Measuring Overhead Costs of Federated Learning Systems by Eavesdropping

Rainer Meindl[1(✉)] and Bernhard A. Moser[1,2]

[1] Software Competence Center Hagenberg (SCCH), Softwarepark 32a, 4232 Hagenberg, Austria
{rainer.meindl,bernhard.moser}@scch.at
[2] Institute of Signal Processing, Johannes Kepler University of Linz, Linz, Austria
bernhard.moser@jku.at

Abstract. This paper addresses the issue of communication overhead costs of federated learning including transmission bandwidth and synchronisation efforts. These costs typically consist of locally observable costs on executing components, but there are also hidden costs that can only be measured from a system-wide perspective. The goal is to provide insight into these hidden costs, measure them and identify strategies for reducing them. We propose an approach to tackle the hidden costs by establishing a methodology consisting of an eavesdropping concept and an evaluation strategy. This way we obtain a refined analysis of directly observable costs contrasting hidden costs, which is underpinned by experiments based on a 40-client-spanning federated learning system and the FEMNIST dataset.

Keywords: Federated Learning · Optimisation · Privacy-Preserving · Transparency

1 Introduction

Federated learning is a distributed machine learning approach that enables the training of a model on decentralized data. This approach has gained popularity recently due to its ability to handle sensitive data while maintaining privacy and security [3,12]. However, the overhead cost of federated learning is an important consideration, as it involves the transmission of large amounts of data over a network, as well as the time spent synchronizing the clients [6]. In this paper, we explore the cost of federated learning, including the directly observable cost in terms of bytes generated, as well as hidden costs, such as bytes transmitted over a network. Our goal is to provide a comprehensive understanding of the overhead cost of federated learning and to identify potential strategies for reducing this cost. This includes the definition of this overhead cost, how and where we can measure it, the influence of the measurement towards the system and finally, the scalability of this approach to large-scale federated learning systems.

G. Kotsis et al. (Eds.): DEXA 2023 Workshops, CCIS 1872, pp. 33–42, 2023.
https://doi.org/10.1007/978-3-031-39689-2_4

The paper is structured as follows. First, we provide a comprehensive overview of the related work in Sect. 2, highlighting key contributions and limitations of previous research. In Sect. 3 we present the methodology and techniques used in our study. We also discuss the rationale behind our design choices and how they contribute to achieving our research goal. Section 4 provides details about a reference implementation, including used libraries, hardware specifications, software infrastructure, data sets, etc. We discuss the challenges we encountered during our implementation and application of the novel framework in Sect. 5. It is followed by the results in Sect. 6. We conclude with Sect. 7 in which we highlight the potential areas for future research, including possible extensions of the framework.

2 Related Work

Many studies have focused on optimizing the performance of federated learning algorithms to minimize computational and communication costs [6]. One key challenge in federated learning is the cost of implementing the approach in software [11–13]. For example, a study by [7] developed a system called Federated Averaging (FedAvg) that reduced the communication cost of federated learning by using stochastic gradient descent (SGD) to aggregate updates from clients. Similarly, a study by [10] developed a system called MOCHA that used model compression techniques to reduce the computational cost of federated learning.

[8] propose a framework for deep learning at the edge. It aims to optimize deep learning on low-energy edge devices by architecture awareness, considering the target inference platform and introducing security and adaptiveness very early in its design. In another study, [9], this framework has been optimized for a cnn use case. The design of this framework is generic enough to also measure the amount of data transferred between the components, although the authors do not specifically mention it. We aim to provide such a mechanism to allow further evaluation of such frameworks, which should also allow us to further explain power consumption on edge devices.

Despite these efforts to optimize the cost and enhance the security of federated, and other machine learning, potential concerns still need to be addressed. For example, studies by [11,13] demonstrated that an eavesdropper could infer private data by analyzing client updates in a federated learning system. The authors showed that this attack was particularly effective when the clients had limited resources, such as memory. This highlights the effectiveness of an eavesdropper, allowing us to inspect (unencrypted) information during system execution.

These scientific works provide great additions to the field of federated learning, security and privacy-preserving machine learning, but none of them provides hard facts on how to measure security or extra overhead incurred by security measures. We aim to provide such a method, as well as a reference implementation to evaluate such overhead costs in the area of federated learning.

3 Design

In this section, we describe the design of our federated learning system and discuss the communication between clients and servers. We then introduce the design additions necessary to measure and visualize the overhead cost of federated learning step by step.

Our system consists of a central server and multiple clients, each with access to local data. An overview of the fundamental idea can be seen in Fig. 1. The goal is to train a global model using the local data on each client without the need to transfer the data to the server. We adopt the Federated Averaging algorithm proposed in [7] implemented in flwr (Federated Learning frameWoRk) [1].

We aim to detect observable costs, which include the bytes generated by each client and the time spent by each client waiting for the central server, as well as the hidden cost, which includes the time spent outside of each client by external influences, such as the network transmission, or the network topology.

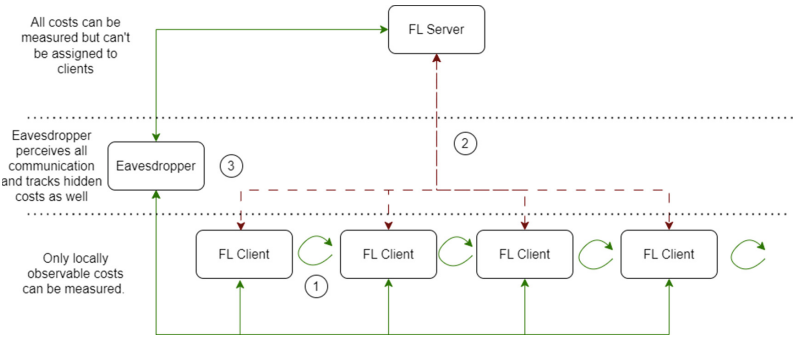

Fig. 1. Design overview of the suggested federated learning system, introducing the eavesdropper. Red marks the generation of hidden costs, green the observable costs. (Color figure online)

Each client initializes their local model, registers to the federated learning server, and starts training the first epochs, instead of synchronizing it beforehand with the federated learning server. They then continue individually training the model using its local data, which already generates bytes and thus directly observable cost, as seen in Fig. 1, item 1. Each client then sends the updated model parameters back to the server [3]. The server aggregates the updated model parameters from all clients using Federated Averaging [7] and broadcasts the updated model to all clients. This process continues for several rounds until the model converges. The difference in cost between item 1 and item 2 is that inside the clients any cost can be measured. Outside of the clients, in item 2, no client can perceive the generated cost.

The clients track the locally incurred overhead cost of the federated learning training themselves. These values include but are not limited to the size of the learning result that is to be sent to the server in bytes, the time spent calculating the size of the learning result bytes, and the time spent waiting for this client to be selected for the next training, i.e. the time spent synchronising. This local reflection cannot be seen as absolutes. For example, the learning result size in bytes in a network is just the payload of a message protocol, which itself creates other overhead. The time spent on calculating the byte size of the learning result again takes time to determine. Thus, the local overhead costs only provide an inaccurate picture of the overhead, which needs to be sharpened by another, external component.

We connect the clients to the server using GRPC [2] since this allows us to keep the components connected throughout the training rounds. Thus, the server can on the fly decide which clients to select for further training rounds and see failed or disconnected clients. But by using such a direct connection we cannot extract any meta-information about the communication, such as the actual number of transmitted bytes, or the time taken for these bytes to be received by one communication partner. To mitigate this problem we introduce the *eavesdropper*, as seen in Fig. 1, item 3.

An *eavesdropper* is a software component that acts like a network proxy. It is part of the communication network and can access the raw information transmitted from and to clients, i.e. it can listen to, or eavesdrop on, the messages transferred between the network clients. The basic premise is similar to the network security concept of a *man in the middle (mitm)* attack [5]. We introduce the *eavesdropper* as a central network component in the network. As the communication between federated learning client and server is based on the clients sending messages to the server and the server just responding, the *eavesdropper* just needs to be aware of how to reach the server and intercept all messages for it. By intercepting these messages we can, in addition to collecting the actual number of bytes being transferred over the network, including any protocol overhead like HTTP-Headers, also pinpoint the time when a message enters and leaves the network.

4 Reference Implementation

In this section, we propose a reference implementation of our federated learning system in a Kubernetes cluster environment. First, we describe the standard federated learning environment used in state-of-the-art applications, then we introduce the additions necessary to measure the overhead cost. Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. Our federated learning system leverages Kubernetes to provide a scalable, fault-tolerant, and easily deployable solution for distributed machine learning.

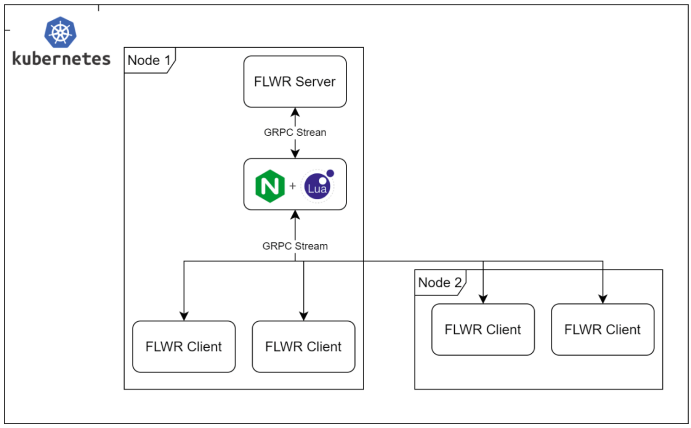

Fig. 2. Detailed overview based on our software environment. Node and pod assignments have been chosen arbitrarily and may differ during test runs.

Our system architecture, as seen in Fig. 2, consists of a central server and multiple nodes, where each node runs Kubernetes pods hosting clients. Note that Fig. 2 does not show the full cluster environment for brevity and clarity. The clients contain a containerized version of our federated learning algorithm and their local training data.

Our federated learning system also utilizes an Nginx proxy as an *eavesdropper* that listens to all communication between clients and the central server pod. The Nginx proxy is extended using Lua scripts for enhanced logging capabilities. It acts as a reverse proxy that sits between the clients and the central server pod. All client requests are routed through the Nginx proxy to the server pod, and all server responses are routed back through the proxy to the clients. By intercepting and analyzing this traffic, we can gain insights into the behaviour of the system and identify potential issues that may arise. For example, we can log the headers and payloads of requests and responses and track metrics such as request and response times. This way, we can gain a deeper understanding of how much overhead is generated in the network as a whole.

This reference implementation of a federated learning system in a Kubernetes cluster provides a scalable, fault-tolerant, and easily deployable solution for distributed machine learning. The use of Kubernetes StatefulSets, Jobs, Services and ConfigMaps enables us to manage the deployment, scaling, and configuration of our system effectively. The combination of the Nginx proxy and Lua scripts as an *eavesdropper*, along with Kubernetes for hosting, provides an efficient and scalable solution for monitoring and analyzing the traffic in our federated learning system. This allows us to not just monitor the overhead cost generated in the training services, but also track the overhead cost incurred by communication over the network.

5 Challenges

Despite the benefits of using federated learning in a Kubernetes cluster, several challenges must be overcome to ensure optimal performance and reliability of the system. In this section, we discuss some key challenges that we have encountered during the development and deployment of our federated learning system, such as the issue of simulation vs. real deployment, limitations of libraries, and optimizing synchronization.

The first challenge is the issue of simulation vs. real deployment. While simulation can be a useful tool for testing and validating a system before deployment, it is important to recognize that the behaviour of a system in a simulated environment may not necessarily reflect its behaviour in a real-world deployment. This is especially true for federated learning systems, which rely on a large number of distributed clients to perform computations and send updates to a central server. As a result, it is important to test the system thoroughly in a real-world deployment environment to ensure that it can perform optimally under real-world conditions. While we do not provide a testing environment on multiple, distributed client devices, we do deploy our reference implementation on a distributed Kubernetes cluster. This cluster itself is a distributed environment, which is closer to the real-world environment than typical federated learning simulations.

The second challenge is the limitations of libraries. While there are several libraries available for implementing federated learning systems, these libraries have certain limitations that can impact the performance of the system. For example, some libraries may not support certain types of machine learning models or may have limited support for the customization of the federated learning algorithm. In addition, some libraries may be less efficient in terms of memory usage or computation time, which can impact the scalability of the system. For *eavesdropping* there are to our knowledge no suitable software libraries, only configurable tools. While this is sufficient for our exploratory study, it could prove to be difficult to integrate the results of the *eavesdropper* from such finished tools. To address these limitations, it may be necessary to develop custom solutions or modify existing libraries to meet the specific requirements of the system.

The third challenge is optimizing synchronization. Federated learning systems rely on the synchronization of client updates to a central server to update the global model. However, the synchronization process can be a bottleneck in the system, particularly when dealing with a large number of clients. To optimize synchronization, it may be necessary to implement strategies such as batching updates, compressing updates, or using more efficient communication protocols.

The final challenge we encountered is the credibility of our measurement. As already mentioned in the previous sections, measuring possible overhead costs can also be seen as overhead, as it takes computation time away from machine learning processes. To address this challenge, we employed various techniques such as measuring the overhead of individual components separately and

comparing the results with those obtained from a control group that did not use an *eavesdropper*. It is important to continually monitor and re-evaluate the measurement process as the system evolves, to ensure that the measurement results remain accurate and reliable.

6 Results

To evaluate the performance of our federated learning system in a Kubernetes cluster, we conducted experiments to measure the bytes generated by training, time spent synchronizing, and actual bytes sent over the network. The experiments were conducted using a testbed consisting of 40 federated learning clients running in Kubernetes pods. We also used a central Kubernetes pod to represent the server in the system. In this testbed, we train the clients using the LEAF FEMNIST dataset [4], 100 server epochs, i.E. the server triggers training and evaluation 100 times on selected clients. We evaluate these clients by the bytes they generate during the training, the time spent waiting on synchronisation and selecting and the actual bytes sent over the network we perceive by employing an *eavesdropper*. Further, we also evaluate the impact of the *eavesdropper* in the federated learning system to create a hypothesis on the scalability of this approach.

In the following subsections, we discuss the results shown in Table 1 in terms of bytes generated and sent over the network and the time spent synchronising the clients, before finally discussing the impact of introducing an *eavesdropper* to the system, which is also shown in Table 1. We sampled ten random clients for evaluation in these tables for brevity.

6.1 Bytes Sent

The size of bytes sent over the network is directly proportional to the data used to train the network. Clients 2 and 4 for example have a more significant divergence in training data sizes. Client 2 uses ~ 3000 samples, while Client 4 uses about twice as many samples, which is reflected in the size of the transmitted data. Surprisingly, the structure of the network, while also relevant for the size of the transmitted data, does not impact the overall size as much as the amount of training data.

Most notable is the divergence in the number of bytes sent and the number of bytes we have detected with an *eavesdropper* in place. As seen in Table 1, the *eavesdropper* makes the communication size visible. Local measurements do not include the overhead of the message protocols and networking procedures, which is the decisive overhead in comparison. We attribute this divergence in size to the selected communication protocol, GRPC fork-join streams.

6.2 Synchronisation Time

The larger the training data, the longer one client needs to train an epoch. In flwr [1], a finished client that already transmitted its results is idling and waiting for a new selection round. We define this time as part of the synchronisation time, hence we can assume the synchronisation time per client to be indirectly proportional to the size of the training data. As seen in Fig. 3, clients with few training samples, such as client 2, finish their training faster, resulting in a longer idle time while they wait for clients with larger training data sets to complete, such as client 5.

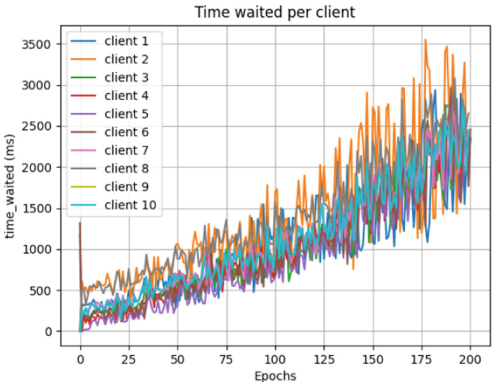

Fig. 3. Time waited per client as per one test run.

Comparing the average time spent synchronization with and without an *eavesdropper* did not result in conclusive results. No major outliers were identified with 40 individual clients, all training individually on different training data sizes. The major discrepancies here can be traced back to random CPU allocations of the Kubernetes cluster.

Table 1. Performance Metrics for Federated Learning Clients, with *eavesdropping*.

Client ID	Total Bytes Sent	Avg. Time Synced	Avg. Bytes Sent	*Eavesdropped Bytes*	*Eavesdropped Avg. Time Synced*
1	3.41 MB	12.70 s	17.14 KB	13.87 MB	11.90 s
2	2.08 MB	13.44 s	10.46 KB	13.24 MB	14.45 s
3	4.16 MB	8.49 s	21.02 KB	13.73 MB	7.53
4	4.16 MB	8.60 s	21.02 KB	14.07 MB	8.32 s
5	4.70 MB	5.72 s	23.77 KB	14.15 MB	7.10 s
6	4.10 MB	7.21 s	20.74 KB	13.78 MB	9.23 s
7	3.52 MB	9.89 s	17.77 KB	13.74 MB	9.76 s
8	2.07 MB	13.51 s	10.50 KB	13.86 MB	13.05 s
9	3.40 MB	9.19 s	17.20 KB	13.87 MB	9.94 s
10	2.07 MB	13.98 s	10.50 KB	13.87 MB	12.80 s

6.3 The Cost of Eavesdropping

The *eavesdropper* itself does need resources we can track, such as CPU and memory usage. We detected some minor peaks on larger loads, but nothing uncommon when comparing it to state-of-the-art reverse proxies used in modern software systems. We did find potentially larger costs for the system while inspecting the memory usage of the *eavesdropper*. The RAM necessary to properly track the federated learning clients without loss or noise is directly proportional to both

the number of clients and the number of epochs they train. So we can formulate the total memory necessary, just for the *eavesdropper* in MB as $TotalMemory = Clients \times Epochs + k$, where k is a constant representing the initial ram usage of the *eavesdropper*. This formula was derived based on the observation of memory usage in a specific set of scenarios involving different numbers of clients and epochs. While this formula may provide a useful estimate for similar scenarios, it may not accurately predict memory usage in all cases. Other factors, such as the size and complexity of the input data and the implementation of the *eavesdropper* may also have an impact on memory usage.

This behaviour can be explained by the communication protocol we use in our reference implementation, as the connection between the federated clients and the server will not close until the federated model has finished training. This could lead to performance issues when scaling the reference implementation to even larger loads, which would independently occur on the federated learning server, even without the *eavesdropper*.

7 Conclusion and Future Work

In conclusion, this paper presents a detailed examination of the cost of federated learning using a reference implementation of federated learning on a Kubernetes cluster and an Nginx proxy with Lua scripts for eavesdropping. The implementation was evaluated using the LEAF FEMNIST dataset and performance metrics such as bytes sent and time spent synchronizing was measured. The results show that the system is effective and scalable, with good performance even with a large number of clients. However, the *eavesdropper*'s RAM usage was found to be a potential drawback, as it increased significantly with the number of clients and epochs in the system.

Overall, the reference implementation presented in this paper provides a useful starting point for those interested in exploring the overhead cost of their federated learning environments. The performance metrics demonstrate the system's effectiveness and scalability, while the identified drawbacks can help guide future improvements to the system. In particular, addressing the issue of RAM usage for *eavesdroppers* will be an important area of future work.

Acknowledgements. S3AI is a COMET Module within the COMET - Competence Centers for Excellent Technologies Programme and funded by BMK, BMAW and the State of Upper Austria. The COMET Programme is managed by FFG. The research reported in this paper has been funded by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK), the Federal Ministry for Labour and Economy (BMAW), and the State of Upper Austria in the frame of the COMET Module Security and Safety for Shared Artificial Intelligence by Deep Model Design (S3AI) [(FFG grant no. 872172) and the SCCH competence center INTEGRATE [(FFG grant no. 892418)] within the COMET - Competence Centers for Excellent Technologies Programme managed by Austrian Research Promotion Agency FFG.

References

1. FLWR: a federated learning framework. https://flwr.dev/ (2023). Accessed 23 Feb 2023
2. gRPC (2023). https://grpc.io/. Accessed 6 June 2023
3. Bonawitz, K., et al.: Towards federated learning at scale: system design. Proceed. Mach. Learn. Syst. **1**, 374–388 (2019)
4. Caldas, S., et al.: LEAF: a benchmark for federated settings (2018). arXiv:1812.01097
5. Callegati, F., Cerroni, W., Ramilli, M.: Man-in-the-middle attack to the https protocol. IEEE Secur. Priv. **7**(1), 78–81 (2009)
6. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
7. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics, pp. 1273–1282. PMLR (2017)
8. Meloni, P., et al.: ALOHA: an architectural-aware framework for deep learning at the edge. In: Martina, M., Fornaciari, W. (eds.) Proceedings of the Workshop on Intelligent Embedded Systems Architectures and Applications, INTESA@ESWEEK 2018, Turin, Italy, 04 October 2018, pp. 19–26. ACM (2018). https://doi.org/10.1145/3285017.3285019
9. Meloni, P., et al.: Optimization and deployment of CNNs at the edge: the ALOHA experience. In: Palumbo, F., Becchi, M., Schulz, M., Sato, K. (eds.) Proceedings of the 16th ACM International Conference on Computing Frontiers, CF 2019, Alghero, Italy, 30 April - 2 May 2019, pp. 326–332. ACM (2019). https://doi.org/10.1145/3310273.3323435
10. Smith, V., Chiang, C.K., Sanjabi, M., Talwalkar, A.S.: Federated multi-task learning. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems, vol. 30. Curran Associates, Inc. (2017). https://proceedings.neurips.cc/paper_files/paper/2017/file/6211080fa89981f66b1a0c9d55c61d0f-Paper.pdf
11. Wang, L., Xu, S., Wang, X., Zhu, Q.: Eavesdrop the composition proportion of training labels in federated learning (2019)
12. Zellinger, W., et al.: Beyond federated learning: on confidentiality-critical machine learning applications in industry. Procedia Comput. Sci. **180**, 734–743 (2021). https://doi.org/10.1016/j.procs.2021.01.296. https://www.sciencedirect.com/science/article/pii/S1877050921003458. proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020)
13. Zou, Y., Wang, G.: Intercept behavior analysis of industrial wireless sensor networks in the presence of eavesdropping attack. IEEE Trans. Industr. Inf. **12**(2), 780–787 (2015)