electronics

MDPI

Article

An Efficient Checkpoint Strategy for Federated Learning on Heterogeneous Fault-Prone Nodes

Jeonghun Kim and Sunggu Lee *

Department of Electrical Engineering, Pohang University of Science and Technology, Pohang 37673, Republic of Korea; salmon@postech.ac.kr
* Correspondence: slee@postech.ac.kr

Abstract: Federated learning (FL) is a distributed machine learning method in which client nodes train deep neural network models locally using their own training data and then send that trained model to a server, which then aggregates all of the trained models into a globally trained model. This protects personal information while enabling machine learning with vast amounts of data through parallel learning. Nodes that train local models are typically mobile or edge devices from which data can be easily obtained. These devices typically run on batteries and use wireless communication, which limits their power, making their computing performance and reliability significantly lower than that of high-performance computing servers. Therefore, training takes a long time, and if something goes wrong, the client may have to start training again from the beginning. If this happens frequently, the training of the global model may slow down and the final performance may deteriorate. In a general computing system, a checkpointing method can be used to solve this problem, but applying an existing checkpointing method to FL may result in excessive overheads. This paper proposes a new FL method for situations with many fault-prone nodes that efficiently utilizes checkpoints.

Keywords: federated learning; distributed system; checkpointing; performance modeling

check for updates

Citation: Kim, J.; Lee, S. An Efficient Checkpoint Strategy for Federated Learning on Heterogeneous Fault-Prone Nodes. *Electronics* **2024**, *13*, 1007. https://doi.org/10.3390/electronics13061007

Academic Editors: Leonardo Pantoli, Costas Psychalinos, Gaetano Palumbo, Egidio Ragonese and Paris Kitsos

Received: 29 December 2023
Revised: 5 March 2024
Accepted: 5 March 2024
Published: 7 March 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

According to recent deep-learning-based machine learning research, collecting more training data guarantees a higher accuracy and generalized models [1]. However it also comes with privacy leak issues and excessively long training times. Federated learning (FL) is a distributed machine learning method that does not directly send local data collected from many nodes (or clients) to the server, but instead trains on local nodes and then sends only the trained models to the server. The server creates a global model by integrating the local models trained at each node. FL can learn using a massive amount of training data through parallel computing on numerous nodes while reducing privacy issues caused by sensitive data collection. The abundance of training data improves the generalization of the global model and has the effect of increasing practical accuracy through fine tuning with local data at each node.

In practical usage, there are many different situations that can arise in FL. The number of nodes participating in training typically ranges from a few thousand to 10^7. In each round, all nodes may participate in training simultaneously, or only a subset of the nodes (e.g., 50 to 5000) may participate. Training times can be as long as 1 to 10 days [2]. In each round, the global training coordinator (usually the central server) decides which nodes will participate in training and initiates the training. Because the computing performance of client nodes can vary greatly, the time it takes to train the same amount of data can also vary greatly. Several nodes may even be turned off during training. Therefore, the coordinator does not wait until all nodes complete training, but waits only until a predetermined deadline and ignores the training results of nodes whose local training has not been completed by then.

Because FL is inherently based on a distributed computing system, taking appropriate action when problems occur at a node is also an important issue. Especially in FL, if appropriate action is not taken on the node where the problem occurs, training may fail because one or more bad local models are merged into the global model. Factors that can result in bad local models include the presence of a malicious attacker, hardware malfunctions, or training on outlier data. To deal with problems that arise during local training, various fault-tolerant methods can be used. As an example, a protocol that guarantees Byzantine fault tolerance can be used in a distributed learning system using FL [3,4]. In this case, even if problems occur in a certain number of nodes (fewer than a predetermined threshold), major problems in FL can be prevented.

Since nodes in FL are often mobile or edge devices that run on batteries and use wireless communication [2], it is desirable to avoid excessive computation or communication overhead. In particular, in the case of FL, each node can have a different computing performance, network environment, and failure probability. So, an appropriate level of action must be taken for each node to ensure efficient failure tolerance.

Checkpointing is one way to ensure efficient fault-tolerant operation of a computing system. Checkpointing means that the computer periodically saves the current memory state to trusted storage and restarts the process from the latest checkpoint when an error is detected. It has the advantage of being able to start from the middle of training when a problem occurs in a node without having to restart the operation from the beginning.

Incremental checkpointing (IC) is an efficient checkpointing method that results in a reduced checkpoint memory size. At the first checkpoint, the entire memory is stored in storage, and from the next checkpoint, only the parts of the memory that have been changed are saved. Although IC can have a large effect in general computing systems and algorithms, it has little effect in most FL systems. This is because the local model parameters, which occupy most of the memory in FL, change values even after only one training epoch. Even if there are a few parameters whose values do not change, the overhead is not reduced. This is because IC determines whether the value has changed on a virtual memory page basis.

Therefore, a new checkpointing method suitable for FL is needed. Although there are several studies utilizing checkpointing during FL [5–7], to the best of our knowledge, no study has focused on efficient checkpointing methods specifically targeted for FL.

The method proposed in this paper can help perform federated learning reliably, even with many fault-prone nodes, with as little overhead as possible. The proposed model can be used to determine an optimal checkpoint interval for each local node. Although it does not have much effect in cases where errors rarely occur, it is expected that this method will enable FL to be used in a much wider variety of situations than was previously possible.

The main contributions of this paper are as follows.

1. An efficient checkpointing method for FL that mitigates performance loss while reducing communication and computing overhead is proposed.
2. A systematic and analytical model for estimating the overhead of checkpointing at heterogeneous nodes where faults may exist is presented.
3. A method is presented for finding the optimal checkpoint interval for each node in a fault-tolerant FL system.

The rest of this paper is organized as follows. Section 2 provides necessary background regarding federated learning and checkpointing for distributed systems. Section 3 presents the system model that is used for analysis and proposes an efficient checkpointing method. Section 4 compares the communication overhead for the proposed method with previously proposed comparable checkpointing methods. Section 5 describes related work and the differences with the proposed model. Finally, Section 6 provides concluding remarks.

2. Background

2.1. Federated Learning

FL is a method that simultaneously meets the goals of training using as much data as possible and preventing user data from leaking during training. The most basic FL method

is to send a global model to local nodes, update the models locally using the local dataset only once, and then send the models back to the main server to merge them.

However, because this learning method incurs too much communication overhead, it updates each local model multiple times and then merges the global model. The process of sending a global model to multiple nodes, going through several local updates, and then merging them into one global model is called a round. If there are too many local training sessions per round, each local model may overfit the local dataset, thereby degrading the final global performance. The time taken for one round is proportional to the number of local training sessions per round and can vary from a few minutes to several hours.

2.2. Checkpointing and Fault Recovery

Checkpointing is a method in which a computing system periodically stores the current state in non-volatile memory or a stable remote server. This is a method that allows the system to restart from the most recently saved state when a problem occurs in the system instead of losing all work completed up to that point.

The most basic checkpointing method involves saving a copy of the entire current system memory at each checkpoint. If the size of the checkpoint is the same as the size of the entire memory, a lot of overhead will occur in creating and saving the checkpoint. There are two ways to reduce this overhead: one is to reduce the size of each checkpoint, and the other is to increase the checkpoint creation interval. The first method can be further divided into two methods: a method of compressing the checkpoint and a method of selecting information to be stored in the checkpoint. The latter method is typically implemented as incremental checkpointing, which reduces the checkpoint overhead by storing only the parts of memory whose values have changed when compared to the previous checkpoint. Figure 1 shows the checkpointing methods described above.

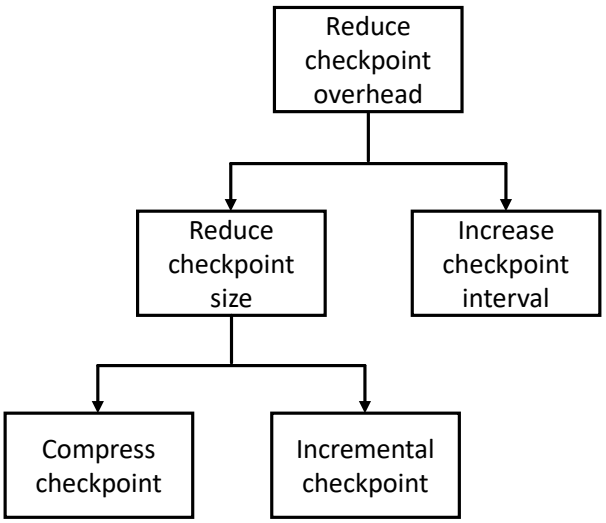


Figure 1. Different approaches used to reduce checkpoint overheads.

Checkpointing and recovery in an FL system is fundamentally different from the situation in general computing systems. First, in a typical FL situation, incremental checkpointing does not result in a reduced memory storage overhead since all trained parameter values in the local model change during each training phase. Second, a fault in one local node does not invalidate the entire global training process—it simply results in a slightly degraded level of training accuracy since the trained model from the faulty local node cannot be used in the accumulated training model. However, global training is conducted over a long time interval, and if many local nodes fail at least once during that long interval, this may result in a significantly degraded global training accuracy. Thus, a checkpointing and recovery approach specifically targeted for FL systems is necessary.

3. Proposed Method

A formal and analytical model is needed to systematically analyze the effect of checkpointing when running FL on a distributed system where faults exist. This section describes the distributed computing system assumed in this paper and the potential faults that can occur in FL. Then, the communication overhead on the network caused by checkpointing and the local training efficiency at each node are formulated and analyzed. Finally, based on these models and an analysis, we propose a method to find the optimal checkpointing method in a given FL environment. The notation used for model and analysis is summarized in Table 1.

Table 1. Symbols and units used in the analysis.

Type	Symbol	Description
Global	R	The number of global update rounds
	S	Model parameter size
	U	Time for global update deadline
	C	The number of local training steps
	N	The number of clients participating in training per round
Local	B	Network bandwidth
	T	Training time taken per one training step
	λ	Average number of fault occurrences per unit time
	p	Probability that a fault occurs within a local training step
	r	Ratio of compressed model size to original size
	I	Checkpoint interval
	O_t	Checkpoint time overhead
	O_e	Checkpoint energy overhead
	F	Fault penalty

3.1. System Model

The FL system assumed in this paper consists of a central server, multiple clients, and checkpoint storage. Clients have different computing performances, network environments, and defect occurrence probabilities. There may be multiple checkpoint repositories or just one. Figure 2 represents the described federated learning system.

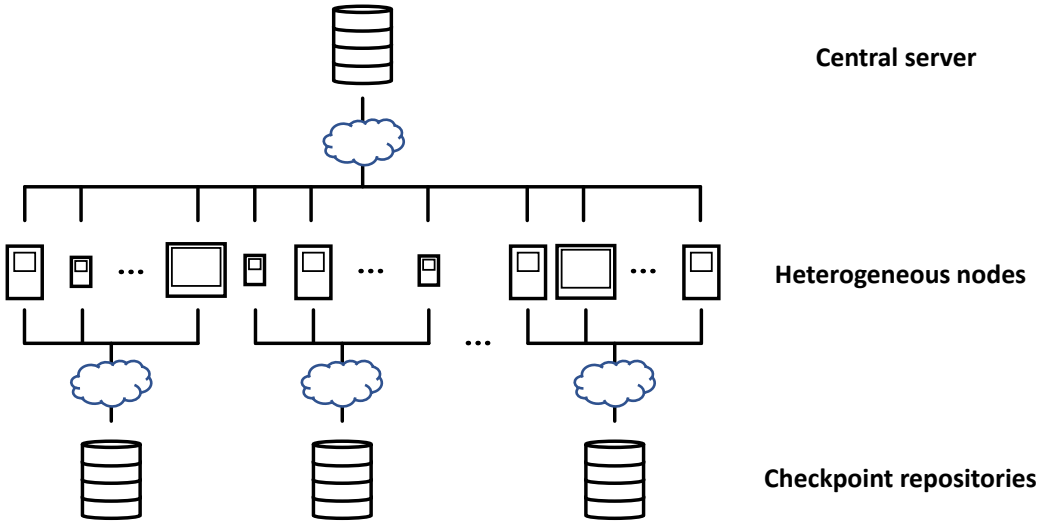


Figure 2. A graphical overview that explains FL in a heterogeneous node network with checkpointing.

Recent FL uses local training multiple times and then updates the global model [8,9]. However, because each client has a different computational performance, if the central server waits for all clients to complete local training, a few slow clients will hinder the

overall training speed. To alleviate this, a certain deadline time can be used, and if there is a client that has not completed local training within that time, the global update is performed excluding that client. It is also possible to consider merging local models that have not yet been trained, but if there is a difference in the amount of training of the local models to be merged, the variance of the local models may increase, which may hinder the convergence of the global model.

Although the above explanation is sufficient for a general FL system, this paper additionally considers the occurrence of defects in the client. In this paper, a defect refers to a situation where a local training progress is lost due to an unstable power supply or hardware malfunction. In the case of general FL, when this problem occurs in a client, the client is immediately excluded from the round and, like clients that do not meet the deadline, does not affect the global model. However, if there are many faulty nodes, the number of clients participating in global model training will be greatly reduced, and, as a result, FL performance will decrease. In this paper, it is assumed that when a fault occurs in the client, this local training is not completely excluded from global training, but training is restarted. If local training is completed before the deadline, it is merged into the global model like any other local model. However, the higher the probability of defects, the lower the probability of completing training before the deadline.

To alleviate this problem, checkpoints are periodically saved during the local training process, and then training starts again from the checkpoint if a problem occurs. This will significantly increase the chances of completing local training within the deadline if an appropriate checkpoint frequency is chosen. The checkpoints are stored in a reliable remote repository because if a local copy of the model is stored inside that node, the copy may become corrupted when the fault occurs. A central server where global models are stored can also be used as a checkpoint repository. Figure 3 and Algorithm 1 describe the FL process used in this paper. The sum of local training times in all nodes is $O(RNC)$.

Note that Algorithm 1 as shown does not include deadline considerations in order to simplify the explanation. In actual usage, the local models of all nodes are not merged, but only the local models of nodes that have completed local training within the deadline. To take this into consideration, the aggregation expression in line 19 of Algorithm 1 can be rewritten as follows:

$$w_i = \frac{\sum_{j=1}^{N} \mathbb{1}_A(j) w_{i,C}^j}{\sum_{j=1}^{N} \mathbb{1}_A(j)} \tag{1}$$

where $\mathbb{1}_A(x)$ is an indicator function, which returns 1 if $x \in A$, and 0 otherwise. A is the set of indices of nodes that have completed local training within the deadline.

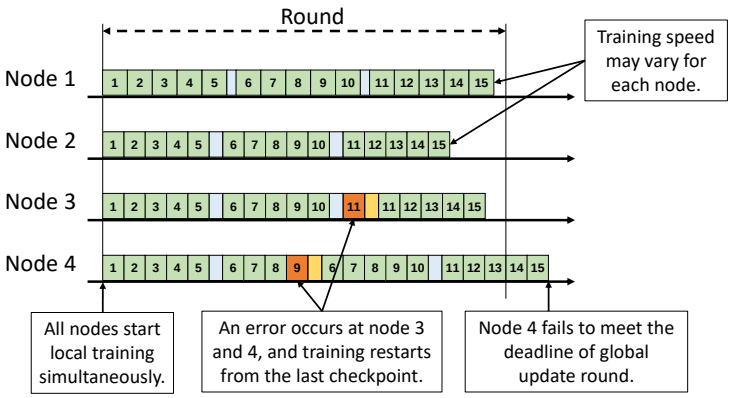


Figure 3. An example of how each node operates over time during one round. The number indicates which local training step is being processed. Green means a fault has not occurred, and red means a fault has occurred. Blue is the overhead for creating a checkpoint, and yellow shows where training is restarted from a previous checkpoint. (N = 4, C = 15, I = 5).

Algorithm 1 A checkpointing FL algorithm for a heterogeneous system with fault-prone nodes

Input: the number of global update rounds R, the number of nodes that participate in a training round N, the number of local training steps at each node C, the learning rate η, and a set of local datasets $\{\mathcal{D}^1, \mathcal{D}^2, \ldots, \mathcal{D}^N\}$

Output: w_R, the weight parameter after R training round.

initialize w_0
for each round $i = 1, 2, \ldots, R$ **do**
  for each node $j = 1, 2, \ldots, N$ **in parallel do**
    $w^j_{i,0} \leftarrow w_{i-1}$
    for each local training step $k = 1, 2, \ldots, C$ **do**
      $\Delta w^j_{i,k} \leftarrow -\eta \nabla L(w^j_{i,k-1}; \mathcal{D}^j)$
      if a fault occurs **then**
        $w^j_{i,k} \leftarrow \tilde{w}^j_{i,k-1}$
      else
        $w^j_{i,k} \leftarrow w^j_{i,k-1} + \Delta w^j_{i,k}$
        if $k\%I = 0$ **then**
          $\tilde{w}^j_{i,k} \leftarrow w^j_{i,k-1}$
        else
          $\tilde{w}^j_{i,k} \leftarrow \tilde{w}^j_{i,k-1}$
        end if
      end if
    end for
  end for
  $w_i \leftarrow \frac{1}{N} \sum_{j=1}^{N} w^j_{i,C}$
end for

3.2. Modeling Checkpoint Overhead and Fault Penalty

Creating checkpoints alleviates the performance degradation of the global model by reducing the chance of faulty clients being dropped during a round. However, creating excessive checkpoints actually has disadvantages in terms of the energy consumption and training time. Therefore, it is necessary to model the overhead that occurs each time a checkpoint is created and the fault occurrence penalty that occurs each time a fault occurs.

There are two type of overhead to consider: the checkpoint time overhead, O_t, which refers to the time required to send a checkpoint to a remote server and is affected by the model size, model compression rate, and network bandwidth, and the checkpoint energy overhead, O_e, which refers to the energy used to send a checkpoint to a remote server and is affected by the model size, model compression rate, and unit energy usage for data transmission. The model size and compression ratio are the same for all nodes, but the network bandwidth and unit energy usage for data transfer vary across nodes. Therefore, the overhead of checkpointing is also different for every node. The equations below represent two types of overhead. Here, S, B, r, and E represent the model size, network bandwidth, model compression ratio, and unit energy usage for data transfer.

$$O_t = \frac{r \cdot S}{B} \tag{2}$$

$$O_e = r \cdot S \cdot E \tag{3}$$

The fault occurrence penalty refers to the number of training steps lost when training is restarted from the last checkpoint after a fault occurs. For example, let us say that local training needs to be performed for 30 steps and a checkpoint is created every 10 steps, but an error occurs at step 25. In this case, local training restarts from the checkpoint at step 20, and the local model is merged into the global model after 10 more training steps. It can be said that the time spent on five training steps (=5T) was wasted.

The penalty per fault occurrence is a discrete random variable that is between 1 and I, where I represents the checkpoint interval. Note that the distribution is not uniform, and it is actually similar to a geometric distribution with an upper bound. The probability density of this distribution is determined by p, which represents the probability that a fault occurs within a local training step.

Calculating the expected value of the penalty is not difficult. We can also calculate the expected value of the sum of all penalties in a round. To calculate this, first, the expected value of the number of training steps performed to reach the next checkpoint from one checkpoint is calculated. If a fault does not occur I times in a row, the next checkpoint can be reached, and if a fault occurs at the j(<I)th step, it must restart from the previous checkpoint. Thus, the following equation holds:

$$x = (1-p)^I I + \sum_{j=0}^{I-1} p(1-p)^j (x+1+j) \tag{4}$$

where x represents the expected value of the number of training steps performed to reach the next checkpoint from one checkpoint. Although the expression looks complicated, it can be organized into an expression for x by multiplying both sides by $(1-p)$ and then subtracting it from the original expression.

$$x = \frac{1}{p}\left(\frac{1}{(1-p)^I} - 1\right) \tag{5}$$

In order to complete a total of C local training steps while creating a checkpoint at each I step, xC/I steps must be learned. Since the penalty is defined as the number of wasted steps, subtracting the C steps originally needed results in the following equation:

$$\mathbb{E}\left[\sum (\text{Fault penalty})\right] = \frac{C}{pI}\left(\frac{1}{(1-p)^I} - 1\right) - C \tag{6}$$

Using the above equations, the average time taken for a node to perform C local training steps during one round is obtained.

$$\begin{aligned} T_{total} &= \frac{S \cdot r}{B}\left\lceil \frac{C}{I} \right\rceil + T\left(\frac{C}{p \cdot I}\left(\frac{1}{(1-p)^I} - 1\right) - C\right) + C \cdot T \\ &= \frac{S \cdot r}{B}\left\lceil \frac{C}{I} \right\rceil + \frac{C \cdot T}{p \cdot I}\left(\frac{1}{(1-p)^I} - 1\right) \end{aligned} \tag{7}$$

In the first line, the first term is the time required to create a checkpoint, the second term is due to the fault penalty, and the third term is the time originally required for training regardless of a fault or checkpoint.

3.3. Optimal Checkpoint Interval

The system model described above and the checkpoint overhead and penalty calculations can be useful in determining checkpoint intervals to improve FL performance with less overhead. The main factors used when evaluating FL are the training convergence time and the final accuracy of the global model, but these factors are difficult to predict until actual training, except in special cases such as convex optimization.

As the number of clients that cannot complete local training by the end of the round increases, the number of rounds required for convergence increases and the accuracy of the final global model decreases. Therefore, in this paper, the optimal checkpoint is defined as the maximum checkpoint interval that can reduce the probability of a client being dropped below a certain level. Specifically, the optimal checkpoint I_{opt} is defined as follows:

$$I_{opt} = \arg \min_{\substack{1 \le I \le C, \\ T_{total} \le m \cdot U}} O_e \tag{8}$$

Here, m is a constant that indicates the allowable margin for failure probability. T_{total} and O_e are functions on I, as explained above. T_{total} bounds the range of I that ensures a low client failure rate, and O_e represents the cost that will be minimized.

4. Experimental Results

4.1. Simulation for Checkpoint Overhead and Fault Penalty

To verify the fault penalty model that is explained in Section 3.2, a simulation was conducted using MATLAB to evaluate the fault penalty. Figure 4 shows the fault penalty according to the fault occurrence probability p, assuming that the checkpoint interval is fixed at 100 and the number of local training steps per round is 500 ($I = 100, C = 500$). The blue line in the graph is the value calculated using Equation (6) in Section 3.2, and the orange line is the value obtained through simulation. Since the two results are very similar, it can be confirmed that the model is correct.

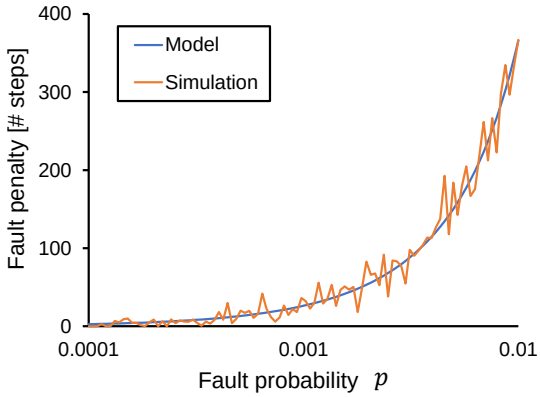


Figure 4. The simulation result and model comparison for the fault penalty.

Looking more closely at the values, when p is as small as 10^{-4}, the fault penalty is close to 0. This means that even without making checkpoints frequently, it is very likely that the client will send its trained local model back to the global server by the deadline. On the other hand, if p is 10^{-2}, the calculated fault penalty value is approximately 366, which means that approximately 70% more time is needed compared to when no fault occurs. This means that there is a high probability that the client will not be able to complete local training by the deadline, and more frequent checkpoints will need to be created to address this situation.

4.2. Federated Learning with Faulty Nodes and Checkpointing

In order to determine the negative impact of fault occurrence on the convergence speed and final accuracy of FL and how much checkpointing can alleviate the negative impact, two artificial neural networks that classify the MNIST dataset [10] and Google Speech Command (GSC) dataset [11] were trained using FL. For the GSC classification, the number of classes is 12.

A simple CNN architecture consisting of two convolutional layers (5×5 kernels with max pooling and ReLU activation) and two fully connected layers was used for MNIST classification. The output channel size of the first convolutional layer is 32, and the output channel size of the second convolutional layer is 64. The output size of the first fully connected layer is 512. The output size of the last layer is 10, which is the number of MNIST classes. BC-ResNet-8 [12] was used for GSC classification. BC-ResNet is a neural network model that was developed for efficient keyword spotting. It differs from the simple MNIST CNN model in that it has much more layers than the MNIST model, including depth-

wise and point-wise convolutional layers. The MNIST training dataset was divided into 100 clients so that none of them had identical data. When splitting the data, two cases were considered: a independent and identically distributed (iid) training dataset and a non-iid training dataset. In the case of the iid dataset, the entire training dataset is divided by the number of nodes regardless of class and then distributed. This is the result of assuming that the data distribution at all nodes is the same as the original dataset. In the case of the non-iid dataset, it is assumed that the distribution of learning data is different for each node. There are several ways to divide the dataset so that it has a non-iid distribution [13].

The method used in this set of experiments is as follows. First, sort the entire training data according to the class they belong to. Then, the sorted data are divided into (number of nodes)*G partitions and G groups are randomly distributed to each node. G is a parameter that indicates how unbalanced the local dataset is. Dividing the data in this way means that only some classes of data exist in one node. Although this data partitioning method seems simple, similar methods are often used in federated learning when assuming non-iid data partitioning [14–16]. The G value used in this set of experiments is 2 for MNIST and 4 for GSC.

For both MNIST and GSC, the number of clients participating in each round is 10, which is 10% of the total clients. A total of 30 rounds of global updates were conducted, with 50 local training steps per round. The batch size used in local training was 20 and the learning rate was 0.03. It was assumed that a fault occurs randomly in each node once every 12 h, and that one local training step takes 30 min. Measurements were made by changing the checkpoint interval to 1, 10, 25, and infinity. The code for the experiment was written in PyTorch 2.0 [17], a Python library for developing artificial neural networks, and was executed using GPUs on a Linux server with an Ubuntu 22.04 OS installed.

The experimental results are shown in Figure 5 and Table 2. Figure 6 shows the learning curve of training accuracy and Table 2 shows the final test accuracy. As a result of the experiment, it can be seen that fault occurrence has a significant impact on the convergence speed and accuracy, and that the smaller the checkpoint interval, the better the convergence speed and accuracy. In particular, in the case of the non-iid dataset, the degree of accuracy reduction due to fault occurrence was confirmed to be greater than that of the iid dataset. The reason why the accuracy drops significantly when there are not many checkpoints in non-iid situations is because the variance between local models is large, so even a small number of clients dropping out has a large impact on aggregation.

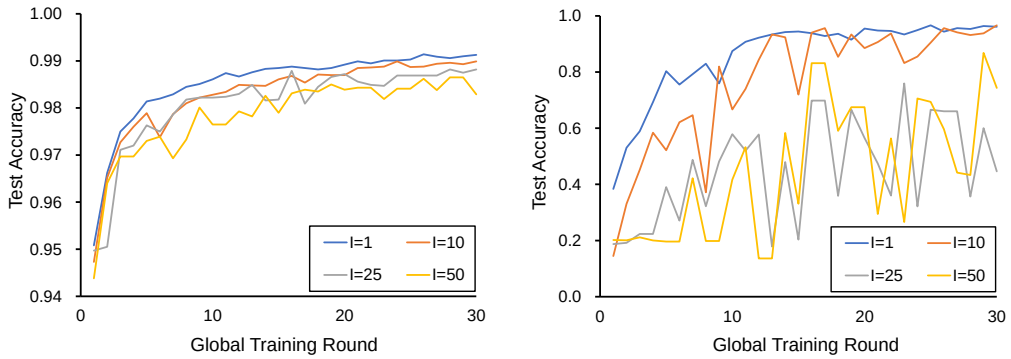
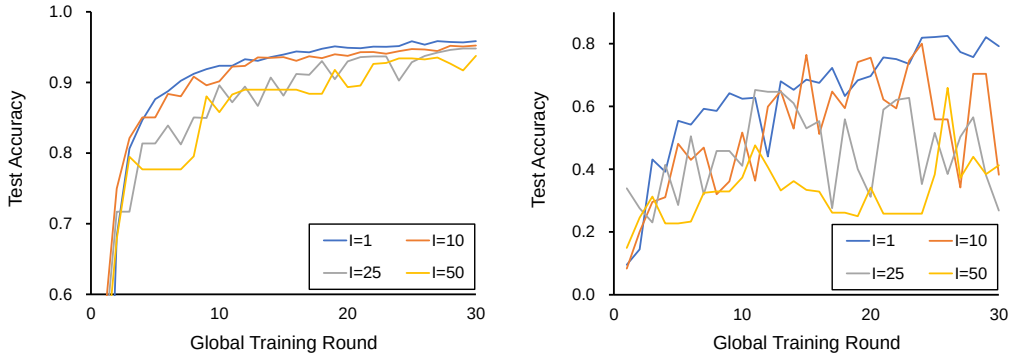


Figure 5. The MNIST classification task accuracy versus the number of global updates with (**a**) iid and (**b**) non-iid training datasets.



(**a**) (**b**)

Figure 6. The GSC classification task accuracy versus the number of global updates with (**a**) iid and (**b**) non-iid training datasets.

Table 2. The final test accuracy of different checkpoint intervals.

Dataset	Checkpoint Interval	Test Acc. (iid) [%]	Test Acc. (Non-iid) [%]
MNIST	1	99.07	94.40
	10	99.08	92.61
	25	98.87	87.74
	inf.	98.42	49.02
GSC	1	95.85	79.20
	10	95.24	38.22
	25	94.81	26.83
	inf.	93.76	41.25

4.3. Checkpoint Interval Decision: Heuristic vs. Modeling

An experiment was conducted to verify the effectiveness of the optimal checkpoint interval. A neural network performing MNIST classification, as used in Section 4.2, was used. However, unlike in the previous experiment, where the fault occurrence probability of all nodes was the same at the rate of once every 12 h, in this experiment, it was set to be evenly distributed between 0.1 every 12 h and 2 every 12 h. This was only tested for the non-iid case, because the iid case is too idealistic to represent the real situation.

Looking at the learning curve in Figure 7, if the checkpoint interval of all nodes is the same and the value is greater than 5, the learning curve fluctuates greatly and does not converge stably. If the checkpoint interval of all nodes is small, the final test accuracy appears to be high, but Table 3 shows that the number of clients that failed to complete training within the time was relatively high, which may have a negative impact on performance. When the checkpoint interval was set to 1, few nodes failed, but the overhead was large because too many checkpoints were created. When using the proposed optimal checkpoint interval, the final accuracy was high and the number of deleted nodes was low. In particular, compared to when the checkpoint interval of all nodes was fixed to 3, the number of nodes eliminated could be reduced by 42.5% for MNIST and 50% for GSC with almost the same overhead. Although there was no significant difference in the final accuracy between the two cases, the significant reduction in the number of dropped nodes means that it is resistant to fault occurrence.

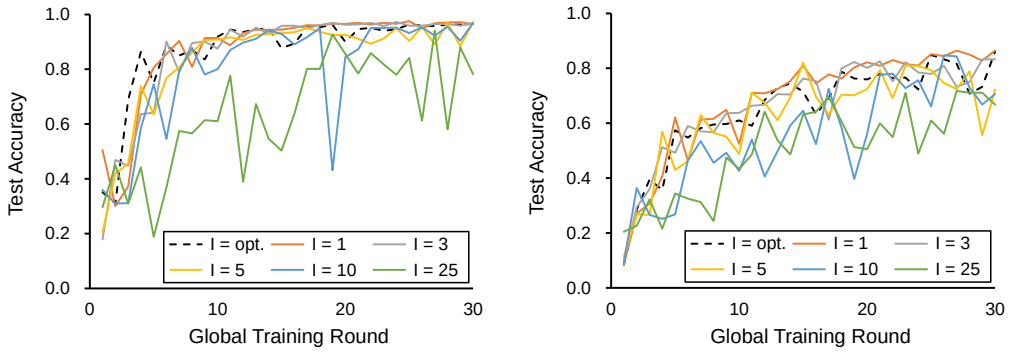


Figure 7. Learning curves of different checkpoint intervals ((**a**): MNIST, (**b**): GSC).

Table 3. The final test accuracy of FL on heterogeneous faulty nodes.

Dataset	Checkpoint Interval	# Failed Clients	Overhead *	Test Accuracy [%]
MNIST	Opt.	23	1694	96.4
	1	2	5000	96.3
	3	40	1600	96.1
	5	85	1000	96.1
	10	152	500	94.4
	25	211	200	82.1
GSC	Opt.	23	1582	86.1
	1	0	5000	86.5
	3	46	1600	83.3
	5	78	1000	72.3
	10	183	500	70.6
	25	194	200	66.7

* In this table, overhead means the sum of the number of checkpoints created on all local nodes during the entire FL process.

5. Related Works

Recently, there have been FL studies that consider defect occurrence. In [18], a federated reinforcement learning method was proposed that theoretically guarantees fault tolerance against random system faults or adversarial attacks by adding a subroutine to the aggregation process. In [19], a secure and fault-tolerant aggregation method for FL was proposed, which makes recovery possible even if failure occurs in some clients during the secure aggregation process.

There are papers that use checkpoints when implementing FL. In [5], checkpoints were used to reduce communication costs by checking the need for communication before starting communication between the server and client. In [6], checkpoints were used to search various training configurations while simultaneously training weights and hyperparameters. There is no research yet that proposes a method to determine the optimal checkpoint cycle for different clients with frequent defects.

There are also studies that are not related to FL but can be compared with the proposed method. Ref. [20] used checkpoints when training a recommendation model in a distributed system. This study used differential checkpointing, which stores only part of the model, making it suitable for use in recommendation models rather than general models. Ref. [21] proposed a method called Distributed Incremental Learning across the Computing Continuum (DILoCC), which can perform incremental learning [22,23] in a distributed system by additionally learning data in an already learned model. This method is related to FL in that data obtained from the edge device's sensor are directly used for learning. The difference between the two is that in DIL, new data are used only for learning the local model at the corresponding node, whereas in FL, they are used for learning the

global model. Ref. [24] proposed an analytical method to optimize checkpoint intervals in distributed stream processing systems. Ref. [25] proposed an adaptive checkpoint interval for distributed data parallel training. It uses an algorithm together with a simple analytical model to adaptively determine checkpoint intervals. Ref. [26] uses an asynchronous multi-level checkpointing method along with several techniques to reduce overhead, including efficient serialization. Table 4 compares the proposed method with related studies.

Table 4. A comparative table of the proposed method and other studies related to distributed learning in faulty systems.

	Uses Checkpoint	Uses an Analytical Method to Determine Checkpoint Interval	Local Dataset Is Private
Proposed.	Yes	Yes	Yes
[5]	Yes	No	Yes
[6]	Yes	No	Yes
[18]	No	No	Yes
[19]	No	No	Yes
[20]	Yes	No	No
[21]	No	No	Yes
[24]	Yes	Yes	No
[25]	Yes	Yes	No
[26]	Yes	No	No

6. Conclusions

This paper has introduced a new checkpointing FL method that is especially useful when there are many fault-prone clients. The checkpointing method helps clients successfully complete local training and send the local model back to the central server, resulting in a relatively high accuracy even when there are many unstable clients. Additionally, this paper proposes an analytical method to select the optimal checkpoint interval for each client. Thanks to this analytical method, efficient performance improvements with a minimal overhead are guaranteed because checkpoints are used only on clients with a high probability of fault occurrence. According to the experimental results, by using the checkpointing method, the FL accuracy did not drop even with a high fault occurrence rate. Furthermore, the optimal checkpoint interval decision based on proper analysis reduced the number of failed clients by 42.5% for the MNIST dataset and 50% for the Google Speech Command dataset with marginal overheads compared to a heuristic greedy method. To the best of our knowledge, this paper is the first study to analyze the use of client-side checkpoints in FL and can be applied in parallel with methods proposed in other studies.

Author Contributions: Conceptualization, J.K.; methodology, J.K.; software, J.K.; validation, J.K.; formal analysis, J.K.; investigation, J.K. and S.L.; writing—original draft preparation, J.K.; writing—review and editing, S.L.; visualization, J.K.; supervision, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by an Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2019-0-01906, Artificial Intelligence Graduate School Program (POSTECH)).

Data Availability Statement: The data presented in this study are available on request after the publication of the paper.

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

References

1. Sun, C.; Shrivastava, A.; Singh, S.; Gupta, A. Revisiting unreasonable effectiveness of data in deep learning era. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 843–852.
2. Kairouz, P.; McMahan, H.B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A.N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. Advances and open problems in federated learning. *Found. Trends® Mach. Learn.* **2021**, *14*, 1–210. [CrossRef]

3. Blanchard, P.; El Mhamdi, E.M.; Guerraoui, R.; Stainer, J. Machine learning with adversaries: Byzantine tolerant gradient descent. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 119–129.
4. Fang, M.; Cao, X.; Jia, J.; Gong, N. Local model poisoning attacks to Byzantine-Robust federated learning. In Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), Boston, MA, USA, 12–14 August 2020; pp. 1605–1622.
5. Xie, Y.; Wang, Z.; Gao, D.; Chen, D.; Yao, L.; Kuang, W.; Li, Y.; Ding, B.; Zhou, J. Federatedscope: A flexible federated learning platform for heterogeneity. *arXiv* **2022**, arXiv:2204.05011.
6. Hossain, I.; Puppala, S.; Talukder, S. Collaborative differentially private federated learning framework for the prediction of diabetic retinopathy. In Proceedings of the 2023 IEEE 2nd International Conference on AI in Cybersecurity (ICAIC), Houston, TX, USA, 7–9 February 2023; IEEE: Piscataway, NJ, USA , 2023; pp. 1–6.
7. Morell, J.Á.; Alba, E. Dynamic and adaptive fault-tolerant asynchronous federated learning using volunteer edge devices. *Future Gener. Comput. Syst.* **2022**, *133*, 53–67. [CrossRef]
8. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the Artificial Intelligence and Statistics, PMLR, Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
9. Reddi, S.; Charles, Z.; Zaheer, M.; Garrett, Z.; Rush, K.; Konečnỳ, J.; Kumar, S.; McMahan, H.B. Adaptive federated optimization. *arXiv* **2020**, arXiv:2003.00295.
10. Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142. [CrossRef]
11. Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv* **2018**, arXiv:1804.03209.
12. Kim, B.; Chang, S.; Lee, J.; Sung, D. Broadcasted residual learning for efficient keyword spotting. *arXiv* **2021**, arXiv:2106.04140.
13. Hsu, T.M.H.; Qi, H.; Brown, M. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv* **2019**, arXiv:1909.06335.
14. Zhao, Y.; Li, M.; Lai, L.; Suda, N.; Civin, D.; Chandra, V. Federated learning with non-iid data. *arXiv* **2018**, arXiv:1806.00582.
15. Sattler, F.; Wiedemann, S.; Müller, K.R.; Samek, W. Robust and communication-efficient federated learning from non-iid data. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *31*, 3400–3413. [CrossRef] [PubMed]
16. Zhao, Z.; Zhang, Y.; Guo, D.; Zhao, S.; Zhu, X. Communication-efficient federated continual learning for distributed learning system with Non-IID data. *Sci. China Inf. Sci.* **2023**, *66*, 122102. [CrossRef]
17. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*.
18. Fan, X.; Ma, Y.; Dai, Z.; Jing, W.; Tan, C.; Low, B.K.H. Fault-tolerant federated reinforcement learning with theoretical guarantee. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 1007–1021.
19. Mansouri, M.; Önen, M.; Ben Jaballah, W. Learning from failures: Secure and fault-tolerant aggregation for federated learning. In Proceedings of the 38th Annual Computer Security Applications Conference, Austin, TX, USA, 5–9 December 2022; pp. 146–158.
20. Eisenman, A.; Matam, K.K.; Ingram, S.; Mudigere, D.; Krishnamoorthi, R.; Nair, K.; Smelyanskiy, M.; Annavaram, M. Check-N-Run: A checkpointing system for training deep learning recommendation models. In Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), Renton, WA, USA, 4–6 April 2022; pp. 929–943.
21. Cicceri, G.; Tricomi, G.; Benomar, Z.; Longo, F.; Puliafito, A.; Merlino, G. DILoCC: An approach for Distributed Incremental Learning across the Computing Continuum. In Proceedings of the 2021 IEEE International Conference on Smart Computing (SMARTCOMP), Irvine, CA, USA, 23–27 August 2021; IEEE: Piscataway, NJ, USA , 2021; pp. 113–120.
22. He, H.; Chen, S.; Li, K.; Xu, X. Incremental learning from stream data. *IEEE Trans. Neural Netw.* **2011**, *22*, 1901–1914. [PubMed]
23. Joshi, P.; Kulkarni, P. Incremental learning: Areas and methods—A survey. *Int. J. Data Min. Knowl. Manag. Process* **2012**, *2*, 43. [CrossRef]
24. Jayasekara, S.; Harwood, A.; Karunasekera, S. A utilization model for optimization of checkpoint intervals in distributed stream processing systems. *Future Gener. Comput. Syst.* **2020**, *110*, 68–79. [CrossRef]
25. Mohan, J.; Phanishayee, A.; Chidambaram, V. CheckFreq: Frequent, Fine-Grained DNN Checkpointing. In Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST 21), Santa Clara, CA, USA, 23–25 February 2021; pp. 203–216.
26. Nicolae, B.; Li, J.; Wozniak, J.M.; Bosilca, G.; Dorier, M.; Cappello, F. Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models. In Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, VIC, Australia, 11–14 May 2020; IEEE: Piscataway, NJ, USA , 2020; pp. 172–181.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.