A Survey on Federated Learning in Distributed Systems

Abie Safdie
asafdie@uoregon.edu

***Abstract*—With the monumental amount of data circulating modern computing systems, efficient ways of conducting Machine Learning are required. Utilizing Federated Learning, which focuses on computing local machine learning models via local data, provides a secure and efficient approach to this problem. Notably, to have Federated Learning models function across multiple nodes, model aggregation algorithms– such as Federated Averaging and Federated Stochastic Gradient Descent– are needed. These algorithms compute a global model that was trained via distributed resources. Further, applying such approach in distributed systems — physically independent systems working together as one logical system — can provide improved efficiency, fault tolerance, and decentralization of data. A distributed system can leverage Federated Learning among its physical nodes to develop a sophisticated model that is representative of all of the data that passes through the distributed system. This is achieved through a leader node aggregating local models into a global model and disseminating it to each physical node. The potential of Federated Learning in distributed systems is exemplified by its performance in privacy and security sensitive domains such as healthcare and computer networking.**

I. Introduction

The growing demand for Machine Learning (ML) models has greatly influenced the current state of computing, leading to a situation where data collection risks outpacing computing power. For example, the volume of data is so vast that certain machine learning workloads, such as speech recognition or computer vision, could take weeks or even months to process on a single node [1]. To address this issue, many workloads currently run in large data centers on parallel platforms [1]. However, a major bottleneck of this approach is data centralization; data must be routed through a centralized data center for an ML model to perform its training, which can considerably slow down the process. Additionally, training ML models on centralized data present data security and data privacy concerns [2]. Therefore, it is essential to develop more efficient, secure methods for training large models. Leveraging federated learning in distributed systems presents a promising solution to this pressing issue.

Distributed systems are defined as multiple systems that coordinate and work in unison to create the illusion of one large, efficient global system. Federated Learning (FL), in contrast, is a decentralized approach to ML where local nodes independently train models on their own data. Each node sends its trained model to a central source, which then uses those locally trained models to form a more complete centralized model. Through an iterative process of sending and receiving updates, both local and global models continuously improve. This approach can lead to notable benefits, including greater efficiency and performance in training models, more robust and accurate models, and enhanced data security and privacy.

Effective implementation of distributed systems leads to improved efficiency and specialization. To achieve this, distributed systems seek to accomplish several main goals: resource sharing, transparency, openness, dependability, security, fault tolerance, and efficiency and scalability [3]. Therefore, a distributed system that leverages federated learning would address each of these goals as follows:

- **Resource Sharing**: Share trained models across multiple system nodes to ensure that each system can adequately train its model to perform desired tasks.
- **Transparency**: Ensure that a system's local model is unaware that the model they use was trained using distributed resources.
- **Openness**: Allow a distributed system to be easily integrated and interfaced with other systems that utilize their own FL procedures.
- **Dependability**: Each system node can depend on the learning that other nodes conduct to inform and impact their future decision making. This creates a reliable network of trust, dependability, and correctness.
- **Security**: Each system node can rely on the sharing of information and models to remain secure.
- **Fault Tolerance**: If one system were to go down and cease to contribute, the distributed system would remain functional.

- **Efficiency and Scalability**: By distributing data across system nodes, the efficiency is improved when training local and global models. In addition, the system is designed to be scalable, allowing more nodes to easily join the network and contribute to the FL process. With the addition of more systems and data, a distributed system's desire to improve in effectiveness as it grows is maintained.

By addressing these goals, a distributed system that utilizes FL can complete its desired tasks in a robust, reliable, and efficient manner.

The high-level architecture of a distributed system that utilizes FL can be viewed as follows:

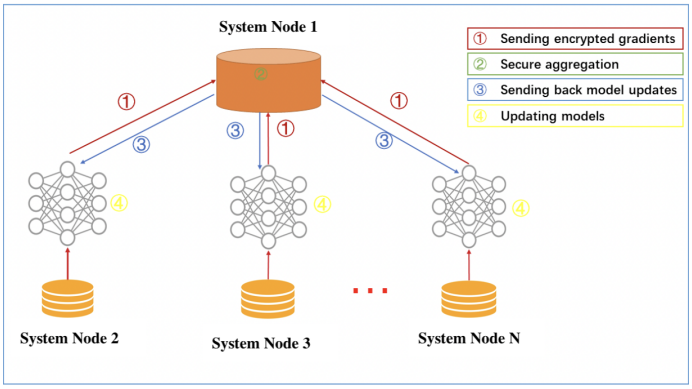

Figure 1: Architecture of a distributed system that uses Horizontal Federated Learning [4]

As seen in Figure 1, a distributed system that uses FL would consist of multiple nodes receiving data updates to train their local models. Subsequently, these nodes would send their models to a leader node which aggregates the local models and forms the central model. The leader node would distribute this model to all nodes in the system. This iterative process continues indefinitely.

FL can be utilized to provide greater efficiency and data privacy [5]. This approach protects sensitive information by sharing models instead of raw data, which is critical for ensuring data privacy and data security. As will be discussed further, FL provides an optimized approach to solve the need for data privacy and developing ML models on scattered data.

II. Background

A. High-Performance Computing

High-Performance Computing (HPC) is at the heart of ML. Systems must be able to perform computationally expensive actions in an optimized manner. Therefore, HPC technologies and libraries have become embedded in ML systems. Most notably through programmable GPUs, which have become extremely commonplace as they offer a substantial amount of parallelization. For instance, the Nvidia Titan V has been measured at roughly 47x faster than a standard server CPU—i.e. an Intel Xeon E5-2690v4—at performing deep learning [6]. In addition, parallel computing frameworks and platforms play a crucial role in the advancement of HPC [1]. Platforms like CUDA and OpenCL have significantly advanced distributed computing, enabling more efficient processing across multiple systems [7]. Similarly, various packages not only serve as frameworks for ML but also provide HPC benefits. Examples include Scikit-learn, PyTorch, and TensorFlow, which are highly versatile, widely adopted throughout the development process, and considered industry standards [1].

For distributed systems to effectively capitalize on the power HPC offers, they must be efficiently scaled-up and scaled-out. That is, each individual system node must be "scaled-up" by leveraging local HPC technologies, but additionally be "scaled-out" by joining a network of HPC systems. This approach introduces fault tolerance (if one HPC system fails, the network can take on the workload) and increased bandwidth input/output (messages and solved computational data can traverse the network at higher rates). Accomplishing scaling-up and scaling-out aid in the completion of the main goals that distributed systems seek to achieve.

B. Machine Learning Algorithms

Machine Learning algorithms seek to solve problems based on making predictions via a given data set. ML algorithms exist within three subcategories: learning, purpose, and method [8]. Together, these algorithms seek to minimize an ML model's loss function, which is a mathematical function that quantifies the difference between a model's output and the actual target value. Simply meaning, it measures how well the model is performing. Therefore, minimizing the loss function is of paramount importance.

1) Learning Algorithms: For an ML model to determine if it is producing the correct results, it needs to receive feedback on its decision-making. This feedback can be provided through various methods, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [8].

- **Supervised Learning**: Supervised Learning works by providing the desired output for a given set of training data. For example, if a group of images depicting the number "42" were given to an ML model, there would be an associated data set telling the model that the desired output should be: "This is the number

42". This framework, however, causes bias and variance errors when solving workloads. Bias refers to error that arises from ML models coming to inaccurate assumptions between the training and the output data. Meanwhile, the variance error reflects the variability in model results when trained on different datasets. Minimizing bias and variance error is critical in developing effective ML models.

Supervised Learning is the most commonly deployed feedback algorithm as it provides superior accuracy [9]. However, it is computationally expensive as it requires large datasets to train, which hinders overall performance and creates the need for HPC technologies.

- **Unsupervised Learning**: Unsupervised learning works by providing input data without the corresponding output data [10]. Returning to the previous example, a dataset containing images of the number "42", the unsupervised learning model would not receive the desired output: "This is the number 42". Therefore, it must find patterns and come to its own conclusions. This can be achieved by utilizing various methods. Some key, popular methods include clustering and dimensionality reduction [10]. By applying these techniques, unsupervised learning models can develop sophisticated interpretations of data.

  Unsupervised Learning is a powerful tool that is most efficient in label-expensive analysis and provides a significant advantage in real-time data analysis [11].
- **Semi-supervised Learning**: Semi-supervised learning works by combining aspects of supervised and unsupervised learning. Typically, in semi-supervised learning, ML models are trained on data sets that contain a small amount of labeled (supervised) data and a large amount of unlabeled (unsupervised) data. This approach has been shown to provide improvements in accuracy without sacrificing too much performance [12].
- **Reinforcement Learning**: Reinforcement learning works by simulating a trial-and-error learning process. Over time, models will learn to make the decision that generates the most favorable or correct output [13]. This learning technique is most commonly used in scenarios where actions have prolonged consequences and feedback is not available right away.

These ML approaches can be seamlessly integrated into a distributed system through FL. For instance, individual system nodes can run distinct learning algorithms on their local datasets, generating local models. By sharing these models, the system enables the nodes to receive a comprehensive global model that has been trained collaboratively using various learning techniques.

2) Purpose Algorithms: Identifying patterns and understanding the nature of specific data are essential in the learning process. As a result, these ML algorithms are frequently used to perform the following tasks:

- **Clustering** seeks to group data that possess similar characteristics. This is a cornerstone of unsupervised learning.
- **Dimensionality Reduction** seeks to reduce the number of characteristics that a data item has so that it is easier to classify.
- **Classification** seeks to label the data into categories based on the training data. This is a cornerstone of supervised learning.
- **Regression** seeks to determine how dependent variables change as a result of changes to other variables.
- **Anomaly detection** seeks to discover outliers among a data set.

These tasks are critical for translating an ML approach to a distributed system that utilizes FL. Each individual system node relies on other nodes to perform accurate data analysis, which ensures the effective development and refinement of their own models. Consequently, performing appropriate data classification across system nodes is necessary.

3) Method Algorithms: The following learning methods are best suited for a distributed approach:

- **Stochastic Gradient Descent (SGD)** algorithms aim to minimize the loss function by adapting a model's parameters in the direction of the negative gradient [8]. In simpler terms, SGD iteratively updates the model's parameters (weights), based on small, random subsets of data, to move closer to the optimal solution. This process continues until the model's predictions align closely with the expected outcome. A key advantage of SGD is its efficiency, as it can quickly learn patterns from data without needing to process the entire dataset. As a result, SGD has become an industry standard. It is widely utilized in various machine learning models, including neural networks. On top of that, SGD is particularly beneficial in federated learning, where individual devices can train models locally on their own data. This approach improves the overall performance of the model while also preserving the privacy of sensitive information, since the data is retained on the local systems rather than shared.

On the other hand, SGD suffers from shortcomings such as noise and stability. Due to SGD's selection of small, random subsets of data, it makes the algorithm highly susceptible to noise. This, in turn, causes SGD to suffer instability. Meaning, it oscillates around the optimal solution as the noise can cause subtle drifts. However, these inefficiencies are widely tolerated as SGD remains an extremely robust and computationally efficient compared to other machine learning and gradient descent algorithms [14].

- **Rule-Based Machine Learning (RBML)** algorithms leverage simple conditional statements to facilitate complex decision-making processes. An ML model trained with RBML uses a dataset to formulate generalized rules. For example, if data shows that many people apply sunscreen when exposed to direct sunlight, the model will establish a rule such as: "If it is sunny, wear sunscreen." This straightforward approach allows for greater transparency, which enables users to analyze the rationale behind the model's decisions. This accountability is a critical aspect of RBML models, as more complex ML models, such as ones fueled by SGD, tend to operate as black boxes. This prevents users and companies from understanding the decision making behind the models that are influencing their operations. Therefore, this clarity behind RBML makes it particularly valuable in scenarios where understanding the basis of decisions is essential. However, this clarity and simplicity comes with multiple weaknesses. These include, the inability to scale efficiently, extrapolate and generalize from datasets, handle ambiguity, and self-learn. Meaning, when large batches of data enter the system, RBML struggles to efficiently process this new data as it relies heavily on predefined rules, which must be manually updated to accommodate new patterns or scenarios. This manual intervention makes the system inefficient and error-prone in dynamic environments where data changes frequently. Moreover, RBML's inability to extrapolate and generalize means it cannot identify complex relationships or trends within the data. On top of that, the lack of self-learning capability means RBML systems cannot improve over time without significant human involvement. These weaknesses prevent RBML from being applied to large, dynamic systems. As a result, RBML is best-suited for generally static datasets that are focused on a specific domain.

  In the context of FL, RBML models are most effective when applied to nodes within the system that exhibit specialized and static datasets. However, during FL's model aggregation phase, discrepancies between conclusions drawn by RBML models can be challenging to resolve due to the static nature of rule-based systems. Nevertheless, the clarity and simplicity of RBML can aid in enhancing data privacy. This is achieved by having sensitive information deterministically and transparently processed at the local level.

C. Distributed Systems

Modern distributed systems operate on an exceptionally large scale [3]. The most notable and common example of this is cloud computing, which serves to store, process, and analyze workloads in a distributed environment [15]. Due to their specialized architectures, cloud computing offers a cost-effective solution for many companies. There are many methods that must be implemented to ensure that cloud computing–and other distributed systems–maintain both effectiveness and correctness.

1) Architectures: Distributed Systems utilize unique architectures to ensure that any individual system can be easily interfaced and integrated into the larger system [3]. A leading approach to achieve this is through middleware, which serves as an intermediary layer. Middleware facilitates communication by transforming otherwise incompatible systems. This allows them to interact and function cohesively. Ultimately, distributes systems utilize three architectural styles: layered, publish-subscribe, and resource-based. A layered architecture involves components ordered in a logical "layer", allowing one layer to communicate to the layer above or below it. A common example of this is the OSI model in networking. Further, a publish-subscribe architecture involves system nodes "subscribing" to certain informational updates. Therefore, when a system node publishes information, the subscribed nodes will receive such information. Lastly, resource-based architectures allow for the transferring of specific resources between systems. A common example of this is Representational State Transfer (RESTful) architectures. RESTful architectures allow for the transferring of data via operations such as PUT, GET, POST, and DELETE. These operations can be seen in use on webpages.

2) Synchronization and Coordination Methods: With multiple systems operating in unison, effective synchronization methods are required. Traditional parallel computing techniques, like mutexes, locks, barriers and semaphores, can help maintain data integrity. However, in the absence of a central operating system to manage lock coordination, a "leader node" must

be selected to coordinate synchronization across the distributed environment. This leader node can subsequently take over the administrative tasks typical of an OS, such as distributing locks.

3) Election Algorithms: To assign a leader node, there are various election algorithms. Some of these algorithms include: election by bullying, ring election, proof of work, and proof of stake [3]. Election by bullying involves assigning an ID to each node and having them exchange ID information until the node with the highest ID—or another defining characteristic that is chosen—is the last one standing. Ring election involves placing nodes into a logical ring, sending reachability information to each logical neighbor until only one leader is chosen, as it is reachable by all the nodes. The proof of work election algorithm forces the nodes to solve a computationally complex task. The node that solves the task is elected as the leader. Proof of stake involves the distribution of a logical token. The node that holds the token is the leader. Ultimately, these algorithms are critical to ensure efficient and consistent coordination among system nodes.

4) Communication: Due to the physical separation of the nodes, effective algorithms are needed to ensure efficient communication. Each node must be provided with the most up-to-date information on the state of the distributed system. For instance, if one node is processing a data set, it must ensure that the data it's working on remains accurate and current.

Epidemic algorithms, such as anti-entropy, rumor spreading, and push-pull methods, are commonly employed to manage these state changes and maintain system consistency [3]. For example, rumor spreading works by having each node send its updates to its logical neighbors, which in turn send the updates they have received to their logical neighbors. This continues until the update has propagated throughout the entire network. Ultimately, these algorithms ensure that updates are effective, which enhances the reliability of the distributed system.

In cloud computing, efficient and accurate server-client communication is critical. To achieve this, Remote Procedure Calls (RPCs) are widely used. RPCs enable servers and clients to communicate and perform remote function execution in compatible and readable data formats. This is a necessity to effectively distribute specialized workloads [3].

5) Naming: To provide accurate and optimized communication between nodes, proper identification methods (naming) must be used. That is, the nodes must be able to identify and access other nodes. There are two general techniques used to do this: flat naming and structured naming. Flat naming consists of a non-hierarchical, random ordering of names. Therefore, to resolve a flat name, broadcasting (sending a message to every node) or following a chain of pointers is usually required. This method does not scale well. On the other hand, structured naming is the use of a naming graph (search tree) to locate names. A notable example of such a structured naming method is the DNS (Domain Name System) protocol in networks. DNS is a large-scale search tree that maps domain names to IP (Internet Protocol) addresses.

Using structured naming techniques such as DNS, distributed systems can scale well and improve the accuracy and efficiency of node identification and communication.

6) Consistency and Replication: With systems working concurrently, maintaining consistency is necessary. Replicating states of the distributed system — known as a data-centric consistency model — is a common practice to achieve consistency. In addition, to provide consistent operations on such stored data, standard synchronization techniques, as discussed earlier, can be applied.

7) Fault Tolerance: Achieving fault tolerance is paramount in a distributed system due to the nature of numerous nodes working in tandem. Therefore, if one node fails, the distributed system must be able to function undisturbed. A common method to produce fault tolerance is the introduction of redundancies. These redundancies usually take on three forms: physical, time, and informational. Physical redundancies refer to adding physical hardware and processes to a system to ensure correctness. This can be as straightforward as deploying extra servers or redundant data storage solutions (this practice relates heavily to consistency and replication, as it deals with data). On the other hand, time redundancies involve re-executing actions when necessary, ensuring that tasks complete successfully despite occasional disruptions. A common example is packet retransmission in networking, where a packet is resent if an acknowledgment (ACK) is not received. Lastly, informational redundancies add data to an action so that if it failed it can be recovered into a functioning state. For instance, checksums are commonly used in data transfers to detect and correct transmission errors. Together, these methods greatly enhance a distributed system's ability to withstand faults.

8) Security: A distributed system is only effective if it remains secure, meaning it must be resilient and resistant against attacks that could disrupt its correct operation. To safeguard the system, four fundamental security mechanisms are commonly implemented: encryption, authentication, authorization, and auditing.

Encryption protects data by converting it into unreadable text, preventing unauthorized parties from interpreting or tampering with it. Widely used encryption algorithms, such as SHA-256 and RSA, ensure data confidentiality and integrity. Authentication verifies the identity of users interacting with the system, typically through methods based on "what you have", "what you know", and "who you are". These correspond to methods such as keycards, passwords, and biometrics, respectively, to confirm that only legitimate users gain access. Authorization determines whether a verified user has permission to perform specific actions, such as accessing particular files or executing commands. User permissions in operating systems are a common example. Lastly, auditing maintains a log of user activities, tracking who accessed or modified what data. This creates a detailed record of system operations, which allows administrators to review and detect potential security breaches. Together, these methods provide a strong foundation for creating a secure distributed system.

Overall, by adhering to these eight specific methods, distributed systems can achieve their desired efficiencies and maintain accuracy.

III. Methods for Federated Learning in Distributed Systems

A. Data Privacy

Currently, companies are collecting a significant amount of data on users. In turn, advanced data mining techniques can intelligently extract the useful data which can then be used to train large ML models. [2]. This raises privacy concerns as private data is shared between large-scale systems and models are trained on such data. To address these issues, utilizing FL in distributed systems offers a solution. FL enables the development of sophisticated ML models without exposure to sensitive user data.

Jia et al. introduced a method to secure privacy in data classification and similarity evaluation within distributed systems, ensuring that neither new data nor trained models are directly exposed during these processes [2]. To achieve this, they proposed various schemes on known algorithms including support vector machine, the oblivious transfer protocol, and the oblivious evaluation of multivariate polynomials [2]. Overall, their research introduces methods to ensure that data remains private and sensitive information – i.e., metadata – of specific models are not compromised when models are shared across a distributed system. This advancement has been critical for maintaining data privacy, specifically ensuring that models cannot be reversed engineered. Additionally, it enables companies to protect sensitive user data, which will ideally enhance user trust and encourage the voluntary collection of more data to train smarter, more sophisticated models.

B. Aggregation Algorithms

The aggregation of machine learning models, as seen in Figure 1, is a critical component of FL. To accomplish effective model aggregation, model merging algorithms were developed. These include Federated Stochastic Gradient Descent (FedSGD) and Federated Averaging (FedAvg)[16]. For FedSGD, a select number of system nodes compute the gradient on their local data. Subsequently, the selected leader node aggregates these gradients and applies the update to the global model. FedAvg, on the other hand, builds on top of FedSGD by having each system node conduct multiple local updates before global model aggregation.

Algorithm 1 FederatedAveraging and Federated Stochastic Gradient Descent Algorithm [16]

1: **Leading Node executes:**
2: Initialize w_0
3: **for** each round $t = 1, 2, \dots$ **do**
4: $\quad m \leftarrow \max(C \cdot K, 1)$
5: $\quad S_t \leftarrow$ (random set of m nodes)
6: $\quad$ **for** each node $k \in S_t$ **in parallel do**
7: $\quad\quad w_{t+1}^k \leftarrow \text{LocalNodeUpdate}(k, w_t)$
8: $\quad$ **end for**
9: $\quad m_t \leftarrow \sum_{k \in S_t} n_k$
10: $\quad w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$
11: **end for**
12: **LocalNodeUpdate(**k, w**):**
13: Split $\mathcal{P}_k$ into batches of size B
14: **if** FedAvg **then**
15: $\quad$ **for** each local epoch i from 1 to E **do**
16: $\quad\quad$ **for** each batch $b \in B$ **do**
17: $\quad\quad\quad w \leftarrow w - \eta \nabla \ell(w; b)$
18: $\quad\quad$ **end for**
19: $\quad$ **end for**
20: **else if** FedSGD **then**
21: $\quad$ **for** each batch $b \in B$ **do**
22: $\quad\quad w \leftarrow w - \eta \nabla \ell(w; b)$
23: $\quad$ **end for**
24: **end if**
25: **return** w to server

Algorithm 1 depicts pseudo-code for the FedAvg and FedSGD algorithms. Due to their similarity, line 14 is their respective branching off point. FedAvg executes multiple local updates, whereas FedSGD performs only

one. The variables in the algorithm represent the following:

- Leading Node:
  - w: model weights
  - C: the percentage of participating nodes
  - K: the total number of nodes
  - m: number of participating nodes
  - w_{t+1}^k: the local updated model weights of node k
  - n_k: data points held by node k
  - m_t: total data points across all selected nodes for round t
  - w_{t+1}: the new model weight, calculated as a weighted average of the local weights from the nodes
- System Nodes
  - E: number of learning epochs
  - w: model weight
  - P_k: local client dataset, split into B batches
  - η: the learning rate used to update the weights
  - $\ell(w; b)$: the loss function used to compute the gradient

As a whole, FedAvg and FedSGD compute the gradient via their loss functions on their local datasets. They use the gradient to update and then send their weights to the leader node. The leader node subsequently computes a weighted average of these updates. This weighted average serves as the weight for the global model.

Although similar, FedAvg and FedSGD have different use cases. FedAvg is better suited for systems that are large in scale, need frequent local updates, and have non-identically distributed data sets. Further, FedAvg has reduced communication overhead, as fewer updates are exchanged between the system nodes. However, this comes at the cost of higher local computation due to the multiple training iterations. In contrast, FedSGD is better suited for systems that require real-time global updates and less local computation. In turn, FedSGD struggles with a high communication overhead due to the constant transferring of information among nodes.

Despite their differences, both FedAvg and FedSGD ensure that sensitive data remains abstracted and is never directly shared with the leading system node. They achieve this by conducting their training locally. Ultimately, these approaches are critical in preserving data privacy and conducting efficient model aggregation.

IV. Existing Application of Federated Learning in Distributed Systems

A. NVIDIA Clara for Healthcare

The healthcare industry provides an exciting opportunity to leverage Federated Learning. Consequently, NVIDIA proposes using Clara, their healthcare-specialized FL platform, to create and deploy FL solutions to the highly regulated and privacy-focused industry of healthcare [17].

The high-level overview of Clara's architecture can be seen in the center image of Figure 2. This architecture mirrors greatly to that seen in Figure 1. However, Clara takes extra steps to ensure data privacy as it is handling extremely sensitive healthcare information. These methods include utilizing homomorphic data encryption and differential privacy (DP). Homomorphic data encryption allows for model update computations to be performed on encrypted data. This ensures that (a) sensitive data cannot be intercepted as it traverses the network and (b) the leading node performing the model aggregation cannot access the underlying local data. DP is used to prevent the reverse engineering of models. DP is conducted in two ways:

- **Selective Parameter Update** is where nodes share a fraction of the models updates.
- **Sparse Vector Technique** is when random noise is added to the model updates.

Both of these methods make it more difficult to reverse engineer the model and reconstruct the original data [17].

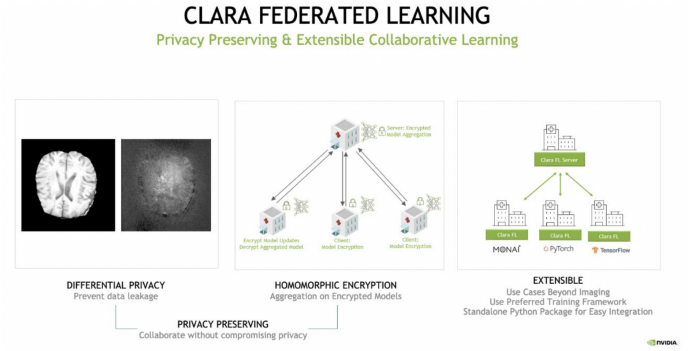

Figure 2: Clara FL Architecture, which highlights the focus on data privacy [17]

NVIDIA has conducted the following case studies showcasing the application of FL in a distributed environment. All of which deployed the architecture that is depicted in Figure 2.

1) Breast Mammography: This study focused on improving breast density classification for early breast cancer detection by leveraging Federated Learning

across numerous healthcare institutions (American College of Radiology (ACR), Diagnosticos da America (DASA), Ohio State University (OSU), Partners HealthCare (PHS), and Stanford University).

Initially, NVIDIA trained a standard 2D mammography classification model on data solely accumulated at PHS. Subsequently, they retrained the model using Clara Federated Learning with data from PHS and all other healthcare sites. These re-trainings utilized the data privacy techniques that are associated with FL. As a result, each institution obtained a better performing model that had overall superior predictive power when applied to their local datasets [18].

2) COVID-19 X-Rays: This study focused on improving real-world models for COVID-19 diagnoses solely based on chest X-Rays. There were three participating institutions (University of Minnesota, Indiana University, and Emory University). These institutions utilized distributed data that was stored in the cloud (Amazon Web Services and Microsoft Azure) and local servers. Their data consisted of about 80,000 labeled images with a 30/70% positive/negative COVID diagnosis. Subsequent results showed a 5-8 percent model performance improvement when utilizing the FL trained model [17].

3) Prostate Cancer: This study focused on developing an FL model to detect Prostate cancer in men. The ProstateX Challenge dataset was used to gather prostate cancer data. There were three participating client sites (State University of New York (SUNY), Uni. of Calif., Los Angeles (UCLA), and the National Institutes of Health (NIH)). The subsequently trained FL model provided equivalent results to that of a model trained directly on the pooled data from the three institutions [19].

4) Pancreatic Cancer: This study focused on developing an FL model to detect pancreatic cancer. NVIDIA teamed up with institutions in Taiwan and Japan to build models. The global Federated Learning model that was built via the distributed data from the institutions was a success as it received an 82.3% Dice score on healthy pancreatic patients [20]. Meaning, it was highly accurate at medical image segmentation, which is critical in identifying the nature of the pancreatic tissue.

5) Oxygen Requirements for COVID Patients: As seen by the COVID-19 pandemic, efficient allocation of medical resources is paramount when the healthcare system is stressed. As a result, this study focused on determining if COVID-19 patients would need to be put on oxygen after their COVID diagnosis. There were roughly 20 participating institutions in this study. Each institution received a copy of the FL trained model to use on their local dataset. When put into practice, the model produced exceptional results in predicting the need for oxygen in COVID patients [17].

Overall, the outcomes of these case studies highlighted the effectiveness of Federated Learning in a distributed healthcare environment, along with preserving data privacy. Further, they achieved the several key goals of a distributed system:

- **Resource Sharing:** Each local healthcare institution shared their data and models with the rest of the partnering institutions.
- **Transparency:** Each local healthcare institution was unaware that their models were trained via distributed healthcare data. This is a critical component in maintaining data privacy.
- **Openness:** Each local healthcare institution was easily integrated into the Federated Learning system.
- **Dependability:** Each local healthcare institution relied on the learning conducted by other institutions to accurately influence their ML model.
- **Security:** Each local healthcare institution trusted that the sharing of models and data remained secure.
- **Fault Tolerance:** There were no documented cases of failure, but one of the local healthcare institutions could have ceased to contribute to the learning process and the FL and distributed system processes would have continued to function.
- **Efficiency and Scalability:** Each local healthcare institution added to the list of institutions increased the amount data available for the FL without sacrificing performance.

Due to the novel nature of utilizing FL in healthcare, NVIDIA has raised a myriad of logistical concerns. Mainly establishing cooperation among healthcare institutions due to the differing legal and ethical boundaries that are currently set, especially institutions that operate in separate countries. Tackling these issues can help pave the way for further deployment of FL in the healthcare industry.

V. Potential Future Application of Federated Learning in Distributed Systems

A. Distributed Security Monitoring of BGP Announcements

Border Gateway Protocol (BGP) is an essential internet protocol that exchanges routing information among Autonomous Systems (ASes) [21]. Therefore, malicious BGP announcements, also known as BGP Prefix Hijacking, pose a significant security threat to today's internet.

Federated Learning offers a promising solution to detect and mitigate malicious BGP announcements. In this context, individual systems—specifically ASes—would create local models to learn about BGP announcements that traverse their network. A FedAvg approach would be ideal for this application as it is well-suited for handling rapidly changing, AS-specific BGP updates. Further, to adhere to the architecture in Figure 1, a leader AS must be designated to perform model aggregation. This leader AS could be selected through election by bullying using AS Numbers as unique identifiers.

Applying this approach to BGP announcements would allow networks to leverage an ML model that was trained on distributed data. Ideally, this will provide a sophisticated image of what denotes a malicious BGP announcement. Further, this approach aligns with the several key goals of a distributed system:

- **Resource Sharing:** Each AS shares their trained model with the rest of the network.
- **Transparency:** Each AS is unaware that their model was trained via BGP announcements from other networks.
- **Openness:** Each AS system can easily be integrated into the Federated Learning system.
- **Dependability:** Each AS can rely on the learning conducted by other systems to accurately impact their decision making.
- **Security:** Each AS system can trust that the sharing of models and BGP data remains secure.
- **Fault Tolerance:** An AS could cease to contribute to the learning process and the FL and distributed system processes would continue to function.
- **Efficiency and Scalability:** Each AS added to the network increases the efficiency of processing BGP data and improves the FL models with scale.

As a whole, utilizing FL among ASes can provide an exciting solution to the security threat of BGP Prefix Hijacking.

Note that this potential application does not have to be limited to BGP announcements, and rather could be applied to other aspects and protocols of computer networking and cybersecurity.

VI. Discussion

Leveraging Federated Learning in distributed systems has the potential to create a transformative shift in how ML models are trained and utilized. Additionally, with their prowess to handle distributed data sets, maintain data privacy, and scale efficiently, FL offers the chance to be ubiquitously applied.

A. Strengths of Federated Learning in Distributed Systems

Federated Learning addresses several challenges associated with standard machine learning. First, it reduces the negative effects of data centralization by training models locally and only sharing aggregated results. As a result, FL is able to ensure data privacy, which standard machine learning lacks as data is directly shared and centralized. Therefore, FL is an exciting candidate to be applied in other privacy-centered areas such as finance and government.

B. Challenges of Federated Learning in Distributed Systems

Despite the strengths associated with FL, there are some significant challenges and limitations. One of the most hindering obstacles is communication overhead. In large-scale distributed systems the frequent communication and exchanging of models between system nodes can create inefficiencies and slow down the FL process. Secondly, another challenge is heterogeneity among participating nodes. The differences in computational capabilities, physical location, network connectivity, and data distributions can all adversely affect the FL process. Third, there are security vulnerabilities associated with FL models. They are susceptible to various adversarial attacks, such as model poisoning and data inversion [22]. Further, if malicious nodes join the system, they can disrupt the learning process by introducing false updates. Or, they can attempt to reverse-engineer the model they receive to expose sensitive data. Finally, the use of a leader node handling the model aggregation introduces a bottleneck and single point of failure. The use of election algorithms greatly hinders the potential threat of this issue, however, the computational capabilities of the newly elected leader could lead to greater inefficiencies.

C. Ethical Considerations

The integration of FL into distributed systems also presents significant ethical considerations. On top of the ethical issues of data security as discussed previously, there are significant ethical qualms regarding equity and inclusivity. For instance, the deployment—or lack thereof—of FL in resource-limited environments raises questions about whether these regions can fairly contribute to the learning process. For example, in the healthcare industry, this concern extends to whether patients from lower socioeconomic backgrounds will be adequately represented in the training data. Moreover, there are concerns about whether these populations will be able to fully access and reap the benefits that FL has to offer in the healthcare industry.

D. Future of FL in Distributed Systems

To solve the challenges discussed, there are strategies that could be developed. To reduce the communication overhead, hybrid aggregation models of FedAvg and FedSGD could be put into practice. In addition, investing more money into improving computational capabilities and network connectivity between nodes can help improve efficiency. Furthermore, novel security measures to prevent the reverse-engineering of models would aid in data security. Lastly, promoting equity among system nodes could involve developing a more balanced and impartial approach to data analysis. This would ensure that no system's data is disproportionately prioritized, even if it is of higher quality or larger volume. This would ideally provide data-scarce regions with a more equitable say in the learning process.

VII. Conclusion

Federated Learning represents an exciting and rapidly evolving subfield of Machine Learning. Further, when integrated into a distributed system, FL offers enhanced efficiency, effective data security, and impressive scalability. These attributes make it a compelling approach for enabling collaborative and effective learning.

Specifically, this survey explored the methodologies underpinning FL; its foundations in high-performance computing, machine learning, and distributed systems; model aggregation techniques; new methods to protect user data and ensure data privacy; and its practical deployment in real-world scenarios.

Ultimately, leveraging FL in distributed systems is an exciting approach that possesses significant promise in solving many pressing issues.

References

[1] S. W. D. Chien, S. Markidis, V. Olshevsky, Y. Bulatov, E. Laure, and J. Vetter, "Tensorflow doing hpc," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 509–518.

[2] Q. Jia, L. Guo, Z. Jin, and Y. Fang, "Preserving model privacy for machine learning in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 8, pp. 1808–1822, 2018.

[3] M. van Steen and A. Tanenbaum, "Distributed Systems," 2020, [Accessed 10-10-2024].

[4] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," 2019. [Online]. Available: https://arxiv.org/abs/1902.04885

[5] J. Liu, J. Huang, Y. Zhou, X. Li, S. Ji, H. Xiong, and D. Dou, "From distributed machine learning to federated learning: a survey," vol. 64, no. 4, 2022. [Online]. Available: https://doi.org/10.1007/s10115-022-01664-x

[6] NVIDIA, "NVIDIA Corporation. 2017. Nvidia Tesla V100." https://www.nvidia.com/en-us/data-center/tesla-v100/., 2017, [Accessed 10-10-2024].

[7] A. Asaduzzaman, A. Trent, S. Osborne, C. Aldershof, and F. N. Sibai, "Impact of cuda and opencl on parallel and distributed computing," in *2021 8th International Conference on Electrical and Electronics Engineering (ICEEE)*, 2021, pp. 238–242.

[8] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," vol. 53, no. 2, 2020. [Online]. Available: https://doi.org/10.1145/3377454

[9] A. Yakimovich, A. Beaugnon, Y. Huang, and E. Ozkirimli, "Labels in a haystack: Approaches beyond supervised learning in biomedical applications." *Patterns, 2(12)*, vol. 19, p. 100860, 2021.

[10] B. Rajoub, "Chapter 3 - supervised and unsupervised learning," in *Biomedical Signal Processing and Artificial Intelligence in Healthcare*, ser. Developments in Biomedical Engineering and Bioelectronics, W. Zgallai, Ed. Academic Press, 2020, pp. 51–89. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128189467000032

[11] J. Wang and F. Biljecki, "Unsupervised machine learning in urban studies: A systematic review of applications," *Cities*, vol. 129, 2022.

[12] S.-S. Learning, "Semi-supervised learning," *CSZ2006. html*, vol. 5, p. 2, 2006.

[13] L. P. Kaelbling and M. L. Littman, "Reinforcement learning: A survey," 1996.

[14] "Ml | stochastic gradient descent (sgd)," https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/, accessed: 2024-11-30.

[15] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1*. Springer, 2009, pp. 626–631.

[16] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2023. [Online]. Available: https://arxiv.org/abs/1602.05629

[17] "Federated learning for healthcare using nvidia clara," https://developer.download.nvidia.com/CLARA/Federated-Learning-Training-for-Healthcare-Using-NVIDIA-Clara.pdf, accessed: 2024-11-30.

[18] H. R. R. et al, ""federated learning for breast density classification: A real-world implementation,"," *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning,*, vol. 24, no. 3, pp. 181–191, 2020.

[19] K. v Sarma et al., ""federated learning improves site performance in multicenter deep learning without data sharing,"," *Journal of the American Medical Informatics Association,*, vol. 28, no. 6, pp. 1259–1264, 2021.

[20] P. Wang, C. Shen, H. R. Roth, D. Yang, D. Xu, M. Oda, K. Misawa, P.-T. Chen, K.-L. Liu, W.-C. Liao, W. Wang, and K. Mori, "Automated pancreas segmentation using multi-institutional collaborative deep learning," 2020. [Online]. Available: https://arxiv.org/abs/2009.13148

[21] J. Kurose and K. Ross, "Computer Networking: A Top Down Approach," Kurose,J.F.,&Ross,K.W.(2022).Computernetworking: Atop-downapproach.PearsonEducationLimited., 2022, [Accessed 10-10-2024].

[22] X. Zhou, M. Xu, Y. Wu, and N. Zheng, "Deep model poisoning attack on federated learning," *Future Internet*, vol. 13, no. 3, 2021. [Online]. Available: https://www.mdpi.com/1999-5903/13/3/73