Journal of Parallel and Distributed Computing 148 (2021) 109–124

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Decentralized learning works: An empirical comparison of gossip learning and federated learning☆

István Hegedűs [a], Gábor Danner [a], Márk Jelasity [a,b,*]

[a] University of Szeged, Szeged, Hungary
[b] MTA SZTE Research Group on Artificial Intelligence, Szeged, Hungary

ARTICLE INFO

Article history:
Received 15 October 2019
Received in revised form 8 October 2020
Accepted 18 October 2020
Available online 4 November 2020

Keywords:
Federated learning
Gossip learning
Decentralized machine learning

ABSTRACT

Machine learning over distributed data stored by many clients has important applications in use cases where data privacy is a key concern or central data storage is not an option. Recently, federated learning was proposed to solve this problem. The assumption is that the data itself is not collected centrally. In a master–worker architecture, the workers perform machine learning over their own data and the master merely aggregates the resulting models without seeing any raw data, not unlike the parameter server approach. Gossip learning is a decentralized alternative to federated learning that does not require an aggregation server or indeed any central component. The natural hypothesis is that gossip learning is strictly less efficient than federated learning due to relying on a more basic infrastructure: only message passing and no cloud resources. In this empirical study, we examine this hypothesis and we present a systematic comparison of the two approaches. The experimental scenarios include a real churn trace collected over mobile phones, continuous and bursty communication patterns, different network sizes and different distributions of the training data over the devices. We also evaluate a number of additional techniques including a compression technique based on sampling, and token account based flow control for gossip learning. We examine the aggregated cost of machine learning in both approaches. Surprisingly, the best gossip variants perform comparably to the best federated learning variants overall, so they offer a fully decentralized alternative to federated learning.

1. Introduction

Performing data mining over data collected by edge devices, most importantly, mobile phones, is of very high interest [34]. Collecting such data at a central location has become more and more problematic in the past years due to novel data protection rules [1] and in general due to the increasing public awareness to issues related to data handling. For this reason, there is an increasing interest in methods that leave the raw data on the device and process it using distributed aggregation.

Google introduced *federated learning* to answer this challenge [22,25]. This approach is very similar to the well-known parameter server architecture for distributed learning [11] where worker nodes store the raw data. The parameter server maintains the current model and regularly distributes it to the workers who in turn calculate a gradient update and send it back to the server. The server then applies all the updates to the central model. This is repeated until the model converges. In federated learning, this framework is optimized so as to minimize communication between the server and the workers. For this reason, the local update calculation is more thorough, and compression techniques can be applied when uploading the updates to the server.

In addition to federated learning, *gossip learning* has also been proposed to address the same challenge [15,27]. This approach is fully decentralized, no parameter server is necessary. Nodes exchange and aggregate models directly. The advantages of gossip learning are obvious: since no infrastructure is required, and there is no single point of failure, gossip learning enjoys a significantly *cheaper scalability and better robustness*. A key question, however, is how the two approaches compare in terms of performance. This is the question we address in this work.

We compare the two approaches in terms of convergence time and model quality, assuming that both approaches utilize the same amount of communication resources in the same scenarios. In other words, we are interested in the question of whether—by communicating the same number of bits in the same time-window—the two approaches can achieve the same model quality. We train linear models using stochastic gradient descent

☆ This work was supported by the Hungarian Government and the European Regional Development Fund under the grant number GINOP-2.3.2-15-2016-00037 (“Internet of Living Things”), by grant TUDFO/47138-1/2019-ITM of the Ministry for Innovation and Technology, Hungary, and by the University of Szeged Open Access Fund, Hungary (Grant Number 4411).

* Corresponding author at: University of Szeged, Szeged, Hungary.
E-mail address: jelasity@inf.u-szeged.hu (M. Jelasity).

https://doi.org/10.1016/j.jpdc.2020.10.006

(SGD)
based on the logistic regression loss function.

Our experimental methodology involves several scenarios, including smartphone churn traces collected by the application Stunner [5]. We also vary the communication pattern (continuous or bursty) and the network size. In addition, we also evaluate different assumptions about the label distribution, that is, whether a given worker has a biased or unbiased subset of the training samples.

To make the comparison as fair as possible, we make sure that the two approaches differ mainly in their communication patterns. However, the computation of the local update is identical in both approaches. Also, we apply subsampling to reduce communication in both approaches, as introduced in [22] for federated learning. Here, we adapt the same technique for gossip learning. We also introduce a token account based flow control mechanism for gossip learning, for the case of bursty communication.

We note that both approaches offer mechanisms for explicit privacy protection, apart from the basic feature of not collecting data. In federated learning, Bonawitz et al. [7] describe a secure aggregation protocol, whereas for gossip learning one can apply the methods described in [8]. Here, we are concerned only with the efficiency of the different communication patterns and we do not compare security mechanisms.

The result of our comparison is that gossip learning is in general comparable to the centrally coordinated federated learning approach. This result is rather counter-intuitive and suggests that decentralized algorithms should be treated as first class citizens in the area of distributed machine learning overall, considering the additional advantages of decentralization.

This paper is a thoroughly revised and extended version of a conference publication [16]. The novel contributions relative to this conference publication are:

- A new evaluation scenario involving bursty traffic, in which nodes communicate in high bandwidth bursts. This is a very different scenario from continuous communication because here, it is possible to have fast "hot potato" chains of messages without increasing average bandwidth.
- The presentation and evaluation of a novel application of a token based flow control mechanism for gossip learning in the bursty transfer scenario.
- The introduction of a new subsampling technique based on partitioned models where each partition has its own age parameter. We show that this partitioning technique is beneficial for token-based learning algorithms.
- New experimental scenarios including larger networks, and an additional variant of compressed federated learning where the downstream traffic is also compressed using subsampling.
- A thorough hyperparameter analysis in Section 5.4.

The outline is as follows. Sections 2–4 describe the basics of logistic regression, gossip learning, and federated learning, respectively. These sections describe our novel algorithms as well, including the token account based flow control mechanism, the model partitioning technique, and a number of smaller design decisions that allow all the evaluated algorithms to use shared components. Section 5 presents our experimental setup that includes the applied datasets, as well as our system model. We also discuss the problem of selecting hyperparameters. Section 6 describes our experimental results in a large number of scenarios with many algorithm variants. Section 7 presents related work and Section 8 concludes the paper.

2. Machine learning basics

Here, we give a concise summary of the main machine learning concepts. We are concerned with the classification problem, where we are given a data set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n examples. An example (x, y) consists of a feature vector $x \in R^d$ and the corresponding class label $y \in C$, where d is the dimension of the problem and C is the set of class labels.

The problem is to find the parameters w of a function $f_w : R^d \to C$ that can correctly classify as many examples as possible in D, as well as outside D (this latter property is called generalization). Expressed formally, we wish to minimize an objective function $J(w)$ in w:

$$w^* = \arg\min_w J(w) = \arg\min_w \frac{1}{n}\sum_{i=1}^{n} \ell(f_w(x_i), y_i) + \frac{\lambda}{2}\|w\|^2, \qquad (1)$$

where $\ell()$ is the loss function (the error of the prediction), $\|w\|^2$ is the regularization term, and λ is the regularization coefficient.

Stochastic gradient descent (SGD) is a popular approach for finding w^*. Here, we start with some initial weight vector w_0, and we apply the following update rule:

$$w_{t+1} = w_t - \eta_t(\lambda w_t + \frac{\partial \ell(f_w(x_i), y_i)}{\partial w}(w_t)). \qquad (2)$$

Here, η_t is called the learning rate. This update rule requires a single example (x_i, y_i), and in each update we can choose a random example.

In this study we use *logistic regression* as our machine learning model, where the specific form of the objective function is given by

$$J(w, b) = \frac{\lambda}{2}\|w\|^2 - \frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln f_{(w,b)}(x_i) + (1 - y_i) \times \ln(1 - f_{(w,b)}(x_i))\right], \qquad (3)$$

where $y_i \in \{0, 1\}$ and

$$f_{(w,b)}(x_i) = P(y_i = 1|x_i, w, b) = \frac{1}{1 + e^{(w^T x + b)}}. \qquad (4)$$

Note that $P(y_i = 0|x_i, w, b) = 1 - P(y_i = 1|x_i, w, b)$. In fact, the loss function above is the log-likelihood of the data under this probabilistic model. The parameter b is called the bias of the model.

While our method can be applied to solve any model-fitting optimization problem, we evaluated it only with logistic regression. This might seem restrictive, however, the practical applicability of such a simple linear model is greatly extended if one uses it in the context of transfer learning [31]. The idea is that arbitrarily complex pre-trained machine learning models are used as feature extractors, over which a simple (often linear) model is trained over a given new dataset. In linguistic applications, this is becoming a very popular approach, often using BERT [12] as the pre-trained model. This can approximate the performance of training the entire complex model over the new dataset, while using much fewer resources.

3. Gossip learning

Gossip Learning is a machine learning approach over fully distributed data without central control [27]. Here, we assume that the data set D is horizontally distributed over a set of nodes, with node k storing its own shard D_k. The task is to collectively find a machine learning model in such a way that it emulates the case when the data set is stored centrally.

Let us discuss the basic notions of gossip learning. Each node k runs Algorithm 1. First, the node initializes its local model

Algorithm 1 Gossip Learning

1: $(t_k, w_k, b_k) \leftarrow (\mathbf{0}, \mathbf{0}, 0)$
2: **loop**
3: wait(Δ_g)
4: $p \leftarrow$ selectPeer()
5: send sample(t_k, w_k, b_k) to p
6: **end loop**
7:
8: **procedure** ONRECEIVEMODEL(t_r, w_r, b_r)
9: $(t_k, w_k, b_k) \leftarrow$ merge$((t_k, w_k, b_k), (t_r, w_r, b_r))$
10: $(t_k, w_k, b_k) \leftarrow$ update$((t_k, w_k, b_k), D_k)$
11: **end procedure**

Algorithm 2 Partitioned Model Merge

1: S : the number of partitions
2: **procedure** MERGE$((t, w, b), (t_r, w_r, b_r))$
3: $j \leftarrow$ index of received partition ▷ $j = i \mod S$, for any coordinate i within the sample
4: **for** coordinate i is included in sample **do**
5: $w[i] \leftarrow (t[j] \cdot w[i] + t_r[j] \cdot w_r[i])/(t[j] + t_r[j])$
6: **end for**
7: $b \leftarrow (t[S] \cdot b + t_r[S] \cdot b_r)/(t[S] + t_r[S])$
8: $t \leftarrow \max(t, t_r)$ ▷ element-wise maximum, where t_r is defined
9: **return** (t, w, b)
10: **end procedure**

(w_k, b_k) and its age t_k. A subset of the model parameters is then periodically sent to another node in the network. When a node receives such a parameter sample, it merges it into its own model and then it performs a local update step. Note that the rounds are not synchronized, although all the nodes use the same period Δ_g. Any received messages are processed immediately. Different variants of the algorithm can be produced with different implementations of the methods SAMPLE, MERGE, and UPDATE. In the simplest case, SAMPLE sends the entire model (no sampling), MERGE computes the average, and UPDATE performs a mini-batch update based on the local data. Later on, we will define more sophisticated implementations in Section 3.1.

The node selection in line 4 is supported by a so-called peer sampling service. Applications can utilize a peer sampling service implementation to obtain random samples from the set of participating nodes. The implementations of this service might be based on several different approaches that include random walks [32], gossip [19], or even static overlay networks that are created at random and repaired when necessary [28]. We will assume a static, connected, random overlay network from now on.

In the following, we shall describe three optimizations of the original gossip learning algorithm, that are interrelated. Very briefly, the basic ideas behind the three optimizations are the following:

sampling: Instead of sending the full model to the neighbor, a node can send only a subset of the parameters. This technique is often used as a compression mechanism to save bandwidth.

model partitioning: Related to sampling, instead of a random subset, it is also possible to define a fixed partitioning of the model parameters and to send one of these subsets as a sample.

token accounts [10]: We can add flow control as well, that is, we can speed up the spreading of information through a self-organizing enhancement of the communication pattern, without increasing the number of messages overall.

These three techniques are interrelated. For example, if one wishes to use token accounts along with sampling-based compression, then using the implementation based on model partitioning is imperative, as we will see. In fact, that is our main motivation for discussing model partitioning. Let us now discuss these techniques in turn.

3.1. Random sampling and model partitioning

As for the method SAMPLE, we will use two different implementations. The first implementation, SAMPLERANDOM(t, w, b, s) returns a uniform random subset of the parameters, where $s \in (0, 1]$ defines the size of the sample. To be precise, the size of the sample is given by $s \cdot d$ (randomly rounded) where d is the dimension of vector w.

The other implementation is based on a partitioning of the model parameters. Let us elaborate on the idea of model partitioning here. The model is formed by the vector w and the bias value b. We partition only w. We define $S \geq 1$ partitions, by assigning a given vector index i to the partition index $(i \mod S)$. When sampling is based on this partitioning, we return a given partition. More precisely, SAMPLEPARTITION(t, w, b, j), where $0 \leq j < S$ is a partition index, returns partition j. The bias b is always included in every sample, in both implementations of method SAMPLE.

It is important to stress that the random sampling method SAMPLERANDOM should be applied only without model partitioning (that is, when $S = 1$). It is possible to define a combination of partition-based and random sampling, where we could sample from a given partition, but we do not explore this possibility in this study.

Upon receiving a model, the node merges it with its local model, and updates it using its local data set D_k. Method MERGE is used to combine the local model with the incoming one. The most usual choice to implement MERGE is to take the average of the parameter vectors [27]. This is supported by theoretical findings as well, at least in the case of linear models and when there is only one round of communication [35].

If there is no partitioning ($S = 1$) then the implementation shown as Algorithm 2 computes the average weighted by model age. This implementation can deal with subsampled input as well, since we consider only those parameters that are actually included in the sample. When partitioning is applied (that is, when $S > 1$) each partition of the parameter vector has its own age parameter. This is crucial especially when we apply flow control to speed up communication, as we explain later on. This means that every model now has a vector of age values t of length $S + 1$ where the ages of the partitions are $t[0], \ldots, t[S-1]$ and the age of the bias is $t[S]$.

Method UPDATE is shown in Algorithm 3. This implementation requires a full model as input but it does take into account the partitioning of the model in that all the partitions have their own dynamic learning rate that is determined by the age of the partition.

3.2. Token gossip learning

In our previous work, we introduced the token account algorithm for improving the efficiency of gossip protocols in a wide range of applications [10]. The basic intuition is that the token account algorithm allows chains of messages to form that travel fast in the network like a "hot potato", even if the budget of

Algorithm 3 Partitioned Model Update Rule

1: S : the number of partitions
2: d : the dimension of w
3: **procedure** UPDATE($(t, w, b), D$)
4: **for all** batch $B \subseteq D$ **do** ▷ D is split into batches
5: $t \leftarrow t + |B| \cdot \mathbf{1}$ ▷ increase all ages by $|B|$
6: **for** $i \in \{1, ..., d\}$ **do**
7: $h[i] \leftarrow -\frac{\eta}{t[i \mod S]} \sum_{(x,y)\in B} (\frac{\partial \ell(f_{w,b}(x), y)}{\partial w[i]}(w[i]) + \lambda w[i])$
8: **end for**
9: $g \leftarrow -\frac{\eta}{t[S]} \sum_{(x,y)\in B} (\frac{\partial \ell(f_{w,b}(x), y)}{\partial b}(b) + \lambda b)$
10: $w \leftarrow w + h$
11: $b \leftarrow b + g$
12: **end for**
13: **return** (t, w, b)
14: **end procedure**

Algorithm 4 Partitioned Token Gossip Learning

1: $(t_k, w_k, b_k) \leftarrow (\mathbf{0}, \mathbf{0}, 0)$
2: $a_k \leftarrow \mathbf{0}$
3: **loop**
4: wait(Δ_g)
5: $j \leftarrow$ selectPart() ▷ select a random partition
6: **do with probability** proactive($a_k[j]$)
7: $p \leftarrow$ selectPeer()
8: send samplePartition(t_k, w_k, b_k, j) to p
9: **else**
10: $a_k[j] \leftarrow a_k[j] + 1$ ▷ we did not spend the token so it accumulates
11: **end do**
12: **end loop**
13:
14: **procedure** ONRECEIVEMODEL(t_r, w_r, b_r, j)
15: $(t_k, w_k, b_k) \leftarrow$ merge($(t_k, w_k, b_k), (t_r, w_r, b_r)$)
16: $(t_k, w_k, b_k) \leftarrow$ update($(t_k, w_k, b_k), D_k$)
17: $x \leftarrow$ randRound(reactive($a_k[j]$))
18: $a_k[j] \leftarrow a_k[j] - x$ ▷ we spend x tokens
19: **for** $i \leftarrow 1$ to x **do**
20: $p \leftarrow$ selectPeer()
21: send samplePartition(t_k, w_k, b_k, j) to p
22: **end for**
23: **end procedure**

each node is limited. This is due to the flow control mechanism implemented with the help of token accounts that allow these hot potatoes to avoid waiting for the next cycle at each node. So far, the token account approach has not been applied to merge-based gossip learning. Here, we omit the general introduction of the entire family of token account algorithms and instead we focus on the novel variant that is applicable in this study, shown as Algorithm 4.

It is very important that the algorithm uses SAMPLEPARTITION as an implementation of sampling. Our preliminary experiments have shown that random sampling (SAMPLERAND) is not effective along with the token account technique. This is because if we sample independently at each hop, we work strongly against the formation of long "hot potato" message chains that represent any fixed model parameter. This is a key insight, and this is why the partitioned approach is expected to work better. It allows for hot potato message chains to form based on a single partition. In other words, we can have the benefits of sampling-based compression and hot potato message passing at the same time.

Accordingly, each partition j at node k now has its own token account $a_k[j]$ that stores the number of available tokens. Apart from the flow control extension, the algorithm is very similar to general gossip learning. The main difference is that we might also send messages reactively, that is, as a direct reaction to receiving a model. The only two methods that we need to define are PROACTIVE and REACTIVE. Method PROACTIVE returns the probability of sending a proactive message as a function of the number of tokens. Method REACTIVE returns the number of reactive messages to be sent. Here, we used the implementations

$$\text{PROACTIVE}(a) = \begin{cases} 0 & \text{if } a < A - 1 \\ \dfrac{a - A + 1}{C - A + 1} & \text{if } a \in [A - 1, C] \end{cases} \quad (5)$$

and REACTIVE(a)$= a/A$, with parameters $A = 10$ and $C = 20$, based on our previous work [10]. Here, $C > 0$ is the maximal number of tokens the account can store. Note that the maximum value of a is indeed C because initially $a = 0$ and when $a = C$, we always send a proactive message, thus a will not grow further. Parameter $A \in \{1, \ldots, C\}$ plays a role in both reactive and proactive message sending, and it can be interpreted as the motivation level for saving tokens. If $A = 1$, then we spend all our tokens on sending reactive messages in every call to ONRECEIVEMODEL and we also send proactive messages with a positive probability if the account is not empty. If $A = C$ then we send at most one reactive message, and we send a proactive message only if the account is full.

With each partition having a separate account, each partition will perform random walks independently, using its own communication budget. Of course, the model update is not independent, it is the same as in partitioned gossip learning. We also track the age of each partition, as discussed previously in connection with the merge and update functions. We will argue that even plain gossip learning can benefit from using partitions, although to a lesser extent.

If the number of partitions is $S = 1$, then we get a special case of the algorithm without any compression (that is, no sampling), we always send the entire model.

As a final note, let us mention that the methods SELECTPART and SELECTPEER were implemented using sampling without replacement, re-initialized when the pool of available options becomes empty. This is slightly better than sampling with replacement, because it results in a lower variance.

4. Federated learning

Federated learning is not a specific algorithm, but more of a design framework for edge computing. We discuss federated learning based on the algorithms presented in [22,25]. While we keep the key design elements, our presentation contains small adjustments and modifications to accommodate our contributions and to allow gossip learning and federated learning to share a number of key methods.

The pseudocode of the federated learning algorithm is shown in Algorithms 5 (master) and 6 (worker). The master periodically sends the current model w to all the workers asynchronously in parallel and collects the answers from the workers. In this version of the algorithm, communication is compressed through sampling the parameter vectors. The rate of sampling might be different in the case of downstream messages (line 4 of Algorithm 5) and upstream messages (line 11 of Algorithm 6). We require that $s_{up} \leq s_{down}$. Although this is not reflected in the pseudocode for presentation clarity, the sample produced in line 11 of Algorithm 6 is allowed to include only indices that are also included in the

Algorithm 5 Federated Learning Master

1: $(t, w, b) \leftarrow$ init()
2: **loop**
3: **for** every node k **in parallel do** ▷ non-blocking (in separate threads)
4: send sample(t, w, b, s_{down}) to k
5: receive (n_k, h_k, g_k) from k ▷ n_k: #examples at k; h_k: sampled model gradient; g_k: bias gradient
6: **end for**
7: wait(Δ_f) ▷ the round length
8: $n \leftarrow \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} n_k$ ▷ $\mathcal{K}$: nodes that returned a model in this round
9: $t \leftarrow t + n$
10: $h \leftarrow$aggregate$(\{h_k : k \in \mathcal{K}\})$
11: $w \leftarrow w + h$
12: $g \leftarrow \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} g_k$
13: $b \leftarrow b + g$
14: **end loop**

Algorithm 6 Federated Learning Worker

1: $(t_k, w_k, b_k) \leftarrow$ init() ▷ the local model at the worker
2:
3: **procedure** onReceiveModel(t, w, b) ▷ w: sampled model
4: $t_k \leftarrow t$
5: **for** $w[i] \in w$ **do** ▷ coordinate i is defined in w
6: $w_k[i] \leftarrow w[i]$
7: **end for**
8: $b_k \leftarrow b$
9: $(t_k, w_k, b_k) \leftarrow$ update$((t_k, w_k, b_k), D_k)$ ▷ D_k: the local database of examples
10: $(n, h, g) \leftarrow (t_k - t, w_k - w, b_k - b)$ ▷ n: the number of local examples, h: the gradient update
11: send sample(n, h, g, s_{up}) to master
12: **end procedure**

Algorithm 7 Variants of the aggregate function

1: **procedure** aggregate(H)
2: $h' \leftarrow \mathbf{0}$
3: **for** $i \in \{1, ..., d\}$ **do**
4: $h'[i] \leftarrow \frac{1}{s|H|} \sum_{h \in H : h[i] \in h} h[i]$ ▷ $s \in (0, 1]$: sampling rate used to create H
5: **end for**
6: **return** h'
7: **end procedure**
8:
9: **procedure** aggregateImproved(H)
10: $h' \leftarrow \mathbf{0}$
11: **for** $i \in \{1, ..., d\}$ **do**
12: $H_i \leftarrow \{h : h \in H \wedge h[i] \in h\}$
13: $h'[i] \leftarrow \frac{1}{|H_i|(1-(1-s)^{|H|})} \sum_{h \in H_i} h[i]$ ▷ skipped if $|H_i| = 0$
14: **end for**
15: **return** h'
16: **end procedure**

received model. For example, if $s_{up} = s_{down}$ then the worker selects exactly those indices that were received in the incoming sample.

Any answers from workers arriving with a delay larger than Δ_f are simply discarded. After Δ_f time units have elapsed, the master aggregates the received gradients and updates the model. We also send and maintain the model age t (based on the average number of examples used for training) in a similar fashion, to make it possible to use dynamic learning rates in the local learning algorithm.

We note that, although in this version of the algorithm the master sends the model to every worker, it is possible to use a more fine-grained method to select a subset of workers that get the model in a given round. For example, if the workers have a very limited budget of communication, it might be better to avoid talking to each worker in each round. Indeed, we will study such a scenario during our experimental evaluation, but we did not want to include this option in the pseudocode for clarity of presentation.

These algorithms are very generic, the key characteristics of federated learning lie in the details of the update method (line 9 of Algorithm 6) and the aggregation mechanism (line 10 of Algorithm 5). The update method is typically implemented through a minibatch gradient descent algorithm that operates on the local data, initialized with the received model w. The implementation we use here is identical to that of gossip learning, as given in Algorithm 3. Note that here we do not partition the model (that is, $S = 1$). As for sampling, we use sampleRandom as described in Section 3.1.

Method aggregate is used in Algorithm 5. Its function is to aggregate the received sampled gradients. Possible implementations are shown in Algorithm 7. Both implementations are unbiased estimates of the average gradient. This also implies that when there is no actual sampling (that is, we have $s = 1$) then simply the average of the gradients is computed by both methods. The improved version averages each coordinate separately, that is, it takes the average of only those coordinates that are included in the sample. This is a more accurate estimate of the true average of the given coordinate. However, in order to get an unbiased estimate, we have to divide by the probability that there is at least one gradient in which the given coordinate is included. This probability equals $1 - (1 - s)^{|H|}$. Note that this probability is independent of the coordinate i, so its effect can be thought of correcting the learning rate, especially when $|H|$ is small.

5. Experimental setup

5.1. Datasets

We used three datasets from the UCI machine learning repository [13] to test the performance of our algorithms. The first is the Spambase (SPAM E-mail Database) dataset containing a collection of emails. Here, the task is to decide whether an email is spam or not. The emails are represented by high level features, mostly word or character frequencies. The second dataset is Pendigits (Pen-Based Recognition of Handwritten Digits) that contains downsampled images of 4×4 pixels of digits from 0 to 9. The third is the HAR (Human Activity Recognition Using Smartphones) [2] dataset, where human activities (walking, walking upstairs, walking downstairs, sitting, standing and laying) were monitored by smartphone sensors (accelerometer, gyroscope and angular velocity). High level features were extracted from these measurement series.

The main properties, such as size or number of features, are presented in Table 1. In our experiments we standardized the feature values, that is, we shifted and scaled them so as to have a mean of 0 and a variance of 1. Note that the standardization can be approximated by the nodes in the network locally if the approximation of the statistics of the features are fixed and known, which can be ensured in a fixed application.

In our simulation experiments, each example in the training data was assigned to one node when the number of nodes was

Table 1
Data set properties.

	Spambase	Pendigits	HAR
Training set size	4140	7494	7352
Test set size	461	3498	2947
Number of features	57	16	561
Number of classes	2	10	6
Class-label distribution	≈ 6:4	≈ uniform	≈ uniform

100. This means that, for example, with the HAR dataset each node gets 73.5 examples on average. The examples were assigned evenly, that is, the number of examples at the nodes differed by at most one due to the number of samples not being divisible by 100. When the network size equaled the database size, we mapped the examples to the nodes so that each node had exactly one example. We also experimented with a third scenario that combines the two settings above. That is, the number of examples per node was the same as in the 100 node scenario, but the network size equaled the database size. To achieve this, we replicated the examples, that is, each example was assigned to multiple nodes.

In the scenarios where a node had more than one example, we considered two different class label distributions. The first one is *uniform assignment*, which means that we assigned the examples to nodes at random independently of class label. The second one is *single class assignment* when every node has examples only from a single class. Here, the different class labels are assigned uniformly to the nodes, and then the examples with a given label are assigned to one of the nodes with the same label, uniformly. These two assignment strategies represent the two extremes in any real application. In a realistic setting the class labels will likely be biased but much less so than in the case of the single class assignment scenario.

5.2. System model

In our simulation experiments, we used a fixed random k-out overlay network, with $k = 20$. That is, every node had $k = 20$ fixed random neighbors. As described previously, the network size was either 100 or the same as the database size. In the churn-free scenario, every node stayed online for the whole experiment. The churn scenario is based on a real trace gathered from smartphones (see Section 5.3 below). We assumed that a message is successfully delivered if and only if both the sender and the receiver remain online during the transfer. We also assume that the nodes are able to detect which of their neighbors are online at any given time with a delay that is negligible compared to the transfer time of a model. Nodes retain their state while being offline.

We assumed that the server has unlimited bandwidth. In practice, unlimited bandwidth is achieved using elastic cloud infrastructure, which obviously has a non-trivial cost in a very large system. Gossip learning has *no additional cost at all related to scaling*. Thus, ignoring the cost of the cloud infrastructure clearly favors federated learning in our study, so this assumption must be kept in mind.

We assumed that the worker nodes have identical upload and download bandwidths. This needs explanation, because in federated learning studies, downstream communication is considered free, citing the fact that the available upload bandwidth is normally much lower than the download bandwidth. But this distinction is only relevant if all the nodes are allowed to use their full bandwidth completely dedicated to federated learning continuously. This is a highly unlikely scenario, given that federated learning will not be the only application on most devices. It is much more likely that there will be a cap on the bandwidth usage, in which light the difference between upstream and downstream bandwidth fades. For the same reason, we also assumed that all the worker nodes have the same bandwidth because the bandwidth cap mentioned above can be expected to be significantly lower than the average available bandwidth, so this cap could be assumed to be uniform.

One could, of course, set a higher cap on downstream traffic. Our study is relevant also in that scenario, for two reasons. First, the actual bandwidth values make no qualitative difference if the network is reliable (there is no churn), they result only in the scaling of time. That is, scaling our results accordingly provides the required measurements. Second, in unreliable networks, a higher downstream bandwidth would result in a similar scaling of time. In addition, it would result in better convergence as well, since the downstream messages could be delivered with a strictly higher probability.

In the churn scenario, we need to fix the amount of time necessary to transfer a full model. (If the nodes are reliable then the transfer time is completely irrelevant, since the dynamics of convergence are identical apart from scaling time.) The transfer time of a full model was assumed to be $60 \cdot 60 \cdot 24/1000 = 86.4$ s, irrespective of the dataset used, in the *long transfer time* scenario, and 8.64 s in the *short transfer time* scenario. This allowed us to simulate around 1000 and 10,000 iterations over the course of 24 h, respectively. Note that the actual models in our simulation are relatively small linear models, so they would normally require only a fraction of a second to be transferred. Still, we pretend here that our models are very large. This is because if the transfer times are very short, the network hardly changes during the learning process, so effectively we learn over a static subset of the nodes. Long transfer times, however, make the problem more challenging because many transfers will fail, just like in the case of very large machine learning models such as deep neural networks.

5.3. Smartphone traces

The trace we used was collected by a locally developed openly available smartphone app called STUNner, as described previously [5]. In a nutshell, the app monitors and collects information about charging status, battery level, bandwidth, and NAT type.

We have traces of varying lengths taken from 1191 different users. We divided these traces into 2-day segments (with a one-day overlap), resulting in 40,658 segments altogether. With the help of these segments, we were able to simulate a virtual 48-hour period by assigning a different segment to each simulated node. We use only the second 24-hour period for learning; the first day is used for achieving a token distribution that reflects an ongoing application. This warm-up period can represent a previous, unrelated learning task executed on the same platform, or the sending of empty messages; it does not count towards the communication costs. For fair comparison, we use the same period for learning also in the case of algorithms that do not use tokens.

To ensure our algorithm is phone and user friendly, we defined a device to be online (available) when it has been on a charger and connected to the internet for at least a minute, hence we never use battery power at all. In addition, we also treated those users as offline who had a bandwidth of less than 1 Mb/s.

Fig. 1 illustrates some of the properties of the trace. The plot on the right illustrates churn via showing, for every hour, what percentage of the nodes left, or joined the network (at least once), respectively. We can also see that at any given moment about 20% of the nodes are online. The average session length is 81.368 min.

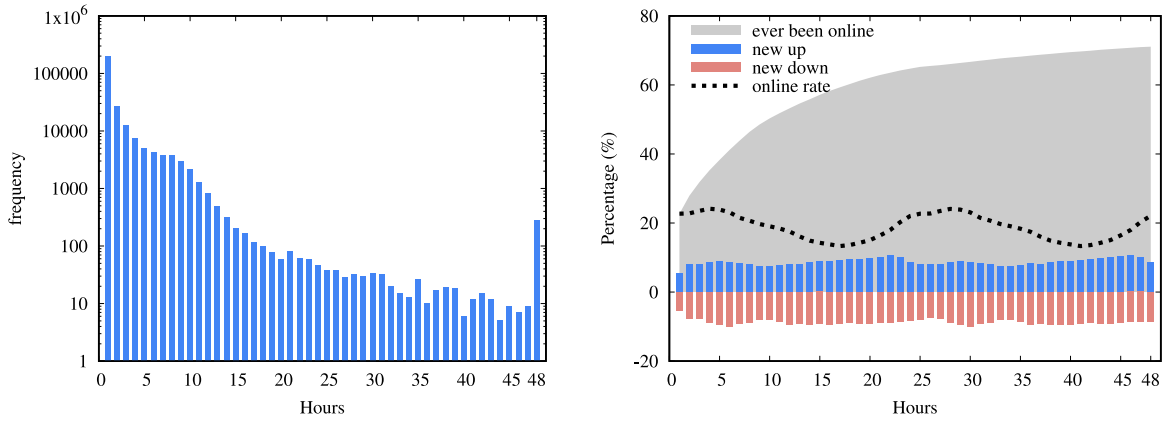

Fig. 1. Online session length distribution (left) and dynamic trace properties (right).

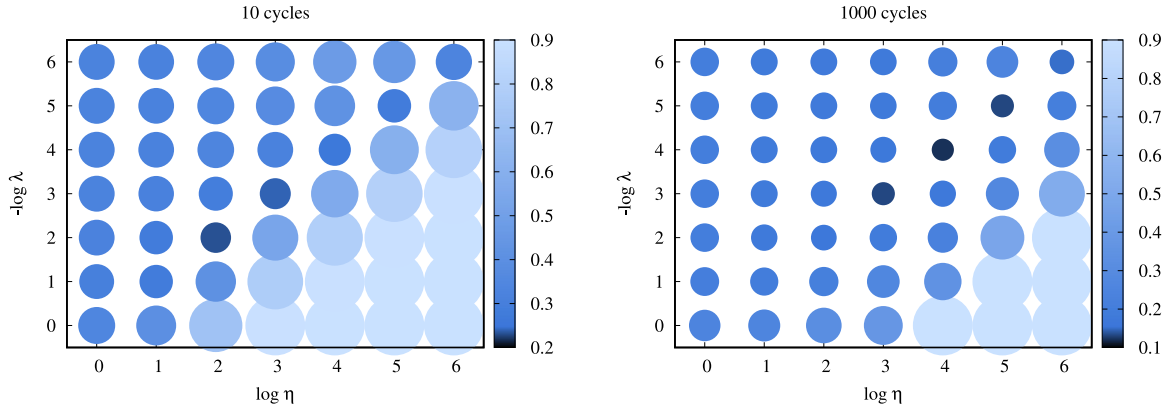

Fig. 2. Error of partitioned gossip learning with $S = 10$ and $N = 100$ on the Pendigits dataset as a function of η and λ after 10 cycles (left) and after 1000 cycles (right).

5.4. Hyperparameters

The cycle length parameters Δ_g and Δ_f were set in two different ways. In the *continuous transfer* scenario, the goal was to make sure that the protocols communicate as much as they can under the given bandwidth constraint. The gossip cycle length Δ_g is thus exactly the transfer time of a full model, that is, all the nodes send messages continuously. The cycle length Δ_f of federated learning is the round-trip time, that is, the sum of the upload and download transfer times. When compression is used, the transfer time is proportionally less as defined by the compression rate, and this is also reflected in the cycle length settings.

In the *bursty transfer* scenario, we assume that we transfer only during a given percentage of the time, say, 1% of the time. Let δ denote the transfer time and let $p \in (0, 1]$ be the proportion of the time we use for transfer. Here, we set the gossip cycle length $\Delta_g = \delta/p$. To implement the bursty model in federated learning, we have many choices for the cycle length depending on how many nodes the master wants to contact in a single cycle. If we set $\Delta_f = (\delta_{up} + \delta_{down})/p$ (where δ_{up} and δ_{down} are the upload and download transfer times, respectively) then the master should contact all the nodes as before so the only effect is to slow the algorithm down. If we set a shorter cycle length then the master will contact only a subset of the nodes to achieve the required proportion of p overall. We will examine this latter case when the cycle length is set so that 1% of the nodes are contacted in a cycle: $\Delta_f = \delta_{up} + \delta_{down}$, where we need $p = 1/100$ to hold as well. When compression is used, δ_{up} and δ_{down} might differ.

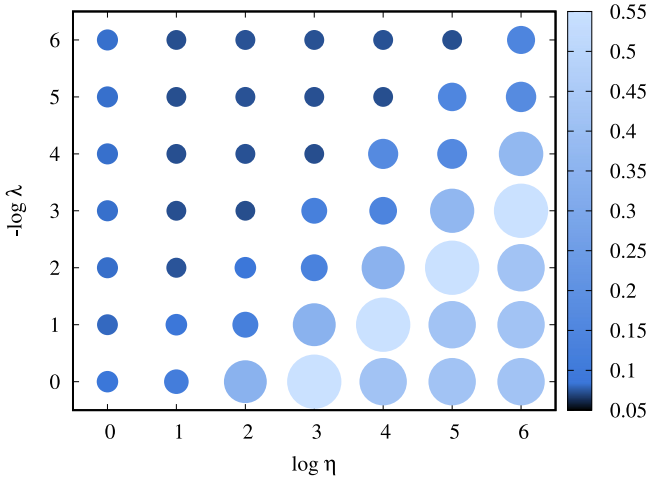

Fig. 3. Error of partitioned gossip learning with $S = 10$ and $N = 4140$ on the Spambase dataset as a function of η and λ after 1000 cycles.

In both the bursty and the continuous transfer scenarios, on average, the two algorithms transfer the same number of bits overall in the network during the same amount of time. Furthermore, continuous transfer is the special case of bursty transfer with $p = 1$.

As for subsampling, we explore the sampling probability values $s \in \{1, 0.5, 0.25, 0.1\}$. In federated learning, both the upstream and the downstream messages can be sampled using a

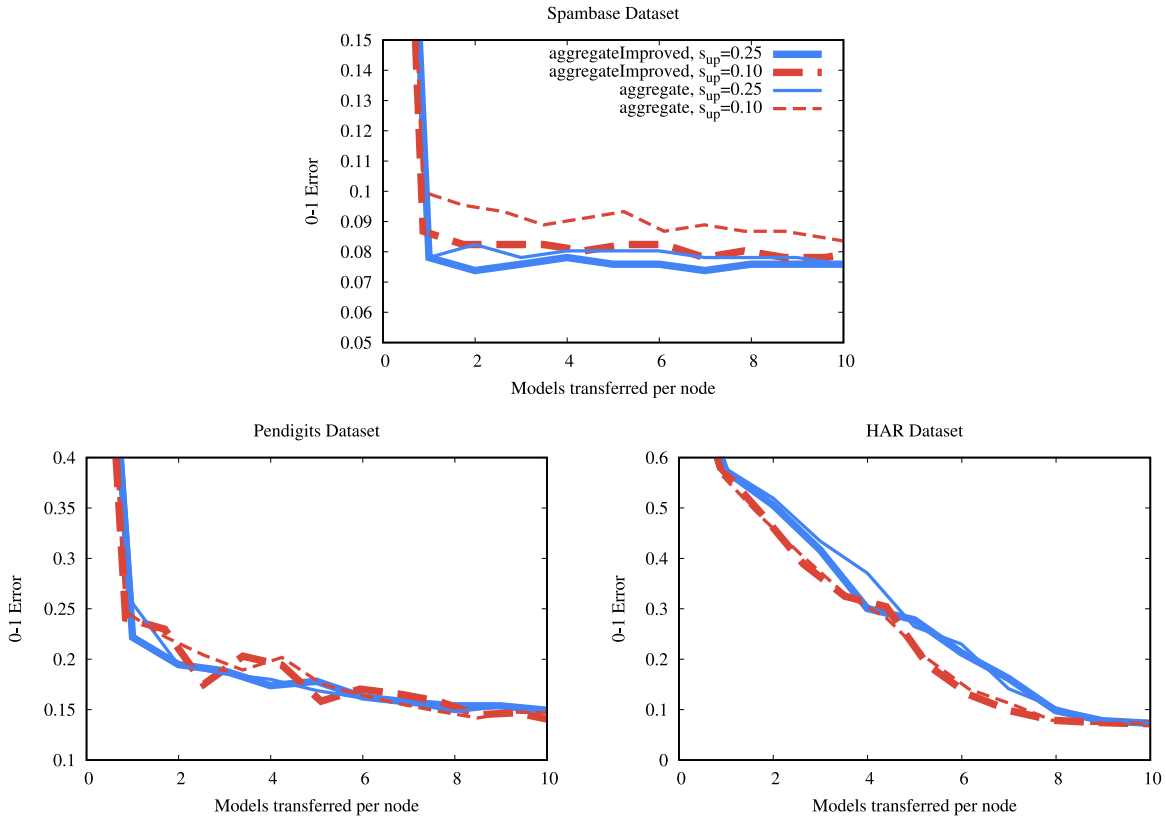

Fig. 4. Federated learning, 100 nodes, long transfer time, no failures, different aggregation algorithms and upstream subsampling probabilities and with $s_{down} = 1$.

different rate, denoted by s_{up} and s_{down}, respectively. The setting we use in our experiments is $s_{down} = s_{up}$. However, we shall also show runs where we fix $s_{down} = 1$ and experiment with upstream sampling only. When the subsampling is based on partitioning, the number of partitions S defines the sampling probability, which equals $1/S$. On the plots $s = 1/S$ will be used even in the partitioned case to indicate the compression rate.

We train a logistic regression model. For the datasets that have more than two classes (Pendigits, HAR), we embedded the model in a one-vs-all meta-classifier. The learning algorithm was stochastic gradient descent. The learning rate η and the regularization coefficient λ used in our experiments are shown in Table 2. We used grid search to optimize these parameters in various scenarios, and found these values relatively robust. However, one should keep in mind that including additional hyperparameters in the search, such as the number of iterations (that we simply fixed here), could result in a different outcome.

Although we fixed the parameters shown in Table 2 in all the scenarios, it is interesting to have a more fine-grained look at the behavior of these hyperparameters. This sheds some light on possible heuristics to pick the right parameters. All the examples here are measurements with gossip learning without token accounts but with model partitioning and $S = 10$. With network size $N = 100$, over the Pendigits dataset, after 10 gossip cycles, the optimal hyperparameters are $\eta = 10^2$ and $\lambda = 10^{-2}$, as shown in Fig. 2. However, after 100 cycles, the optimal values are $\eta = 10^3$ and $\lambda = 10^{-3}$, and after 1000 cycles, $\eta = 10^4$ and $\lambda = 10^{-4}$. We often observed similar trends also with other algorithms and datasets. Notice that settings where $\eta\lambda = 1$ (points on the diagonal of the grid) are often a good choice; however, there are exceptions. For instance, when we have only one example per node, over the Spambase dataset, this is not the case, as shown in Fig. 3. As we can see, here, the points above the diagonal tend to be somewhat better. Also note that settings with $\eta\lambda > 1$ (points below the diagonal) tend to perform very poorly.

Table 2
Hyperparameters.

	Spambase	Pendigits	HAR
Parameter η	10^3	10^4	10^2
Parameter λ	10^{-3}	10^{-4}	10^{-2}

6. Experimental results

We ran the simulations using PeerSim [26]. As for the hardware requirements for reproducing our results, we used a server with 8 2 GHz CPUs each with 8 cores, for a total of 64 cores. The server had 512 GB RAM. On this configuration, the experiments we include in this work can be completed within a month.

We measure learning performance with the help of the 0–1 error, which gives the proportion of the misclassified examples in the test set. In the case of gossip learning the loss is defined as the average loss over the online nodes. This means that we compute the 0–1 error of all the local models stored at the nodes over the same test set, and we report the average. In federated learning we evaluate the central model at the master. Note that this is often more optimistic than evaluating the average of the online nodes. For example, in the bursty transfer scenario, or if the downstream communication is compressed (that is, $s_{down} < 1$), the local models will always be more outdated than the central one, because the nodes will not receive the complete model, or they receive it with a delay.

The presented measurements are averages of 5 runs with different random seeds. The only exceptions are the measurements with our gossip algorithms in the scenarios over the HAR dataset when the network size was the database size. These scenarios are expensive to simulate so we show a single run.

The 0–1 error is measured as a function of the total amount of bits communicated anywhere in the system normalized by

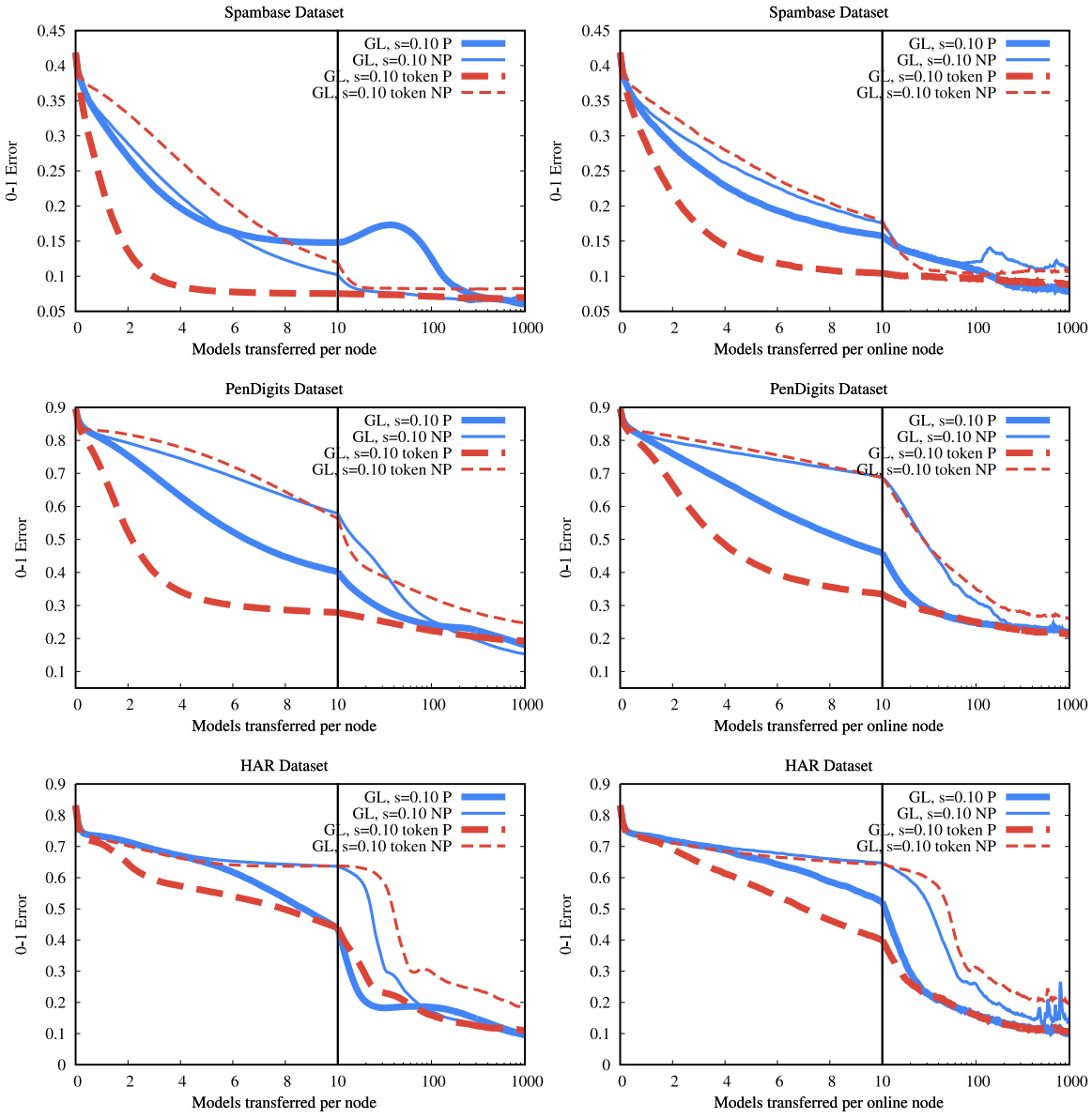

Fig. 5. Gossip learning with one node for each example, bursty transfer, subsampling probability $s = 0.1$, no-failure (left) and smartphone trace with long transfer time (right). Variants with and without model partitioning, indicated as P and NP, respectively.

the number of online nodes. We use the size of a full machine learning model as the unit of the transferred information.

6.1. Basic design choices

First, we compare the two aggregation algorithms for subsampled models in Algorithm 7 (Fig. 4) in the no-failure continuous transfer scenario. The results indicate a slight advantage of AGGREGATEIMPROVED, although the performance depends on the database. In the following we will apply AGGREGATEIMPROVED as our implementation of method AGGREGATE.

Another design choice is whether we should apply model partitioning. We introduced model partitioning to help token gossip learning. However, this technique has other advantages as well. To verify this, we compared partitioned and non-partitioned variants in several scenarios (Fig. 5). We show the scenario where the effect in question is the clearest. Clearly, the partitioned implementations consistently outperform the non-partitioned ones, although in the no-failure scenario, classical gossip learning appears to suffer a temporary setback during convergence. Note that this is a case where the hyperparameters are not exactly optimal, as we explained in Section 5.4, Fig. 3.

The improved performance in the partitioned case is due to the more fine-grained handling of the age parameter. Recall that in the partitioned implementation, all the partitions have their own age and are updated accordingly. In the smartphone trace scenario, this feature is especially useful, since when a node comes back online after an offline period, its model is outdated. During the first merge operation on the model, only those parameters will get an updated age parameter that were indeed updated, that is, that are included in the merged partition. Without partitioning, only a random subset of the parameters will be merged, but the entire model will get a new age value. This is a problem because in the first merge operation that they are included in, the weight of these old parameters will be too large. Due to these observations, from now on, all the experiments are carried out with model partitioning, and this fact will not be explicitly indicated.

The third design choice that we study is the subsampling (that is, compression) strategy for federated learning. Recall that we

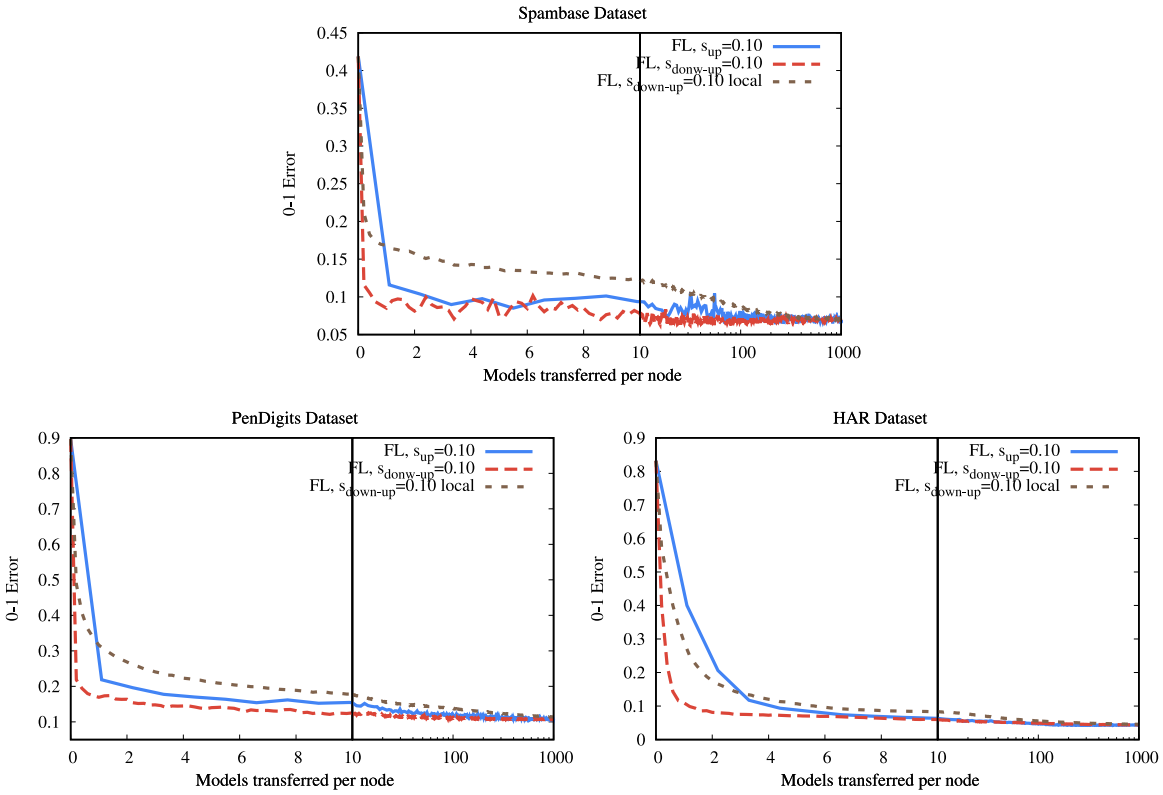

Fig. 6. Federated learning with 100 nodes, no-failure scenario, with different subsampling strategies and $s = 0.1$. The "local" plot indicates the average of the models that the clients store, otherwise the master's model is evaluated.

have a choice to subsample only the model that is sent by the client to the master or we can subsample in both directions. Note that if we subsample in both directions then we can achieve a much higher compression rate but the convergence will be slower. Overall, however, it is possible that, as a function of overall bits communicated, it is still preferable to compress in both directions. Note that subsampling only the model from the master to the client is meaningless because that way the client will send mostly outdated parameters that the master has already received in previous rounds.

Fig. 6 compares the two meaningful strategies for the case of $s = 0.1$. Subsampling in both directions is clearly the better choice. However, it also has a downside, because in that case the clients no longer receive the full model from the master so they cannot use the best possible model locally. To illustrate this problem, we include the average performance of the models stored locally. Depending on the application, this may or may not be a problem. Nevertheless, from now on, we will apply subsampling in both directions in the remaining experiments.

6.2. Continuous transfer

Here, we study the continuous transfer setup where all the nodes try to minimize their idle time taking advantage of the available bandwidth as best as they can. The comparison of the different algorithms and subsampling probabilities is shown in Fig. 7. The stochastic gradient descent (SGD) method is also shown, which was implemented by gossip learning with no merging, where the received model replaces the current model at the nodes. Clearly, the methods using merge are all better than SGD. Also, it is very clear that subsampling helps both federated learning and gossip learning.

Most importantly, in the 100 node setup (left column of Fig. 7), gossip learning is competitive with federated learning in the case of high compression rates (that is, low sampling probability). This was not expected, as gossip learning is fully decentralized, so the aggregation is clearly delayed compared to federated learning. Indeed, with no compression, federated learning performs better.

Fig. 7 (right) also illustrates the extreme scenario, when each node has only one example, and the size of the network equals the dataset size. This is a much more difficult scenario for both gossip and federated learning. Also, federated learning is expected to perform relatively better, because of the more aggressive central aggregation of the relatively little local information. Still, gossip learning is in the same ballpark in terms of performance. In terms of long range convergence (recall, that our scenarios cover approximately a day's time) all the methods achieve good results.

Fig. 8 contains our results over the smartphone trace churn model. Here, all the experiments shown correspond to a period of 24 h, so the horizontal axis has a temporal interpretation as well. The choice of long or short transfer time causes almost no difference (apart from the fact that the shorter transfer time obviously corresponds to proportionally faster convergence). Also, more interestingly, churn only causes a minor increase in the variance of the 0–1 error but otherwise we have a stable convergence. This is due to the application of model partitioning. Previous results without model partitioning showed a much higher variance. It is also worth pointing out that federated learning and gossip learning shows a practically identical performance under high compression rates. Again, gossip learning is clearly competitive with federated learning.

Fig. 9 contains the results of our experiments with the single class assignment scenario, as described in Section 5.1. In this extreme scenario, the advantage of federated learning is more apparent, although on the long run gossip learning also achieves good results. Interestingly, in this case the different compression rates do not have a clear preference order. For example, on the Pendigits database (containing 10 classes) the compressed variant

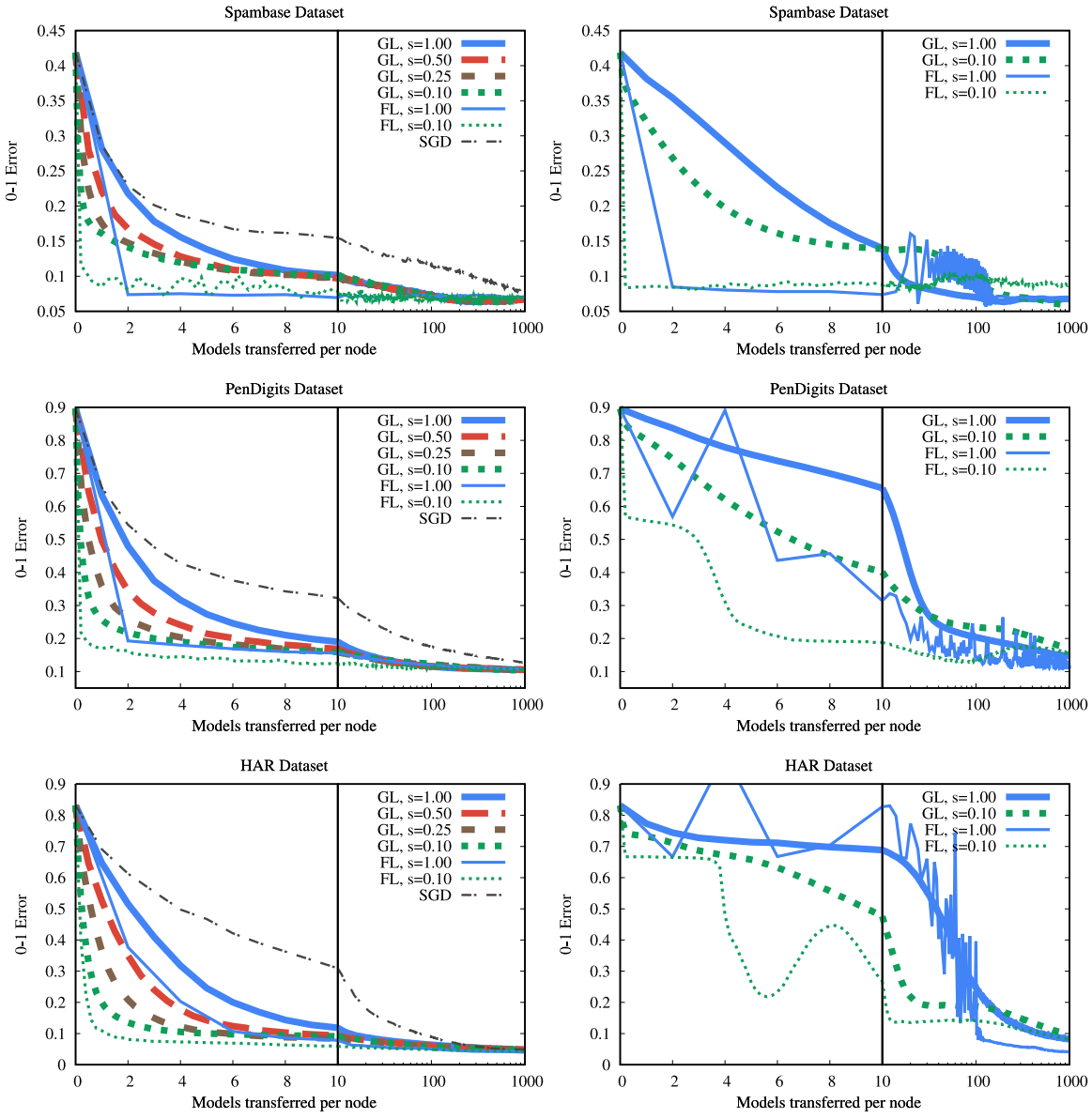

Fig. 7. Federated learning and gossip learning with 100 nodes (left) and with one node for each sample (right), no-failure scenario, with different subsampling probabilities. Stochastic Gradient Descent (SGD) is implemented by gossip learning with no merging (received model replaces current model).

is inferior, while on HAR (with 6 classes) the compressed variant appears to be preferable.

Let us also point out the similarity between the results on Figs. 9 and 7 (right). Indeed, the scenario where each node has one sample is also a single class assignment scenario by definition.

6.3. Bursty transfer

Here, we study the bursty transfer setup where all the nodes communicate only during a given percentage of the time, as described in Section 5.4. Without any modification, the behavior of the algorithms we have discussed so far is very similar to their behavior in the continuous transfer scenario. Although gossip algorithms do work slightly better due to the reduced number of parallel transfers, the bursty transfer scenario offers a possibility to implement specialized techniques that take advantage of bursty transfer explicitly.

In the case of gossip learning, we introduced the token account flow control technique, as described previously. This technique motivated the partition-based sampling technique, to allow for the formation of "hot potato" chains for each partition separately. Recall that in Section 6.1 we decided that all the experiments would use model partitioning, after observing that it helps even traditional gossip.

In the case of federated learning, one such technique is when the master communicates only with a subset of the workers in each round, selecting a different subset each time. This way, although the workers communicate in a bursty fashion, the global model still evolves relatively faster. Here, we will assume that the nodes communicate only 1% of the time. In this case, the master will talk to 1% of the nodes in each round. This way, the master communicates continuously while the nodes communicate in bursts.

Fig. 10 illustrates the performance in the bursty transfer scenario. Clearly, the convergence of each algorithm becomes faster than in the continuous communication case. This suggests that it is better to allow for short but high bandwidth bursts as opposed to long but low bandwidth continuous communication. We can also observe that token gossip converges faster than

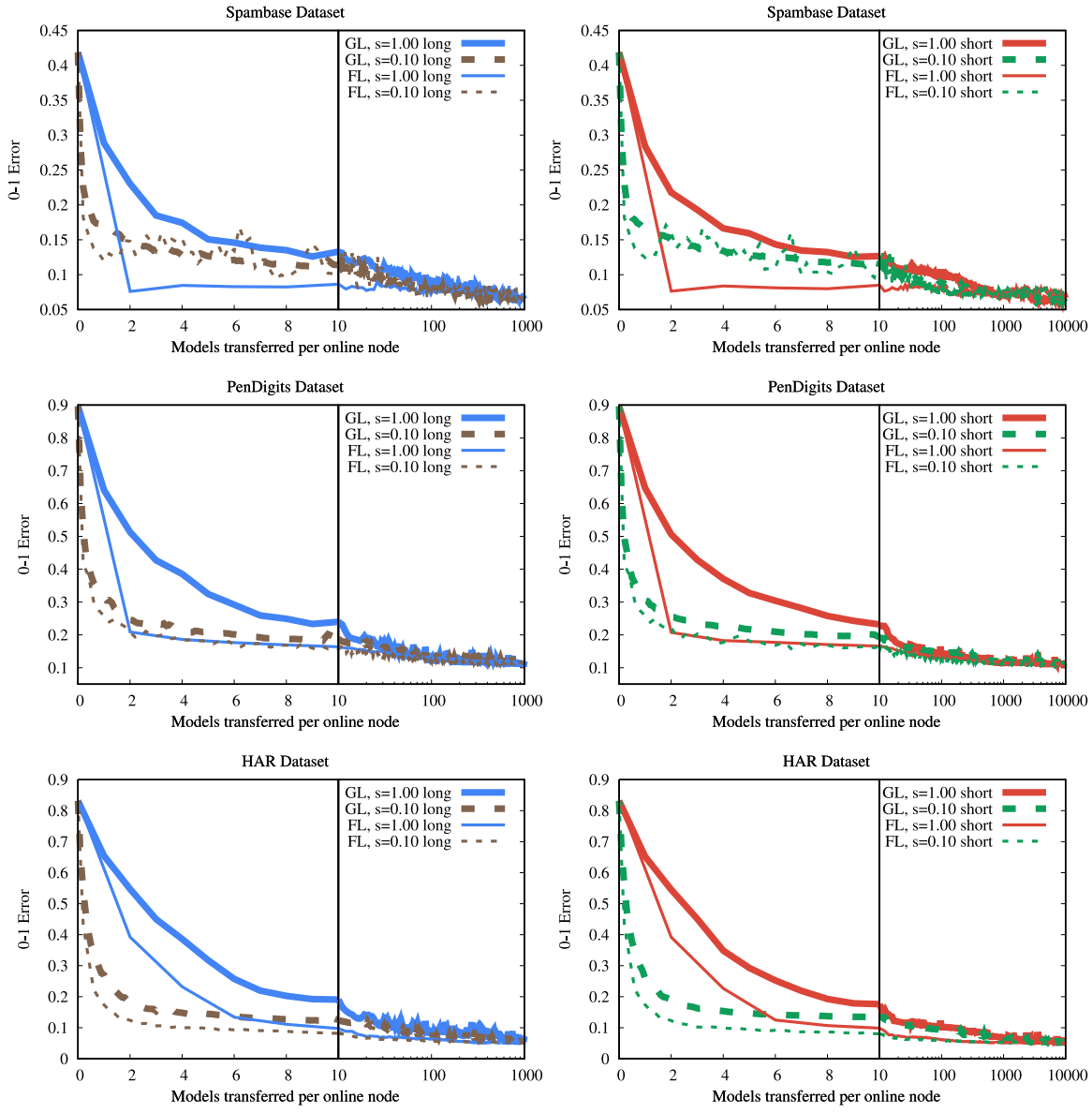

Fig. 8. Federated learning and gossip learning over the smartphone trace with long (left) and short (right) transfer time, in the 100 node scenario.

regular gossip in most cases. Also, the best gossip variant is, again, competitive to the federated learning algorithm.

6.4. Large scale

Here, we experiment with the scenario where the number of examples per node was the same as in the 100 node scenario, but the network size equaled the database size. To achieve this, we replicated the examples, that is, each example was assigned to multiple nodes (see Section 5.1). We call this the large scale scenario.

The results are shown in Fig. 11. We can see that the best gossip variants are competitive with the best federated learning variants. The first observation we can make is that in the bursty transfer scenario a faster convergence can be achieved. This is due to the algorithms that exploit burstiness. Besides, the most important feature of this large scale scenario seems to be whether the label distribution is biased (single label assignment) or not (random assignment). The single class assignment (biased) scenario results in a slower convergence for both approaches. However, compared with previous experiments, increasing the size of the network in itself does not slow the protocols down.

7. Related work

The literature on machine learning and, in general, optimization based on decentralized consensus is vast [30]. Our contribution here is a comparison of the efficiency of decentralized and centralized solutions that are based on keeping the data local. Thus, we focus on works that target the same problem. Savazzi et al. [29] study a number of gossip based decentralized learning methods in the context of industrial IoT applications. They focus on the case where the data distribution is not identical over the nodes. They do not consider compression techniques or other algorithmic enhancements such as token-based flow control.

Hu et al. [17] introduce a segmentation mechanism similar to ours, but their motivation is different. Their focus is on saturating the bandwidth of all the nodes using P2P connections that have a relatively smaller bandwidth, which means they propose that the nodes should communicate to several peers simultaneously.

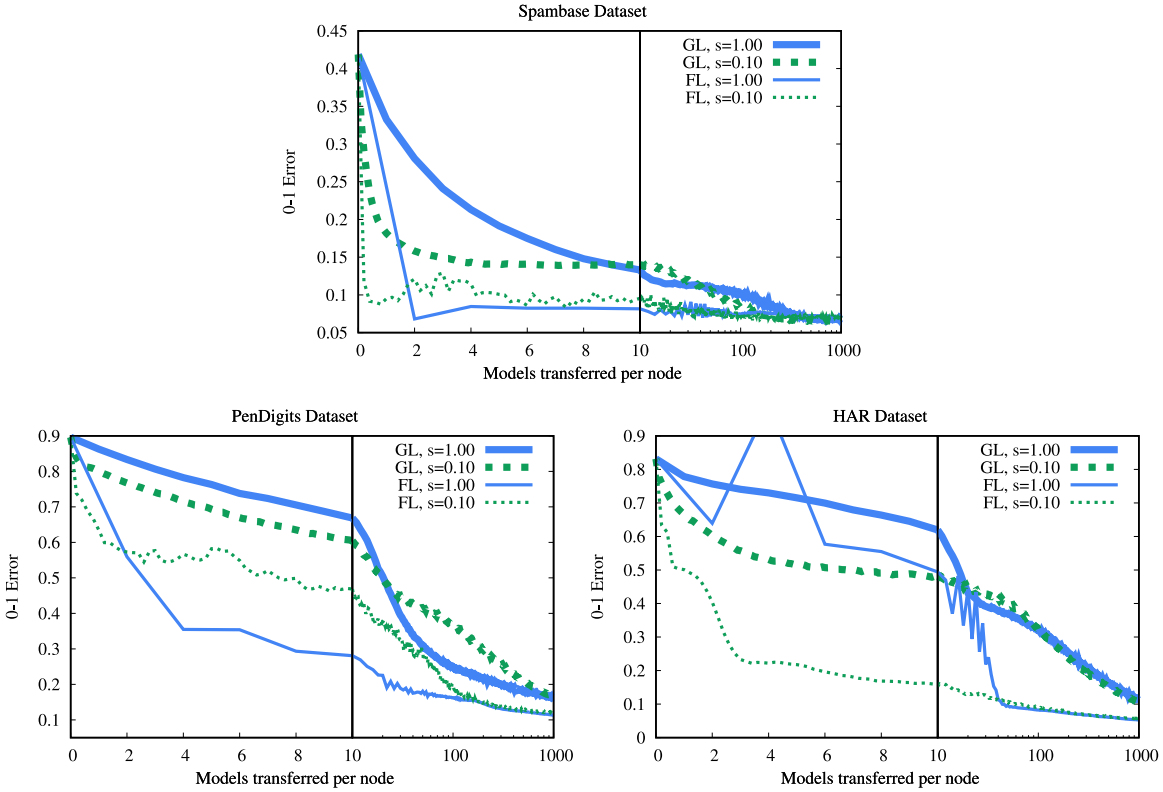

Fig. 9. Federated learning and gossip learning with 100 nodes, no-failure scenario, with single class assignment.

Sending only a part of the model appears to be beneficial in this scenario. In our case, we focused on convergence speed as a function of overall communication, and we used this technique to optimize our token-based flow control mechanism.

Blot et al. [6] compare a number of aggregation schemes for the decentralized aggregation of gradients, including a gossip version based on the weighted push sum communication scheme [20]. Although the authors do not cite federated learning or gossip learning as such, their theoretical analysis and new algorithm variants are relevant and merit further study.

Lalitha et al. [23] study the scenario where each node can possibly observe only a subset of parameters and the task is to learn a Bayesian model collaboratively without a server. The work is mainly theoretical, experimental evaluation is done with two nodes only, as an illustration.

Jameel et al. [18] focus on the communication topology, and attempt to design an optimal topology that is both fast and communication efficient. They propose a superpeer topology where superpeers form a ring and they all have a number of ordinary peers connected to them.

Lian et al. [24] and Tang et al. [33] introduce gossip algorithms and compare them with the centralized variant. Koloskova et al. [21] improve these algorithms via supporting arbitrary gradient compression. The main contribution in these works is a theoretical analysis of the synchronized implementation. Their assumptions on the network bandwidth are different from ours. They assume that the server is not unlimited in its bandwidth usage, and they characterize convergence as a function of the number of synchronization epochs. In our study, due to our edge computing motivation, we focus on convergence as a function of system-wide overall communication in various scenarios including realistic node churn. We perform asynchronous measurements along with optimization techniques, such as token-based flow control.

Giaretta and Girdzijauskas [14] present a thorough analysis of the applicability of gossip learning, but without considering federated learning. Their work includes scenarios that we have not discussed here including the effect of topology, and the correlation of communication speed and data distribution.

Ben-Hun and Hoefler [4] very briefly consider gossip alternatives claiming that they have performance issues.

8. Conclusions

Here, we compared federated learning and gossip learning to see to what extent doing away with central components—as gossip learning does—hurts performance. The first hurdle was designing the experiments. One has to be careful what system model is selected and what constraints and performance measures are applied. For example, the best algorithm will be very different when we grant a fixed overall communication budget to the system overall but allow for slow execution, or when we give a fixed amount of time and allow for utilizing all the available bandwidth at all the nodes.

Our choice was to allow the nodes to communicate within a configurable bandwidth cap that is uniform over the network, except the master node. This can be done in a continuous fashion, or in a bursty fashion. In the latter case, the cap is interpreted in terms of the average bandwidth usage in any time window of some fixed length. These models cover most application scenarios. Within these models, we were interested in the speed of convergence after a given amount of overall communication.

We observed several interesting phenomena in our various scenarios. In the random class assignment case (when nodes have a random subset of the learning examples) gossip learning is clearly competitive with federated learning. In the single class assignment scenario, federated learning converges faster, since it can mix information more efficiently. This includes the case

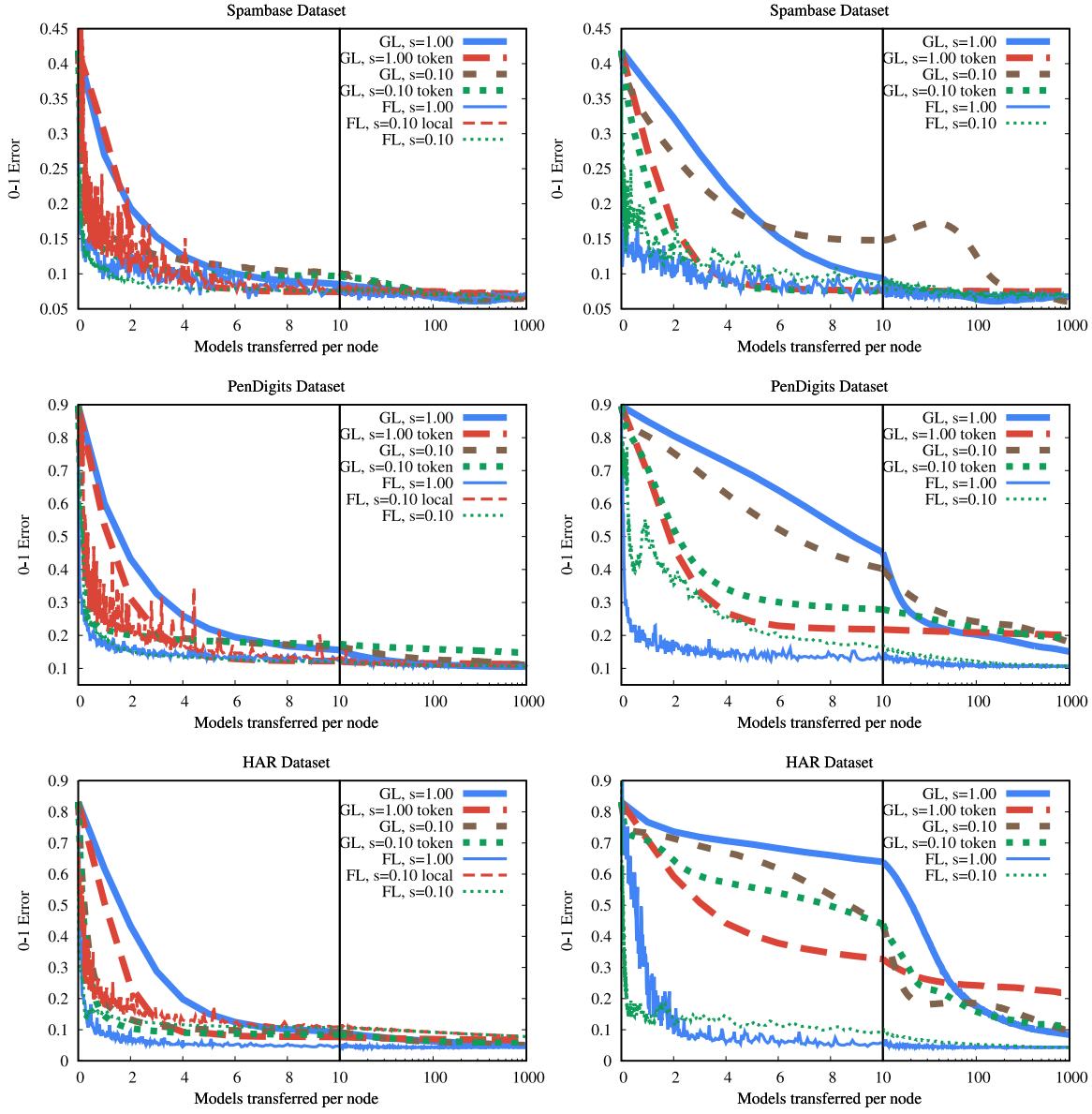

Fig. 10. Federated learning and gossip learning with 100 nodes (left) and with one node for each sample (right), no-failure scenario, in the bursty transfer scenario.

when every node has only a single example, as it is also a special case of single class assignment. However, gossip learning is able to converge as well in a practically realistic time frame. Here, we believe that gossip learning could be improved via applying more sophisticated peer sampling methods that are optimized to increase the efficiency of mixing different updates, or via applying a different learning rule, based on momentum methods, for example [3].

In our experimental setup we opted for putting the same cap on both upstream and downstream traffic in federated learning, as motivated in Section 5.2. But even if one removes this assumption and considers downstream traffic completely free, the downstream-compressed federated learning variant will converge only twice as fast, a relatively modest difference. Note that in gossip learning there is only peer-to-peer traffic, so there is only one cap.

The bursty traffic model broadens the algorithm design space, allowing for an array of techniques for scheduling the messages to achieve a speedup relative to the random schedule. We experimented with token-based flow control in gossip learning, and could achieve a performance comparable to federated learning in the random label assignment scenarios. In general, our results indicate that it is better to allow clients to communicate in bursts (maximal bandwidth for a short time) as opposed to setting a low bandwidth cap but allowing for continuous communication.

The token-based flow control approach outperforms the random gossip baseline that does not use any flow control. However, to achieve these advantages, the compression mechanism must be based on partitioning, as opposed to simple subsampling. The reason is that this way, the different partitions can form "hot potato" chains separately, whereas with subsampling, these chains cannot form because sampling picks different weights in every step.

In addition, we have only examined subsampling as a compression mechanism. There are more sophisticated compression techniques [9] that could potentially be applied in both federated and gossip learning.

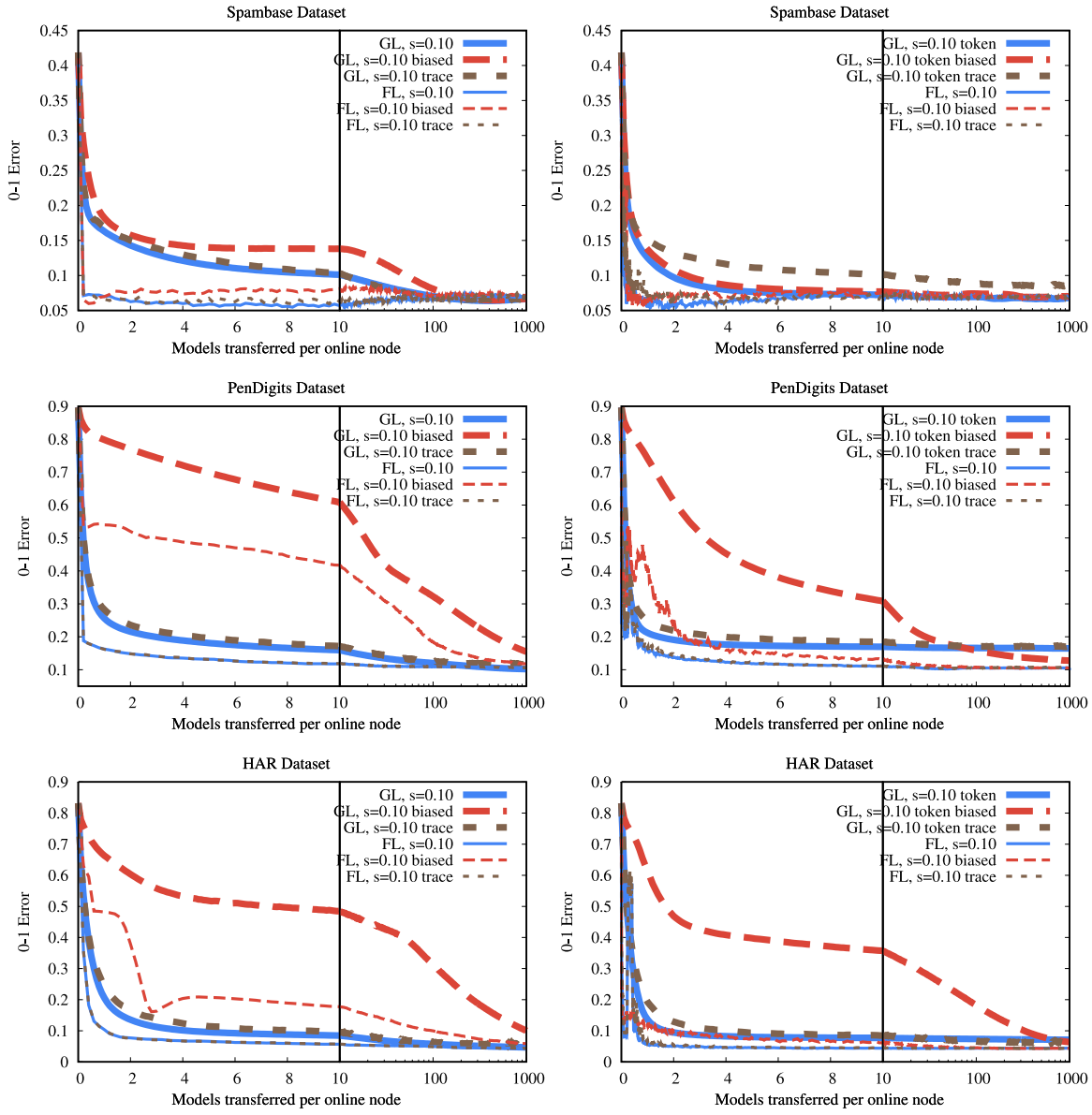

Fig. 11. Selected experiments in the large scale scenario. Continuous transfer (left) and bursty transfer (right). 'Biased' indicates single class assignment, 'trace' indicates the smartphone trace scenario.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] European Commission, General data protection regulation (GDPR), 2018, URL: https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules.

[2] D. Anguita, A. Ghio, L. Oneto, X. Parra, J.L. Reyes-Ortiz, A public domain dataset for human activity recognition using smartphones., in: 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), 2013.

[3] J. Ba, D. Kingma, Adam: A method for stochastic optimization, in: 3rd International Conference on Learning Representations (ICLR), 2015.

[4] T. Ben-Nun, T. Hoefler, Demystifying parallel and distributed deep learning: An in-depth concurrency analysis, ACM Comput. Surv. 52 (4) (2019) http://dx.doi.org/10.1145/3320060.

[5] Á. Berta, V. Bilicki, M. Jelasity, Defining and understanding smartphone churn over the internet: a measurement study, in: Proceedings of the 14th IEEE International Conference on Peer-to-Peer Computing (P2P 2014), IEEE, 2014, http://dx.doi.org/10.1109/P2P.2014.6934317.

[6] M. Blot, D. Picard, N. Thome, M. Cord, Distributed optimization for deep learning with gossip exchange, Neurocomputing 330 (2019) 287–296, http://dx.doi.org/10.1016/j.neucom.2018.11.002.

[7] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H.B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth, Practical secure aggregation for federated learning on user-held data, in: NIPS Workshop on Private Multi-Party Machine Learning, 2016.

[8] G. Danner, Á. Berta, I. Hegedűs, M. Jelasity, Robust fully distributed minibatch gradient descent with privacy preservation, Secur. Commun. Netw. 2018 (2018) 6728020, http://dx.doi.org/10.1155/2018/6728020.

[9] G. Danner, M. Jelasity, Robust decentralized mean estimation with limited communication, in: M. Aldinucci, L. Padovani, M. Torquati (Eds.), Euro-Par 2018, Springer International Publishing, 2018, pp. 447–461, http://dx.doi.org/10.1007/978-3-319-96983-1_32.

[10] G. Danner, M. Jelasity, Token account algorithms: The best of the proactive and reactive worlds, in: Proceedings of the 38th International Conference on Distributed Computing Systems (ICDCS 2018), IEEE Computer Society, 2018, pp. 885–895, http://dx.doi.org/10.1109/ICDCS.2018.00090.

[11] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, A.Y. Ng, Large scale distributed deep networks, in: Proceedings of the 25th International Conference on Neural

Information Processing Systems - Volume 1, Curran Associates Inc., USA, 2012, pp. 1223–1231.
[12] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186, http://dx.doi.org/10.18653/v1/N19-1423.
[13] D. Dua, C. Graff, UCI machine learning repository, 2019, URL: http://archive.ics.uci.edu/ml.
[14] L. Giaretta, Š. Girdzijauskas, Gossip learning: Off the beaten path, in: 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 1117–1124, http://dx.doi.org/10.1109/BigData47090.2019.9006216.
[15] I. Hegedűs, Á. Berta, L. Kocsis, A.A. Benczúr, M. Jelasity, Robust decentralized low-rank matrix decomposition, ACM Trans. Intell. Syst. Technol. (TIST) 7 (4) (2016) 62:1–62:24, http://dx.doi.org/10.1145/2854157.
[16] I. Hegedűs, G. Danner, M. Jelasity, Gossip learning as a decentralized alternative to federated learning, in: J. Pereira, L. Ricci (Eds.), Proceedings of the 19th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2019), Springer International Publishing, 2019, pp. 74–90, http://dx.doi.org/10.1007/978-3-030-22496-7_5.
[17] C. Hu, J. Jiang, Z. Wang, Decentralized federated learning: A segmented gossip approach, in: The 1st International Workshop on Federated Machine Learning for User Privacy and Data Confidentiality (IJCAI Workshop), 2019.
[18] M. Jameel, J. Grabocka, M. ul Islam Arif, L. Schmidt-Thieme, Ring-star: A sparse topology for faster model averaging in decentralized parallel SGD, in: Decentralized Machine Learning At the Edge (ECML PKDD 2019 Workshop), 2019.
[19] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, M. van Steen, Gossip-based peer sampling, ACM Trans. Comput. Syst. 25 (3) (2007) 8, http://dx.doi.org/10.1145/1275517.1275520.
[20] D. Kempe, A. Dobra, J. Gehrke, Gossip-based computation of aggregate information, in: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03), IEEE Computer Society, 2003, pp. 482–491, http://dx.doi.org/10.1109/SFCS.2003.1238221.
[21] A. Koloskova, S. Stich, M. Jaggi, Decentralized stochastic optimization and gossip algorithms with compressed communication, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, PMLR, Long Beach, California, USA, 2019, pp. 3478–3487.
[22] J. Konecný, H.B. McMahan, F.X. Yu, P. Richtárik, A.T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, in: Private Multi-Party Machine Learning (NIPS 2016 Workshop), 2016.
[23] A. Lalitha, S. Shekhar, T. Javidi, F. Koushanfar, Fully decentralized federated learning, in: Bayesian Deep Learning (NIPS 2018 Workshop), 2018.
[24] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, J. Liu, Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 5330–5340.
[25] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: A. Singh, J. Zhu (Eds.), Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, PMLR, Fort Lauderdale, FL, USA, 2017, pp. 1273–1282.
[26] A. Montresor, M. Jelasity, Peersim: A scalable p2p simulator, in: Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P 2009), IEEE, Seattle, Washington, USA, 2009, pp. 99–100, http://dx.doi.org/10.1109/P2P.2009.5284506, extended abstract.
[27] R. Ormándi, I. Hegedűs, M. Jelasity, Gossip learning with linear models on fully distributed data, Concurr. Comput.: Pract. Exper. 25 (4) (2013) 556–571, http://dx.doi.org/10.1002/cpe.2858.
[28] R. Roverso, J. Dowling, M. Jelasity, Through the wormhole: Low cost, fresh peer sampling for the internet, in: Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing (P2P 2013), IEEE, 2013, http://dx.doi.org/10.1109/P2P.2013.6688707.
[29] S. Savazzi, M. Nicoli, V. Rampa, Federated learning with cooperating devices: A consensus approach for massive IoT networks, IEEE Internet Things J. (2020) http://dx.doi.org/10.1109/JIOT.2020.2964162.
[30] A. Sayed, Adaptation, learning, and optimization over networks, Found. Trends Mach. Learn. 7 (4–5) (2014) 311–801, http://dx.doi.org/10.1561/2200000051, URL: http://iracema.icsl.ucla.edu/publications/books/now_2014/book.pdf.
[31] H. Shin, H.R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, R.M. Summers, Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning, IEEE Trans. Med. Imaging 35 (5) (2016) 1285–1298.
[32] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, W. Willinger, On unbiased sampling for unstructured peer-to-peer networks, IEEE/ACM Trans. Netw. 17 (2) (2009) 377–390, http://dx.doi.org/10.1109/TNET.2008.2001730.
[33] H. Tang, X. Lian, M. Yan, C. Zhang, J. Liu, D^2: Decentralized training over decentralized data, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, pp. 4848–4856.
[34] J. Wang, B. Cao, P.S. Yu, L. Sun, W. Bao, X. Zhu, Deep learning towards mobile applications, in: IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 2018, pp. 1385–1393, http://dx.doi.org/10.1109/ICDCS.2018.00139.
[35] Y. Zhang, J.C. Duchi, M.J. Wainwright, Communication-efficient algorithms for statistical optimization, J. Mach. Learn. Res. 14 (1) (2013) 3321–3363.

István Hegedűs is a research fellow at the Institute of Informatics of the University of Szeged. He obtained his Ph.D. degree in computer science in 2017 from the University of Szeged and he received his M.Sc. in computer science from the same university. He received the award for the most innovative Ph.D. thesis in 2017 at the University of Szeged. He has been a teaching artificial intelligence since 2009. His main research interests include fully distributed algorithms and decentralized machine learning. He has participated in several successful research and industrial projects in these areas.

Gábor Danner is currently pursuing a Ph.D. degree at the Doctoral School of Computer Science of the University of Szeged, Szeged, Hungary, where he is supervised by Márk Jelasity. He received his M.Sc. degree in Computer Science from the same institution in 2014. In 2015, he received a 1st prize at the National Scientific Students' Associations Conference (OTDK) with his work entitled "Calculating Ultra-Strong and Extended Solutions for Nine Men's Morris, Morabaraba, and Lasker." The main focus of his thesis is to make decentralized machine learning algorithms more efficient in terms of communication cost. His main research interests are fully distributed algorithms, data mining, and artificial intelligence in games.

Márk Jelasity is a full professor at the Institute of Informatics of the University of Szeged. He is the head of the Department of Algorithms and Artificial Intelligence. He received his Ph.D. degree in computer science from the University of Leiden in 2001. From 2000 until 2006 he was a postdoc at the Free University of Amsterdam and the University of Bologna. He visited Cornell University in 2013 as a Fulbright Scholar. He is the recipient of the 10 years best paper award at the ACM/IFIP/USENIX Middleware Conference (2014) and the Bolyai Plaquette of the Hungarian Academy of Sciences (2015). He worked in several areas including self-organizing systems, distributed computing, machine learning, and the intersections of these.