future internet

MDPI

Article

Micro-FL: A Fault-Tolerant Scalable Microservice-Based Platform for Federated Learning

Mikael Sabuhi, Petr Musilek * and Cor-Paul Bezemer

Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada; sabuhi@ualberta.ca (M.S.); bezemer@ualberta.ca (C.-P.B.)
* Correspondence: pmusilek@ualberta.ca

Abstract: As the number of machine learning applications increases, growing concerns about data privacy expose the limitations of traditional cloud-based machine learning methods that rely on centralized data collection and processing. Federated learning emerges as a promising alternative, offering a novel approach to training machine learning models that safeguards data privacy. Federated learning facilitates collaborative model training across various entities. In this approach, each user trains models locally and shares only the local model parameters with a central server, which then generates a global model based on these individual updates. This approach ensures data privacy since the training data itself is never directly shared with a central entity. However, existing federated machine learning frameworks are not without challenges. In terms of server design, these frameworks exhibit limited scalability with an increasing number of clients and are highly vulnerable to system faults, particularly as the central server becomes a single point of failure. This paper introduces Micro-FL, a federated learning framework that uses a microservices architecture to implement the federated learning system. It demonstrates that the framework is fault-tolerant and scalable, showing its ability to handle an increasing number of clients. A comprehensive performance evaluation confirms that Micro-FL proficiently handles component faults, enabling a smooth and uninterrupted operation.

Keywords: federated learning; microservices; fault tolerant system design

Citation: Sabuhi, M.; Musilek, P.; Bezemer, C.P. Micro-FL: A Fault-Tolerant Scalable Microservice-Based Platform for Federated Learning. *Future Internet* **2024**, *1*, 0. https://doi.org/

Received: 15 January 2024
Revised: 14 February 2024
Accepted: 19 February 2024
Published:

1. Introduction

Over the last decade, the rapid progression of machine learning technologies has propelled a wave of artificial intelligence applications, encompassing fields such as computer vision, anomaly detection, fault diagnosis, and natural language processing, among others. The rise of machine learning can be largely attributed to two key factors: the accessibility of vast volumes of data and significant advancements in computational techniques and resources.

Nevertheless, the availability of extensive data, metaphorically a "double-edged sword" [1], poses significant risks of personal information leakage when customer, industrial or public data are not properly managed and used. As an illustration, stringent regulations such as the European Union's General Data Protection Regulation (GDPR) [2] and the United States' California Consumer Privacy Act (CCPA) [3] have been implemented to enhance the protection of personal data and privacy by regulating corporate behaviour [4].

With the increasing focus on data privacy, ownership, and confidentiality in contemporary society, there is a growing apprehension that personal information could be exploited for commercial or political purposes without the individual's consent. This concern has catalyzed the emergence of a novel era in machine learning, characterized by approaches specifically designed to safeguard user data privacy. An example of such an approach is Federated Learning (FL), a technique introduced by McMahan et al. [5].

Federated learning serves as a privacy-focused alternative to machine learning approaches that require central data collection, allowing models to be trained directly at the data storage site of each user. This approach eliminates the need for data transmission, as only the locally trained model parameters are used to develop and refine a more effective global model.

Federated learning systems, depending on the communication scheme between components, can be implemented in a centralized (client-server) or decentralized (peer-to-peer) fashion [6,7]. In a centralized scheme, the central server primarily orchestrates the training process and sets up the communication infrastructure among users. However, its pivotal role also introduces a potential vulnerability, rendering it a single point of failure within the system. Conversely, in a decentralized scheme, all clients can autonomously coordinate to acquire the global model, facilitating model updates and aggregations via peer-to-peer client interactions.

Although decentralized federated learning methods, such as those based on blockchain [8–13], can mitigate the challenges of centralized federated learning by eliminating the central server, they introduce their own challenges [14,15]. These include performance degradation as well as increased computational and storage costs. Consequently, this study will focus on federated learning systems that employ a centralized communication scheme and tackle its specific challenges.

In a centralized federated learning system, a central server might become a vulnerability, acting as a single point of failure due to physical damage, server node failure, or network disruptions. This can potentially interrupt the federated learning process. Although large organizations may handle such server roles in some scenarios, collaborative learning often faces constraints regarding the availability and reliability of a robust central server [16]. The server may also become a bottleneck when serving numerous clients, as highlighted by Lian et al. [17].

Hence, when conceptualizing a federated learning system based on a centralized design, it is essential to adopt a design pattern that is both fault-tolerant and performant. Although there are numerous platforms and frameworks for federated learning, challenges related to performance, scalability, and fault tolerance remain. While these platforms often emphasize user scalability and fault management at the user-end, they frequently neglect the vital aspect of server-side fault management and system scalability as the user base expands. Ideally, a federated learning system should inherently possess scalability and fault tolerance.

Scalability pertains to the capacity of the system to include additional devices in the federated learning process. More devices can improve the accuracy and speed up the convergence of the federated learning process [18,19]. Techniques such as resource optimization, prioritizing devices with high computational power, and implementing compression schemes for learning model parameter transfer, can aid in scalability. Effective resource optimization allows more devices to participate in the federated learning process, thereby enhancing performance. However, increasing device participation requires an expansion of server-side computational resources.

Fault-tolerance in the context of federated learning indicates the system's capability to manage the federated learning process effectively, even when the server fails. Traditional federated learning, which relies on a centralized cloud server for global aggregation, can be disrupted if the aggregation server malfunctions [20]. Current concerns about fault tolerance in federated learning systems often revolve around adversarial or Byzantine attacks targeting the central server [21–24] and data related faults such as handling missing data [25]. While numerous studies have investigated the system performance, research into the effects of server faults, such as physical damage, on the federated learning system's performance is relatively scant.

To achieve these properties for a performant federated learning system, this paper presents Micro-FL: a microservice-based federated learning platform designed to handle an expanding user base, guarantee high availability, and offer fault tolerance. Suitable for

deployment on-site or in the cloud, it leverages the flexibility, modularity, scalability, and reliability that microservices provide. Micro-FL streamlines the testing, deployment, and maintenance of federated learning algorithms, while allowing dynamic resource allocation based on workload or user count, thus improving resource efficiency.

The principal contributions of this research paper are as follows:

1. Introduction of a Microservice-based Federated Learning (Micro-FL) Platform: This research paper proposes a novel Micro-FL platform that leverages microservices architecture to address the challenges of fault tolerance and scalability of federated learning systems. This approach significantly differs from the traditional centralized federated learning frameworks by decomposing the monolithic architecture of the traditional centralized federated learning frameworks into smaller, manageable microservices. This decomposition enhances scalability, fortifies fault tolerance, and facilitates efficient resource management for the federated learning server.
2. Emphasis on Server-side Fault Management: Unlike other federated learning frameworks that mainly concentrate on user-end scalability and fault management, the current study emphasizes server-side challenges. This paper presents solutions for managing faults in the communication system of the federated learning system, which is crucial for maintaining the integrity of the federated learning process.
3. Scalability and Dynamic Resource Allocation: The Micro-FL platform facilitates dynamic resource allocation, which allows for the more efficient management of computational resources. This feature is crucial for accommodating a growing number of federated learning clients without compromising performance or reliability.

These contributions underscore the innovation and significance of the proposed Micro-FL platform in enhancing the resilience, efficiency, and scalability of federated learning systems.

2. Background

This section describes the fundamental concepts used in this study, specifically focusing on federated learning and microservices.

2.1. Federated Learning

Federated learning, introduced by Google [5], is a distributed machine learning strategy focused on data privacy. It brings together numerous clients, such as edge devices and organizations, to collaboratively train a shared statistical model, known as a global model. The process, facilitated by a central server, occurs across remote client devices without directly sharing data.

Federated learning is characterized by two main features: (1) It involves a multi-party collaboration, with at least two entities, to construct a machine learning model. Each participant holds unique data that contributes to model training. (2) During the training process, each party's data are kept localized and are not transferred elsewhere.

Federated learning can be formulated as follows: Consider a scenario with N clients, denoted as $\{F\}_i^N$, each possessing unique datasets $\{D_i\}_i^N$. In traditional machine learning, these datasets $\{D_i\}_i^N$ would be sent to a central server to train the unified model M_{SUM}. However, this process requires each client F_i to disclose its dataset D_i to the central server, which poses potential data leakage risks.

On the other hand, in federated learning, clients work together to train a model M_{FED} without having to share their respective datasets $\{D_i\}_i^N$. Suppose V_{SUM} and V_{FED} represent the performance metrics (such as accuracy, recall, or F1-score) of the traditional machine learning model M_{SUM} and the federated model M_{FED}, respectively. Assuming that δ is a non-negative real number, the federated learning model M_{FED} is said to experience δ-performance loss if:

$$|V_{\text{SUM}} - V_{\text{FED}}| < \delta. \tag{1}$$

Equation (1) implies that while federated learning builds a machine learning model using decentralized data sources, the model performance on unseen data closely matches that of a model built on centrally collected data [4].

The federated learning training process can be broadly classified into three steps, as defined by Lim et al. [26]. The model trained at each client's end is termed the local model, whereas the model synthesized by the federated learning server is denoted the global model. Figure 1 provides a visual representation of the federated learning architecture and its training process.

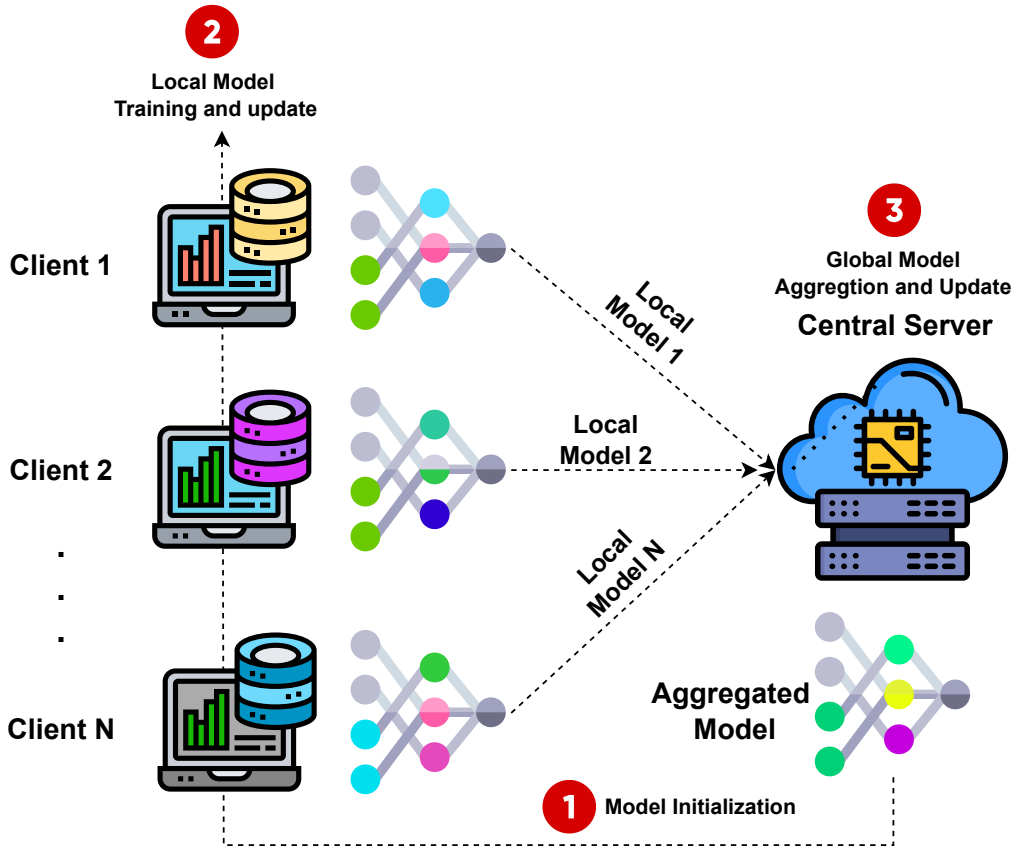


Figure 1. An example of a federated learning architecture: client-server model.

1. Initialization: The server defines the machine learning task, data prerequisites, and training hyperparameters. It then broadcasts the initial global model parameters w_G^0 to selected clients [27–29].
2. Local model training and update: Each client i downloads the broadcast model parameters w_G^t from the server, for the iteration number t. They train the model on their local data to obtain updated model parameters, w_i^t, which minimize a specific loss function, $L(w_i^t)$. These updated model parameters are then sent back to the server.
3. Global model aggregation and update: The server aggregates the local models generated by the clients and prepares the model parameters for the subsequent training iteration, w_G^{t+1}, with the aim of minimizing the global loss function, $L(w_G^t)$.

   Steps 2 and 3 are repeated until the global loss function converges or a targeted training accuracy is achieved.

Federated averaging (FedAvg) is a straightforward and commonly used method for aggregating local models in federated learning, proposed by McMahan et al. [5]. This algorithm averages the updated weights from each client's local model to create a new global model.

2.2. Microservices

Monolithic and microservice-based architectures currently dominate the realm of business application development [30]. Monolithic architecture, a traditional approach, constructs an application as a single extensive codebase or repository that encompasses various services that are not executable independently [31]. This tightly coupled architecture operates as a singular process in the application server's environment during request handling, with all internal communications managed by an intra-process mechanism. However, as new features are continually integrated in today's fast-paced development cycle, the growing codebase and complexity make code understanding and modification more challenging [32,33], leading to slower deployment. Another issue with the monolithic architecture is its lack of fault tolerance. In other words, there is no provision for a system component to function independently when another component fails [34], which is possible with the microservices-based architecture.

In contrast, microservices, a rising trend in software architecture, emphasize the design and development of highly maintainable and scalable software [35]. Thus, by functionally breaking down large systems into a collection of independent smaller systems, microservices help manage the growing complexity of software systems.

Microservices typically employ containerization technologies, such as Docker, which encapsulate each service and run it within a container. This structure allows for effortless scalability with minimal latency and hardware resource footprints. Docker containers, which are lightweight, efficient, and can scale quickly based on needs [36], prove particularly beneficial for a microservices architecture. For this reason, Docker containers have been selected for the federated learning platform proposed in this study.

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications [37]. It offers scalability by dynamically adjusting containers based on resource demands. It ensures high availability through the automatic restart or rescheduling of failed containers, promotes portability, and prevents vendor lock-in through infrastructure abstraction. Furthermore, it optimizes resource utilization by efficiently scheduling containers.

Hence, integrating a federated learning framework that uses a microservices architecture can facilitate the creation of a performant federated learning system that is both scalable and fault-tolerant.

3. Related Work

Addressing the single point of failure and improving fault tolerance in centralized federated learning has garnered substantial research attention. One approach is the implementation of a decentralized federated learning design, which eliminates the central server from the federated learning system, thus averting any single point of failure. This is made possible through the use of different blockchain technologies [8,9,38], such as proof-of-work [11], proof-of-authority [13], and proof-of-contribution [14], in conjunction with smart contracts [10,39]. Such a setup enables model updates and aggregations via direct client-to-client interactions [12], enhancing the overall robustness and fault tolerance of the system.

However, blockchain-integrated federated learning systems face several challenges [14,15], including: (1) Performance issues due to a limited number of transactions, which can result in high latency; (2) the high computational cost of aggregation processes due to typically limited resources on client devices; (3) increased storage demands, as machine learning models must be stored on all client devices, leading to considerable strain on storage resources; and (4) potential data privacy risks as all models are accessible to client devices.

Given these challenges with decentralized federated learning, our research proposes an alternative approach to improve the fault tolerance of centralized federated learning. This approach involves implementing a microservice-based design pattern for federated learning.

The following section examines previous research relevant to the area of microservice-based platforms for federated learning. In particular, it delves into existing tools that use a centralized server design for federated learning and their unique features.

TensorFlow Federated (TFF) [40] is an open-source framework for machine learning on decentralized data. Its interfaces include the high-level Federated Learning API for federated learning training with pre-existing TensorFlow models and the lower-level Federated Core API for developing new federated learning algorithms. While it supports various aggregation functions, it currently does not allow the use of GPU for ML model training and only supports the simulation mode. The framework is still under development, and its current limitations suggest that it might not be suitable for all use cases.

Federated AI Technology Enabler (FATE) [41] is an open-source project by Webank's AI Department, designed to provide a secure computational framework for a federated AI ecosystem. FATE offers a suite of features such as federated statistics, feature engineering capabilities, machine learning algorithm support, and secure protocols. It can be deployed in simulation or federated modes, with installation streamlined via Docker containers. However, its high resource requirements, including 6GB RAM and 100GB disk space on both the client and server side, may render it impractical for real-world federated learning scenarios.

Paddle Federated Learning (PFL) [42] is an open-source platform that supports both horizontally and vertically partitioned data, and can handle neural networks and linear regression models. It leverages techniques like Federated Averaging, Secure Aggregation, and Differentially Private Stochastic Gradient Descent for model construction. Communication in PFL is managed using the ZeroMQ protocol, and it supports both simulation and federated modes, making it adaptable for various deployment scenarios.

PySyft [43] is an open-source project focusing on secure, private deep learning. It comprises components like PyGrid for connecting data owners and data scientists in a peer-to-peer network, KotlinSyft for training PySyft models on Android, SwiftSyft for iOS, and Syft.js for web interfacing. These elements collectively enable the secure, collaborative training of models using PySyft.

The Federated Learning and Differential Privacy (FL&DP) [44] Framework is an open-source framework that uses TensorFlow for deep learning tasks and the SciKit-Learn library for linear models and clustering. It offers various aggregation algorithms and uses adaptive Differential Privacy and randomized response coins to enhance data privacy protection during the learning process.

LEAF [45] is an open-source benchmark tailored for federated learning settings. It provides open-source datasets suitable for federated learning, metrics for evaluating federated learning algorithms, and a repository of standard methods such as minibatch Stochastic Gradient Descent and Federated Averaging. Serving as a valuable resource, LEAF aids in benchmarking and comparing federated learning algorithms.

Flower [46] is an open-source framework that supports large-cohort training on edge devices and compute clusters. It offers aggregation methods like SecAgg [47] and SecAgg+ [48], and supports both simulation and federated modes of operation. Notably, Flower is language and machine learning framework-agnostic, ensuring broad compatibility.

Serverless federated learning (FedLess) [49] is a system designed for federated learning on diverse Function-as-a-Service platforms, supporting major commercial FaaS platforms such as AWS Lambda, Google Cloud Functions, Azure Functions, and IBM Cloud Functions. Implemented in Python3, FedLess provides a command-line tool for orchestrating the training process and supports TensorFlow and Keras for deep learning models. Its default federated learning strategy is the FedAvg algorithm, commonly used for the aggregation of model updates. Other research projects, such as [50], have also adopted a serverless design for federated learning.

FedML [51] is an open-source research library and benchmark platform designed to aid the development of Federated Learning algorithms and provide objective performance

comparisons. It supports on-device training, distributed computing, and single machine simulation. FedML offers resources such as algorithmic implementations, benchmarks with evaluation metrics, access to real-world datasets, and validated baseline results. It is organized into FedML-API for high-level APIs and FedML-core for low-level APIs, using the Message Passing Interface for system communication. FedML supports various federated learning algorithms including FedAvg, Decentralized FL, Vertical Federated Learning, and Split Learning.

Numerous other tools are designed to address scalability issues in federated learning systems, mainly with regard to scaling in the number of clients [52–56]. Despite providing several beneficial features, these platforms fail to implement an efficient central server design that is scalable and fault tolerant. Micro-FL is introduced to mitigate these deficiencies. Based on the principles of microservices system architecture, Micro-FL is designed to enhance the scalability and robustness of centralized federated learning systems, effectively addressing these significant gaps in contemporary solutions.

4. Micro-FL

Figure 2 contrasts the building blocks of a commonly used federated learning server design (monolithic architecture) with the microservices-based design proposed in this study. In a monolithic federated learning server design, all components (e.g., user interface and communication services) are encapsulated into a single process, using a single database. A fault in any of these components can completely halt the federated learning process. This issue, known as a 'single point of failure' within the server, could cause substantial downtime and undermine system reliability. More importantly, scaling monolithic applications requires scaling the whole application, necessitating a considerable increase in resource requirements.

Conversely, when employing a microservices architecture, these components are decoupled from each other (i.e., isolated), each operating as an individual scalable process (microservice) with its own dedicated database. Furthermore, a communication mechanism is implemented to allow these services to interact with each other. As these microservices are horizontally scalable, multiple instances of each microservice can be created to enhance the fault tolerance of the federated learning system. The proposed architectural framework is called Microservice-based Federated Learning (Micro-FL) (https://github.com/asgaardlab/Micro-FL, (accessed on 21 February 2024)).

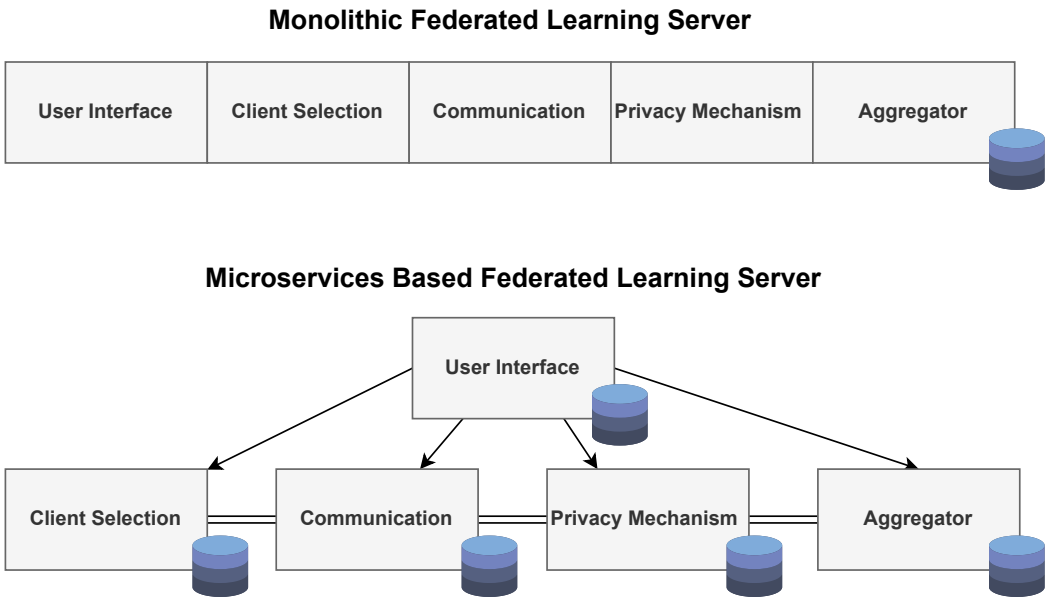


Figure 2. Comparison of monolithic and microservices-based federated learning systems architectures.

Micro-FL accommodates both Linear and Deep Neural Network (DNN) models, leveraging TensorFlow and Keras. Additionally, as a microservice-based application, Micro-FL possesses several notable attributes:

- Micro-FL's distinct modules handle specific functions, contributing to a compact codebase and easier debugging, while also enabling incremental upgrades. These upgrades allow for the coexistence of old and new versions for compatibility testing, and changes in a module do not require a system-wide reset, thus reducing the re-deployment cycle.
- The framework enhances fault tolerance capabilities with Kubernetes; even with a communication microservice failure, the federated learning process continues without interruption.
- Using containerization, Micro-FL allows for an extensive customization of the deployment environment and facilitates the scaling of individual microservices without impacting the whole application. This functionality supports the easy deployment or retraction of services based on demand and accommodates both horizontal and vertical scaling.

4.1. Micro-FL workflow

The following subsection provides an overview of the workflow of the Micro-FL system. The process is characterized by a series of steps that ensure the smooth execution of federated learning tasks. Each of these steps is explained in detail below:

❶ Clients initiate a registration process with Micro-FL via a dedicated web application interface. ❷ Based on the number of clients registered and prepared to contribute to the federated learning process, the Micro-FL administrator notifies registered clients ready to contribute, signaling the start of a new training iteration. ❸ The Aggregator service actively monitors connected clients and their statuses. When a certain number of participating clients is reached, it triggers the initialization of a model, which is subsequently disseminated to all the clients. ❹ Clients continuously listen for updates from the aggregator service. Upon receipt of the model from the aggregator service, they start training on their local datasets. ❺ After training, the clients transmit their model parameters to the server through the communication service. ❻ All messages submitted by clients are transmitted securely through the communication service and are logged into the database. ❼ The aggregator microservice continuously monitors the client messages during each iteration. When the number of messages is equal to the total number of clients, the aggregator synthesizes a new global model using the individual client models. ❽ The Aggregator service dispatches a fresh message to the clients, and the cycle from steps ❸ to ❽ repeats. This iterative process continues and is monitored until a specified number of iterations are completed or a pre-defined model performance metric is achieved.

4.2. Framework Design

A minimalistic implementation of the proposed Micro-FL architectural design is presented in Figure 3. All services operate as Docker containers and are orchestrated using Kubernetes. Additionally, load balancing is used to distribute client requests between the user interface and communication microservices. Building on the previous section, this part discusses the essential components, services, and applications used to actualize Micro-FL. These components are selected based on their performance, scalability with Kubernetes, and open-source nature. Each microservice is briefly explained, as follows:

1. User Interface. The web application is developed using the Flask library for Python and Nginx as the web server. A federated learning client can register and authenticate using this web interface, and Google Cloud Load Balancing is used to balance the workload of the web application. Both the web application and the server are deployed with three replicas to improve their performance and reliability.
2. Database. The database governs the access and caching of the federated learning system. Elasticsearch (ES), a NoSQL database, retains federated Learning models,

accuracy metrics, and client information. The proposed Micro-FL platform employs Elasticsearch due to its scalable and fault-tolerant design, which incorporates index replication and sharding. ES uses REST APIs for data storage and search, with a document-based structure in place of tables and schemas. Additionally, Kibana visualizes the federated learning process.

3. Communication. This microservice enables data exchange across various applications, services, and systems, which is essential for effective communication between microservices and clients. Apache Kafka is used as a message broker, renowned for its scalability, fault tolerance, and capacity to handle trillions of daily messages with minimal latency. Within Kafka, partitions serve as the foundational units for parallelism and scalability. These partitions segment a Kafka topic into multiple smaller, immutable, ordered sequences of records, each hosted on a distinct Kafka broker within a Kafka cluster. Multiple partitions in a Kafka topic enable parallel processing and enhance scalability. Producers can write to various partitions simultaneously, and consumers can consume data from numerous partitions concurrently. This design promotes high throughput and fault tolerance. Kafka employs replication to ensure fault tolerance and high availability. Each partition is replicated across multiple brokers for redundancy. The replication factor determines how many copies of each partition are maintained in the cluster. Strimzi Kafka [57] is used for Kafka broker deployment on Kubernetes and the Apache Camel Elasticsearch sink connector (Kafka Connect) for message transfer to the Elasticsearch database.
4. Aggregator. Aggregator microservice is responsible for aggregating client updates. It retrieves model updates from the database. After all selected users have reported their local model updates, it creates a new global model for the next federated learning iteration. Although numerous methods and structures can be used for aggregation, the simple and commonly used FedAvg algorithm has been selected for testing. Since the aggregation, a synchronous process, occurs at the end of the federated learning cycle, this microservice is not replicated. In other words, if this microservice fails, the Kubernetes controller manager will automatically restart it, ensuring no impact on the training process.

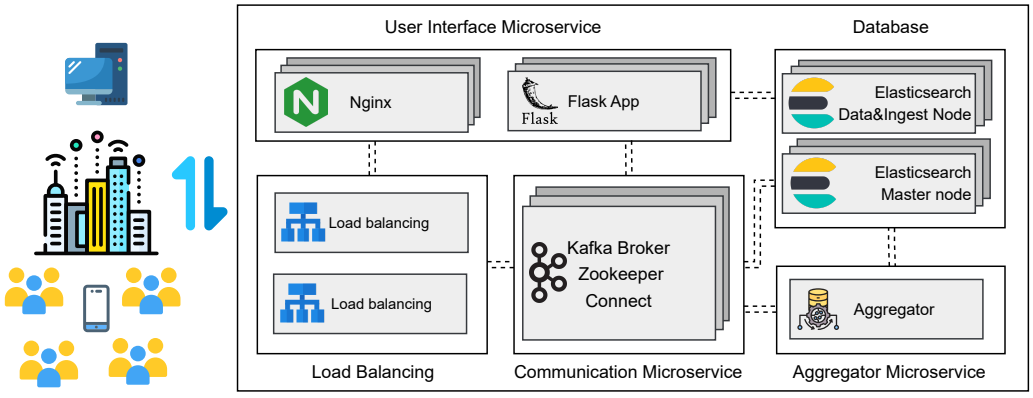


Figure 3. Scalable and Fault-Tolerant Microservice Architecture of the Micro-FL Framework.

5. Methodology

This section describes the methodology for evaluating the fault tolerance capabilities of the proposed Micro-FL platform. The working environment will be provisioned using Google Cloud Platform (GCP) and Google Kubernetes Engine. The various steps that comprise the methodology are illustrated in Figure 4. Subsequent subsections will provide an in-depth explanation of each of these steps.

5.1. Deploying the Micro-FL Framework

Table 1 demonstrates the configuration of the Kubernetes cluster and the specifics of the Micro-FL deployment are detailed in Table 2. The Google Cloud Kubernetes Engine is used to deploy the Micro-FL frameworks, employing three Kubernetes nodes. This setup will be used to evaluate the fault tolerance behaviour of the microservices, maintaining system robustness in the face of potential faults in instances.

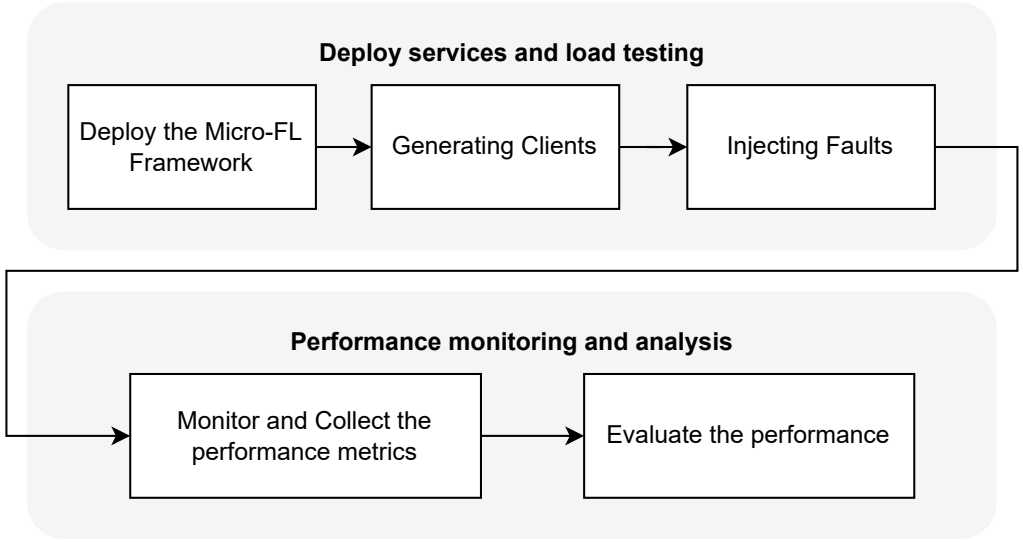


Figure 4. Overview of the methodology to evaluate the fault tolerance of Micro-FL.

Table 1. Configuration of the Micro-FL Kubernetes instance.

Property	Value
Machine Family	E2-Standard-16
Number of Nodes	3
vCPUs	16
RAM	64GB
Image Type	Ubuntu With Containerd
Boot Disk Type	Balanced Persistent Disk
Boot Disk Size	100GB
Zone	us-central-c
GKE Ver.	1.25.8-gke.500

For both Kafka brokers and Kafka Connect, a replication factor of three is implemented across all Kafka topics (which is recommended for the production level [58]). Additionally, the minimum in-sync replicas parameter was configured to two, providing resilience against any single instance failure. Three replicas for the ZooKeeper microservice are also established. In the case of Elasticsearch, indices were configured with a replication factor of 3. Kibana is used for data visualization tasks within the Micro-FL platform.

Table 2. Micro-FL deployment allocated resources.

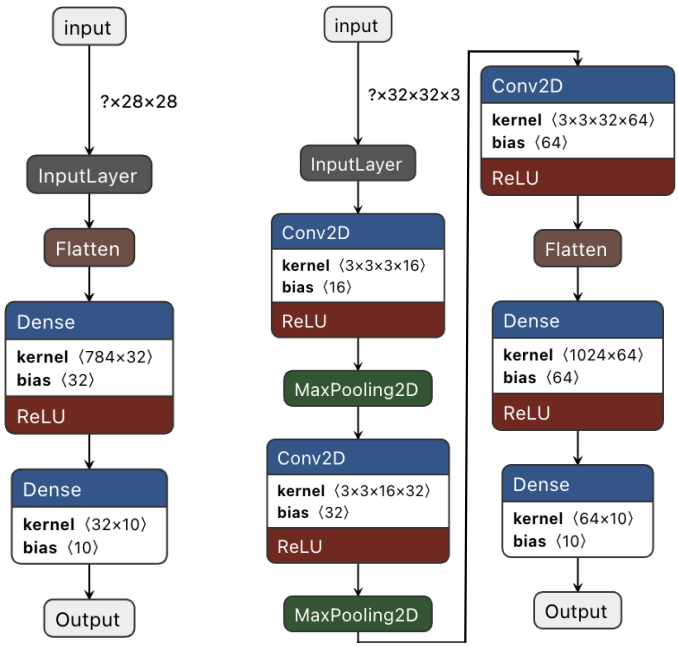
Service Name	vCPU	RAM(GB)	Replica	Ver.
Kafka	2	8	3	3.4.0
Zookeeper	0.5	4	3	3.7.1
Connect	2	8	3	3.4.0
ES-Master Node	1	4	3	8.7.0
ES-Data/Ingest Node	4	16	3	8.7.0
Kibana	1	4	1	8.7.0
Aggregator	1	4	1	N/A

5.2. Generating Clients

Clients are integral to the federated learning process. The performance of the proposed Micro-FL system under varying user counts is evaluated using simulated clients. Given the substantial resources required to run multiple clients concurrently, the simulations are performed using Docker containers and Kubernetes. Google Cloud Kubernetes Engine is used to orchestrate variable client counts. Clients contribute to the federated learning process using two popular datasets, MNIST [59] and CIFAR-10 [60], in two distinct models, as shown in Figure 5a and Figure 5b, respectively. Datasets are randomly selected and evenly distributed to ensure no overlap among clients, as described in Table 3. Client-side training employs five epochs with a batch size of 10. Optimization is performed using Adam optimizer with a learning rate of 0.001 and training process of 100 iterations. For brevity, these datasets and their corresponding models will be referred to as MNIST and CIFAR-10 workloads. Additionally, the computational resources assigned to each workload are outlined in Table 4. It is important to clarify that the primary focus is server-side faults. Therefore, the simulations are run under the assumption that the clients function as expected, without any faults.

Table 3. Distribution of MNIST and CIFAR-10 datasets for different number of users.

Dataset	Clients	Training Samples	Testing Samples
MNIST	100	600	100
	500	120	20
	1000	60	10
CIFAR-10	100	500	100
	500	100	20
	1000	50	10



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

Figure 5. Trained models for performance analysis of the Micro-FL framework. The model trained on MNIST has 25,450 trainable parameters and for CIFAR-10, it has 89,834. (**a**) Model trained on MNIST. (**b**) Model trained on CIFAR-10.

Table 4. Resource allocation for each client for the MNIST and CIFAR-10 datasets.

Dataset	Instance Type	$\frac{\text{vCPU}}{\text{Client}}$	$\frac{\text{RAM}}{\text{Client}}$	Zone
MNIST	General Purpose	0.1	0.6 GB	us-central1 (a,b,c,f)
CIFAR-10	General Purpose	0.25	1.4 GB	us-central1 (a,b,c,f)

5.3. Injecting Faults

Chaos-Mesh [61], a robust chaos engineering platform specifically designed for Kubernetes, is used to evaluate the fault tolerance properties of the Micro-FL system. Chaos-Mesh supports a comprehensive array of fault types, including pod and network faults, while also being safe and manageable. For the experiments, Chaos-Mesh version 2.5.2 is employed.

Kafka broker is considered to be the most essential component of the Micro-FL system, as all communications between services and clients pass through it. Message loss may result from a malfunction in this module, severely impeding the federated learning process. Thus, the fault analysis is restricted to Kafka brokers.

Federated learning is conducted using the MNIST and CIFAR-10 workloads under two distinct conditions, specifically healthy and faulty scenarios. Under the healthy scenario, the system operates without any injected faults. In the faulty scenario, a `POD_FAIL` fault is injected every 20 iterations (in 100 iterations) during the federated learning training process and lasts 5 min. This fault is randomly inserted into one of the Kafka brokers. The purpose of this experiment is to assess whether the Micro-FL system can effectively manage these faults and maintain a seamless operation throughout the process.

5.4. Monitoring and Evaluating the Performance

The performance of Micro-FL is assessed from two perspectives: (1) Federated learning performance, which encompasses the efficiency of the global model and execution time of the experiments, and (2) Software system performance, which includes CPU utilization and metrics pertinent to the message broker.

5.4.1. Federated Learning Performance Metrics

Performance metrics associated with federated learning are gathered and assessed in the Aggregator microservice and are defined as follows:

- Global Model Performance: is measured as the accuracy of training and testing. These metrics indicate the efficacy of the global model in the federated learning process. The aim is to obtain a high performance during training and testing. Given that the datasets address a classification issue, the goal is to achieve high training and testing accuracies. Notably, in this study, the aggregator evaluates these accuracies on the entire training and testing datasets, to which it has complete access.
- Experiment Execution Time: denotes the time it takes to complete each experiment. In assessing the fault tolerance of the proposed architecture, the objective is to maintain consistent execution times for each experiment, regardless of whether the operation conditions are healthy or faulty.

5.4.2. Software Performance Metrics

To collect and evaluate software performance metrics, Prometheus is used as the monitoring system and Grafana for data visualization. The performance of the messaging system (Kafka) is assessed using the Kafka exporter. It assists in gathering metrics like CPU utilization and partition status of the brokers, as well as their throughput. Simultaneously, cAdvisor metrics are used for pods and containers along with their associated performance metrics such as CPU utilization. The following software performance metrics have been collected:

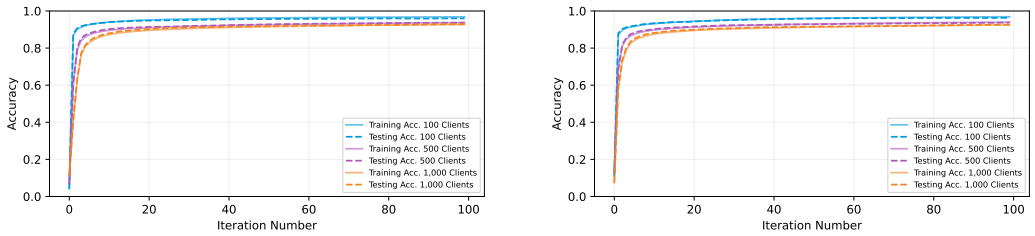
- Online Partitions: are active Kafka partitions, also called leaders, that handle data service. In the leader–follower model, the leader broker manages read/write requests, while others replicate its data for high availability and fault tolerance. If a leader fails or a broker goes offline, Kafka automatically assigns a new leader or marks replicas as 'under replicated', respectively.
- Under Replicated Partitions: In Kafka, they have fewer replicas than the set replication factor. Kafka actively manages this by monitoring the replication status, electing new leaders for under-replicated partitions, and initiating replication processes. Once replication is caught up, the partitions become fully replicated again.
- Partitions at Minimum In-Sync Replicas (ISR): Kafka ensures data integrity by defining a minimum number of replicas (ISR) that must sync with the leader for successful writes. When a message is sent, it is written to the leader and copied to follower replicas. If enough replicas acknowledge the message to meet the ISR, the write is successful. The Kafka cluster has an ISR of 2, ensuring that messages are stored in at least two brokers, providing tolerance against a single broker failure. The replication factor and ISR can be customized to meet the SLA needs.
- Offline Partitions: In Kafka, partitions lacking a leader replica are called off-line partitions. They cannot perform read/write operations if all replicas are unavailable or have failed. This can disrupt data availability, hindering data writing and consumption. The goal is a federated learning platform without offline partitions, ensuring continuous operation and data availability, even during Kafka cluster faults.
- Broker CPU Utilization: reflects the proportion of CPU cores used by each Kafka broker during the federated learning process. The objective is to utilize the allocated resources efficiently, avoiding overuse.
- Broker Throughput: signifies the data transmission rate to or from Kafka brokers. As a critical performance indicator of a Kafka cluster, it demonstrates the speed with which producers can relay messages to brokers. High throughput, a core aspect of Kafka's design, enables it to manage the real-time processing of substantial, rapid data streams.

6. Results

This section discusses the results of the experiments. The federated learning performance of the Micro-FL framework is evaluated first to show whether the platform can carry out federated learning with a good model convergence for the MNIST and CIFAR-10 workloads.

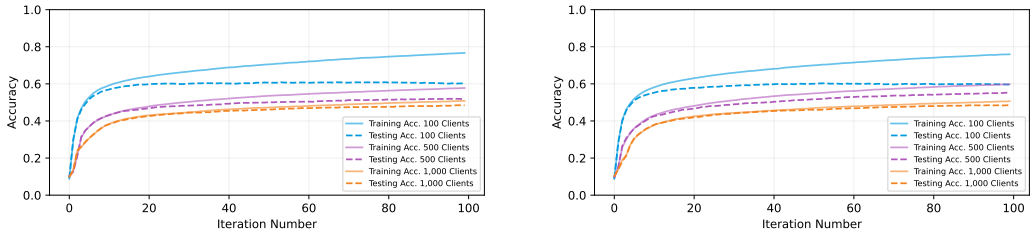
6.1. Federated Learning Performance

The global model of the federated learning system, even under faulty conditions, performs robustly without any significant adverse effects. Figures 6 and 7 show the training and testing accuracy of the global model for each iteration under healthy and faulty conditions for the workloads discussed earlier. It is evident that faults in the communication microservice do not adversely affect the global model's performance, demonstrating its robust machine learning operation. Table 5 details the accuracy achieved for different numbers of users under both healthy and faulty conditions, following 100 iterations of training. These results corroborate the expected performance and uninterrupted convergence of the simple Federated Averaging algorithm, even amidst faults. Minor variations in training and testing accuracies (such as in the CIFAR-10 workload with 500 clients) are expected due to random dataset sampling.



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

Figure 6. Global model's training and testing accuracies on the MNIST dataset across 100 iterations of training. (**a**) MNIST in the healthy operation scenario. (**b**) MNIST in the faulty operation scenario.



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

Figure 7. Global model's training and testing accuracies on the CIFAR-10 dataset across 100 iterations of training. (**a**) CIFAR-10 in the healthy operation scenario. (**b**) CIFAR-10 in the faulty operation scenario.

For an optimal machine learning performance, a robust and fault-tolerant federated learning system is necessary, particularly when handling many clients and large models.

In the MNIST workload, an increase in the number of users from 100 to 1000 induces a minor change in training accuracy (∼2%) and testing accuracy (∼4%). However, this impact is amplified for CIFAR-10, resulting in a substantial ∼25% decrease in training and testing accuracy, which is attributed to larger model parameters. Enhanced model performance can be achieved by extending federated aggregation to more iterations and prolonging training durations [18,19], albeit raising the risk of faults. This observation underscores the need for a robust, persistent federated learning platform. Such a platform should be reliable, fault-tolerant, and capable of integrating an increasing number of users, allowing them to contribute to the model and improve its performance.

The proposed Micro-FL platform maintains consistent execution times, with minor fluctuations even under faults, demonstrating its robustness and fault tolerance. Table 6 provides the execution times for the same workload and healthy and faulty scenarios. From these data, it can be observed that for the MNIST dataset, there is a minimal variation of less than 3.5% in the execution time of the experiment. This variation occurred during the experiments with 100 and 500 users on the MNIST dataset, but these fluctuations can be deemed negligible given the short experiment durations and fluctuations in the cloud infrastructure. For more extensive federated training procedures, such variations become even less pronounced. For example, with the CIFAR-10 dataset and 1000 users, the discrepancy between execution times under healthy and faulty conditions is a mere 0.06%. This observation highlights that the proposed Micro-FL platform and its fault-tolerant design do not allow faults to influence the execution time. Therefore, the federated learning process does not experience delays due to faults.

Table 5. Training and testing accuracies of the aggregator after training the model for 100 iterations.

Dataset	#Clients	Training Accuracy		Testing Accuracy	
		Healthy	Faulty	Healthy	Faulty
MNIST	100	96.86	96.89	96.02	96.35
	500	94.24	93.26	94.07	93.14
	1000	92.76	92.80	92.83	93.12
CIFAR-10	100	76.69	75.94	60.23	59.60
	500	57.78	59.83	51.90	55.24
	1000	50.84	50.69	48.45	48.58

Table 6. Experiment execution time for healthy and faulty scenarios. Positive change indicates shortened execution time and negative change shows extended execution time.

Dataset	#Clients	Experiment Duration (s)		Change
		Healthy	Faulty	
MNIST	100	5230	5070	−3.06
	500	5888	6090	3.43
	1000	10,231	10,144	−0.85
CIFAR-10	100	11,160	11,014	−1.30
	500	18,819	18,757	−0.33
	1000	31,256	31,236	−0.06

6.2. Software Performance Analysis

Micro-FL provides a persistent operation and communication platform for federated learning, even during faults. This demonstrates the robust fault tolerance of the proposed approach. Figure 8 illustrates the partition status during the MNIST experiment with 100 users in the presence of faults. Initially, all partitions are online and accessible. When a fault occurs, the affected Kafka broker's partitions become under-replicated but remain available due to the minimum ISR policy, thus preventing any offline partitions. This confirms the fault tolerance of Micro-FL, which provides a persistent operation and communication platform for the federated learning system, despite the presence of faults. Once the fault is resolved, the Kafka cluster quickly recovers, restoring the number of online partitions and reducing the underreplicated and minimum ISR partitions to zero, thus reaffirming the message availability. This resilient behaviour is consistent across all experiments.

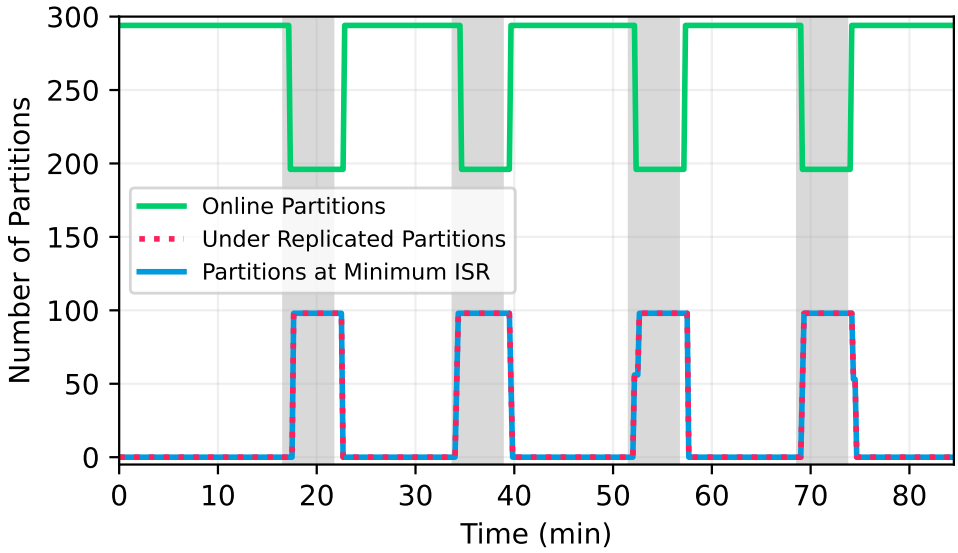


Figure 8. Number of online and under-replicated partitions and partitions at minimum ISR for Kafka broker during the faulty scenario for MNIST with 100 users. The fault period is marked in gray.

The Micro-FL framework effectively maintains constant throughput and CPU utilization under healthy conditions, and adapts to faults by redistributing the load among operational Kafka brokers, thereby demonstrating its fault-tolerant nature. Figure 9 presents the throughput and CPU utilization under the MNIST workload with 100 users across the three Kafka brokers in the cluster in a faulty scenario. In normal operating conditions of Micro-FL, the throughput and CPU utilization of the Kafka brokers remain relatively constant, fluctuating around a specific value. However, according to Figure 9, a distinct change is noticeable during the occurrence of a fault, with darker shades of grey marking the period of time with the faulty Kafka broker. When a fault occurs, there is a decrease in throughput and CPU utilization for the broker in question. Concurrently, a slight increase is observed in the throughput and CPU utilization of the remaining two operational Kafka brokers. This increase is an adaptive response to compensate for the faulty broker, enabling the system to continue processing messages. Once the fault is resolved, the throughput of the faulty broker remains at zero, while the CPU utilization spikes across all Kafka brokers. This is indicative of the faulty Kafka broker retrieving underreplicated partitions from the other brokers.

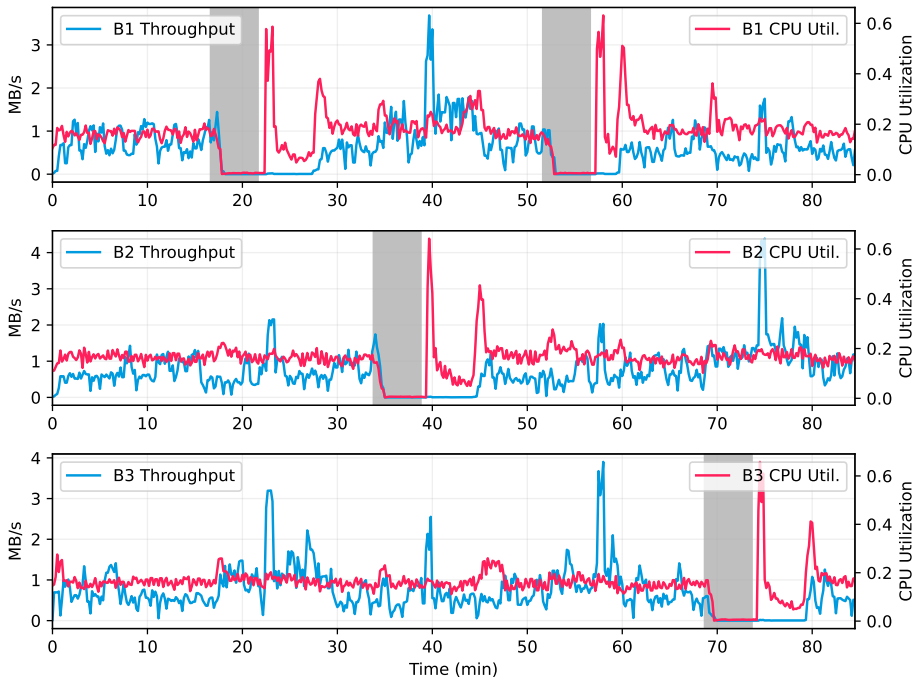


Figure 9. Throughput and CPU utilization for all three Kafka brokers in the cluster during the faulty experiment with the MNIST dataset and 100 users. The darker grey denotes the period when the broker is faulty.

Micro-FL efficiently manages federated learning with minimal resource utilization, allowing a cost-efficient dynamic resource adjustment. As shown in Table 7, with the increasing number of users, the average throughput of brokers increases linearly in both healthy and faulty conditions. The most intensive experiments, involving 1000 users, show a minimal change in the average throughput between healthy and faulty conditions for both MNIST and CIFAR-10 workloads. Despite the increasing workload intensity, the increase in CPU utilization is minimal. Even under the most demanding CIFAR-10 workload with 1000 clients, the overall CPU utilization only reaches 38% and 43% in healthy and faulty scenarios, respectively. These observations confirm that Micro-FL efficiently handles federated learning even with minimal resource allocation, accommodating a dynamic resource adjustment for cost-effective system design.

Table 7. Average throughput (MB/s) and CPU core utilization for different experiments.

Metrics	Scenario	MNIST			CIFAR-10		
		100	**500**	**1000**	**100**	**500**	**1000**
Throughput	Healthy	1.12	5.43	10.28	2.83	17.52	35.59
	Faulty	1.45	6.26	11.06	3.18	17.60	35.76
CPU	Healthy	0.17	0.54	0.74	0.23	0.61	0.76
	Faulty	0.24	0.57	0.85	0.29	0.55	0.87

7. Conclusions

This paper identifies the shortcomings of traditional centralized server designs for federated learning, highlighting the need for improved fault tolerance, scalability, and resource management. It introduces Micro-FL, a microservices-based, fault-tolerant system design uniquely created for centralized federated learning setups. The empirical performance analysis of Micro-FL, conducted across varying user counts and two different workloads, showcased its ability to seamlessly manage faults while ensuring an uninterrupted federated learning process. The proposed design facilitates dynamic resource allocation,

promoting efficient computational resource management, and represents a significant advancement towards more resilient and efficient system designs in federated learning. Possible directions for future research include the optimization of the communications of federated learning systems using message compression and aggregation techniques that leverage message queuing systems like Kafka to decrease the aggregation execution time.

Author Contributions: Conceptualization, M.S., P.M. and C.-P.B.; methodology, M.S., P.M. and C.-P.B.; software, M.S.; validation, M.S.; formal analysis, M.S.; investigation, M.S.; resources, P.M.; data curation, M.S.; writing—original draft preparation, M.S.; writing—review and editing, P.M. and C.-P.B.; visualization, M.S.; supervision, P.M. and C.-P.B.; project administration, P.M.; funding acquisition, P.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Government of Alberta under the Major Innovation Fund, project RCP-19-001-MIF and by the Natural Sciences and Engineering Research Council (NSERC) of Canada, project RGPIN-2017-05866.

Data Availability Statement: The data presented in this study is available on our Github repository (https://github.com/asgaardlab/Micro-FL).

Acknowledgments: The authors would like to thank Google for supporting this research by providing research credits to access the Google Cloud Platform (GCP).

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

References

1. Li, Z.; Sharma, V.; Mohanty, S.P. Preserving data privacy via federated learning: Challenges and solutions. *IEEE Consum. Electron. Mag.* **2020**, *9*, 8–16.
2. Regulation, P. Regulation (EU) 2016/679 of the European Parliament and of the Council. *Regulation* **2016**, *679*, 2016.
3. California Privacy Rights Act: Californians for consumer privacy, 2021. https://www.caprivacy.org/ (accessed on 21 February 2024).
4. Yang, Q.; Liu, Y.; Cheng, Y.; Kang, Y.; Chen, T.; Yu, H. Federated learning. *Synth. Lect. Artif. Intell. Mach. Learn.* **2019**, *13*, 1–207.
5. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-Efficient Learning of Deep Networks from Decentralized Data. *arXiv* **2016**, arXiv:1602.05629.
6. 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.
7. Li, Q.; Wen, Z.; Wu, Z.; Hu, S.; Wang, N.; Li, Y.; Liu, X.; He, B. A survey on federated learning systems: vision, hype and reality for data privacy and protection. *IEEE Trans. Knowl. Data Eng.* **2021**, *35*, 3347–3366.
8. Chang, Y.; Fang, C.; Sun, W. A blockchain-based federated learning method for smart healthcare. *Comput. Intell. Neurosci.* **2021**, *2021*, 4376418,.
9. Wang, R.; Tsai, W.T. Asynchronous federated learning system based on permissioned blockchains. *Sensors* **2022**, *22*, 1672.
10. Wang, Y.; Zhou, J.; Feng, G.; Niu, X.; Qin, S. Blockchain assisted federated learning for enabling network edge intelligence. *IEEE Netw.* **2022**, *37*, 96–102.
11. Kim, H.; Park, J.; Bennis, M.; Kim, S.L. Blockchained on-device federated learning. *IEEE Commun. Lett.* **2019**, *24*, 1279–1283.
12. Chen, Q.; Wang, Z.; Zhou, Y.; Chen, J.; Xiao, D.; Lin, X. CFL: Cluster Federated Learning in Large-Scale Peer-to-Peer Networks. In Proceedings of the Information Security: 25th International Conference, ISC 2022, Bali, Indonesia, 18–22 December 2022; pp. 464–472.
13. Korkmaz, C.; Kocas, H.E.; Uysal, A.; Masry, A.; Ozkasap, O.; Akgun, B. Chain fl: Decentralized federated machine learning via blockchain. In Proceedings of the 2020 Second International Conference on Blockchain Computing and Applications (BCCA), Antalya, Turkey, 2–5 November 2020; pp. 140–146.
14. Tian, Y.; Guo, Z.; Zhang, J.; Al-Ars, Z. DFL: High-Performance Blockchain-Based Federated Learning. *arXiv* **2021**, arXiv:2110.15457.
15. Lo, S.K.; Lu, Q.; Zhu, L.; Paik, H.Y.; Xu, X.; Wang, C. Architectural patterns for the design of federated learning systems. *J. Syst. Softw.* **2022**, *191*, 111357.
16. Vanhaesebrouck, P.; Bellet, A.; Tommasi, M. Decentralized collaborative learning of personalized models over networks. *Artif. Intell. Stat.* **2017**, *54*, 509–517.
17. Lian, X.; Zhang, C.; Zhang, H.; Hsieh, C.J.; Zhang, W.; Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 5336–5346.
18. Stich, S.U. Local SGD converges fast and communicates little. *arXiv* **2018**, arXiv:1805.09767.
19. Gao, W.; Zhao, Z.; Min, G.; Ni, Q.; Jiang, Y. Resource allocation for latency-aware federated learning in industrial internet of things. *IEEE Trans. Ind. Inform.* **2021**, *17*, 8505–8513.

20. Khan, L.U.; Saad, W.; Han, Z.; Hossain, E.; Hong, C.S. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1759–1799.
21. Bhagoji, A.N.; Chakraborty, S.; Mittal, P.; Calo, S. Analyzing federated learning through an adversarial lens. *Int. Conf. Mach. Learn.* **2019**, *97*, 634–643.
22. Tolpegin, V.; Truex, S.; Gursoy, M.E.; Liu, L. Data poisoning attacks against federated learning systems. In Proceedings of the Computer Security—ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, 14–18 September 2020; Proceedings, Part I 25; pp. 480–501.
23. Shejwalkar, V.; Houmansadr, A. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In Proceedings of the NDSS, virtual, 21–25 February 2021.
24. Yang, J.; Zheng, J.; Baker, T.; Tang, S.; Tan, Y.a.; Zhang, Q. Clean-label poisoning attacks on federated learning for IoT. *Expert Syst.* **2023**, *40*, e13161.
25. Oishi, K.; Sei, Y.; Tahara, Y.; Ohsuga, A. Federated Learning Algorithm Handling Missing Attributes. In Proceedings of the 2023 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS), Bali, Indonesia, 28–30 November 2023; pp. 146–151.
26. Lim, W.Y.B.; Luong, N.C.; Hoang, D.T.; Jiao, Y.; Liang, Y.C.; Yang, Q.; Niyato, D.; Miao, C. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2031–2063.
27. Wang, J.; Liu, Q.; Liang, H.; Joshi, G.; Poor, H.V. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 7611–7623.
28. Nishio, T.; Yonetani, R. Client selection for federated learning with heterogeneous resources in mobile edge. In Proceedings of the ICC 2019–2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–7.
29. Cho, Y.J.; Wang, J.; Joshi, G. Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. *arXiv* **2020**, arXiv:2010.01243.
30. Fowler, M. Microservice Premium. Available online: https://martinfowler.com/bliki/MicroservicePremium.html (accessed on 21 February 2024).
31. Bjørndal, N.; Bucchiarone, A.; Mazzara, M.; Dragoni, N.; Dustdar, S.; Kessler, F.B.; Wien, T. Migration from monolith to microservices: Benchmarking a case study. *Tech. Rep.* **2020**. DOI: 10.13140/RG.2.2.27715.14883.
32. Fowler, M. Microservices. Available online: https://martinfowler.com/articles/microservices.html (accessed on 21 February 2024).
33. Richardson, C. Microservices pattern: Monolithic Architecture Pattern. Available online: https://microservices.io/patterns/monolithic.html (accessed on 21 February 2024).
34. Baboi, M.; Iftene, A.; Gîfu, D. Dynamic microservices to create scalable and fault tolerance architecture. *Procedia Comput. Sci.* **2019**, *159*, 1035–1044.
35. Dragoni, N.; Giallorenzo, S.; Lafuente, A.L.; Mazzara, M.; Montesi, F.; Mustafin, R.; Safina, L. Microservices: yesterday, today, and tomorrow. *Present Ulterior Softw. Eng.* **2017**, pp. 195–216.
36. Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *2014*, 2.
37. Kubernetes. Production-Grade Container Orchestration. Available online: https://kubernetes.io (accessed on 21 February 2024).
38. Qi, Y.; Hossain, M.S.; Nie, J.; Li, X. Privacy-preserving blockchain-based federated learning for traffic flow prediction. *Future Gener. Comput. Syst.* **2021**, *117*, 328–337.
39. Wu, X.; Wang, Z.; Zhao, J.; Zhang, Y.; Wu, Y. FedBC: Blockchain-based decentralized federated learning. In Proceedings of the 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 27–29 June 2020; pp. 217–221.
40. Google. Tensorflow Federated. Available online: https://www.tensorflow.org/federated (accessed on 21 February 2024).
41. Fate. An Industrial Grade Federated Learning Framework. Available online: https://fate.fedai.org (accessed on 21 February 2024).
42. Baidu. Baidu PaddlePaddle. Available online: http://research.baidu.com (accessed on 21 February 2024).
43. OpenMined. A World Where Every Good Question Is Answered. Available online: https://www.openmined.org (accessed on 21 February 2024).
44. Sherpa. Privacy-Preserving Artificial Intelligence to Accelerate Your Business. Available online: https://sherpa.ai (accessed on 21 February 2024).
45. Caldas, S.; Duddu, S.M.K.; Wu, P.; Li, T.; Konečnỳ, J.; McMahan, H.B.; Smith, V.; Talwalkar, A. Leaf: A benchmark for federated settings. *arXiv* **2018**, arXiv:1812.01097.
46. Beutel, D.J.; Topal, T.; Mathur, A.; Qiu, X.; Parcollet, T.; de Gusmão, P.P.; Lane, N.D. Flower: A friendly federated learning research framework. *arXiv* **2020**, arXiv:2007.14390.
47. Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H.B.; Patel, S.; Ramage, D.; Segal, A.; Seth, K. Practical secure aggregation for privacy-preserving machine learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1175–1191.
48. Bell, J.H.; Bonawitz, K.A.; Gascón, A.; Lepoint, T.; Raykova, M. Secure single-server aggregation with (poly) logarithmic overhead. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, 9–13 November 2020; pp. 1253–1269.

49. Grafberger, A.; Chadha, M.; Jindal, A.; Gu, J.; Gerndt, M. FedLess: Secure and Scalable Federated Learning Using Serverless Computing. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; pp. 164–173.
50. Jayaram, K.; Muthusamy, V.; Thomas, G.; Verma, A.; Purcell, M. Lambda FL: Serverless Aggregation For Federated Learning. In Proceedings of the International Workshop on Trustable, Verifiable and Auditable Federated Learning, Vancouver, BC, Canada, 1 March 2022; p. 9.
51. He, C.; Li, S.; So, J.; Zeng, X.; Zhang, M.; Wang, H.; Wang, X.; Vepakomma, P.; Singh, A.; Qiu, H.; et al. Fedml: A research library and benchmark for federated machine learning. *arXiv* **2020**, arXiv:2007.13518.
52. Nandi, A.; Xhafa, F.; Kumar, R. A Docker-based federated learning framework design and deployment for multi-modal data stream classification. *Computing* **2023**, *105*, 2195–2229.
53. Kim, J.; Kim, D.; Lee, J. Design and Implementation of Kubernetes enabled Federated Learning Platform. In Proceedings of the 2021 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 20–22 October 2021; pp. 410–412.
54. Zhuang, W.; Gan, X.; Wen, Y.; Zhang, S. Easyfl: A low-code federated learning platform for dummies. *IEEE Internet Things J.* **2022**, *9*, 13740–13754.
55. Quan, P.K.; Kundroo, M.; Kim, T. Experimental Evaluation and Analysis of Federated Learning in Edge Computing Environments. *IEEE Access* **2023**, *11*, 33628–33639.
56. Weber, N.; Holzer, P.; Jacob, T.; Ramentol, E. Fed-DART and FACT: A solution for Federated Learning in a production environment. *arXiv* **2022**, arXiv:2205.11267.
57. Kafka on Kubernetes in a few minutes. Available online: https://strimzi.io (accessed on 21 February 2024).
58. Mellor, P. Optimizing Kafka Broker Configuration. Available online: https://strimzi.io/blog/2021/06/08/broker-tuning/ (accessed on 21 February 2024).
59. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324.
60. Krizhevsky, A.; Hinton, G. Learning multiple layers of features from tiny images. *Technical report*, University of Toronto, **2009**.
61. A Powerful Chaos Engineering Platform for Kubernetes: Chaos Mesh. Available online: https://chaos-mesh.org (accessed on 21 February 2024).