Federated Learning Systems: Architecture Alternatives

Hongyi Zhang
Chalmers University of Technology
Gothenburg, Sweden.
hongyiz@chalmers.se

Jan Bosch
Chalmers University of Technology
Gothenburg, Sweden.
jan.bosch@chalmers.se

Helena Holmström Olsson
Malmö University
Malmö, Sweden.
helena.holmstrom.olsson@mau.se

2020 27th Asia-Pacific Software Engineering Conference (APSEC) | 978-1-7281-9553-7/20/$31.00 ©2020 IEEE | DOI: 10.1109/APSEC51365.2020.00047

***Abstract*—Machine Learning (ML) and Artificial Intelligence (AI) have increasingly gained attention in research and industry. Federated Learning, as an approach to distributed learning, shows its potential with the increasing number of devices on the edge and the development of computing power. However, most of the current Federated Learning systems apply a single-server centralized architecture, which may cause several critical problems, such as the single-point of failure as well as scaling and performance problems. In this paper, we propose and compare four architecture alternatives for a Federated Learning system, i.e. centralized, hierarchical, regional and decentralized architectures. We conduct the study by using two well-known data sets and measuring several system performance metrics for all four alternatives. Our results suggest scenarios and use cases which are suitable for each alternative. In addition, we investigate the trade-off between communication latency, model evolution time and the model classification performance, which is crucial to applying the results into real-world industrial systems.**

***Index Terms*—Federated Learning, System Architecture, Machine Learning, Artificial Intelligence**

I. Introduction

Federated learning is a new basic technology of artificial intelligence. It was originally proposed by Google in 2017, with the aim to solve the problems of local model training and updating in mobile edge devices [1] [2] [3]. The design goal of Federated Learning is to carry out efficient machine learning among multiple participants or computing nodes on the premise of ensuring the information security during massive data exchange, protecting the privacy of terminal data and personal data and ensuring legal compliance. Federated learning has the potential to be the foundation of the next generation of AI collaborative algorithms and networks [4].

Federated learning defines a machine learning framework, in which a global model is designed to solve the problem of collaboration between multiple data owners without exchanging data [1]. The global model is the optimal model which is the aggregated knowledge from all parties. Federated learning requires that the modelling result should be infinitely close to the traditional pattern, that is, the data belonging to multiple owners should be gathered in one place for modelling results [5]. Since the data is not exchanged, it will not take the risk of leaking the user's privacy or affecting the data specification which meets the requirements of legal compliance (such as GDPR [6]). Figure 1 shows the system architecture of Federated learning with two data owners (edge A and edge B) as an example. The system can be extended to scenarios with multiple edge data owners. Suppose that edge A and B want to train a machine learning model jointly, and their business systems have the relevant data of their respective users. If A and B are both allowed to exchange data directly, for example, because of the data privacy and security issues, we may apply the Federated Learning system to build the model.

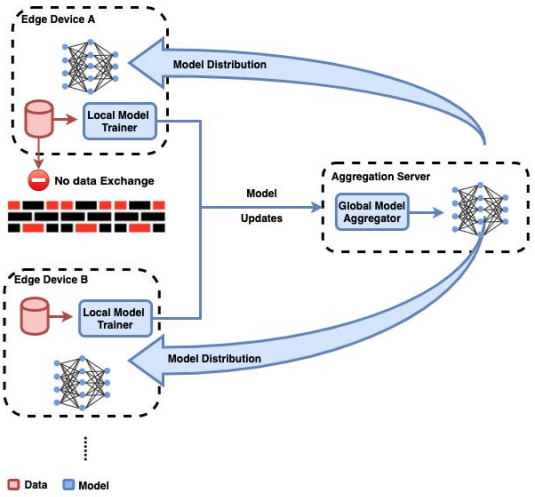


Fig. 1. System architecture of Federated learning

However, our research shows the challenges of deploying Federated Learning into a real-world industrial context. As defined in *"Engineering AI Systems: A Research Agenda"* [7], AI engineering refers to AI/ML-driven software development and deployment in production contexts. Also, our previous research shows that **the transition from prototype to the production-quality deployment of ML models proves to be challenging for many companies** [8] [9]. The situation also applies to Federated Learning systems [10]. Currently, the majority of deployments utilize a single-server centralized architecture which may inevitably face the risk of component failure, system scalability, communication efficiency, etc [4]. Those problems will prevent the AI/ML components from being continuously serviceable in real-world industrial deployments, which can compromise the system and lead to terrible accidents in the end.

2640-0715/20/$31.00 ©2020 IEEE
DOI 10.1109/APSEC51365.2020.00047

To the best of our knowledge, there is limited research that provides an overview of the different architecture alternatives for the Federated Learning systems. In this paper, based on our simulation, we describe and suggest several applicable scenarios and use cases for four different architecture reported in this paper which can be applied to an industrial Federated Learning system. We conduct the study using two well-known image classification data sets, MNIST and CIFAR-10. All the training data are distributed to edge devices that follow a statistical distribution to simulate real-world scenarios. In order to provide comprehensive suggestions, for each alternative, communication latency, model evolution time and model classification performance are measured and compared.

The contribution of this paper is threefold. First, we introduce four architecture alternatives which have been or can be applied to a Federated Learning system and we identify the advantages and disadvantages of each alternative. Second, we evaluate the system performance, including weights update latency, model evolution speed and model classification performance with each of the architecture alternatives. Third, by studying the trade-off between model performance and the overhead of latency and evolution speed, we describe for which industrial scenario each architectural alternative reported in this paper is the optimal choice.

The remainder of this paper is structured as follows. Section II introduces four architecture alternatives. Section III details our research method, including the simulation testbed, the method of distributing the training data set, the utilized machine learning method and the evaluation metrics. Section IV presents the algorithms utilized in each alternative. Sections V evaluates four architecture alternatives applied to the data traces. Section VI outlines the discussion on suitable scenarios and use cases for each alternative. Finally, Section VII presents conclusions and future work.

II. Architecture Alternatives

As described in Section I, current Federated Learning systems may face the problem of components failure, system scalability, communication efficiency, etc. Inspired from the empirical results of existing literature [4] [1] [11] [12], we have defined four alternatives which can be utilized in a Federated Learning system from a centralized to a fully decentralized approach, that is, centralized, hierarchical, regional and decentralized architectures. The terms of each architecture are derived based on their characteristic. Figure 2 illustrates the concepts.

A. Centralized Architecture

The centralized architecture is a widely used setting in the majority of current Federated Learning systems [2] [13] [14]. In this alternative, there is only a single central node which is responsible for communicating all edge devices, aggregating local models, and deploying the global model. The model transmission within this architecture is smooth and elegant and the single central node has a dedicated system which can be modified to suit customized needs. Quick updates become possible and it is efficient for small systems, as the central systems take limited resources to set up. In addition, any edge node can be easily detached from the system by removing the connection between the client node and the server without influencing other active nodes.

However, because of the single management node, the centralized Federated Learning system will encounter a scalability problem. When thousands of client nodes join, the server node will not have improved performance even if the hardware and software capabilities have been optimized. In addition, communication bottlenecks may appear when the amount of traffic increases exponentially and the system can easily break down when the server suffers a Denial-of-Service attack.

B. Hierarchical Architecture

As shown in Figure 2 (b), different from the centralized architecture, a hierarchical architecture introduces several regional coordination nodes to manage different edge clusters, which can ease the work of the central node, such as model updating and aggregation. This alternative has been introduced in [2], which solves part of the communication bottleneck problem and is scalable for a medium system. However, this approach still has the potential problem of the single-point of failure and being vulnerable to DoS attack since the central node still exists. In addition, the management cost will increase and the industrial deployments may need more budgets for more aggregation servers compared to a centralized architecture alternative.

C. Regional Architecture

The regional architecture has a similar setting compared to the hierarchical architecture but removes the central aggregation nodes. Each edge cluster will be assigned to a regional aggregation node where models are aggregated and exchanged. One application which utilizes this alternative is reported in [11]. The results demonstrate the computational efficiency compared to a more centralized architecture. The purpose of this design is to avoid the influence of the central node failure and to increase system robustness. In addition, after defining the frequency of local model exchange among regional aggregation nodes, a system may have a chance to focus more on their local sample clusters instead of the whole data set at the edge. However, with the increasing number of servers, real-world deployments may cost more in terms of hardware purchases and server configuration management.

D. Decentralized Architecture

As shown in Figure 2 (d), a decentralized Federated Learning system only contains edges nodes. Compared to the three alternatives above, a decentralized architecture moves the aggregation function to the edge. The idea is firstly tried and reported in [15]. The system is able to minimize the problem of performance bottlenecks since the entire load gets balanced on all the nodes. Furthermore, due to the flexibility of node connections, the system has better autonomy and is able to quickly adapt its local environment changes.

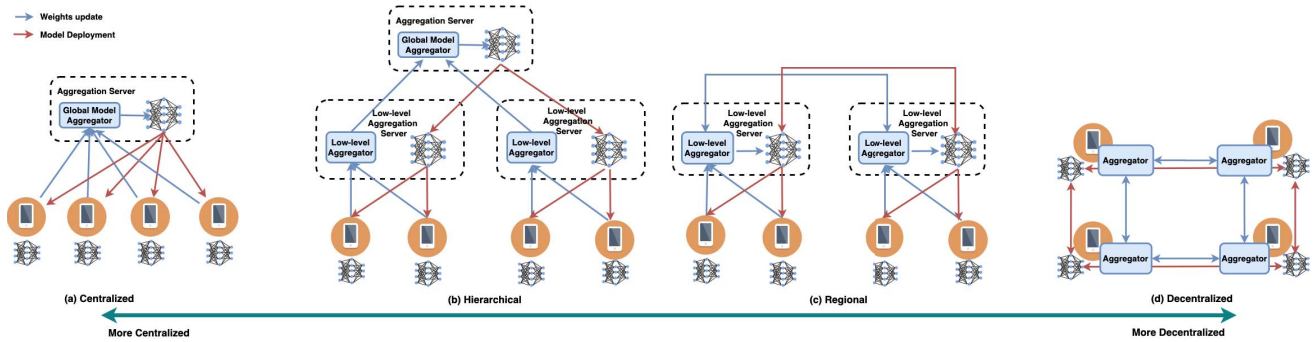


Fig. 2. Architecture alternatives for Federated Learning systems: centralized, hierarchical, regional and decentralized architecture. For a centralized Federated Learning system ((a)), all the edge nodes are connected to the central aggregation node in order to update local weights and distribute models. An improved way ((b)) is to add several coordinators, the regional aggregation nodes, which aims to reduce data exchange and be in charge of managing local devices. The regional architecture ((c)) will totally remove the central management point in order to remove the risk of the single-point of the failure. A more elegant way ((d)) is to completely move the aggregation function to the edge. Each edge node can perform local training and model aggregation. This is a potential alternative when a global or regional sever faces the problem of heavy traffic and then becoming a bottleneck.

Nevertheless, decentralized architecture can lead to the problem of coordination. Since every node is the owner of its own behaviour, it is difficult to achieve collective tasks and global knowledge. Normally, the models vary a lot which is not optimal for some scenarios. Additionally, it is not suitable for small systems since industries cannot benefit from building and operating small decentralized systems due to inefficient system management and performance.

III. Research Method

In this research, the empirical method and learning procedure described in Zhang [16] was applied to make a quantitative measurement and comparison with four architecture alternatives. In the following sections, we present our simulation testbed, the method used for splitting and distributing data sets, evaluation metrics and the machine learning methods used in the experiments.

A. Simulation Testbed

Figure 3 outlines our testbed topology. In order to simulate aggregation and edge functions, we adopted two of the total six machines as our server cluster and the rest work as the edge. (Table I shows the hardware setup for all the servers) Each edge nodes were implemented as a small process running in one of the edge nodes server cluster (server 3-6).

TABLE I
Hardware setup for testbed server

CPU	Intel Xeon Processor (Skylake, IBRS)
Cores	8
Frequency	2.59 GHz
Memory	32 GB
OS	Linux 4.15.0-106-generic

More specifically, in the centralized architecture simulation, aggregation functions were deployed on server 1 while the edge nodes in server 3-6 can push and request the latest model to or from server 1 to continuously learn latent patterns.

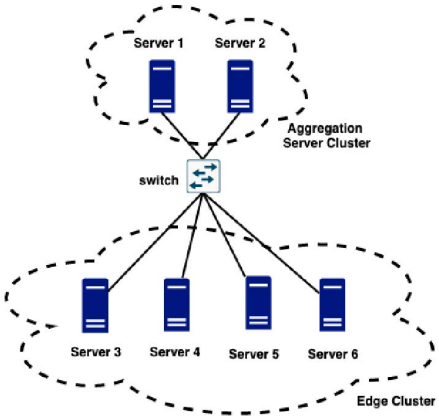


Fig. 3. Topology of the simulation testbed

In the hierarchical architecture, the central aggregation function was deployed in server 1 while we assigned four regional aggregation processes in server 1 and 2. Edge nodes in each edge server were assigned to one of the regional aggregation processes, which means that those nodes will only contact their corresponding regional aggregation process.

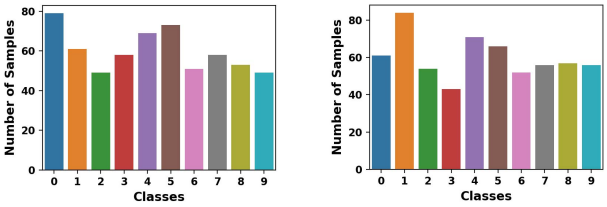
For the regional architecture simulations, the aggregation functions were deployed both on server 1 and 2. Similar to the hierarchical architectures simulation, edge nodes were assigned to one of the aggregation processes once they joined in the system and only communicated with that unique aggregation node. In the decentralized simulation, we removed the aggregation server cluster and moved the aggregation functions to all the edge nodes in order to simulate decentralized features. In each edge nodes, their neighbour nodes were predefined based on their edge ID.

B. Training Data Distribution

For the purpose of this study, we used two kinds of the edge data distribution to analyze system performance under

different architecture alternatives.

1) Uniform Distribution: Under this setting, we distributed training data samples to the edge follows the uniform distribution, which means the number of data samples of each target classes is equally likely. Figure 4 outlines the data distribution in two example edge nodes.



(a) Edge 0 (b) Edge 49

Fig. 4. Uniform training data distribution

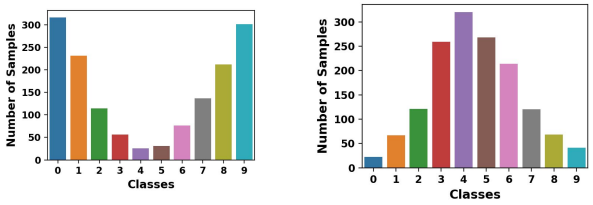
2) Normal Distribution: With this setting, in each edge nodes, the number of samples in each class follows the normal density function as shown below.

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

Here, μ and σ are defined as follows:

$$\mu = \frac{k \times N}{K}, \sigma = 0.2 \times N$$

where k is the ID of each edge node, K is the total number of edge nodes and N equals to the total number of target classes in training data. The purpose of this configuration is to provide various distribution in different edge nodes, where each class can have the probability to have the majority number of samples in one node. Figure 5 outlines the data distribution in two example edge nodes.



(a) Edge 0 (b) Edge 49

Fig. 5. Normal training data distribution

C. Machine Learning Method

The models used in this paper were implemented in Python, using torch 1.4.0 [17], torchvision 0.5.0 [18] and scikit-learn [19] libraries for model building.

In order to achieve a satisfying classification result, two different convolutional neural networks (CNN) [20] were trained for the MNIST and CIFAR-10 data sets. In the MNIST data set experiment, the CNN network contains two 5x5 convolution layers, (The first layer has 10 output channels, while the second has 20, each followed with 2x2 max pooling.) a fully connected layer with 50 units and the ReLu activation, and a linear output layer.

For the CIFAR-10 data set, the CNN network contains four 5x5 convolution layers, (The first layer has 66 output channels; the second has 128 output channels and the stride of convolution equals 2; the third has 192 channels; the fourth has 256 channels and the stride of convolution equals 2.), two fully connected layers (ReLu activation) with 3000 and 1500 units, and a linear output layer.

D. Evaluation Metrics

In order to demonstrate fruitful results of systems under different architecture alternatives, we selected three metrics including weights update latency, model evolution time and model classification performance (local and global).

1) Weights update latency: The weights updated latency is defined as the time difference of the model transmission from edge nodes to the aggregation nodes (In the centralized, hierarchical, regional architecture, aggregation nodes are central or regional servers which are responsible for collecting models. In the decentralized architecture, since aggregation function is moved to the edge, the aggregation node can be regarded as the peer node which is ready for receiving the updated model). The result is the average of all edge nodes during one training round. This metric indicates the network situation and communication overhead of each architecture alternatives. The metrics were measured in all the model receivers by checking the sending and receiving timestamp.

2) Model Evolution time: Evolution time is defined as the time difference between two different versions of the deployed global model at the edge nodes. The result is the average of all edge nodes during one training round. This metric demonstrates the speed of local edge devices updating their knowledge which is crucial and important for those systems which need to quickly evolve to adapt to the rapidly-changed environment. The metrics were measured in all the edge nodes by checking model deployment timestamp.

3) Model Classification Performance: Classification performance is the most important metric which indicates the quality of the training model. It is defined as the percentage of correctly recognized images among the total number of testing images. Furthermore, in order to have a better understanding of the influence of different architectures on local edge devices. Here, the *local classification performance* was tested on each edge devices by using their updated global model. The test sample distribution should be the same as the training samples (local test set). The result of local classification performance is the average value from all edge nodes. The *global classification performance* is tested by using the global test set, where the number of samples in different classes should be equally likely.

IV. Algorithms Used in Each Architecture Alternative

In order to simulate and compare characteristics of the system with the architecture alternatives reported in this paper, we select Federated Averaging (FedAvg) [1] as the base Federated Learning algorithm during our experiments. This

algorithm has been widely used in research and industrial communities for model aggregation. Thus, it is also compelling to see how FedAvg behaves with the architecture alternatives introduced in section II. In a centralized architecture, the original Federated Average algorithm is applied while for the other three alternatives, the base algorithm is modified to fit different architectures.

A. Centralized Architecture

The algorithm used in the centralized architecture is outlined in Algorithm 1. Since this architecture has been widely used in various fields, we didn't change any components and make the setting remain the same as all existing research. The steps of FedAvg algorithm in the centralized architecture is straightforward:

Step 1: Edge devices locally compute the model; After reaching the number of local epochs, they send updated model results w to the central aggregation node.

Step 2: The central node performs aggregation by averaging all updated models to form a global knowledge of w_{t+1}.

Step 3: The node sends back the aggregated result to each edge device k.

Step 4: Edge device replaces the local model and performs further local training by using the global deployed model.

Algorithm 1: FedAvg - Centralized: In the system, total K edge devices are indexed by k; B is the local mini-batch size; E represents the number of local epochs, and γ is the learning rate.

Function `Server_Function()`**:**
  initialize w_0
  for *each round t = 1, 2, ...* **do**
    $m \longleftarrow max(C \times K, 1)$;
    $S_t \longleftarrow$(random set of m clients);
    for *each client* $k \in S_t$ ***in parallel*** **do**
      $w_{t+1}^k \longleftarrow Client\_Update(w_t)$;
    end
    $w_{t+1} \longleftarrow \sum_{k=1}^{K} \frac{1}{K} w_{t+1}^k$;
  end
End Function

Function `Client_Update(w)`**:**
  $\beta \longleftarrow$(split P_k into batches of size B);
  for *each local epoch i from 1 to E* **do**
    for *batch* $b \in \beta$ **do**
      $w \longleftarrow w - \gamma \nabla l(w; b)$;
    end
  end
  return w to server
End Function

B. Hierarchical Architecture

The algorithm (Algorithm 2) used in this alternative is modified based on the Federated Averaging algorithm. Since the system has several regional coordination nodes, all the edge nodes send their weights updates only to their corresponding regional nodes. After receiving local models, a regional coordination node sums all models and counts the number of received models. Then, these information will then be updated to the central node. Therefore, the central node only needs to process the information sent from coordinator nodes without contacting numerous edge devices, which largely releases and balances the computation work at the central point. The steps can be summarized as follows:

Step 1: Edge devices locally compute the models; After reaching the number of local epochs, they send updated model results w to the regional aggregation nodes.

Step 2: The regional nodes perform aggregation by adding all updated models and calculate the number of updated models. Then, these information will be sent to the central node to form a global knowledge of w_{t+1}.

Step 3: The central node sends back the aggregated result to each regional nodes. Regional nodes will then forward

Algorithm 2: FedAvg - Hierarchical

Function `Server_Function()`**:**
  initialize w_0
  for *each round t = 1, 2, ...* **do**
    $S_l \longleftarrow (localserverset)$
    for *each local server* $s \in S_l$ ***in parallel*** **do**
      $w_{t+1}^s, k^s \longleftarrow Local_Server\_Update(w_t)$;
    end
    $K_{t+1} \longleftarrow \sum_{s=1}^{S} k^s$
    $w_{t+1} \longleftarrow \sum_{s=1}^{S} \frac{1}{K_{t+1}} w_{t+1}^k$;
  end
End Function

Function `Local_Server_Update(`w_t`)`**:**
  for *each round t = 1, 2, ...* **do**
    $m \longleftarrow max(C \times K, 1)$;
    $S_t \longleftarrow$(random set of m clients);
    for *each client* $k \in S_t$ ***in parallel*** **do**
      $w_{t+1}^k \longleftarrow Client\_Update(k, w_t)$;
    end
    $w_{t+1} \longleftarrow \sum_{k=1}^{K} w_{t+1}^k$;
  end
  return w_{t+1}, $len(S_t)$ to central server
End Function

Function `Client_Update(`*k, w*`)`**:**
  $\beta \longleftarrow$(split P_k into batches of size B);
  for *each local epoch i from 1 to E* **do**
    for *batch* $b \in \beta$ **do**
      $w \longleftarrow w - \gamma \nabla l(w; b)$;
    end
  end
  return w to local server
End Function

the global model to all registered edge devices k.

Step 4: Edge device replaces the local model and performs further local training by using the global deployed model.

C. Regional Architecture

In order to remove the central node and move the aggregation functions to the regional nodes, we further modified the algorithm used in hierarchical architecture. In each training epochs, regional nodes are only responsible for aggregating models for their registered edge devices. After a certain number of training iterations, all regional nodes exchange their model information with each other to form a global knowledge. The algorithm is outlined in Algorithm 3 and the steps can be summarized as follows:

Step 1: Edge devices locally compute the models; After reaching the number of local epochs, they send updated model results w to corresponding regional aggregation nodes.

Step 2: The regional nodes perform aggregation by averaging all updated models to form regional knowledge. In addition, every f iterations, there is an exchanging iteration in which the node applies another aggregation function by adding all updated models and calculate the number of updated models. Then, this information will be spread to all the regional nodes to form a global knowledge of w_{t+1}. (If the exchanging iteration is not reached, regional nodes will only aggregate a regional model and send it to all the edge nodes)

Step 3: After calculating the aggregated result in each regional nodes. Regional nodes will then forward the global model to all registered edge devices k.

Step 4: Edge device replaces the local model and performs further local training by using the global deployed model.

D. Decentralized Architecture

In order to realize decentralized characteristics, we remove the aggregation functions from central points but attach them to the edge. Algorithm 4 illustrates the idea. Each edge nodes has an independent process to train, send and receive model weights. There is also a frequency parameter which can control edge nodes exchange their model to their neighbours after several training epochs. The steps can be concluded as follows:

Step 1: Edge devices locally compute training gradients; After reaching the exchanging iteration, they send updated model results w to their registered neighbours.

Step 2: After receiving all the models from the neighbours, each node performs aggregation by averaging all updated models.

Step 3: Edge device replaces the old model and performs further local training by using the updated model.

V. Evaluation

In this section, we present the experiment results for four different architecture alternatives and compare them with the

Algorithm 3: FedAvg - Regional: The new parameter f defined the frequency of exchanging the models.

Function `Server_Update(`w_t`)`**:**
- initialize w_0 $S \longleftarrow$ (all neighbour servers)
- **for** *each round t = 1, 2, ...* **do**
  - $K \longleftarrow 0$;
  - $m \longleftarrow max(C \times K, 1)$;
  - $S_t \longleftarrow$(random set of m clients);
  - **for** *each client* $k \in S_t$ ***in parallel*** **do**
    - $w_{t+1}^k \longleftarrow Client\_Update(k, w_t)$;
    - K ++;
  - **end**
  - $w_{t+1} \longleftarrow \sum_{k=1}^{K} w_{t+1}^k$;
  - **if** t *mod* f == *0* **then**
    - **for** *each server* $s \in S$ ***in parallel*** **do**
      - send(W_{t+1}, K);
      - $w_{t+1}^s, k^s \longleftarrow Server^s\_send(w_t)$;
      - $K_{t+1} \longleftarrow \sum_{s=1}^{S} k^s$
    - **end**
    - $w_{t+1} \longleftarrow \sum_{s=1}^{S} \frac{1}{K_{t+1}} w_{t+1}^s$;
  - **else**
    - $w_{t+1} \longleftarrow \frac{1}{K} w_{t+1}$;
  - **end**
- **end**

End Function

Function `Client_Update(`*k, w*`)`**:**
- $\beta \longleftarrow$(split P_k into batches of size B);
- **for** *each local epoch i from 1 to E* **do**
  - **for** *batch* $b \in \beta$ **do**
    - $w \longleftarrow w - \gamma \nabla l(w; b)$;
  - **end**
- **end**
- **return** w to regional server

End Function

Algorithm 4: FedAvg - Decentralized

Function `Client_Update(`*k, w*`)`**:**
- $\beta \longleftarrow$(split P_k into batches of size B);
- $C \longleftarrow$ (all neighbour clients)
- **for** *each round t = 1, 2, ...* **do**
  - **for** *batch* $b \in \beta$ **do**
    - $w_{t+1} \longleftarrow w_t - \gamma \nabla l(w_t; b)$;
  - **end**
  - **if** t *mod* f == *0* **then**
    - **for** *each client* $c \in C$ ***in parallel*** **do**
      - send(W_{t+1});
      - $w_{t+1}^c \longleftarrow client^c\_send(w_t)$;
    - **end**
    - $w_{t+1} \longleftarrow \sum_{c=1}^{C} \frac{1}{len(C)} w_{t+1}^c$;
  - **end**
- **end**

End Function

system performance in three aspects (The metrics are defined in section III-D) - (1) Weights update latency: time used to transmit model from edge to the aggregation nodes, (2) Model evolution time: time used to train and deploy a new global model, (3) Local and Global model classification accuracy: classification accuracy tested on the local and global test set.

To have a clear comparison, the MNIST data set was used to measure all three metrics while CIFAR-10 data set was used to further validate the result of local and global classification accuracy. During the experiments, we conduct the simulation with the different number of edge nodes which varies from 10 to 1,000 and all the nodes participate training procedure.

A. Weights update latency

Figure 6 present the result of weights updating latency, which illustrates a linear increasing trend of weights latency based on the number of connected nodes and more detailed values are listed in Table II.

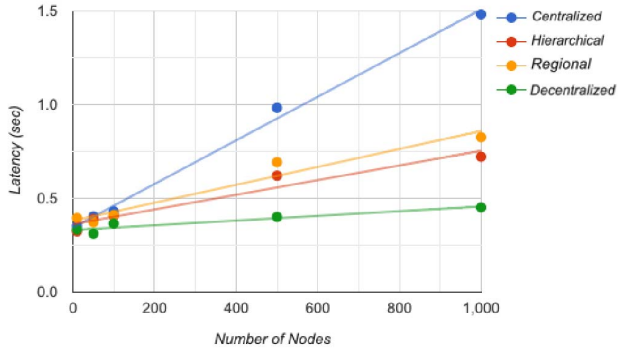


Fig. 6. Weights latency with different number of nodes in four architecture alternatives

The above figure shows that centralized architecture has the largest weights update latency when the number of nodes is bigger than 500. In a centralized architecture, a single central node needs to handle all receiving, training and sending tasks which may highly influence system performance. It can easily lead to communication bottleneck and single-point failure.

Relatively, in the hierarchical and regional alternatives, after introducing regional nodes, the load on the central node is balanced by multiple regional nodes. The red and orange lines show a linearly increasing trend but slower than the centralized architecture.

Furthermore, in the decentralized architecture, since each node can establish equal connections, the server work is further distributed to the edge. Since nodes can only communicate with their neighbours, every node can balance the weights updating traffic, which leads to the smallest growth rate among four architecture alternatives.

In addition to weights updating latency, the number of retransmission is also measured. From Table III, it can be observed that, when dealing with a large number of edge devices, centralized architecture causes more transmission mistake and less communication efficiency than other alternatives. It also proves our findings in weights updating latency.

TABLE II
WEIGHTS UPDATING LATENCY

	Latency (sec)			
Number of Nodes	Central	Hierarchical	Regional	Decentral
10	0.353	0.324	0.395	0.334
50	0.404	0.389	0.373	0.312
100	0.431	0.406	0.411	0.366
500	0.983	0.621	0.693	0.401
1000	1.482	0.722	0.826	0.452

TABLE III
AVERAGE NUMBER OF MODEL RETRANSMISSION DURING ONE TRAINING ITERATION

Number of Nodes	Central	Hierarchical	Regional	Decentral
10	-	-	-	-
50	-	-	-	-
100	-	-	-	-
500	6	-	-	-
1000	147	17	21	-

B. Model Evolution Time

We then calculated the average model evolution time in all edge nodes, which is outlined in Table IV. In our experiments, the model evolution time is influenced by model training time and the weights update latency. With the increasing number of nodes, the training time in each training epoch largely decreases, due to the distribution of training data in each edge nodes and the model training task is separated in numerous workers. However, with the increasing number of nodes, latency may increase as well. In our results, the best number of nodes in the previous three alternatives is 500 while evolution time further increases with the growth of the number of edge nodes.

TABLE IV
AVERAGE MODEL EVOLUTION TIME

	Model evolve (sec)			
Number of Nodes	Central	Hierarchical	Regional	Decentral
10	44.218	45.036	46.020	45.017
50	10.052	10.910	12.741	10.311
100	4.839	4.657	4.166	4.327
500	2.584	2.183	2.031	2.049
1000	3.602	2.990	3.016	1.553

C. Classification Accuracy

In this section, we present model classification accuracy under two different training sample distributions. Here we only present the result measured with 100 edge nodes as the number of edge nodes doesn't have too much obvious influence on classification accuracy.

1) Uniform Distribution: As described in section III-B, the number of classes in each edge device with this distribution are equally likely. Under this setting, the global model classification accuracy (with global test set) can reach 98% in MNIST data set and 88% in the CIFAR-10 data set. The results are outlined in Table V for MNIST and Table VI for CIFAR-10.

TABLE V
GLOBAL PREDICTION PERFORMANCE WITH MNIST DATA SET FOLLOWS A UNIFORM DATA DISTRIBUTION ON THE EDGE

	MNIST Global			
Number of Epochs	Central	Hierarchical	Regional	Decentral
10	96.63	96.42	96.01	94.91
30	98.10	97.80	97.66	96.87
50	98.55	98.47	98.39	97.08

TABLE VI
GLOBAL PREDICTION PERFORMANCE WITH CIFAR-10 DATA SET FOLLOWS A UNIFORM DATA DISTRIBUTION ON THE EDGE

	CIFAR-10 Global			
Number of Epochs	Central	Hierarchical	Regional	Decentral
10	78.75	78.42	77.33	75.89
30	83.21	81.94	81.24	80.30
50	87.92	88.01	87.37	86.45

However, we see a slight difference in four alternatives where the regional and decentralized architecture has 1% worse accuracy, which we explain that in a more decentral architecture, a model may cost more time to form the global knowledge due to their algorithm. (Especially in the decentralized architecture, the model needs more training rounds to spread and aggregate.) This feature becomes more obvious while the model is trained on the data which is distributed and follows a normal density function.

2) Normal Distribution: Normal sample distribution is closer to a real-world data set, however, the accuracy of image classification results is worse than the model which is trained on the data set with a uniform distribution. We observe 1% lower accuracy with MNIST global test set under the centralized architecture alternative. Furthermore, in the decentralized architecture, a model needs more time to converge and form a global knowledge on the whole training data set. The results are presented in Table VII.

TABLE VII
GLOBAL PREDICTION PERFORMANCE WITH MNIST DATA SET FOLLOWS A NORMAL DATA DISTRIBUTION ON THE EDGE

	MNIST Global			
Number of Epochs	Central	Hierarchical	Regional	Decentral
10	89.05	88.95	68.72	33.69
30	95.96	94.16	86.22	45.93
50	97.12	96.31	93.70	81.39

However, when it comes to model performance on the local test set, we find that the decentralized architecture outperforms the rest of the architectures. Compared to architectures with the central aggregation server, the decentralized architecture focuses more on a local data set which results in a slower process of forming the global model but achieves higher accuracy on local set classification. The results are outlined in Table VIII.

TABLE VIII
LOCAL PREDICTION PERFORMANCE WITH MNIST DATA SET FOLLOWS A NORMAL DATA DISTRIBUTION ON THE EDGE

	MNIST Local			
Number of Epochs	Central	Hierarchical	Regional	Decentral
10	89.51	89.42	92.70	95.84
30	95.48	95.05	93.51	96.91
50	97.07	96.00	95.29	98.02

In order to further validate our findings, CIFAR-10 data set was also used to conduct image classification under predefined architecture alternatives. The results (Table IX and Table X) also shows that a centralized architecture have quicker global model convergence while a decentralized architecture is better to perform classification on local edge data sets.

TABLE IX
GLOBAL PREDICTION PERFORMANCE WITH CIFAR-10 DATA SET FOLLOWS A NORMAL DATA DISTRIBUTION ON THE EDGE

	CIFAR-10 Global			
Number of Epochs	Central	Hierarchical	Regional	Decentral
10	72.14	71.02	63.44	25.51
30	78.74	76.93	71.24	44.34
50	86.82	86.03	80.68	70.65

TABLE X
LOCAL PREDICTION PERFORMANCE WITH CIFAR-10 DATA SET FOLLOWS A NORMAL DATA DISTRIBUTION ON THE EDGE

	CIFAR-10 Local			
Number of Epochs	Central	Hierarchical	Regional	Decentral
10	73.01	71.83	75.22	79.31
30	77.68	75.35	79.56	83.67
50	86.07	86.23	87.95	88.24

VI. DISCUSSION

According to experiment results, each architectural alternative demonstrates its advantages and disadvantages. In order to help industries easily understand the requirements and suitable scenarios for setting up a Federated Learning system, we summarize our findings and suggestions in the following sections.

A. Centralized

A centralized architecture is suitable for a small scale Federated Learning system. Since there is only a single central point which manages all the participating nodes and provides model aggregation service. Thus, there is a high probability to cause the communication bottleneck if further increase the number of edge nodes.

In other words, companies that would like to speed up training speed and benefit from parallel training but only have small budgets should consider applying this alternative. They

TABLE XI
COMPARISON BETWEEN DIFFERENT ARCHITECTURE ALTERNATIVES

	Centralized	Hierarchical	Regional	Decentralized
Number of Edge Nodes	Small scale	Small-Medium scale	Medium-Large scale	Large scale
Model variation	Identical	Identical	Localized	Localized
Weights update latency	High	Medium	Medium	Low
Model evolution	Slow	Slow	Fast	Fast
Example Domain	Medical Applications, Human Activity Recognition	Mobile Applications, Wireless Systems	Weather Prediction, Geographic Applications, Vehicle and Traffic Application	IoT

can also benefit from the advantages of easy configurations and nodes management with a centralized architecture compared to other options.

As for the model performance, use cases which require centralized knowledge of all distributed data samples should choose a more centralized architecture. For example, in a medical system, human activity recognition, etc [21] [22], the number of participated edge nodes is usually small and those cases all need a common knowledge for the target prediction whose input training sample has similar distribution in different edge devices.

B. Hierarchical

A hierarchical architecture is an improved option compared with the centralized alternative. It is more suitable for a medium or relatively large scale system. The traffic of model updating is balanced because of the introduction of the regional nodes. This architecture is more suitable for companies which need their system to be scalable and able to tolerant node failure. For example, in mobile applications and wireless systems [3], due to numerous connected devices, management and traffic balance point have to be introduced. Hierarchical architecture is the optimal choice to realize serviceable Federated Learning system. However, due to those extra servers, the system requires more budget and needs more resource for system setting and management.

C. Regional

Different from the previous two options, regional architecture removes the central aggregation node and replace it with several regional nodes. Similar to the hierarchical architecture, it supports a medium or relatively large scale system and needs a medium budget due to more server deployed.

Nevertheless, since the system removes the central point, the aggregated model could gain more knowledge from the local side, especially for those nodes whose data samples may have a similar distribution with their neighbours. This feature is most suitable for use cases such as weather prediction, geographic location detection, vehicle and traffic applications, etc [23] [11]. Furthermore, the system can perform a faster model evolution based on local data but still can partially benefit from global knowledge.

D. Decentralized

For a decentralized architecture, the aggregation functions are moved to the edge devices. This option is suitable for a large and scalable system. Systems such as IoT and network constraint system [12] [24] which don't want to waste resources on transmitting large amounts of data ought to consider this alternative. Furthermore, if a system which needs quickly model evolution and more knowledge from local samples (Sensors, etc.), it should choose the decentralized architecture.

However, a decentralized architecture requires a large budget to realize, as all edge devices need to support the local training and model transmission functions.

VII. CONCLUSION AND FUTURE WORK

In this paper, we introduce and compare four architecture alternatives for a Federated Learning system. We analyze the system performance with three important metrics, i.e. weights update latency, model evolution time, classification accuracy. For the model classification accuracy, a centralized system can formalize the global knowledge which covers all participated data samples while a decentralized alternative focuses more on local data sets in edge devices. Additionally, the weights update latency and model evolution time are much shorter in decentralized architectures than in centralized alternatives. Table XI illustrates some of the insights we gained from the study we conducted in this paper.

Future work will include algorithm improvement on the architectures, such as traffic control, peer finding mechanism and neighbour selection methods, etc. Furthermore, additional efforts in studying hardware cost in those four architecture alternatives will take into consideration. Finally, we aim to realize real-world systems based on architecture alternatives reported in this paper.

ACKNOWLEDGMENTS

This work was funded by the Chalmers AI Research Center. The simulations were performed on resources at Chalmers Centre for Computational Science and Engineering (C3SE) provided by the Swedish National Infrastructure for Computing (SNIC).

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[2] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[3] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," *arXiv preprint arXiv:1906.04329*, 2019.

[4] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[5] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[6] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[7] J. Bosch, I. Crnkovic, and H. H. Olsson, "Engineering ai systems: A research agenda," 2020.

[8] A. L'heureux, K. Grolinger, H. F. Elyamany, and M. A. Capretz, "Machine learning with big data: Challenges and approaches," *IEEE Access*, vol. 5, pp. 7776–7797, 2017.

[9] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic, "A taxonomy of software engineering challenges for machine learning systems: An empirical investigation," in *International Conference on Agile Software Development*. Springer, Cham, 2019, pp. 227–243.

[10] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[11] B. Hu, Y. Gao, L. Liu, and H. Ma, "Federated region-learning: An edge computing based framework for urban environment sensing," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–7.

[12] Y. Zhao, J. Zhao, L. Jiang, R. Tan, and D. Niyato, "Mobile edge computing, blockchain and reputation-based crowdsourcing iot federated learning: A secure, decentralized and privacy-preserving system," *arXiv preprint arXiv:1906.10893*, 2019.

[13] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *International journal of medical informatics*, vol. 112, pp. 59–67, 2018.

[14] S. Lu, Y. Yao, and W. Shi, "Collaborative learning on the edges: A case study on connected vehicles," in *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[15] I. Hegedűs, G. Danner, and M. Jelasity, "Gossip learning as a decentralized alternative to federated learning," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2019, pp. 74–90.

[16] D. Zhang and J. J. Tsai, "Machine learning and software engineering," *Software Quality Journal*, vol. 11, no. 2, pp. 87–119, 2003.

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[18] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of torch," in *Proceedings of the 18th ACM international conference on Multimedia*, 2010, pp. 1485–1488.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[21] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[22] K. Sozinov, V. Vlassov, and S. Girdzijauskas, "Human activity recognition using federated learning," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. IEEE, 2018, pp. 1103–1111.

[23] P. J. Navarro, C. Fernandez, R. Borraz, and D. Alonso, "A machine learning approach to pedestrian detection for autonomous vehicles using high-definition 3d range data," *Sensors*, vol. 17, no. 1, p. 18, 2017.

[24] N. H. Tran, W. Bao, A. Zomaya, N. M. NH, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1387–1395.