privacy, heavy communication, by allowing workers to train models using distributed data, and exchanging the model updates (e.g., gradients or parameters, etc.) instead of raw data.

A general federated learning system uses a central parameter server to coordinate the large federation of the participating workers. The workers train a local model with their dataset and send the model updates periodically to a centralized server for synchronization. To reduce the risk of single point failure, a couple of decentralized synchronization methods have been proposed. All-reduce [4] adopts an all-to-all scheme, that is, each worker sends the local model updates to all other workers. It achieves the same synchronization effect as a parameter server but consumes much bandwidth

resources between works. When the model updates from all nodes in the system are sent to all other nodes, the performance is highly degraded. To reduce the transmission cost, gossip based model synchronization [5,6] is proposed—workers send local updates to only one or a group of selected workers.

In real-world federated learning scenarios, the network capacities between nodes are highly uniformly distributed and smaller than that in a datacenter [7]. Thus, it is still extremely bandwidth costly when workers send the full model updates (e.g., the size can be up to 1360MB in $BERT_{LARGE}$ [1]). An intuitive question is then, is it possible for workers to synchronize the model partially, from/to only a part of the workers, and still achieve good convergence performance?

Our answer to this question is a novel decentralized, federated learning design, introducing a segmented gossip approach and the bandwidth-aware worker selection, which not only makes full utilization of sufficient node-to-node bandwidth by transmitting model segmentations in a peer-to-peer manner, but also has good training convergence. In particular, the details of the design and the contributions are summarized as follows.

- First, we propose a model segmentation level synchronization mechanism. We "split" a model into a set of segmentations—subsets that contain the same number of model parameters that are not overlapped with each other. Workers perform segmentation level updates by aggregating a local segmentation with the corresponding segmentation from k other workers. Based on our analysis, k can be much smaller than the number of all workers, to achieve better convergence.
- Second, we propose a decentralized, federated learning design, borrow the idea from gossip protocol; each worker stochastically selects a few workers to transfer the model segment for each training iteration. Our objective is to maximize the utilization of bandwidth capacities between workers and split the bill of communication cost. To improve the convergence performance of our solution, we introduce "Model Replica" to guarantee enough information from different workers. The theory analysis proves that our solution has convergence property.
- Third, to speed up the convergence rate further, we propose a bandwidth-aware worker selection method, taking advantage of the epsilon-greedy algorithm, the workers monitor the average bandwidth between peers over time and select the fast peers to transmit the segments with high probability.
- Finally, we implement the segmented gossip and bandwidth-aware worker selection strategy into a prototype called BACombo, and design experiments to evaluate its performance. Our results show that our design significantly reduces (up to 18×) the training time in practical network topology and bandwidth setup, without accuracy degrade.

2. Related Work

2.1. Distributed ML

Conventional distributed machine learning (ML) systems are centralized, and workers periodically send the local updates to an (a set of) parameter servers (PS) such as SparkNet [8], Tensorflow [9] and traditional federated learning systems [10,11]. To avoid bottleneck and single point failure [12,13], aim to scale PS for better network utilization. Although these scaling methods could increase the accumulative bandwidth at the server side, they are still suffering the long convergence time when the network is poor.

An alternative solution is a decentralized architecture; the workers exchange updates directly using all-reduce scheme, with communication cost $O(n^2)$ for n workers. To reduce the huge communication costs, an intuitive approach is to take the advantage of topology. Baidu first introduced Ring-allreduce [14], which is a bandwidth-optimal way to do an allreduce. The workers involved are arranged in a ring, each worker sends gradients to the next clockwise worker and receives from the previous one. In this way, it reduces the communication complexity to linear growth in scale. similarly,

the tree [15] and graph [16] topologies are proposed to reduce the communication cost. However, these approaches may need multiple hops between workers, resulting in slow convergence. Instead of the topology-based method, Ako [17] proposes a partial gradient updates method. In each synchronization round, each worker sends a gradient partition to every other worker. Ako reduces the synchronization time, and the communication overhead depends on the partition size and the worker number.

Although these existing approaches perform well in distributed ML, they aggregate gradients every epoch, which still face high communication cost and is not practical in federated learning with slow internet connections.

2.2. Communication Efficient FL

The main research focus of federated learning is to reduce communication cost. Reference [11] propose structured updates and sketched updates to reduce the exchange data size at the cost of accuracy loss. Reference [3] proposes the federated averaging algorithm (FedAvg) to reduce the parameter updates significantly. FedAvg aggregates parameters after several epochs. In each synchronization round, it selects a fraction of workers and computes the gradient of the loss over all the data held by these workers. These methods are based on the PS architecture, which faces the network congestion when the updates arrive at the PS concurrently.

2.3. Gossip Protocol in ML

The gossip protocol is widely used in distributed systems [6,18]; each worker sends out the message to a set of other workers, the message propagates through the whole network worker by worker. Reference [19] first introduced the gossip protocol in deep learning. They propose GoSGD, using sum-weight gossip protocol to share the updates with selective workers. The results show good consensus convergence properties. Reference [5] proposed GossipGraD, which is a gossip-based SGD algorithm for large scale deep learning systems and reduces the communication complexity to $O(1)$.

However, in federated learning, network connections between geo-distributed workers usually could not be fully utilized because of the bottleneck, which are ignored in these approaches.

3. Proposed Framework: `BACombo`

In this section, we introduce BACombo, a decentralized federated learning framework. We begin by describing BACombo's two key components: (1) Segmented Gossip Aggregation and (2) Bandwidth-Aware Worker Selection, and then introduce the algorithm.

In the distributed federated learning scenarios, we consider the network topology with N workers, each worker has a network connection among the other $N-1$ workers. The goal of each worker i is to minimize the following objective function:

$$\min_{\mathcal{W}_i} F_i(\mathcal{W}) = \sum_{k}^{N} p_k F_k(\mathcal{W}), \tag{1}$$

where N is the total number of workers, $p_k \geq 0$ and $\sum_k p_k = 1$. The objective of each worker $F_k(\mathcal{W})$ can be defined by empirical risk over their local data, i.e., $F_k(\mathcal{W}) = \frac{1}{n_k} \sum_{j_k}^{n_k} f_{j_k}(\mathcal{W})$, where n_k is the number of samples of worker. We set p_k to be $\frac{n_k}{n}$, where $n = \sum_k n_k$ is the number of samples over the entire distributed dataset.

3.1. Segmented Gossip Aggregation

An all-reduce worker pushes $N-1$ local model replicates to the other workers through $N-1$ links while a gossip worker is expected to push one local model replicate out through only one link. Within a datacenter where the workers are connected by the local area network, they can always communicate with each other at maximum bandwidth thus, the gossip worker can achieve great speed up as the transmission size is drastically reduced.

However, in the federated learning context where the workers are geo-distributed, the real bandwidth between the workers is typically small due to the potential bottleneck of WAN. Thus the traditional gossip-based schemes can not make full use of the worker's bandwidth because the transmissions are limited in one or a few links. We propose the Segmented Gossip Aggregation to solve this problem by "splitting" the transmission task and feeding them into more links.

3.1.1. Segmented Pulling

Figure 1a illustrates the transfer procedure with segmented gossip aggregation which we name it segmented pulling. In the aggregation phase, the worker needs to receive the model parameters from others. While the naive gossip-based synchronization schemes require the worker to collect the whole model parameters, segmented pulling allows the worker to pull different parts of the model parameters from different workers and rebuild a mixed model for aggregation.

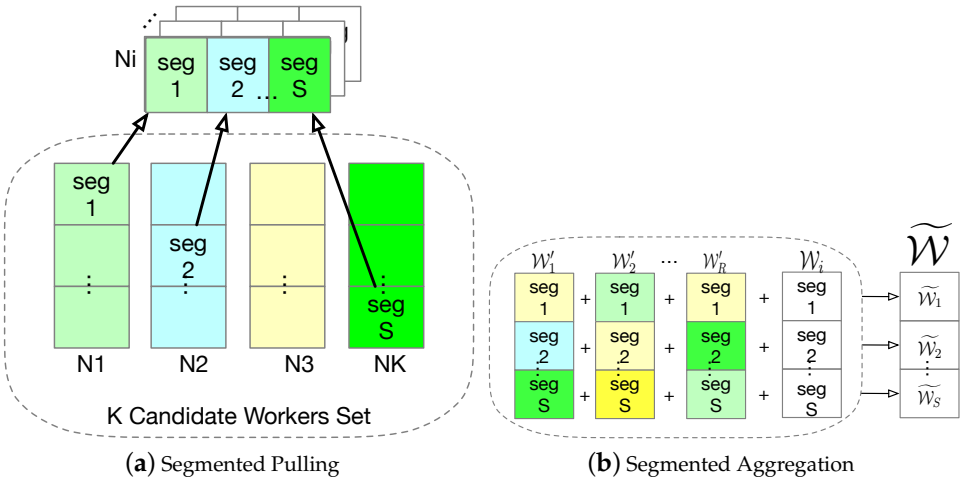

Figure 1. Segmented Gossip Aggregation.

Let $\mathcal{W}$ denote the model parameters. The worker firstly breaks the structure of $\mathcal{W}$ into S segments without overlapping such that

$$\mathcal{W} = (\mathcal{W}[1], \mathcal{W}[2], \ldots, \mathcal{W}[S]). \tag{2}$$

For each segment l, the worker chooses a peer worker which we denote it as j_l and then actively pulls the corresponding segment $\mathcal{W}_{j_l}[l]$ from it. Note that this step is parallelized to make full use of the bandwidth. When the worker fetches all the model segments back, a new mixed model $\mathcal{W}'$ can be rebuilt from the segments such that

$$\mathcal{W}' = (\mathcal{W}_{j_1}[1], \mathcal{W}_{j_2}[2], \ldots, \mathcal{W}_{j_l}[S]). \tag{3}$$

The naive gossip-based scheme pulls all the segments from a single peer worker. However, with segmented pulling, if we choose a different peer for each segment, the total transmission size is still equal to one model, like the naive gossip-based schemes, but the traffic is dissolved among not one but S links.

3.1.2. Model Replica

In traditional distributed ML scenario within the datacenter, the gossip-based solutions can choose only one other worker for aggregation but still achieve excellent convergence, because the workers "gossip" with each other frequently such that the update of each worker are propagated through the whole network before they become too stale [5]. However, for communication efficient FL systems,

the staleness of the model updates is hard to bound as the models are trained separately for up to a few epochs.

Thus as a compromise, we set a hyper-parameter model replica R, which represents the number of the mixed models gathered by segmented pulling. To rebuild R mixed models, the worker will pull $S \times R$ segments from peers. Thus increasing the value of R means more segments have to be transferred through the network, which may cause bandwidth overhead. But this is necessary to accelerate the propagation and ensure the model quality. Since there is no centralized server bottleneck, the model training speed could still be faster even with the extra transmission.

3.1.3. Segmented Aggregation

Typically the model aggregation uses weighted averaging of the received model parameters with the worker's dataset size as weight. But in segmented gossip aggregation, the mixed models are patched together from different workers, so it is hard to set a reasonable weight for the mixed model as a whole. For such a case, we use a segment-wise model aggregation.

Assume the worker i has fetched all the segments and rebuilt R mixed models which we represent as $\mathcal{W}'_1, \mathcal{W}'_2, \ldots, \mathcal{W}'_R$. Then for each segment l, we have R mixed models and one local model to aggregate. Let P_l denote the set of the workers which provide the segment l (worker i itself is contained too) and $|D_j|$ denote the dataset size of worker j, then we can aggregate segment l by:

$$\widetilde{\mathcal{W}}[l] = \frac{\sum_{j \in P_l} |D_j| \mathcal{W}'_j[l]}{\sum_{j \in P_l} |D_j|}. \tag{4}$$

Combine all the aggregated segments, and we can rebuild the final aggregation result by

$$\widetilde{\mathcal{W}} = (\widetilde{\mathcal{W}}[1], \widetilde{\mathcal{W}}[2], \ldots, \widetilde{\mathcal{W}}[S]), \tag{5}$$

and then the worker can continue its training until next aggregation phase comes.

3.2. Bandwidth-Aware Worker Selection

The Wide Area Network (WAN) bandwidth not only varies significantly between different regions (e.g., up to 12× of bandwidth variance between geographically-close regions and distant regions), but also is time-varying (e.g., up to 10× of bandwidth variance within a day), as shown by other systems [20,21]. Thus, to reduce the transmission time, we tend to select the workers with faster bandwidth. However, we are unaware of the time-varying bandwidth of each peer workers. We seek to design a multi-armed bandit based online learning algorithm to estimate the bandwidth over time and make worker selection decisions accordingly. In our design, for each worker, the arm is the peer worker to pull segments from. At each communication iteration, we decide the worker selection decisions, that is, a set of arms to pull. The bandwidth of selected workers can be estimated during the transmission.

Although the existing multi-armed bandit algorithms, such as UCB [22] and EXP3 [23], achieve a good performance of exploring and exploiting trade-off. While in our problem, due to the stochastic nature of the WAN bandwidth, we could observe a noisy measurement; besides, the worker also needs to explore enough parameters from peer workers. Hence each worker should make some explorations to pull again workers who have estimated the small bandwidth. On the other hand, each worker should not explore too much either, and we may slow down the total training time by introducing the network congestion when choosing the peer workers with small bandwidth.

Therefore, we take epsilon-greedy [22] at each communication round. Each worker decides whether to explore the network conditions or exploit the peer workers with the best bandwidth with a probability ϵ. To explore the network means to randomly select workers channels while exploiting

network means to select the workers greedy. By doing so, we exploit large bandwidth while also exploring the bandwidth distribution by pulling random peer workers.

3.3. Algorithm

To put the components above together, we now introduce our proposed framework, BACombo as Algorithm 1. The initial model parameter of each is the same, and denoted as $\mathcal{W}_0$. At any iteration t, each worker first conducts local training to obtain the local update $\mathcal{W}_{t+1,i}$. Note that, the parameter after segment aggregation is denoted as $\widetilde{\mathcal{W}}_{t,i}$ at $t-1$ iteration. Then the worker decides whether to sample workers at this iteration greedily; each worker shares the same seed t, which means each worker makes the same decision at any iteration. After worker selection, each worker pulls and aggregates segments based on the selection results P, to obtain the new aggregated update $\widetilde{\mathcal{W}}_{t+1,i}$.

Algorithm 1 Bandwidth-Aware Combo (BACombo)

Input: $N, T, \eta, \tau, \mathcal{W}_0, N, R, \epsilon$

Each worker i **executes:**
$B \leftarrow \mathbf{0}$
for $t = 0, \ldots, T-1$ **do**
  $r_t \leftarrow$ `UniformRandom`(t)
  updates $\widetilde{\mathcal{W}}_{t,i}$ for τ epochs of SGD on F_i with step size η to obtain $\mathcal{W}_{t+1,i}$
  if $r_t < \epsilon$ **then**
    $P \leftarrow$`RandomWorkerSelection`(R, N, i)
    update bandwidth table B based on the `BandwidthPrediction`
  else
    $P \leftarrow$`BandAwareWorkerSelection`(R, N, i)
  end if
  $\widetilde{\mathcal{W}}_{t+1,i} \leftarrow$ `SegGossipAggregation` $(i, \mathcal{W}_{t+1,i}, S, R, P, \eta)$
end for

`SegGossipAggregation`$(i, \mathcal{W}_{t+1,i}, S, R, P, \eta)$ //Run on worker i
  worker i splits the local update into S segments based on Equation (2) and pulls $S \times R$ segments from P
  worker i aggregates the segments Equations (4) and (5) to obtain $\widetilde{\mathcal{W}}_{t+1,i}$
return $\widetilde{\mathcal{W}}_{t+1,i}$

4. Convergence Analysis

Generally, the deep learning uses a gradient descent algorithm to find the model parameters that minimize a user-defined loss function which we denote it as $F(\mathcal{W})$. For the loss function, we make the following assumptions.

Assumption 1 (Loss function). *$F(\mathcal{W})$ is a convex function with bounded second derivative such that*

$$\mu \leq ||\nabla^2 F(\mathcal{W})|| \leq L. \tag{6}$$

In a centralized learning system, the model parameters are updated with the gradient $\nabla F(\mathcal{W})$ calculated from the whole dataset. But with the federated settings, the worker i updates the model with the gradient of a subset of data and we denote it as $\nabla F_i(\mathcal{W})$. To capture the divergence of these two gradients, we make the next definition.

Define 1 (**Gradient Divergence**). *For any worker i and model parameter $\mathcal{W}$, We define δ as the upper bound of the divergence between local and global gradients.*

$$||\nabla F_i(\mathcal{W}) - \nabla F(\mathcal{W})|| \leq \delta. \tag{7}$$

For a worker i in our proposed system, at iteration t, the local model parameter $\mathcal{W}_{t,i}$ is an aggregation result of the local model and a few mixed models rebuilt from segments. As a contrast, we denote $\mathcal{W}_t$ as the aggregation result of all the nodes, which is the output of *FedAvg* algorithm. Like the gradient divergence, we define aggregation divergence to measure the aggregation result.

Define 2 (**Aggregation Divergence**). *For any worker i at iteration t, we define ρ as the upper bound of the divergence between partial and global aggregation.*

$$||\mathcal{W}_{t,i} - \mathcal{W}_t|| \leq \rho. \tag{8}$$

With the above assumption and definitions, we can present the convergence result of BACombo.

Theorem 1. *Let $\mathcal{W}^*$ denote the global optimum and $\mathcal{W}_0$ denote the initial model parameters, worker i performs gradient descent on local dataset for τ times with learning rate $\alpha \leq \frac{1}{L}$ and then pulls the segments to aggregate, the aggregation result is $\mathcal{W}_{t,i}$, the convergence upper bound of BACombo is given by*

$$||\mathcal{W}_{t,i} - \mathcal{W}^*|| \leq \theta^{t\tau}||\mathcal{W}_0 - \mathcal{W}^*|| + (1 - \theta^{t\tau})[\frac{\rho}{1 - \theta^{\tau}} + \frac{\alpha\delta}{1 - \theta}], \tag{9}$$

where $\theta = 1 - \alpha\mu$.

Proof. Based on the Definition 2, we could have:

$$\begin{aligned} \|\mathcal{W}_{t,i} - \mathcal{W}^*\| &= \|\mathcal{W}_{t,i} - \mathcal{W}_t + \mathcal{W}_t - \mathcal{W}^*\| \\ &\leq \|\mathcal{W}_{t,i} - \mathcal{W}_t\| + \|\mathcal{W}_t - \mathcal{W}^*\| \\ &\leq \|\mathcal{W}_t - \mathcal{W}^*\| + \rho. \end{aligned} \tag{10}$$

Then based on the definition of $\mathcal{W}_t$ and $\mathcal{W}^*$, we have:

$$\begin{aligned} \|\mathcal{W}_t - W^*\|_1 &= \|\sum_{j\in N} \frac{|D_j|\widetilde{\mathcal{W}_{t,j}}}{|D|} - \mathcal{W}^*\|_1 \\ &= \|\sum_{j\in N} \frac{|D_j|\widetilde{\mathcal{W}_{t,j}}}{|D|} - \sum_{j\in N} \frac{|D_j|\mathcal{W}^*}{|D|}\|_1 \\ &= \|\sum_{j\in N} \frac{|D_j|}{|D|}(\widetilde{\mathcal{W}_{t,j}} - \mathcal{W}^*)\|_1 \\ &\leq \sum_{j\in N} \frac{|D_j|}{|D|}\|\widetilde{\mathcal{W}_{t,j}} - \mathcal{W}^*\|_1 \\ &\leq \|\widetilde{W_{t,\hat{j}}} - \mathcal{W}^*\|_1, \end{aligned} \tag{11}$$

where $\hat{j} = argmax \sum_{j\in N} \frac{|D_j|}{|D|}\|\widetilde{\mathcal{W}_{t,\hat{j}}} - \mathcal{W}^*\|_1$. We denote $\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau)$ as the parameter $\widetilde{W_{t,\hat{j}}}$ after τth local SGD passing at iteration t, note that, it equals to $\widetilde{W_{t+1,\hat{j}}}$, and $\widetilde{\mathcal{W}_{t,\hat{j}}}(0)$ equals to $\widetilde{\mathcal{W}_{t-1,\hat{j}}}$. Thus based on the SGD, we have:

$$\begin{aligned}
\|\widetilde{\mathcal{W}_{t+1,\hat{j}}} - W^*\|_1 &= \|\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau-1) - \alpha\nabla f_{\hat{j}}(\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau-1)) - W^* + \alpha\nabla f(W^*)\|_1 \\
&= \|\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau-1) - \alpha\nabla f_{\hat{j}}(\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau-1)) + \alpha\nabla f(\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau-1)) - \alpha\nabla f(\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau-1) - \mathcal{W}^* + \alpha\Delta f(\mathcal{W}^*)\|_1 \\
&= \|\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau-1) - \mathcal{W}^* - \alpha(\nabla f(\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau-1)) - \nabla f(\mathcal{W}^*))\|_1 + \alpha\delta \\
&\leq \theta\|\widetilde{\mathcal{W}_{t,\hat{j}}}(\tau-1) - \mathcal{W}^*\|_1 + \alpha\delta \\
&\leq \theta^\tau\|\widetilde{\mathcal{W}_{t,\hat{j}}}(0) - \mathcal{W}^*\|_1 + \alpha\delta\frac{1-\theta^\tau}{1-\theta} \\
&\leq \theta^\tau\|\widetilde{\mathcal{W}_{t,\hat{j}}} - \mathcal{W}^*\|_1 + \alpha\delta\frac{1-\theta^\tau}{1-\theta},
\end{aligned} \tag{12}$$

where $\theta = |1 - \alpha\mu|$. Based on Equations (10)–(12), we have:

$$\begin{aligned}
\|\mathcal{W}_{t,i} - W^*\| &\leq \|\mathcal{W}_t - \mathcal{W}^*\|_1 + \rho \\
&\leq \|\widetilde{\mathcal{W}_{t,\hat{j}}} - \mathcal{W}^*\|_1 + \rho \\
&\leq \theta^\tau\|\widetilde{\mathcal{W}_{t-1,\hat{j}}} - \mathcal{W}^*\|_1 + \alpha\delta\frac{1-\theta^\tau}{1-\theta} \\
&\leq ||\mathcal{W}_0 - \mathcal{W}^*|| + (1-\theta^{t\tau})[\frac{\rho}{1-\theta^\tau} + \frac{\alpha\delta}{1-\theta}].
\end{aligned} \tag{13}$$

□

Note that this bound is characteristic of stochastic gradient descent bounds that it converges to within a noise ball around the optimum rather than approaching it. The gap between the output and optimum comes from two parts—the gradient divergence δ and the aggregation divergence ρ. The gradient divergence is related to the data distribution of each worker, which is the inherent drawback of the FL system.

According to the above inequality, the influence of ρ is exacerbated when the communication interval τ increases. The aggregation divergence can be ameliorated by aggregating more models from other workers. This explains why we set a hyper-parameter R to control the model replicas received from others. If we let $R = n - 1$, the worker aggregates all the external models and the model divergence decreases to zero. In this situation, BACombo degrades to the all-reduce scheme and has the same training result as the centralized way. However, we argue that the value of R can be much smaller and still maintains the training efficiency, which is validated in the evaluation.

5. BACombo Implementations

In this section, we firstly present the implementation details of BACombo, then discuss how it handles the dynamic nature of FL workers, and finally, we give a brief analysis of the convergence of BACombo.

5.1. Implementation Details

As a decentralized FL system, we focus on the design of the workers as the participation of the centralized server is not required during the training. However, it is important to notice that before the training starts, the server has to initialize the model parameters of each worker with the same value; otherwise, the training may fail to converge.

A BACombo worker follows a stateful training process, as illustrated by the numbered steps in Figure 2. At each iteration, the workers (1) update the model with local dataset and meanwhile, (2) send the segment pulling requests to other workers. Once the update is finished, they (3) send the segments to the requestors as a response of the pulling requests and when all the pulling requests are satisfied, the workers (4) aggregate the model segments and start the next iteration. Next, we describe the implementation details of these steps.

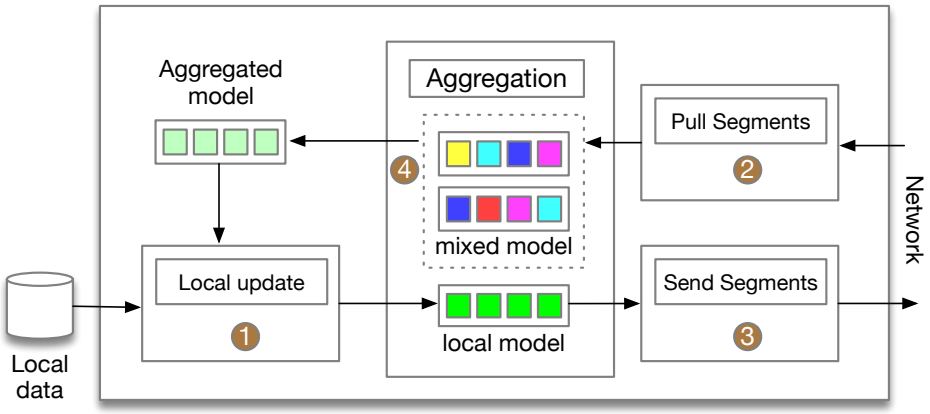

Figure 2. The architecture of BACombo workers.

(1) Local Update.

The learning process starts with the worker updating the model with the local dataset. The worker takes the aggregation result of the last iteration as the input model and updates it using stochastic gradient descent (SGD) with the local data. To reduce the communication cost, the local update may contain multiple SGD rounds before the communication with other workers. We denote the communication interval or the number of SGD rounds as τ, which, in typical FL systems, could be up to a few epochs.

(2) Worker Selection.

The selection has two main components: bandwidth monitor and worker selection. Each worker maintains a bandwidth table of all the other peer workers with default values, once a peer worker is picked in the communication, the monitor will measure the transmission time and estimate the average bandwidth based on the Round-trip delay time (RTT) and several (e.g., 5) bandwidth values in the history of this worker; In the selection part, each worker will select the workers follows the same scheme, either random selection or greedy selection.

(3) Segments Pulling.

The workers firstly decide how to partition the model. They do not have to follow the same partition rule, but for simplicity, we assume they partition the model into S segments in the same way. For each segment, the worker has to select R peers and sends the pulling request, which contains a segment description and a unique identifier of the worker to indicate which part of the model is to be sent and whom it suppose to be sent to.

Each worker has to send $S \times R$ segment pulling requests to the other workers, and BACombo tries to distribute these requests evenly among all the workers to engage more links and balance the transmission workload. Thus for each request, the target worker is randomly selected from all the other workers without replacement until there is no option left, which means when $S \times R \leq n$, all the segments come from different workers. Note that for each iteration, the pulling requests can be sent even before the local update starts; in this way, the target workers can send the segments immediately when the local model is ready.

(4) Segments Sending.

The sending procedure is a twin action of the segments pulling. When the worker finishes the local update, it is ready to send its update result to others. Rather than actively pushing the model, the worker only dispatches the model segments according to the received pulling requests.

(5) Model Aggregation.

While the worker is providing the model segments to others, it is also receiving the segments it has requested previously. The model aggregation phase is blocked until all the pulling requests are satisfied, then the worker aggregates the external model segments with the local model using (4) and puts the aggregated segments together to rebuild the model. With the aggregation result, the worker gets back to the first step and starts the next training iteration.

5.2. Dynamic Workers

In the context of federated learning, the participating workers are more likely to be mobile phones and embedded devices, which are often not connected to a power supply and stable network. Thus the workers in FL system are highly dynamic and unstable, and they can join and exit the federation at any time.

Traditional distributed systems adopt the heartbeat packet and time threshold to check the status of the workers. However, these methods are not applicable with the FL system for the next two reasons: (1) The server has to maintain the heartbeat connection with all the participating workers which limits the scalability of the system. (2) The computation times of each worker vary significantly due to the difference in the computing devices and network environment.

Fortunately, the design of BACombo allows us to solve this problem directly. If the worker exits accidentally, the pulling requests it sends to other workers can be canceled immediately when the target workers find it unreachable. For those workers who have requested segments from the offline worker, they can monitor the status of the target workers, and once they see the connection with the target worker is lost, they can mark it as offline, resend the request to another worker and stop pulling from the offline worker. If it is a false report due to the network fluctuation or the offline worker comes back, the offline flag can be removed as long as the communication is reestablished.

The participation of a new worker is relatively easy to handle. When a new worker comes to the federation, it first requests a worker list either from a server or an old worker. Then it pulls the segments and aggregates them as normal only without its own local model. With the aggregation result, it can start the training with its local dataset. When it sends the pulling requests to the target workers, the target worker adds the newcomer to the worker list. Since the new worker sends the pull requests to many workers in a single iteration, its existence will be quickly noticed by all other workers, and then the new worker successfully joins the federation.

6. Performance Evaluation

6.1. Experimental Setup

6.1.1. Datasets and Models

We use datasets and models from **LEAF** [24], an open-source benchmarking framework for federated settings, including six tasks. We summarized the statistics of datasets in Table 1. Additional details on the models and datasets are presented below.

- **Federated Extended MNIST (FEMNIST).** We study an image classfication problem on EMNIST dataset [25], which has 62-class. The federated version of EMNIST, called FEMNIST, split the dataset into different workers, that is, each worker has a corresponding writer of digits/characters in EMNIST. We create the FEMNIST dataset in **LEAF** by using command `./preprocess.sh -s iid -sf 0.05 -k 100 -t sample -tf 0.8`. The model used takes as input a 28×28 image, followed with two convolution layers and two dense layers, and the output is a class label between 0 and 61.
- **Synthetic.** We create a diverse set of synthetic datasets, with different task numbers, class number, and worker numbers. This dataset follows a similar set up in References [3,26]. The logistic

regression model takes as input a 60 dimension feature. (1) Synthetic-C10-W50: We generate the whole dataset with 5000 tasks, and sample the dataset using command `./preprocess.sh -s iid -sf 1.0 -k 5 -t sample -tf 0.8 -iu 0.001`, to have a 10 prediction classes model and 50 workers dataset. (2) Synthetic-C5-W10: We generate the whole dataset with 1000 tasks, and sample the dataset using command `./preprocess.sh -s iid -sf 1.0 -k 5 -t sample -tf 0.8 -iu 0.001`, to have a 5 prediction classes model and 10 workers dataset. (3) Synthetic-C5-W40: Similarly, we generate a 5 prediction classes model and 40 workers dataset. (4) Synthetic-C5-W80: We generate a 5 prediction classes model and 80 workers dataset.

Table 1. Statistics of Datasets.

Dataset	# Workers	# Param.	Samples/Worker	
			Mean	std
Femnist	35	26,414,840	1144.89	392.77
Synthetic-C5-W10	10	1220	10,755.30	0.46
Synthetic-C5-W40	40	1220	2688.82	0.38
Synthetic-C5-W80	80	1220	1344.41	0.49
Synthetic-C10-W50	50	2440	10,189.80	0.40

6.1.2. Experiment Implementation Details

- **Hardware Device** We simulate the distributed federated learning (each device performs local training and aggregation) on a server with 2 Intel(R) Xeon(R) E5-2650 v4 @ 2.20 GHz CPUs and 4 Nvidia 1080Ti GPUs.
- **Libraries** We implement all code in TensorFlow 1.14.0. The full details could be found at `https://github.com/ginger0106/BACombo`.
- **Hyperparameters.** We split each dataset randomly on each worker into 80% training set and 20% testing set. The learning rates for all the datasets are 0.004; the batch sizes for all the datasets are 10. For the BACombo, the default number of segments S is 8, the default number of replicas R is 5, the default value of epsilon e is 0.5. Each worker uses SGD as a local solver, and the default number of epochs is 1. The bandwidth capacity of each worker is set to 100 Mb/s. The link bandwidth (Mb/s) among workers are uniformly sampled from $\{0.2, 0.4, 0.8 \ldots, 7.8, 8\}$.

6.1.3. Baselines

We compare BACombo with several baselines distributed federated learning methods:

- **Gossip.** Gossip is a distributed version of FedAvg, that is, each worker act as an aggregation server and a local update worker at the same time. At each communication round, each worker randomly samples R peer workers and pull the updates, then conduct the weight averaging using updates transmitted and local updates as FedAvg.
- **Combo** Combo is a special case of BACombo. At each communication round, each worker randomly samples R peer workers and pull S updates segments, then conduct the weight averaging using segments transmitted and local updates as FedAvg.

6.1.4. Metrics

- **Time.** To evaluate the convergence speed, we measure the average time to train 100 iterations of all workers, which consists of local training time, updates/segments synchronizing time, and updates/segments transmission time.
- **Accuracy.** We also measure the average test accuracy of all workers at each synchronizing iteration.

6.2. Experiment Result

We now present the empirical results for `BACombo`, we first investigate the performance of convergence of our proposed approach and compare the end-to-end convergence speed with other baselines, and demonstrate the superior performance of BACombo. Then we present the impact of hyper-parameters in BACombo.

6.2.1. Convergence Speed

We first investigate the performance of convergence about BACombo compared with baselines. We present the whole training process over time, as illustrated in Figure 3, BACombo shows a good convergence performance as the traditional methods, BACombo will convergence at the same test accuracy (84%, 88% and 88% resp.) among all the datasets (FEMNSIT in Figure 3a, Synthetic-C10-W50 in Figure 3b and Synthetic-C5-W80 in Figure 3c resp.) At the same time, BACombo exhibits an obvious speedup (18×, 16×, and 10× resp. compared to Gossip) in the convergence among all the datasets (FEMNSIT, Synthetic-C10-W50, and Synthetic-C5-W80) as shown in Figure 3d.

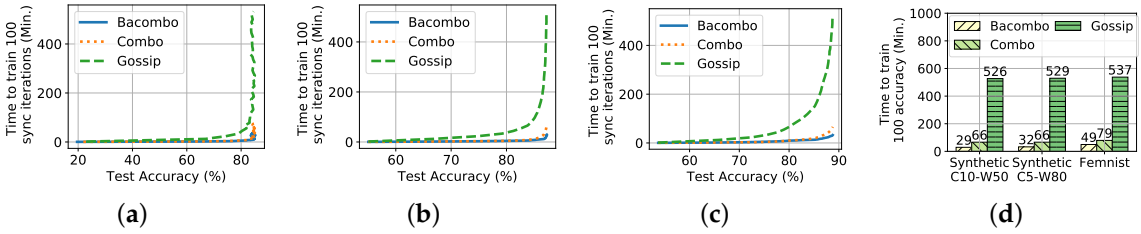

Figure 3. Convergence Speed. (**a**) Feminist, (**b**) Synthetic-C10-W50, (**c**) Synthetic-C5-W80, (**d**) Convergence time comparison.

6.2.2. Benefit of Model Segments

The speedup of decentralized approaches comes from the removal of the bottleneck of the centralized server, and the advantage of BACombo comes from the benefit of model segments. We measure convergence time with a different number of segments S ($S \in \{2, 4, 8, 10\}$) to investigate how model segments affect the training performance.

Compared with the naive gossip solution, BACombo aggregates mixed model parameters made up of multiple segments instead of the complete model. A potential concern is that the result may suffer degradation as the aggregation target is mottled and loses integrality. However, Figure 4a–c show that the accuracy of the aggregated results at each synchronization iterations is not affected by the model segments at all. Partitioning the model into ten segments ($S = 10$) has the same convergence trend as that without partition.

While the model segments do not affect the accuracy at each iteration, the synchronization time is significantly reduced. As illustrated in Figure 4d, by simply splitting the model parameters into four segments can reduce the synchronization time by half. This is because when $S = 4$, the original transmission quantity is divided into two parts and fed into 2× more links. When the bandwidth is not exhausted, the sending and receiving time can be reduced almost proportionally. However, when $S \geq 8$, the bandwidth is almost fully exploited, increasing the number of segments will not improve the time consumption then.

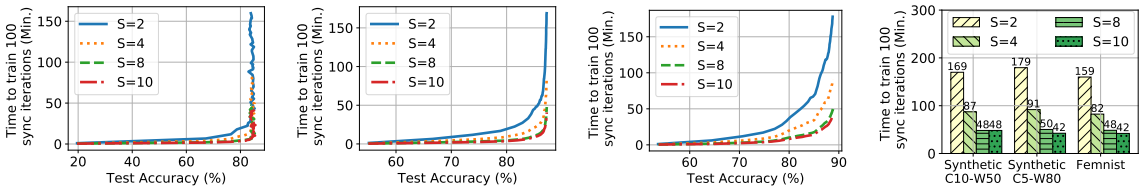

Figure 4. Benefit of model segments. (**a**) Feminist, (**b**) Synthetic-C10-W50, (**c**) Synthetic-C5-W80, (**d**) Convergence time comparison.

6.2.3. Impact of Model Replicas

Next, we evaluate the impact of model replica, which controls the overall information quantity that the workers send and receive at each synchronization iterations.

As illustrated in Figure 5, when the model parameters increase, the better the accuracy is for the larger replica. As we discussed in the convergence analysis of BACombo, the more information a worker receives, the better aggregation result it will get. When the worker receives all the model replicas from other peers, BACombo becomes the All-reduce structure and achieves the same training result as the centralized approach. While, if the model is too small, the larger replica will introduce more noises leading to a slightly slower in convergence.

We can observe from Figure 5d that when the number of replica increases, the convergence time decreases. It is because that when $R = 1$, at the greedy worker selection iteration, it will suffer from the congestion at the bandwidth abundant workers. However, the improvement is not unlimited. We can see that there is no significant gap between $R = 2$ and 4 in the convergence trend and result. This reflects the redundancy of All-reduce structure that the worker does not have to collect all the external models to train a high-quality model. In addition, as the bandwidth of workers is fully utilized with model segments, increasing R leads to the proportional growth of the transmission workload. Thus there exists a trade-off; a larger R increases the convergence rate on synchronization iterations but also the synchronization time.

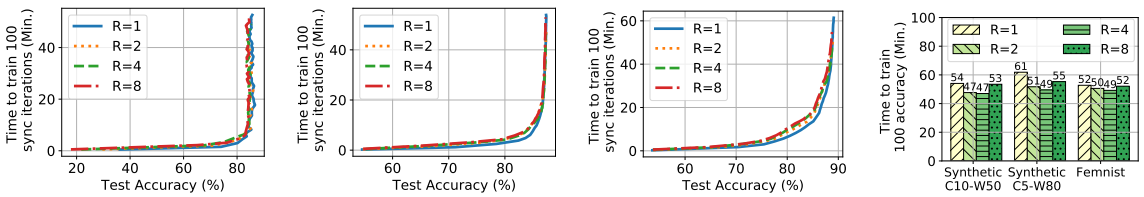

Figure 5. Benefit of model replicas. (**a**) Feminist, (**b**) Synthetic-C10-W50, (**c**) Synthetic-C5-W80, (**d**) Convergence time comparison.

6.2.4. Impact of epsilon

In this section, we investigate the impact of epsilon in bandwidth-aware worker selection. The larger epsilon implies a lower probability of pulling segments based on the bandwidth greedily. From Figure 6a–c, we could observe that the convergence performance is not affected by the change of epsilon among all the datasets. While, as shown in Figure 6d, as the increase of probability of greedy pulling (i.e., the decrease of epsilon), the convergence time will be reduced, that is, it achieves about 1.5× speedup when epsilon increases 2×.

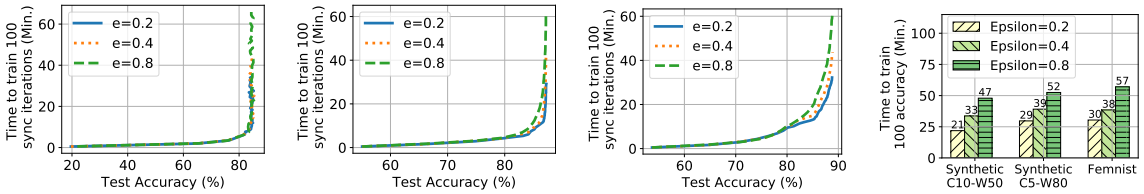

Figure 6. Benefit of model epsilon. (**a**) Feminist, (**b**) Synthetic-C10-W50, (**c**) Synthetic-C5-W80, (**d**) Convergence time comparison.

7. Conclusions

One of the most challenging problems of federated learning is the poor network connection as the workers are geo-distributed and connected with slow WAN. To avoid the drawback of network congestion in centralized parameter servers architecture, which is adopted in today's FL systems, we explore the possibility of decentralized FL solution, called BACombo. Taking the insight that the peer-to-peer bandwidth is much smaller than the worker's maximum network capacity, BACombo could fully utilize the bandwidth by saturating the network with segmented gossip aggregation. The experiments show that BACombo significantly reduces the training time and maintains a good convergence performance.

Author Contributions: Formal analysis, C.H.; Methodology, J.J.; Project administration, J.J.; Resources, Z.W.; Software, J.L.; Supervision, L.H. and Z.W.; Writing—original draft, J.J. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported in part by NSFC under Grant No. 61872215, SZSTI under Grant No. JCYJ20180306174057899, National Key R&D Plan of China under Grant No. 2017YFA0604500, and by National Sci-Tech Support Plan of China under Grant No. 2014BAH02F00, and by National Natural Science Foundation of China under Grant No. 61701190, and by Youth Science Foundation of Jilin Province of China under Grant No. 20160520011JH & 20180520021JH, and by Youth Sci-Tech Innovation Leader and Team Project of Jilin Province of China under Grant No. 20170519017JH, and by Key Technology Innovation Cooperation Project of Government and University for the whole Industry Demonstration under Grant No. SXGJSF2017-4, and by Key scientific and technological R&D Plan of Jilin Province of China under Grant No. 20180201103GX, Project of Jilin Province Development and Reform Commission No. 2019FGWTZC001

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2018**, arXiv:1810.04805.
2. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
3. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A.y. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
4. Patarasuk, P.; Yuan, X. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parall. Distrib. Comput.* **2009**, *69*, 117–124. [CrossRef]
5. Daily, J.A.; Vishnu, A.; Siegel, C.; Warfel, T.; Amatya, V.C. GossipGraD: Scalable Deep Learning using Gossip Communication based Asynchronous Gradient Descent. *arXiv* **2018**, arXiv:1803.05880.
6. Haas, Z.J.; Halpern, J.Y.; Li, L. Gossip-based ad hoc routing. In Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, New York, NY, USA, 23–27 June 2002; Volume 3, pp. 1707–1716.
7. Vulimiri, A.; Curino, C.; Godfrey, B.; Jungblut, T.; Padhye, J.; Varghese, G. Global analytics in the face of bandwidth and regulatory constraints. In Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), Santa Clara, CA, USA, 13 January 2015; pp. 323–336.

8. Moritz, P.; Nishihara, R.; Stoica, I.; Jordan, M.I. SparkNet: Training Deep Networks in Spark. In Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.
9. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A system for large-scale machine learning. In Proceedings of the Operating Systems Design and Implementation, 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
10. Konecny, J.; Mcmahan, H.B.; Ramage, D. Federated Optimization: Distributed Optimization Beyond the Datacenter. *arXiv* **2015**, arXiv:1511.03575.
11. Konečnỳ, J.; McMahan, H.B.; Yu, F.X.; Richtárik, P.; Suresh, A.T.; Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv* **2016**, arXiv:1610.05492.
12. Li, M.; Andersen, D.G.; Park, J.W.; Smola, A.J.; Ahmed, A.; Josifovski, V.; Long, J.; Shekita, E.J.; Su, B. Scaling distributed machine learning with the parameter server. In Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation, Broomfield, CO, USA, 6–8 October 2014; pp. 583–598.
13. Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konecny, J.; Mazzocchi, S.; McMahan, H.B.; et al. Towards Federated Learning at Scale: System Design. *arXiv* **2019**, arXiv:1902.01046.
14. Gibiansky, A. Bringing HPC Techniques to Deep Learning. Available online: http://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/ (accessed on 21 February 2017).
15. Li, H.; Kadav, A.; Kruus, E.; Ungureanu, C. MALT: Distributed data-parallelism for existing ML applications. In Proceedings of the Tenth European Conference on Computer Systems, Bordeaux, France, 21–25 April 2015; p. 3.
16. Agarwal, A.; Chapelle, O.; Dudik, M.; Langford, J. A reliable effective terascale linear learning system. *J. Mach. Learn. Res.* **2014**, *15*, 1111–1133.
17. Watcharapichat, P.; Morales, V.L.; Fernandez, R.; Pietzuch, P.R. Ako: Decentralised Deep Learning with Partial Gradient Exchange. In Proceedings of the Tenth European Conference on Computer Systems, Bordeaux, France, 21–24 April 2016; pp. 84–97.
18. Baraglia, R.; Dazzi, P.; Mordacchini, M.; Ricci, L. A peer-to-peer recommender system for self-emerging user communities based on gossip overlays. *J. Comput. Syst. Sci.* **2013**, *79*, 291–308. [CrossRef]
19. Blot, M.; Picard, D.; Cord, M.; Thome, N. Gossip training for deep learning. *arXiv* **2016**, arXiv:1611.09726v1.
20. Hsieh, K.; Harlap, A.; Vijaykumar, N.; Konomis, D.; Ganger, G.R.; Gibbons, P.B.; Mutlu, O. Gaia: Geo-distributed machine learning approaching {LAN} speeds. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), Boston, MA, USA, 27–29 March 2017; pp. 629–647.
21. Zhang, B.; Jin, X.; Ratnasamy, S.; Wawrzynek, J.; Lee, E.A. Awstream: Adaptive wide-area streaming analytics. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, Budapest, Hungary, 20–25 August 2018; pp. 236–252.
22. Auer, P.; Cesa-Bianchi, N.; Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* **2002**, *47*, 235–256. [CrossRef]
23. Auer, P.; Cesa-Bianchi, N.; Freund, Y.; Schapire, R.E. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.* **2002**, *32*, 48–77. [CrossRef]
24. Caldas, S.; Wu, P.; Li, T.; Konečnỳ, J.; McMahan, H.B.; Smith, V.; Talwalkar, A. Leaf: A benchmark for federated settings. *arXiv* **2018**, arXiv:1812.01097.
25. Cohen, G.; Afshar, S.; Tapson, J.; Van Schaik, A. EMNIST: Extending MNIST to handwritten letters. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; pp. 2921–2926.
26. Li, T.; Sahu, A.K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; Smith, V. Federated optimization in heterogeneous networks. *arXiv* **2018**, arXiv:1812.06127.