UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Decentralized Federated Learning using Distributed Ledgers and Smart Contracts

Diploma Thesis

Panagiotidis Ioannis

Supervisor: Mountanos Ioannis

July 2023

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Decentralized Federated Learning using Distributed Ledgers and Smart Contracts

Diploma Thesis

Panagiotidis Ioannis

Supervisor: Mountanos Ioannis

July 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αποκεντρωμένη Ομόσπονδη Μάθηση χρησιμοποιώντας Κατανεμημένα Κατάστιχα και Έξυπνα Συμβόλαια

Διπλωματική Εργασία

Παναγιωτίδης Ιωάννης

Επιβλέπων: Μούντανος Ιωάννης

Ιούλιος 2023

Approved by the Examination Committee:

Supervisor **Mountanos Ioannis**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Elias N. Houstis**
Emeritus Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Chronaios Alexandros**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

First and foremost, I would like to thank from the bottom of my heart my family for their unending support throughout my life, for enduring my constant whining and for always being eager to listen me mumble about stuff that they don't understand.

Also many thanks to my friends for the their company throughout these years. Among them I hold dearest to my heart my childhood friends, my high school friends and last but not least the friends that I met in UTH's Drama Club. Additional thanks to the ones with whom we shared projects and ideas.

I would also like to thank my teachers in junior high, for teaching me Rhetoric and Programming, therefore sparking my interest in computer science.

Many thanks to my tutors in senior high school without whom I would never have come to Volos.

I would like to express my appreciation to my committee:

First I would like to thank Prof. Mountanos for our long and fruitful cooperation and communication as well as for the depth of his classes and for always setting high standards.

I would like to also thank Prof. Chronaios and Prof. Houstis for the opportunities that they provided me and the high quality of their classes.

An honorable mention to Dr Dimitris Chatzopoulos whose guidance and assistance was essential to the formulation and completion of this thesis.

Thank you all.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Panagiotidis Ioannis

Diploma Thesis

Decentralized Federated Learning using Distributed Ledgers and Smart Contracts

Panagiotidis Ioannis

Abstract

Blockchain based federated learning employs Blockchain technology in an attempt to mitigate concerns about current federated learning environments and in order to implement new features regarding auditability, transparency, immutability and availability of federated learning models as well as to enhance users' security and privacy, to provide incentives for well behaved users and to strike penalties to malicious ones. Most such efforts have been implemented using Smart contracts in the public Ethereum Blockchain or the private Hyperledger platform. As Hyperledger based approaches require trust in a designated authority we focused on public platforms for their decentralized (trustless) operation and public security verification. As Ethereum is known for its low TPS which results in large latencies in Smart contract execution, in this thesis we implemented a BCFL system in the Solana Blockchain which promises faster TPS and lower latencies in Smart contracts. We explore the practicality of implementing a BCFL (Blockchain Federated Learning) system on the Solana Blockchain, examining both its potential advantages and limitations. Our investigation involves conducting experiments with Smart contracts at different levels of model quantization and exploring various techniques for model pruning. The goal is to strike a suitable balance between the number of participants in the system and number of weights of the stored models.

Keywords:

Solana, Distributed ledgers, Smart contracts, Federated learning, Decentralized technologies

Διπλωματική Εργασία

Αποκεντρωμένη Ομόσπονδη Μάθηση χρησιμοποιώντας Κατανεμημένα Κατάστιχα και Έξυπνα Συμβόλαια

Παναγιωτίδης Ιωάννης

Περίληψη

Η ομόσπονδη μάθηση με βάση την τεχνολογία Blockchain (BCFL) χρησιμοποιεί την τεχνολογία Blockchain σε μια προσπάθεια να μετριάσει τις ανησυχίες σχετικά με τα τρέχοντα περιβάλλοντα ομόσπονδης μάθησης και να εφαρμόσει νέα χαρακτηριστικά όσον αφορά την ελεγξιμότητα, τη διαφάνεια, την αμεταβλητότητα και τη διαθεσιμότητα των μοντέλων ομόσπονδης μάθησης, καθώς και να ενισχύσει την ασφάλεια και την ιδιωτικότητα των χρηστών, να παρέχει κίνητρα για τους καλά συμπεριφερόμενους χρήστες όπως και να επιβάλλει κυρώσεις στους κακόβουλους χρήστες. Οι περισσότερες τέτοιες προσπάθειες έχουν υλοποιηθεί με τη χρήση έξυπνων συμβολαίων στο δημόσιο Blockchain Ethereum ή στην ιδιωτική πλατφόρμα Hyperledger. Καθώς οι προσεγγίσεις που βασίζονται στο Hyperledger απαιτούν εμπιστοσύνη σε μια καθορισμένη αρχή τρίτων, επικεντρωθήκαμε στις αποκεντρωμένες (δημόσιες) πλατφόρμες για την αναξιόπιστη λειτουργία τους και τη δημόσια επαληθευσιμότητα της ασφάλειας. Καθώς το Ethereum είναι γνωστό για το χαμηλό TPS του, το οποίο έχει ως αποτέλεσμα μεγάλες καθυστερήσεις στην εκτέλεση των Έξυπνων Συμβολαίων, στην παρούσα διατριβή υλοποιήσαμε ένα σύστημα BCFL στο Blockchain Solana, το οποίο υπόσχεται μεγαλύτερο TPS και χαμηλότερες καθυστερήσεις στα Έξυπνα συμβόλαια. Διερευνούμε την πρακτικότητα της υλοποίησης ενός συστήματος BCFL στην πλατφόρμα Solana, εξετάζοντας τόσο τα πιθανά πλεονεκτήματα όσο και τους περιορισμούς του. Η έρευνά μας περιλαμβάνει τη διεξαγωγή πειραμάτων με Έξυπνα συμβόλαια σε διαφορετικά επίπεδα κβαντισμού μοντέλων και τη διερεύνηση διαφόρων τεχνικών για το κλάδεμα μοντέλων. Στόχος είναι να βρεθεί η κατάλληλη ισορροπία μεταξύ του αριθμού των συμμετεχόντων στο σύστημα και του αριθμού βαρών των αποθηκευμένων μοντέλων.

Λέξεις-κλειδιά:

Solana, Κατανεμημένα κατάστοιχα, Έξυπνα συμβόλαια, Ομόσπονδη μάθηση, Αποκεντρωμένες τεχνολογίες

Table of contents

List of figures

List of tables

Abbreviations

ML	Machine Learning
FL	Federated Learning
BCFL	Blockchain based Federated Learning
BCHFL	Blockchain based Federated Learning
SC	Smart Contract
TPS	Transactions per Second
DLT	Distributed Ledger Technology
DeFi	Decentralized Finance
DAG	Directed Acyclic Graph
DAO	Decentralized Autonomous Organisation
GDPR	General Data Protection Regulation
VITA	Validity, Integrity, Termination and Agreement
NN	Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Nerwork
MLP	Multilayer Perceptron
FedAvg	Federated Average
SGD	Stochastic Gradient Descent
ADAM	Adaptive Moment estimation
SWIFT	Society for Worldwide Interbank Financial Telecommunications
ATM	Automated Teller Machine
RAFT	Reliable, Replicated, Redundant And Fault-Tolerant
PBFT	Practical Byzantine Fault Tolerance
FLP	Fischer Lynch Patterson
PoW	Proof of Work

PoS	Proof of Stake
PoH	Proof of History
PoET	Proof of Elapsed Time
TX	Transaction
SHA	Secure Hash Algorithm
EVM	Ethereum Virtual Machine
NFT	Non-Fungible Token
CLI	Command Line Interface
ETH	Ethereum
SOL	Solana's native coin
SDK	Software Development Kit
REST	Representational State Transfer
JSON	JavaScript Object Notation
RPC	Remote Procedure Call
gRPC	google Remote Procedure Call
LLVM	Low Level Virtual Machine
ELF	Executable and Linkable Format
OOP	Object Oriented Programming
API	Application Programming Interface
SPoF	Single point of Failure
DDoS	Distributed Denial of Service
SVs	Shapley Values
PoSap	Proof of Shapley
CSVES	Class-Sampled Validation-Error Scheme
DSC	Dual skip Chain
ZKP	Zero Knowledge Proof
STARK	Scalable Transparent Arguement of Knowledge
MNIST	Modified National Institute of Standards and Technology
ReLU	Rectified Linear Unit
IoT	Internet of Things
dApps	Decentralized Applications

Chapter 1

Introduction

Federated learning enviroments consist of participating nodes that train models based on their data and upload their gradients to a centralized server architecture which computes their Federated Average in order to create a single model. Despite Federated learning seeing massive success and adoption, the use of a centralized server architecture has faced much criticism due to its challenges and limitations in terms of availability, privacy preservation, communication, security, and transparency.

Distributed ledger technologies have been an research topic which although originally introduced for monetary transactions without the need for third party authorization among peers, now sees massive adaptation due to the support of decentralized code execution through smart contracts.

Blockchain platforms with smart contract support starting with Ethereum have multiplied bringing with attempts to enable faster TPS, lower transaction latencies, scalability as well as new technological capabilities such as interoperability, distributed cloud and data privacy.

Blockchains with smart contract support have been employed as replacements to traditional Federated learning environments where they provide resistance against SPoF and DDos attacks, code and weight redundancy transparency and immutability, privacy preservation, incentives for well behaved participants as well as penalties for malicious ones.

While the defacto smart contract platform has been Ethereum, it has been limited by its low TPS. Solana's high TPS has driven the accelerated adoption of its smart contracts and innovation of dApps in its ecosystem.

In this thesis we use Solana smart contracts to build an aggregating server for Federated learning enviroments.

1.1 Purpose of this thesis

Current centralized Federated learning environments lack decentralization, immutability, traceability, incentives, integrity, and reliability which are crucial for creating more robust, transparent, and trustworthy systems that can harness the potential of Federated learning while addressing privacy concerns and fostering collaboration among participants.

To address the aforementioned limitations, Blockchain solutions have emerged as a potential remedy. By leveraging Blockchain technology, decentralized Federated learning environments can be created to address much of these issues and introduce additional features.

Existing efforts include the implementation of Blockchain solutions using Hyperledger, which is a private Blockchain framework. While Hyperledger provides certain advantages such as permissioned access, improved privacy and great performance, it remains a private Blockchain not suitable for trustless environments. Trustless environments require a decentralized network where participants can interact without relying on a central authority. Private Blockchains, like Hyperledger, rely on trust in a designated authority, thereby limiting their effectiveness in fully decentralized scenarios.

Another popular Blockchain platform that has been explored is Ethereum. While Ethereum offers the advantage of a public Blockchain, it faces challenges in terms of transaction processing speed (TPS) and latency. The current scalability limitations of Ethereum result in lower TPS and higher latencies, which can hinder the performance and real-time nature of Federated learning environments. These issues need to be addressed to fully harness the potential of Blockchain technology in creating robust and efficient decentralized Federated learning systems.

Solana was therefore chosen as the platform to be investigated in this thesis as it is a public Blockchain platform but its flagship feature is that it promises significantly faster TPS than current public Blockchains with smart contract support.

However, the development ecosystem of Solana is still in its early stages, resulting in a slow adoption rate among developers. Additionally, there is a lack of research works dedicated to Solana development.

1.1.1 Contribution

This thesis' contribution is summarised as follows :

- Opportunities in the Solana platform
- The properties of Solana smart contracts
- Development in the Solana ecosystem
- Use of Solana smart Contracts for Blockchain Federated learning
- Evaluation of the of Solana for BCHFL in realistic scenarios
- Model pruning methodologies for deployment in constrained environments

1.2 Content organization

This chapter is the introduction 1 Part I of this thesis sets the theoretical backround. The second chapter 2 presents Federated learning while the third chapter 3 introduces Distributed Ledger Technologies (DLTs) as well as Blockchains. The fourth chapter 4 introduces Smart contracts and describes their operating principles. Chapter 5 descibes the Solana Blockchain which was used to implement the experiments in this thesis and chapter 6 descibes the characteristics of Smart contracts in the Solana platform. Chapter 7 introduces Blockchain based Federated learning which is the application to be developed in Solana Smart contracts in the context of this thesis and chapter 8 discusses previous works and efforts in BCFL. Part II of this thesis describes the propsed system and testing conducted. In chapter 9 the proposed system design is described while in chapter 10 we provide the specifics for the implementation or the proposed system and its components. In chapter 11 we present the Evaluation methodology for the proposed system and its results while in chapter 12 we discuss said results. Part III is about conclusions where chapter 13 discusses challenges faced, while also providing future directions, research as well as the feasability of BCFL in the Solana ecosystem.

Part I

Theoretical Backround

Chapter 2

Federated learning

2.1 Introduction - ML, Distributed ML

Machine learning is the procedure of training models from data automatically [46, 39, 108, 31] . Such a model is defined in terms of its algorithm, and its hyperparameters. Then the training process uses said data to optimize the structure or the parameters of the model with aim to improve the algorithms' performance. New data is processed through the selected algorithm to produce an output result such as a prediction of the classification of the pattern in question. Algorithms differentiate themselves based on whether data are labelled or not into supervised, unsupervised, semi-supervised, and reinforcement learning. In supervised learning the training data consists of (x,y) pairs, where x denotes the pattern and y its label, while unsupervised learning consists only of x and no y. Semi-supervised learning contains a small subset of data with labels but most data do not contain labels. Finally reinforcement learning further trains already pretrained models in order to adapt them to new environments.

Regarding data, in modern applications it spans enormous volume that cannot be handled by single machines or is inherently distributed and lies in multiple resources. In both cases the training procedure must be tweaked in order to accommodate for these new circumstances. Distributed machine learning is a machine learning paradigm which enables training to take place in distributed resources which accounts for distributed and huge data while also providing significant performance benefits as it largely accelerates the training procedure especially when combined with modern parallelization techniques and modern hardware. It is widely employed in organizations' server clusters and datacenters and powers todays commercial machine learning applications.

2.2 Motivation

Collecting users' data in single organisations and datacenters raises serious concerns regarding user privacy and user data security [46, 39, 108, 31, 110]. Privacy related concerns have been amplified by the increase of data breaches and suspicion about organisations' data monetization practices. To combat these practices privacy preserving laws and regulations across multiple nations and states have limited or even prohibited the collection of users' raw data in single organisations or datacenters.

Simultaneously as AI applications see exponential adoption and growth, efforts are focused on ways to train and deploy better machine learning models. As machine learning models' accuracy is highly dependent on the amount of data used to train them, huge amounts of data must be employed if such models are expected to deliver accurate predictions as well as usable results in order to be used in realistic scenarios [46]. Such data may include detailed and extensive personal information, the use of which use is dictated by data privacy regulations.

In classical machine learning applications that rely on data from mobile or IoT devices, data is sent to servers or datacenters where machine learning algorithms are trained and then deployed in commercial applications [46]. The concerns discussed above stem from the fact that users' raw data are sent in single organisations' servers in order to be used in model training. Thus they comprise centralized data used to train centralized models. Moreover once raw data is in the hands of single organizations one cannot be certain about the practices used in its processing and whether they are used for other purposes other than training since they also hold monetary value themselves.

This has spurred critisism on traditional distibuted machine learning as it requires the training process to have access to users' raw training data and therefore raises concerns regarding the users' privacy and data security [31, 110]. It also comes with huge energy and bandwidth costs for the communication needs of the data which have to be aggregated into a central location and then distributed in order to achieve learning even when deployed in datacenters where bandwidth is most available and point to point distances between training nodes are minimal. All these concerns have guided research to investigate new methods for achieving distributed training while cutting down on communication costs and preserving users' privacy.

2.3 Federated learning Definition

Federated learning, also known as collaborative learning, is a distributed machine learning paradigm where a machine learning algorithm is trained by combining local models trained at participating nodes in a single global model which lies in an aggregating server, all without individual training nodes sharing raw data between themselves or the server [110, 108]. This is an efficient approach which enables the utilization of distributed data and distributed computing resources in order to collaboratively train models without sharing data between the nodes and without storing raw data in single organisations' servers or datacenters.

Fl differentiates itself from classical distributed machine learning in three key ways. First, since no raw data is directly exchanged between nodes and the server, law requirements regarding users' privacy and data collection such as GDPR's data minimalization and consent principle (GDPR Article 5 and 6 respectively) are met. Second, Fl efficiently takes advantage of the heterogenous and highly distributed resources and data in a multitude of regions, organizations and devices. This also enables the models to be shared in order to reach collaboration between organisations. Third, Fl addresses data privacy and security concerns by employing encryption and other defensive mechanisms as to preserve the privacy and security of raw data and to protect them against leaks which convey financial risks and loss of reputation.

2.4 Federated learning Types / The How

Federated learning is classified in three types based on the distribution characteristics of training data into three variants. Horizontal, vertical as well as hybrid Fl [110].

- Horizontal : Same features, different samples
- Vertical : Different features, same samples
- Hybrid : Different features, different samples

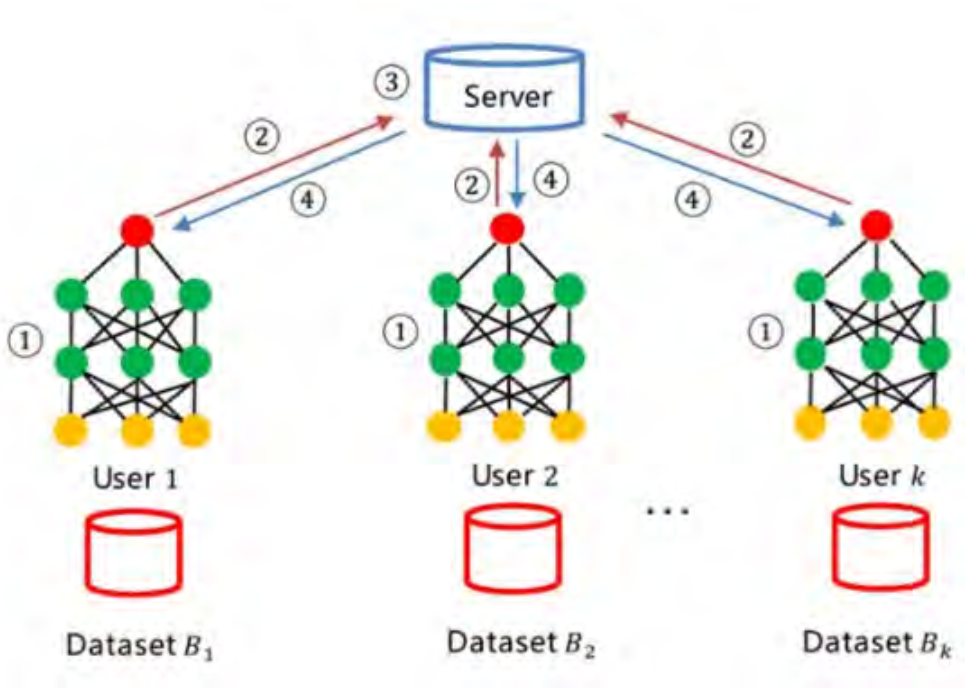

Figure 2.1: Federated learning scheme [28]

2.4.1 Horizontal Federated learning

Horizontal Federated learning is applied when datasets of participating clients data have the same attributes or features but different samples [110, 108]. This does not mean that data between clients is Indepenent and Identically Distributed, merely that the featureset is the same.

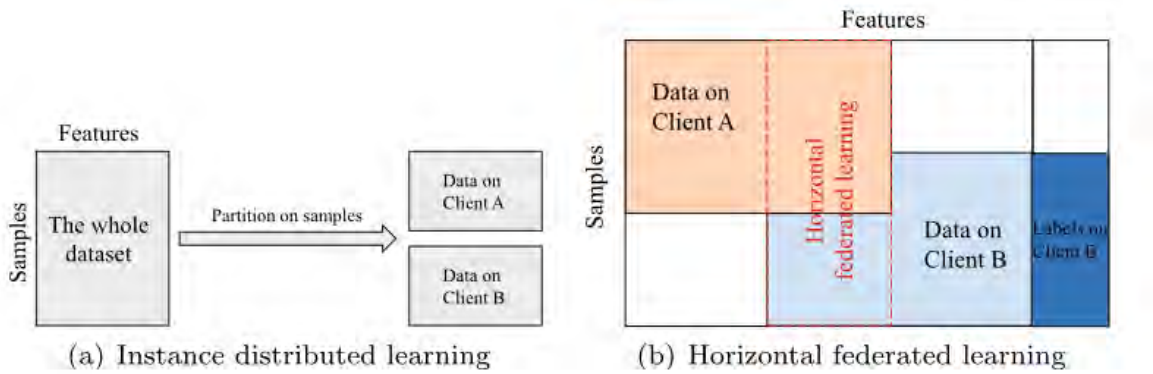

Figure 2.2: Horizontal Federated learning between clients [51]

This is the most widely adopted Federated learning type for its relative ease of implementation and privacy preservation properties. As shown in figure 2.2 models are only trained with the shared features between samples as non shared features require modifications in

models in order to be incorporated into the training process and are therefore simply not utilized. Regarding privacy, each clients' or organizations' data can be mixed and remain anonymous. This is in contrast to vertical Fl where as explained below requires cryptographic identification in order to determine which samples are shared between organizations and which are not [110, 108] . This work focuses on this Fl type which will be explored in depth and will finally be implemented for the experiments to be conducted.

2.4.2 Vertical Federated learning

Vertical Federated learning is applied when clients' datasets have different attributes but the same samples [110, 108]. For example, two financial organizations may hold different data for their customers in order to compute their credit scores, but may want to create joint models based on customers from both organizations.

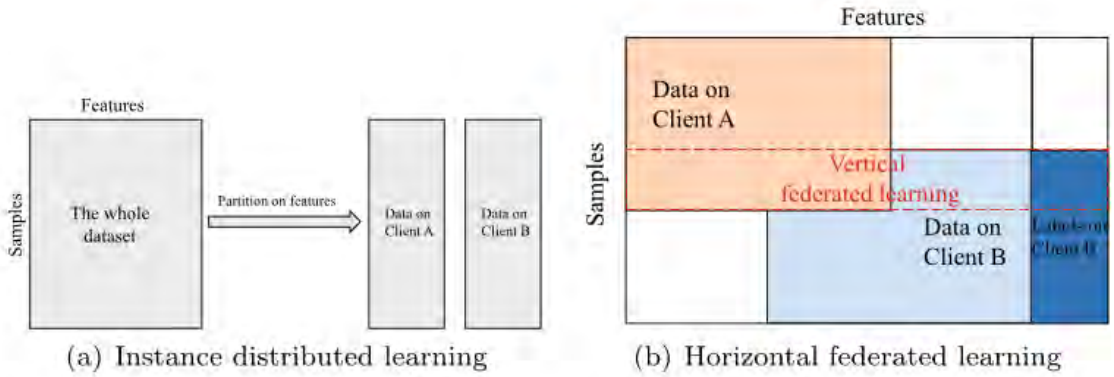

Figure 2.3: Vertical Federated learning between clients [51]

In this scenario the central server is not an aggregator of models but a coordinator as is called since its purpose is essential to training as it calculates the losses used to train model parameters. The coordinator employs encryption methods to identify corresponding samples across organizations by encryption key pairs and uses homomorphic encryption to retain privacy properties of the data.

2.4.3 Hybrid Federated learning

Hybrid Federated learning addresses scenarios where participating clients not only contain different features in their samples but also different samples completely [110, 108] . Such

real world scenarios require much attention as to not break privacy preservation as different training organizations need to share identification of their samples when training. When training with disjoint data across training parties the combination of models is non-trivial and together with efforts to retain privacy make up for the majority of research questions in this area. Protocols are proposed for both such as the Private Set Intersection (PSI) that can be implemented using classical public-key cryptosystems in order to keep unintersected data private.

2.5 Horizontal Fl Algorithm - FedAvg

FedAvg was the first algorithm for horizontal Federated learning proposed by B. McMahan in [58]. FedAvg and related algorithms are comprised of similar steps [46, 39, 108, 31, 110]:

- Random initialization of parameters that lie in the global ML model and distribution of the model to all connected nodes.
- Nodes use their own data to train the received model for a number of epochs. After that they calculate the difference between the trained model and the received model which in this context takes the place of the updating gradient found in traditional ML training. Finally nodes send such computed gradients to the aggregating server
- The aggregating server combines the uploaded models to merge updates into its stored model
- Repetition of steps 2 and 3 until the training ends either by the nodes or by the aggregating server.

This training algorithm is indeed similar to the ones used in synchronous distributed machine learning. Although they share characteristics, horizontal Federated learning implementation have to overcome several challenges that arise from the behaviour of edge devices that participate in the training process who may exhibit availability outages and slow training times. This results in there being no guarantees in terms of the trained models' convergence speed or eventual performance.

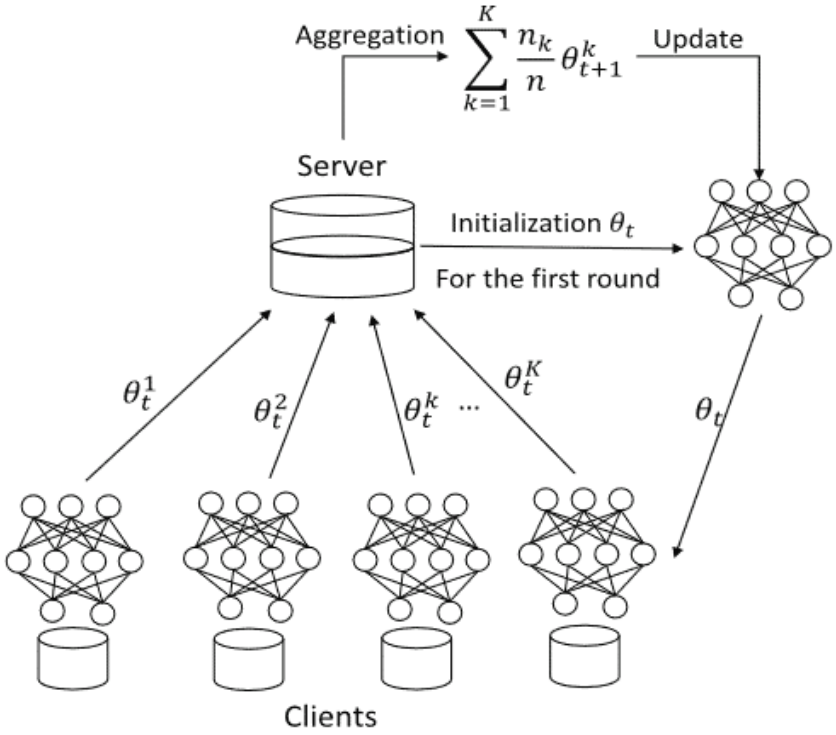

Figure 2.4: Horizontal FedAvg Schema [51]

2.6 Formal definitions

The optimization problem that Fl attempts to solve is formulated in 1 Given n training datasets $\mathcal{A} = A_1, A_2, \ldots, A_n$, which contain points $(x, y) \sim \mathcal{A}$, Fl aims minimize the expectation of loss over the distribution of all the datasets $\mathcal{A}_\setminus$ by learning a function $\widehat{F}$ [58].

$$\widehat{F} = \underset{F \in \mathcal{H}}{\operatorname{argmin}} \underset{(x,y)\in\mathcal{A}}{\mathbb{E}} L(y, F(x)), \tag{2.1}$$

where $L(y, F(x))$ is the loss (given by the selected loss function L) of $F(x)$ to the label y. During the training process, the stochastic gradient descent (SGD) approach is generally used to attempt to find the minima the loss function using Formula 2.1 [58].

$$F_{k+1}(x) \leftarrow F_k(x) - \eta_k \nabla F_k(x) \tag{2.2}$$

$F_k(x)$ is the trained model of iteration k^{th}, $\nabla F_k(x)$ is the gradient of the model at iteration k^{th}, η_k is the learning rate, and $F_{k+1}(x)$ is the model of the k^{th} iteration 2.2. As these calculations are dispersed among numerous computing entities, the gradient vectors or model vectors of each node of computing are aggregated using an aggregation algorithm, in order to achieve consensus of multiple models and to generate a global model. The learning rate

can be dynamically adapted using a local adaptive optimizer, e.g., Adam, and/or cross-round learning rate schedulers.

Algorithm 1 Federated Averaging. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and a is the learning rate [58]

Server executes:

initialize w_0

for each round t = 1,2,.... **do**

m = $max(C \times K, 1)$

S_i = (random set of m clients)

for each client k $\in S_i$ **in parallel do**

${w^k}_{t+1}$ = ClientUpdate(k,w_t)

${w^k}_{t+1}$ = $w_{t+1} + \sum_{k=1}^{K} \frac{n_k}{n} w_{L+1}^k$

Client executes:

procedure ClientUpdate(k, w) {R}un on client K

B = (split P_k into batches of size B)

for each local epoch i from 1 to E **do**

for batch $b \in B$ **do**

w $\leftarrow w - \eta \nabla l(w; b)$

Return w to server

2.7 Opportunities in Federated learning

Techniques currently used in distributed machine learning can be also used in Federated learning. For instance in horizontal Fl the data parallelism can be exploited in order to train models in subsets of the training data in parallel, while in vertical Fl paths of a single model are assigned to different devices to process different feature data.

New techniques are also being developed for the aggregating algorithms and the processing of the final resulting model. Additionally model compression is being employed for the

reduction of data transfers across participating nodes, while differential privacy, homomorphic encryption and robustness aggregation are used to combat attacts that poison or affect models in Fl environments or help retain users' data privacy. All of that to cater to the numerous applications of Fl that include mobile edge networks IoT driven healthcare or Federated recommendation systems [10].

2.8 Aggregation Algorithms

So far no discussion has taken place for the implementation of the aggregating server itself. Much like machine learning which has been classically centralized in a single machine but as efforts and needs evolve has become distributed and eventually decentralized, aggregation algorithms can be implemented with any of these topologies [51]. Aggregation algorithms can also be centralized, hierarchical and decentralized. In general an aggregating server in the Fl context ought to orchestrate and synchronize the participating distributed computation nodes and to gather their models in order to synthesize the global model.

2.8.1 Centralized Aggregation

In centralized aggregation, all of the above takes place in a single centralized server where gradients are sent from distributed computing resources. Contributed gradients are combined with the original model and the resulting model is sent to participating computing resources for further training. One downside with central parameter servers is that they tend to favour computing resources with better network conditions relative to the server be that closer proximity or better bandwidth. This also raises concerns in terms of the use of the resulting model after its training [51]. While the training process in Fl provides security and privacy reassurance for the data no guarantees are given for how models are to be used after training of whether they will be kept as trained by the parameter server. Trust is not guaranteed for the model itself in this scenario. Finally such centralized system are lack performance in when nodes lie in distant regions and are single points of failure i.e. when the aggregating server stops working the whole Fl system halts.

"CR" stands for computer resource. ω are the calculated weights and g are gradients.

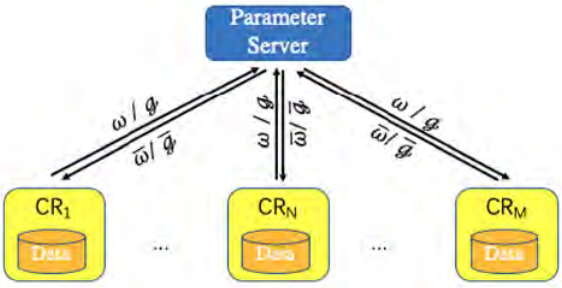

Figure 2.5: Centralized aggregation [51]

2.8.2 Hierarchical Aggregator

Hierarchical aggregation is when the centralized aggregating server is replaced by a hierarchy of parameter servers all of which cooperate in order to accumulate resulting models in a single server [51]. This can be a 2 layer hierarchy architecture in order to decrease latencies and increase performance across distant regions and stability as well as to prevent the aggregating server from being a single point of failure, apart from the final aggregating server at the top of the hierarchy.

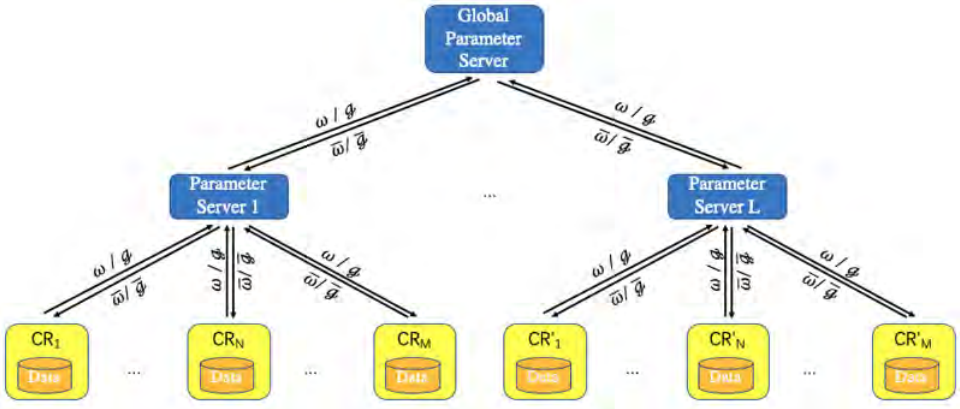

Figure 2.6: Hierarchical aggregation [51]

2.8.3 Decentralized Aggregator

Decentralized aggregation is achieved when participating nodes are connected in a peer-to-peer manner [51]. Here the communication efficiency and convergence speed are determined by the network topology and connectivity. Note that the classical centralized topology can be thought of as a special type of decentralized network in a star like topology. In addition the graph type of the topology can be extended from a star to an exponential graph which can

greatly improve training speed.

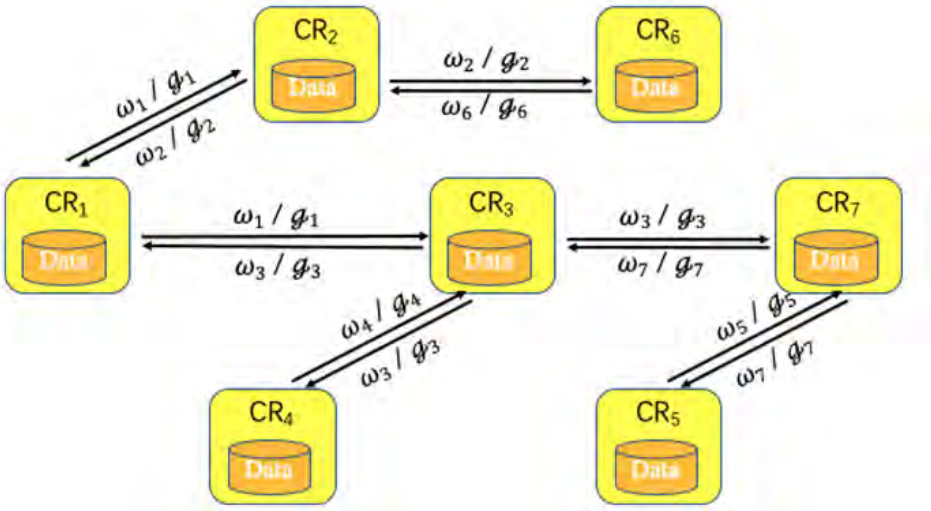

Figure 2.7: Decentralized aggregation [51]

Decentralized aggregation algorithms can be classified as partial or full communication in terms of the amount of neighbors. In full communication algorithms each node computes an averaged model based on all the gradients of the last iteration coming from all its neighbors. In partical communication schemes each node calculates average models based on either a single one or a selected number of neighbors. The selection of neighbors in each iteration can be carried out using a gossip algorithm such as a random selection that protects against poisoning attacts or with a reputation based selection that rewards nodes with a clean record of consistent results.

Centralized architectures exhibit poor performance when training nodes are in sparce geographical locations and they are single points of failure meaning that in the event of failure of the server the whole training process remains ofFline. As no 100% availability exists in such systems, the halting of the training process may be harmless in most cases but catastrophic in mission critical applications. These problems are somewhat addressed when hierarchical architectures are employed but the geographical restrictions may still persist depending on the location of the participating nodes.

Finally in contrast to centralized or hierarchical architectures where the servers or server hierarchies are in a single organization, decentralized nodes by their nature can operate between different organizations and therefore behave more transparently and securely. In centralized or hierarchical architectures the ownership of models is on the hands of single entities or organizations which raises concerns regarding the handling of the models themselves after

training has been completed as well as their deployment in production. Such entities provide no guarantees for the handling of the models nor the validity of the nodes that participated in the training process.

2.9 Challenges in Federated learning

Horizontal Federated learning lacking compared to centralized learning in three district ways. First, the communication costs and resources associated with the sending and receiving of the models as well as the gradients are still a major issue that consumes a lot of bandwidth and slows down the training process [102]. Second, with the combination of the trained models of participating nodes model convergence is harsh and especially poor in terms of speed and number of training required. Third, although the basis of Federated learning is the preservation of users' data privacy, efforts must be focused in ensuring that no users' information is leaked even when sending model updates between the nodes and the aggregating server. Research efforts have been focused on all three of these areas with aim to reduce communication costs with some notable ones being sub-sampling of the clients' model updates and quantization of the trained models [54, 10].

In Horizontal Federated learning, central servers are thought as honest but curious, that is they follow the Federated learning protocol in terms of the learning procedure but simultaneously attempt to infer the original client data used in training by the resulting trained models [14]. As the exchanged model updates of clients may be used to extract information about the raw training data encryption methods are being employed in order to encrypt and protect model gradients against attacks.

Such methods include homomorphic encryption which allows addition and multiplication of the encrypted data and is widely used in learning scenarios [108]. It still comes with drawbacks as instead of reducing the already large communication costs, encrypted messages increase them substantially as they have to be sent back and forth multiple times in order to be decrypted. To address these issues differential privacy is utilized, a method that adds noise in the models of training nodes before sent to the aggregating server which makes it harder to infer the original data.

Chapter 3

Distributed Ledgers

3.1 Introduction

Today's worldwide economy is driven the digital exchange of assets such as currency, stocks and other financial products. A centralized banking system refers to a financial system where a single institution, typically a central bank, has the authority and control over the issuance and distribution of currency, as well as the regulation and supervision of the banking sector. Centralized banking systems control the computer systems responsible for the processing and validation of such monetary transactions and constitute the central authoritative systems that power the economy. Systems like VISA, Mastercard and Paypal are utilized in commercial digital payments while in other cases when banking institutions intent to make cross party or cross-border transactions, protocols like the SWIFT are employed to convey messages. For instance Visa [98], as of 2018 boasts 3.3 billion active cards worldwide with over 65000 transaction messages per second across 200 countries and territories and across 160 currencies.

3.2 Motivation

Such centralized banking systems have long been criticized for their drawbacks. In terms of data privacy, major concerns surround the fact that banking institutions and credit card companies have unlimited and unregulated access to transaction history and data together with sensitive as well as personal information of clients and their credit records. This results in customers not having a say in where their information is located and stored therefore owning none of their personal financial data. Numerous cases of data trading and data breaches have been reported in such centralized institutions that further speak distrust of customers.

As centralized systems are essentially monopolies in the transaction domain, they have frequently been reported to charge high percentage-wise transaction fees that affect ATM withdrawals, wire transfers, and international transactions which makes up for financial burden for low-income individuals or businesses that need to make frequent transactions. Credit card issuers for example are known to charge upwards of 2.5% commission per transaction which makes up their revenue from billions of transactions.

Such systems are also tedious to work with in terms of transaction delay and availability, especially when attempting to make transaction across different banks or across borders. Especially in those cases apart from the extra fees introduced, transactions may be delayed as much as a day. Other reasons for bank transfer delays include different currencies, weekend delays, differences in time zones and finally holidays as transfers are processed only during working hours.

They also have been known to respond poorly in times of crisis, as only a small subset of large financial institutions manage the majority of the economy's liquidity. In times of economic stress banks have been known to shut down withdrawals and transfers or even promptly shut down without notice. This can have severe consequences for individuals, businesses, and the overall economy.

Finally although in recent times almost every bank has e banking infastructure and support, there are still numerous issues regarding operational security and bad security practices that are found in banking applications such as private key leakage, personal information leakage in data breaches that originate from mobile apps or webapps and session hijacking. Few parties have allocated the resources for proper e banking support and adoption in webapps and mobile but even so fraud attempts and scams remain great concerns today.

3.3 Distributed Ledgers Structure

Distributed Ledger Technology (DLT) [29] enables the transactions of digital assets to be registered, shared and synchronized in a decentralized manner which extinguishes the need of a central authority to authorize transactions. DLT has cumulated significant interest as it promises to address problems and deficiencies of traditional centralized ledger systems such as data privacy, security, transparency and accountability. DLT evolves upon traditional distributed systems by combining four key computer science fields such as public key cryptography, data structures and consensus algorithms in a peer-to-peer network enviroment in order evolve existing ledger technologies in terms of decentralization, transparency, openness, immutability, traceability, security and availability among others [3, 29, 4].

While traditional banking uses centralized hierarchical cloud infastructure, databases for storing transactions and client identification in client server neworking schemes, DLTs are the enablers of DeFi (Decentralized Finance) where all banking infrastructure such as ledgers, exchanges and investing are replaced with decentralized equivalents [3, 29, 4]. We have shown the limitations of centralized banking systems and decentralized finance promises to offer a viable alternative. As mentioned DLTs combine joint innovations of these four key computer science domains [40, 35, 69, 97].

- Data structures : Records pending, processed and finalized transactions and implements the backbone of DLTs.
- Public Key cryptography : Provides identification of users and signing of transactions. Each participant in order to record transactions in the Distributed Ledger has a public/private keypair. Signing of transactions and user identification build ownership of data in the Distributed Ledger.
- Distributed peer-to-peer networks : Provides the network protocol that users employ while communicating in the Distibuted Ledger. It attempts to provide scalability of the network, prevent single point of failure and to restrain individuals or user groups to take over the network.
- Consensus protocols : The mechanism which will enable users to agree on the transactions to be accepted (next system state) without being regulated by external third parties.

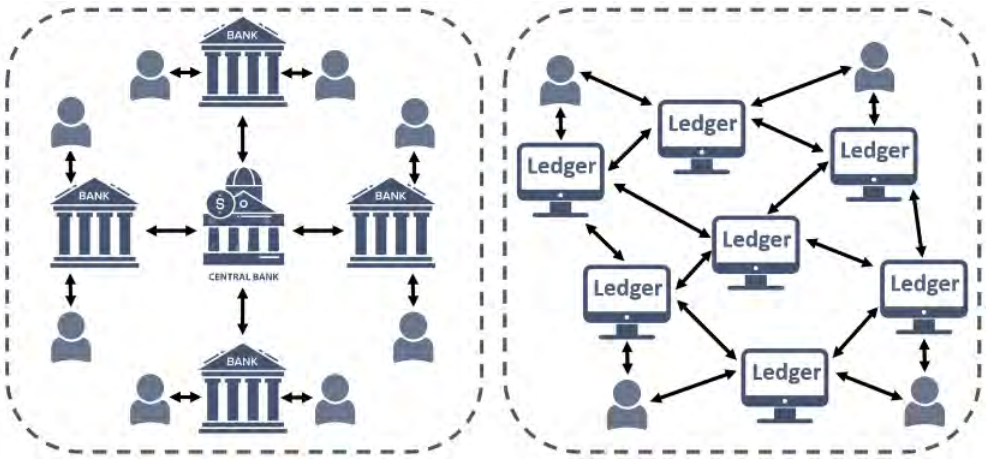

Figure 3.1: Classic banking vs DeFi [40]

3.4 Data Structures

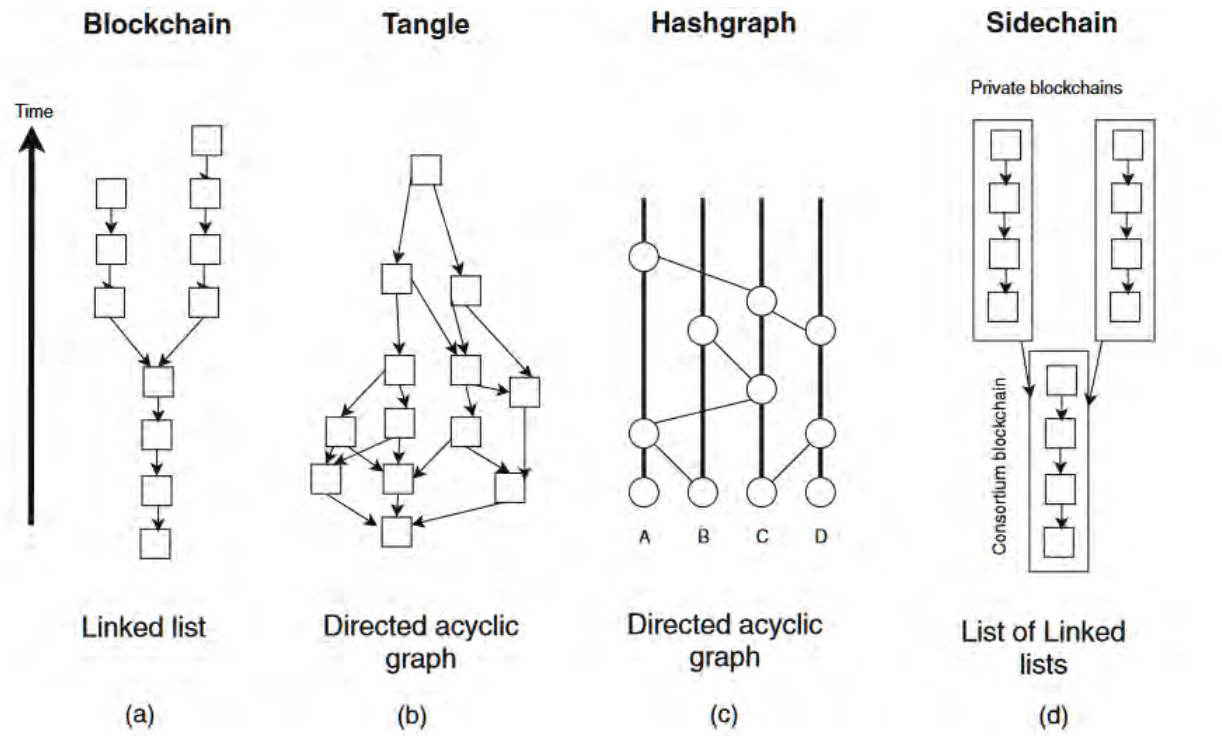

Figure 3.2: Data structure examples as foundations of DLTs [29]

There are various of data structures that DLTs use among linked lists, directed acyclic graphs (DAGs) or trees. There is a difference between the data that the nodes store and the structure that store the nodes themselves. Another important aspect is time as most of the data structures present in DLTs typically aim to stabilize or converge older data entries while remaining flexible for new ones. More specifically in Blockchains the central linked list struc-

ture splits in a procedure called a fork when two or more parties successfully publish a new block while other scaling solutions employ sidechains that store different kinds of transactions and stores sidechain state changes in a central canonical Blockchain.

3.5 Public Key Cryptography

Asymmetric cryptography, commonly referred to as public-key cryptography, is a cryptographic method, that involves a pair of keys consisting of a public key and a private key known as a keypair. Key pairs are generated through carefully crafted mathematical hash functions [69, 97]. In such a public encryption scheme any user can encrypt a message with their private key which can only be can only be deciphered using the corresponding public key of the user. This leads two characteristics that are crucial and essential to DLTs [29, 3], first that users sign their transactions in a uniquely identifiable way and second that public keys can be commonly known in the public domain without risking security. These properties are used to construct ownership of data or monetary assets in such decentralized systems as they can be cryptographically tied to users [4, 40, 35].

3.6 Peer to Peer networks

Peer to peer (P2P) networks are networks whose structure deviates from the traditional client-server model. Such network consists of so called peers i.e., users that are both suppliers and consumers of resources in contrast to the traditional client-server model [29, 3]. Since no hierarchical relations exist all nodes behave both as clients and as servers executing the same protocol [4, 40, 35]. All peers are privileged equally and workloads as well as storage is partitioned among all of them. This is the core difference with classical systems as the discard of a central authority is what ultimately realizes the desired decentralized gonvernance of the protocol [69, 97].

3.7 The need for Consensus

So far we have established that cryptographically idenifiable peers in the p2p network will record transactions in a Linked list or a DAG structure. Does this imply that any peer within the network has the ability to submit proposed transactions, and that all those proposed

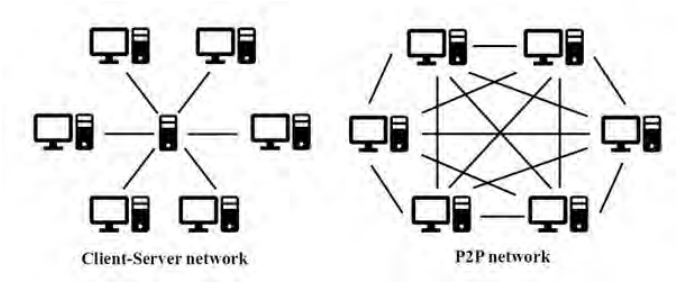

Figure 3.3: Client server model vs peer-to-peer [35]

transactions will be subsequently accepted into the ledger? This orders for a protocol that will select proper transactions among the proposed ones, record them in the data structure and broadcast them in all participating nodes so that there is no misconception about users' assets. In DLTs such a protocol is implemented by employing a consensus algorithm [69, 3, 29].

Consensus algorithms originate from distributed systems where multiple computer nodes collaborate in a peer to peer message passing enviroment [42, 44, 43, 21, 69]. Contrary to other multiprocessing environments such as multi threading, distributed systems nodes do not have a shared memory architecture. Distributed systems nodes can be client server nodes or peer to peer nodes connected via networking. Any distributed system that requires multiple process nodes to maintain a common state boils down to resolving the consensus problem [21, 69].

Use cases of distributed systems include :

- State machine replication i.e., synchronizing replicated state machines ensuring that all state replicas share a consistent view of the system's state.
- electing a leader (e.g., for mutual exclusion)
- fault-tolerance in distributed logging while its sequence remains globally consistent
- ability to commit to a transaction or abort transactions in distributed settings

Distributed ledgers have to maintain copies of the backbone data structure across many participating nodes, so that nodes have access to the latest verified transactions. In every node creation, that is round where transactions are being verified a single verifying node must be selected. Their sequence must also be globally consistent so that no dishonest nodes can alter transaction history. Finally the elected leader must decide on whether to commit on abort on

the proposed transactions which may be monetary transactions or the execution of general purpose code as we will see later on.

3.7.1 Consensus

Let a collection of value proposing processes in a message passing environment and a system which expects only one value. Consensus is the algorithm used to achieve agreement on a particular single value, that is assurance that a single value from the ones proposed is chosen based on the individual votes of each process. Then if a value is chosen, all the participating processes should learn it [42, 44, 43, 21, 69].

Consensus algorithms also assume a synchronization model, which may be either the Synchronous or the Asynchronous model. The former divides time into rounds where values are proposed while the latter does not. DLTs operate on asynchronous messaging [69].

Consensus operates in two phases; the leader election phase, that is the selection of the node that will finalize the value voted and the agreement phase where the proposed value will be finalized by the elected leader and all peers will be notified. Different consensus algorithms employ different leader election algorithms and subsequently based on the algorithm chosen, an agreement algorithm is employed as well.

Consensus algorithms have three safety (validity, agreement, integrity) also abbreviated as VIA and one liveness property (termination) which makes up for the VITA properties [69, 43].

- Only a value that has been proposed my be chosen (Validity)
- Only a single value is chosen (Integrity)
- A process never learns a value is chosen unless its has been chosen (Agreement)
- This algortihm terminates (Termination)

Agreement and Integrity are the core consensus properties that ensure that if the algorithm terminates it will provide a single unanimous results across all nodes while validity is employed in order to rule out trivial solutions and to recognize incorrect protocol outcomes. Finally termination is a requirement as to establish fault tolerance in such systems as faulty nodes will force the protocol to run indefinately.

Consensus algorithms aim to address all of these properties. While this is trivial in non faulty systems in real world scenarios where nodes become faulty it becomes impossible to satisfy all of them, at least not completely. But before diving into that it is important to clarify the definition of faults themselves their classification and why it is important when combined with the VITA properties.

3.7.2 Fault classification

In distributed systems nodes are known to exhibit crash fails, stop and recover faults, as well as Byzantine faults [42, 44, 43, 21, 69].

Crash fails are the simplest ones as participating nodes simply come to a halt and other participating nodes no longer receive messages from them. This may be either to the process itself stopping of its connection failing. There is no distinction between the two in this case.

Crash Recover fails happen when halted processes spontaneously burst into life due to a restart, a late arrival of an already sent message or a message re- transmission. Any case has equivalent results as the newly received messages in the leader will be arrive in an incorrect protocol state.

”Byzantine” refers to arbitrary faults, that is it models the general kind of fault. This includes both crash fails, crash recovers along with any kind of deviation from the standard system protocol, including malfunctions as well as malicious behaviour.

3.7.3 Deterministic Consensus

Consensus protocols are divided into deterministic and non-determinitstic algorithms i.e., algorithms with random element where determinism here can affect the algorithmic steps, either the leader election or the value proposed. Since values proposed cannot be random in any case since else how distributed systems whould be meaningless. The randomized algorithmic elements therefore lie in the leader selection.

Deterministic algorithms include Paxos Raft and PBFT [42, 44, 43].

Consensus among honest and fault-free processes is trivial. This is solved by the two-phase commit protocol. In such a protocol an coordinating node (elected leader) is selected which is assigned with determining the protocol’s result, while the rest of the nodes execute the protocol based solely on the leader’s commands. The leader puts forward a value which

is communicated to all the participating nodes and nodes store the proposed value. In a transaction system this procedure is repeated as long as there are transactions to be processed. The caveat is that in order for this to work all processes must be always honest,always alive and in an environment where communication is flawless.

Faults in this case may be failures in terms of the processes or their communication. When such failures occur the two-phase commit protocol must either wait indefinitely or restart.

3.8 FLP

The central question is, can we achieve consensus in an asynchronous network model?

FLP impossibility [21], also known as Fischer-Lynch-Paterson impossibility, is a theoretical result that states that it is impossible to solve the consensus problem i.e. derive and algorithm that guarantees both safety and liveness deterministically in a message passing asynchronous distributed system in the presence of even a single node that suffers from Crash-Failures.

In other words, FLP impossibility shows that in an asynchronous system, where nodes may fail or have arbitrary (infinite) message delay, there is no consensus algorithm that can always guarantee that all correct nodes agree on the same value within a finite amount of time, even if only one node fails [42, 44, 43, 21, 69]. This means that in an asynchronous system, it is impossible to completely avoid the risk of a consensus failure, where nodes may disagree on the outcome of a consensus even if they follow the protocol correctly.

FLP's proof dictates that in protocols that retain consistency and weak validity in the asynchronous setting there will in any case exist an algorithm run where participating nodes will run indefinitely without outputting a result which therefore violates the liveness property of termination [69, 21]. Immediately it is important to investigate why the FLP result differentiates between deterministic and random based algorithms and this is true because in random based algorithms such non terminating runs have a very small probability of occurring. This implies that runs have a high probability of successful termination.

Randomness or (somewhat) synchronized clocks can be employed to mitigate the FLP impossibility and derive consensus protocols in the asynchronous message passing enviroment [42, 44, 43, 21, 69]. This is the core distinction between consensus algorithms, Classical ones like Paxos Raft PBFT that have existed since Lamport's days and lottery based Con-

sensus algorithms like PoW PoS DPoS etc. Here lottery based refers to the random leader selection process [69].

DLTs have been known to employ any of these algorithms when recording transactions by repeated consensus. Lottery based algorithms have been the core backbone of most Blockchains, i.e. linked list DLTs and were introducted with the emergence of Bitcoin and the first such algorithm Proof of Work [69]. Before talking about proof of Work we will first explain Blockchain's structure and how it stacks up among DLTs.

3.9 Blockchain

Bitcoin [61], introduced in 2009, is a distributed Ledger with a cryptogaphically linked list core and a lottery based consensus algorithm combining the longest chain protocol as well as proof of work [65]. After Bitcoin's success emerged multiple efforts such as Ethereum, Hyperledger, Smart Chain and fast forward to today where Blockchain implementations are custom built to cater for specific applications. Blockchains implement a native coin to the platform, e.g. Bitcoin in the Bitcoin Blockchain and Ether in the Ethrereum Blockchain which is awarded upon block creation to participants in the network or can be purchased using traditional currency [106].

There are three kinds of Blockchain systems depending on the nature of users' priviledges. (i) permissionless (ii) permissioned, and (iii) private. Permissionless Blockchains are public Blockchains that allow any user to operate and submit transactions (e.g., Bitcoin and Ethereum). In permissioned Blockchains network participants are specified and multiple organizations cooperate in order to control and manage Blockchain gonvernace (e.g., Hyperledger. Private Blockchains are single organization managed with such an organization having full control over the network. In terms of our comparison, we mainly consider the properties of permissionless platforms [4, 3, 29].

Blockchain has unique features that distinguish it from other Distributed Ledger Technologies (DLTs). One of its main advantages is the increasing number of implementations, which suggests that the community recognizes its potential across various domains. The existing implementations address different issues and target different application domains, but they are all fundamentally implementations of a distributed ledger.

3.10 Blockchain Structure

Blockchain is a linear log of transactions that achieves repeated consensus for transaction immutability by employing Lottery based consensus protocols. Lottery based protocols attempt to solve the consensus by randomly selecting the round leader either by competitive puzzle solving (PoW) or by a probability proportional to their investment in the platform (PoS). These are the main components of a Blockchain.

- Transactions, which are cryptographically signed pieces of information created by participants and then once accepted are broadcast to the rest of the nodes in the network.
- Blocks, are collections of transactions that are appended to the Blockchain after being validated
- The Blockchain which is a ledger of all the created blocks that make up the network
- The Blockchain relies on cryptographically signed blocks in order to implement block connection and block sequence
- A consensus mechanism is employed to select the blocks which are to be added to the Blockchain.

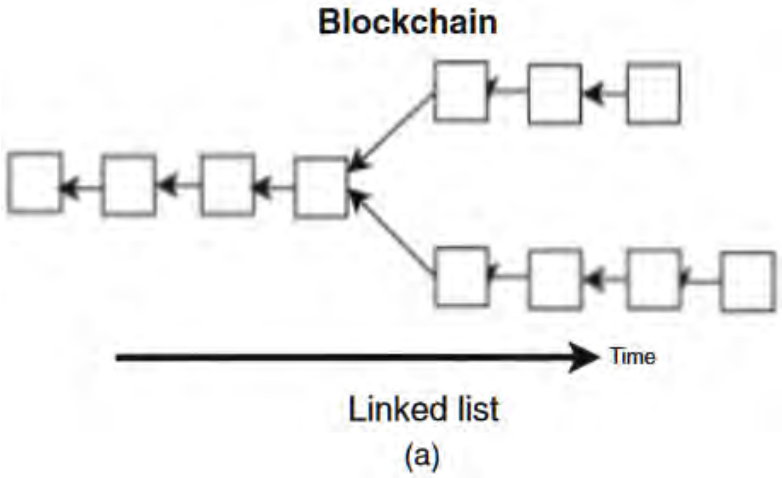

Figure 3.4: Linear log (linked list) of transactions [29]

3.10.1 Blockchain's core data structure

The underlying data structure is a cryptographically linked list also known as a chain. Nodes are identified by hash values which are being used as pointing values to the previous

node. The linked list nodes are also referred to as blocks, hence the name Blockchain. Blocks are comprised of a block header where data about the block are stored and a block body which stores transaction data. In each block header there exists a field which contains the previous block hash which is calculated based on the previous blocks' header. This implies that each block is cryptographically linked to the previous one. Note here that the notion of "previous" requires keeping track of time which is achieved storing timestamps in the block header and using them to compute the blocks' header hash. Finally as time moves forward the chain may split in a process called a fork which essentially means that from the block where the fork was created onward, two chains record transactions and thus continue the Blockchain.

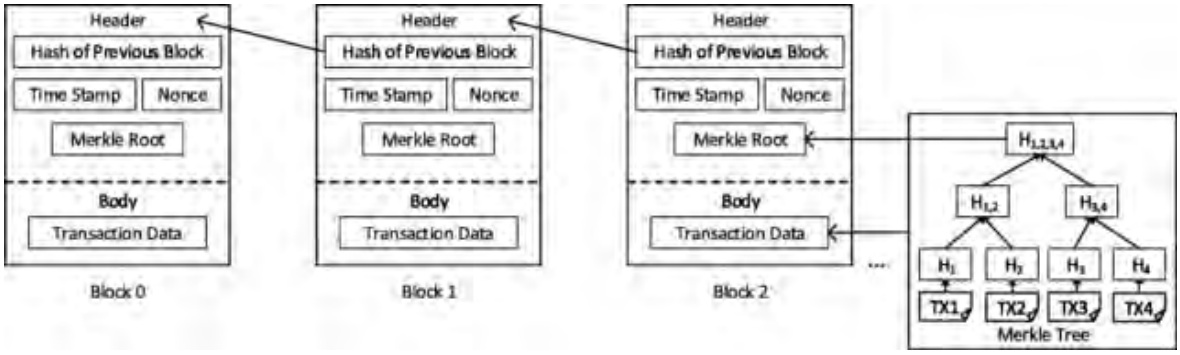

Figure 3.5: High level overview of bitcoin [25]

Bellow we will analyze each component of the Blockchain in-depth starting with individual transactions, block headers, block bodies, timestamps, calculation of block header fields, block creation, Blockchain consensus and fork resolutions using bitcoin as a use case .

3.10.2 Transactions

A transaction refers to the transfer of digital assets or data from one account or entity to another on the network. A transaction typically includes information such as the sender's address, the recipient's address, the amount of digital assets or data being transferred, and transaction fees. Once a transaction is initiated, it is broadcast to the network, validated by nodes in the network, and included in a block. After the block is appended to the Blockchain, the transaction becomes permanent and irreversible, creating an immutable record of the transaction history on the Blockchain. In this system, electronic currency is transferred between users when payees digitally sign a hash of the previous transaction and the public key of the intended recipient, subsequently appending it to the end of the chain.

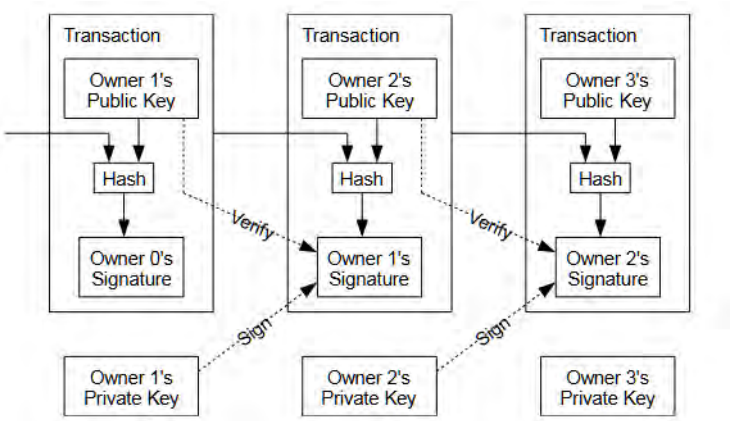

Figure 3.6: Transactions in bitcoin [61]

Transactions are therefore verified by the owner's public key and signed by the owners private key. Each transaction uses the owners' public key to create a hash of a transaction which will be used when blocks are created so that transactions can be verified later.

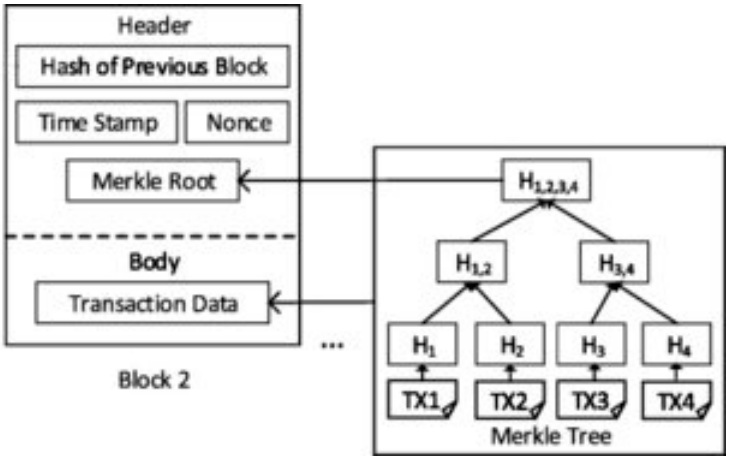

Figure 3.7: Transactions merkle tree [61]

When blocks are being proposed, transactions are gathered from a pool of pending transactions and placed in the proposed block's body. There all hashes of the proposed transactions are gathered and used to construct a merkle tree whose root is identification of the transactions in the Blockchain.

A hash tree, commonly referred to as a Merkle tree, is a cryptographic data structure employed in the field of cryptography used for data integrity verification, that is to securely verify the integrity of large data sets.

A Merkle tree is constructed by recursively hashing pairs of data, resulting in a binary tree structure where each leaf node represents a single piece of data and each non-leaf node

represents the hash of its child nodes. Starting from the leaves which store hashes, tree nodes accumulate their children hashes and obtain the resulting hash which is the hash of the current node up towards the root. The root node of the tree represents the top-level hash, also known as the Merkle root.

Merkle trees are commonly used to allow for efficient and secure verification of transactions. In this context, each leaf node represents a transaction, and the Merkle root is included in the block header. The Merkle root can be used to verify the integrity of the entire data set, as any changes to the data will result in a different Merkle root and is used to identify each blocks' transactions.

3.10.3 Blockchain Header

The block header without transactions would be approximately 80 bytes. Starting with the version of the protocol then continuing onward with the hash of the previous block, the merkle root of the transaction data, the timestamp, the target and finally the nonce 3.8.

The block header's fields along with 48 bytes of padding totaling 128 bytes are split into 2 512-bit segments, which are hashed using the SHA-256 algorithm, concatenated along with padding and hashed once more to create the final 256 bit block hash. This serves as identification of the present block and will be stored in the ”previous block hash” field of the next block as to show their continuity 3.8.

The ”target” and ”nonce” are very important fields which are essential components of the mining problem, the underlying mechanism for leader election in Blockchain's consensus algorithm. This is true for the ”time” field of the struct as it is used to maintain the sequence and order of blocks which is important for the longest chain property, also a component of Blockchain's consensus algorithm.

3.10.4 Mainaining order of transactions

Here Blockchain's solution starts by employing a timestamp server. The timestamp server works by taking the block hash and publishing the hash value. This timestamp proves that at the time of its creation data existed at the time in order to participate in the hash calculation. The timestamps themselves also form a chain, as each timestamp's hash requires the previous hash's value in order to be calculated 3.11.

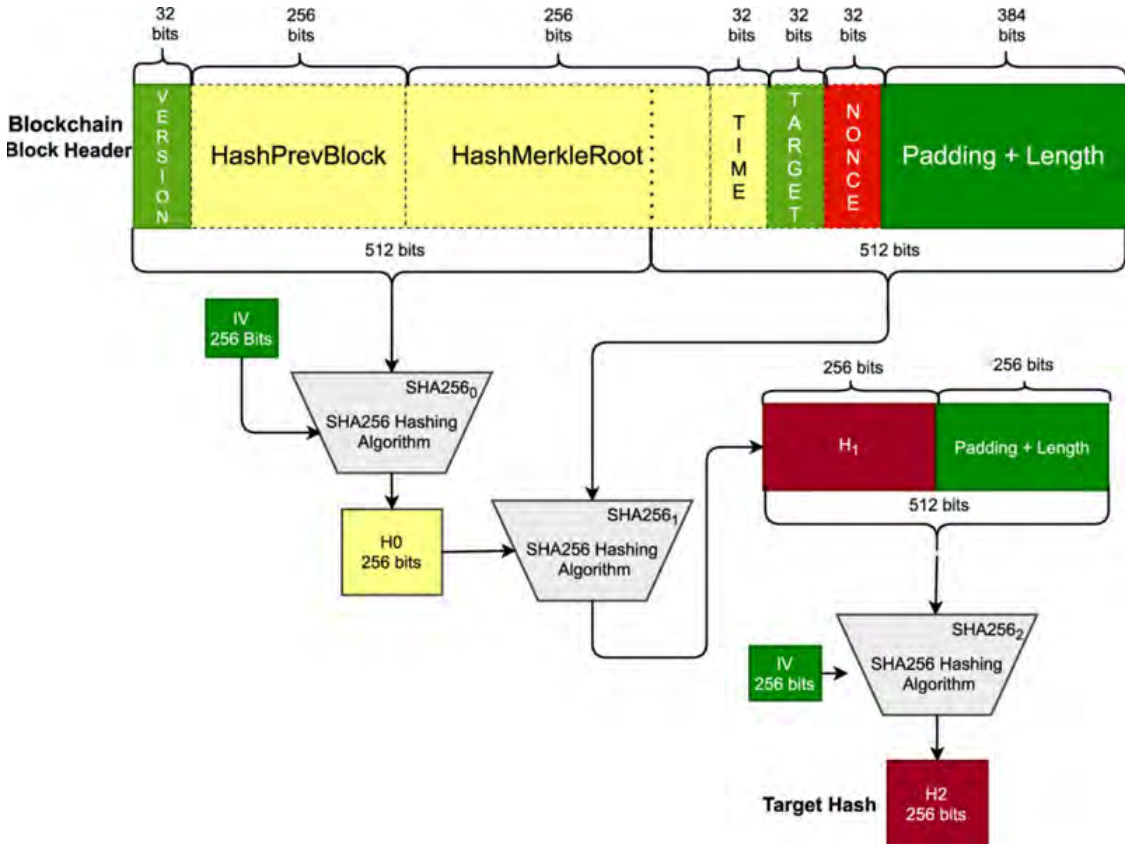

Figure 3.8: Block header hash [97]

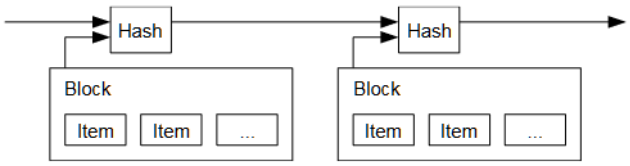

Figure 3.9: Transaction Block hashing [61]

3.11 Block Creation

So far we have established that Blockchain implements a DLT based on peers that store transactions in blocks which are recorded in a cryptographically linked list. We also explored block structure. Whats left is selection of the leader who will authorise the next block and the process of block creation. This is addressed by the Blockchains consensus algorithm, proof of work with longest chain. Proof of work is a competitive protocol and together with the longest chain property select the new block to be placed.

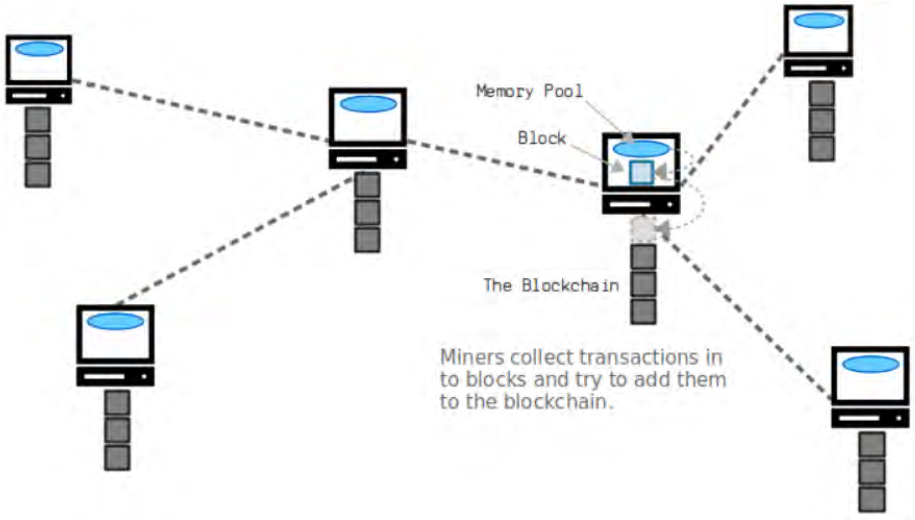

Figure 3.10: Miner [99]

3.12 Proof of work - the mining puzzle

We described the hashing process of the Blockchain header in Figure 3.8. In order to introduce a new block to the chain two things must hold true.

- The first node to solve the mining puzzle will publish its new block
- The new block must be added to the longest chain

The mining puzzle is a cryptographic puzzle where nodes seek to find a correct number (nonce in the present context) that makes the resulting hash contain zeros in a number of most significant places. Here this hash is the block hash.

version	02000000
previous block hash (reversed)	17975b97c18ed1f7e255adf297599b55 330edab87803c8170100000000000000
Merkle root (reversed)	8a97295a2747b4f1a0b3948df3990344 c0e19fa6b2b92b3a19c8e6badc141787
timestamp	358b0553
bits	535f0119
nonce	48750833
transaction count	63

Block hash
0000000000000000 e067a478024addfe cdc93628978aa52d 91fabd4292982a50

Figure 3.11: Transaction Block hashing [68]

As we have seen in Figure 3.8 the nonce is itself concatenated to the other block header fields in order to formulate the double hash input. When competing nodes attempt to formu-

late blocks they gather transactions, and brute force nonces in order for the resulting hash to start to be less than the value in the target field.

Among competing nodes, the first one to collect a valid set of transactions and to find the appropriate nonce so that the block hash is less than the target get to be the elected leader equivalent and will record their block in the Blockchain so long as the block's previous hash value refers to the final block of the longest chain 3.10.

The longest chain rule is key here as it resolves the forks of the Blockchain 3.12. Forks occur when two or more parties manage to simultaneously succesfully mine a correct block and place it in the Blockchain. In this instance the Blockchain splits in two and contains two transaction histories. However, as future blocks continue the longest chain and most nodes build on one side of the fork rather than the other they get resolved by the network itself.

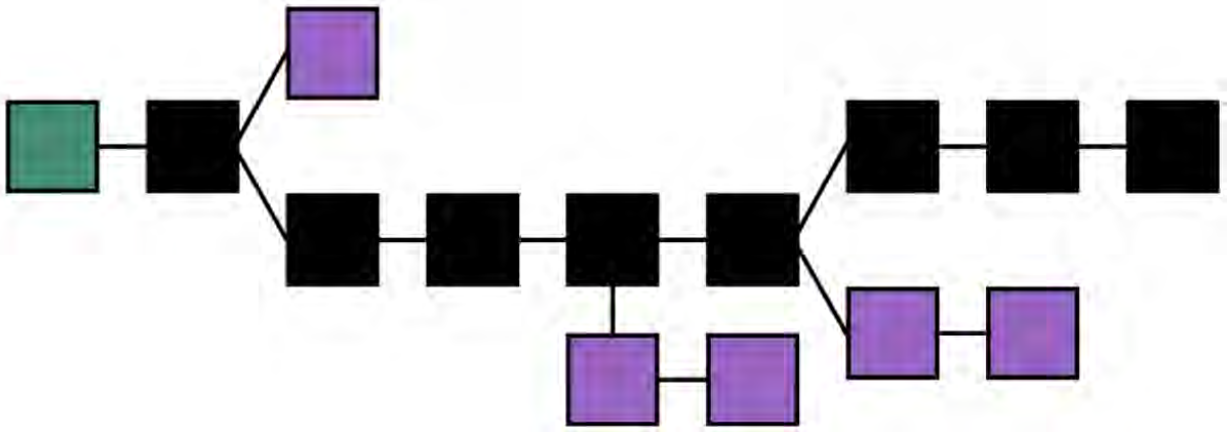

Figure 3.12: Forks in Blockchain [96]

With these two mechanisms Blockchains manage to circumvent the FLP impossibility in order to achieve probabilisitc consensus in this asynchronous setting. Leader election here is random or lottery based as the competition in order to solve the mining puzzle is practically random itself. This also solves the non termination issue as PoW is a probabilistic algorithm but raises the question on what happens when the low probability of PoW failing occurs.

PoW failure does not in the termination property but rather in the aggreement property. Proof of Work definitely terminates in each round of block creation but there is a certain low probability that two nodes will not be in agreement of the chain state. This of course is what happens when forks occur and that probability diminishes to zero as time moves on and forks are resolved. This process of blocks becoming immutable at time marches is called finality. Finality in bitcoin is about 6 blocks and takes about an hour.

3.13 Bitcoin Weaknesses

One of the primary criticisms of Bitcoin is its scalability [69, 100]. The current Bitcoin network can handle only a limited number of transactions per second which can result in delays and high transaction fees during periods of high network traffic. The limited block size and the time required for block confirmation result in slower transaction processing times, making it less suitable for high-volume transactions. Secondly, the energy consumption caused by Bitcoin mining has provoked significant criticism due to its environmental impact. The Proof of Work consensus mechanism employed by Bitcoin requires substantial computational power due to the repeated hashing calculations required, leading to significant energy consumption [106].

3.14 Proof of Stake

Scalability and energy concerns lead to the search for suitable alternatives to Proof of work [106]. Efforts focused on solutions that attempted to solve the consensus problem in the asynchronous setting without the computational and energy requirements of competitive mining as well as also reducing delays and increasing theoretical transaction throughput 3.1.

Proof of State starts with the same basis as Proof of Work, meaning that it also assumes a Blockchain data structure with blocks that store transactions, a peer-to-peer network protocol and cryptographic verification of transactions and blocks [69]. The differentiation between PoW and PoS is in leader selection, where PoS discards cryptographic puzzles.

Validators replace traditional miners in PoW and are chosen by the protocol to validate transactions and create new blocks randomly in each round by a probability value assigned to thembased on their ownership and willingness to ”stake” a certain amount of cryptocurrency as collateral in the platform. Note here that while not all Blockchains house a native coin, PoS requires a currency enabled Blockchain to work. Although in Proof of Work is a public consensus protocol, PoS implements a permmissioned consensus algorithm as the protocol starts with a well defined set of staking validators [69].

So Proof of Stake skips latencies stemming from PoW’s competitive nature and is able to reach must faster block creation times, a much desired feature of modern Blockchains. While bitcoin boasts 15 TPS with 60-minute finality times, the PoS based Blockchains can already reach 2000 (Solana) or even 4500 (Avalance) with finality times in the seconds.

Performance and Energy comparison					
	Bitcoin	Ethereum	Cardano	Solana	Avalance
Transaction speed	5 TPS	15-20 TPS	2 TPS	2000 TPS	4500 TPS
Transaction finality	60 min	14 min	10-60 min	21-46 s	2-3 s
Energy Efficiency	1,449 kWh	238 kWh	0.5 KWh	0.00051 KWh	0.00013 KWh

Table 3.1: Public Blockchains

Finally alternatives to PoS have been proposed with each Blockchain implementing a slightly different protocol such as DPoS PoET PoH.

3.15 Survey of most popular Blockchains

Figure 3.13: Public Blockchain collection [27]

Since Ethereum [100, 49] introduced Smart Contract support, the entire focus of new Blockchain ecosystems has shifted to smart contract support as well as the implementation

and support of new features such as new methods for on-chain storage, privacy and security enhancements all towards the implementation of fully functional decentralized cloud systems. Furthermore, as various chains are being contstantly introduced efforts are focused towards interoperability with cross-chain platforms such as Cosmos and Polkadot that enable cross-chain transactions and perhaps more importantly for our purposes cross-chain smart contracts 3.13.

3.16 Challenges in Distributed Ledgers

Besides Blockchain, there are DLT implementations utilizing various data structures and consensus mechanisms. Blockchain remains highly popular but even Blockchain variants continue to multiply as new efforts try to address problems and deficiencies that trace back since the adoption of bitcoin begun [1].

There are several challenges associated with the adoption and implementation of DLT. Firstly, interoperability between different DLT platforms remains a significant challenge, as there is a lack of standardization and uniformity in DLT protocols [69]. This limits the ability to share data across different platforms, hindering the potential for broader adoption and network effects. Secondly, scalability remains a challenge for many DLT platforms, as they face limitations in processing capacity and transaction speed, particularly in high-volume scenarios [100, 49].

Additionally, the security of DLT platforms is a critical concern, as vulnerabilities and attacks can result in significant financial losses and undermine user confidence in the technology. Another challenge is the regulatory environment, which is still evolving and can vary significantly across jurisdictions, creating legal and compliance challenges for businesses seeking to adopt DLT solutions. Finally, there are challenges related to governance and decision-making within DLT networks, as stakeholders may have differing incentives and goals, requiring consensus-building mechanisms to ensure the integrity and functionality of the network. Despite such challenges, current research and development efforts are exploring solutions to address these issues and promote the wider adoption and success of DLT .

Such developments have resulted in perhaps the most important feature of modern DLTs, that is Smart Contractss which will be discussed in the next chapter.

Chapter 4

Smart Contracts

4.1 Introduction

The commercial success of bitcoin and its mass adoption incentivised other pioneers and organisations to deploy public Blockchain DLTs. Among these efforts was the Ethereum Blockchain which apart from being an alternative to bitcoin also came with significant technological advances. Bitcoin is a Blockchain DLT but a simple one at that as only ledger operations are supported among its users who can only send and receive the platforms native currency, that its bitcoin. Ethereum [100] pioneered with the introduction of the EVM (Ethereum Virtual Machine) when Vitalik Buterin and the Ethereum Team realized that the simple transactions that bitcoin supports can be extended to a fully fledged instruction set for operations in the Ethereum Blockchain.

The Ethereum Virtual Machine (EVM) is a runtime environment for executing such instructions in bytecode or smart contracts on the Ethereum Blockchain. It is a crucial component of Ethereum, enabling the execution of decentralized applications (dApps) and the implementation of complex logic. Since the introduction of Ethereum most Blockchains implement a Virtual Machine which in turn supports code execution.

Ethereum's focus has been on providing a programmable Blockchain platform that supports the execution of smart contracts, enabling developers to build decentralized applications with diverse functionalities. DApps have seen tremendous commercial success and have driven innovations in multiple industries who seek to migrate applications and services in Blockchains instead of traditional centralized or cloud applications.

4.2 Smart Contracts Definition

Smart contracts are programs that run on a Distributed Ledger and extend its use. They are essentially scripts that are executed in a decentralized manner on Blockchains and other DLTs and are immutable much like transactions in a Blockchain [26, 35]. They also undergoes cryptographic verification to ensure their integrity and reliability .

Smart contracts main feature is that much like Blockchains, they operate in a peer-to-peer manner without the need for a central authority to regulate code execution [66, 24, 40]. They enforce and execute the code when specified conditions are met, all without being explicitly activated by a third party. This along with the fact that they are operating on a fault tolerant distributed systems means that they provide great service availability, although sometimes at the cost of execution speed.

4.3 The Ethereum virtual machine

The Ethereum Virtual Machine (EVM) is a stack-based virtual machine, that is, a stack based data structure wihch uses a set of opcodes to execute instructions beyond monetary transactions in the Ethereum Blockchain [100, 49].

Smart contracts on Ethereum are written in high-level programming languages like Solidity and are compiled into bytecode that the EVM can understand.

When a transaction involving a smart contract is initiated, the EVM receives the bytecode and begins executing it. Each opcode represents a specific operation, such as arithmetic calculations or storage manipulation. The EVM executes the opcodes one by one, updating the program's state and modifying the Blockchain as necessary.

Much like transactions though, instructions on the EVM are executed upon block creation. Instruction executions are proposed to the network in the same fashion as transactions which are then gathered in order to create a block and then executed when the current leader election places that block in the Blockchain.

The EVM provides a deterministic environment, meaning that given the same inputs and state, the execution of a smart contract will always produce the same output. This determinism is crucial for maintaining consensus across all Ethereum nodes and ensuring that the Blockchain remains immutable and trustworthy.

4.4 Blockchains with smart contracts

There are multiple implementations of smart contracts as new Blockchains are constantly implemented and introduced. Besides Ethereum, Binance Smart Chain [13] as well as Avalance [16] are both examples of Blockchain platforms compatible with the Ethereum Virtual Machine (EVM) and the Solidity language.

Hyperledger Fabric [2], an enterprise level permissioned Blockchain which focuses on meeting the specific requirements of enterprises, offering features such as modular architecture, permissioned networks, and fine-grained access control. It aims to provide a flexible and scalable infrastructure for building and deploying Blockchain-based solutions within business environments. It provides extensive smart contract support for the permissioned setting.

Tezos [22] is another Blockchain which utilizes on chain administration where participants in the network can propose changes to the protocol which are reviewed by stakeholders. It supports smart contracts in its native language Michelson.

Eos [23] is another Blockchain with smart contracts using the C++ programming language which gained popularity due to its high transaction throughput and low transaction cost.

Stellar [57] is a Blockchain focused on transaction speed and cost-efficiency which is shown by the fact that the platform only supports basic smart contracts releated to monetary functions therefore not being capable of dApp support.

Solana [104] is a high-performance public Blockchain platform designed for decentralized applications and crypto-currencies. It provides fast transaction processing and low fees, making it suitable for applications requiring high throughput. Solana deviates from the EVM while being a public Blockchain and supports smart contracts using the Rust language.

There are other examples of smart contract supporting Blockchain platforms but the key takeaway here is the diversity of the ecosystems proposed. Multiple paradigms and programming languages are present with implementation being custom to each network.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.7;
3
4  contract MyContract {
5      uint256 public myVariable;
6
7      constructor(uint256 initialValue) {    infinite gas 76600 gas
8          myVariable = initialValue;
9      }
```

Figure 4.1: Solidity Smart contract code example

4.5 Deployment and Execution

Smart contracts are written in each chains' supported programming language and then compiled into the supported bytecode of the Blockchain. Then smart contracts are deployed in the platform in a series of transactions in order for the smart contract code to be in every peer node ready to be executed [26, 66].

First step in execution (step 0) involves the initialization of smart contracts. There are multiple ways to interface with each Blockchain network, as smart contracts may handle or process data coming from external systems. Ethereum SDK [100] and the Hyperledger Fabric SDK [2] support building applications with REST API support.

External interfacing applications submit their requests for smart contract functionality in the underlying Blockchain (Step 1). Here when such transactions are being requested sanity checks are performed on whether there are runtime errors associated with the requested code (Step 2). Then authentication and legitimacy checks are performed in order to investigate whether such actions were invoked by members of the network by cross referencing their digital signature [26, 66].

If the node determines that the transaction (Step 3) as a legitimate one, the smart contract code is executed in order to produce a transaction (Step 4). The transasction is processed and appended to the block being processed and the node mines the current block (Step 5). The successfully mined block is then added to the Blockchain (Step 6) and partitipaiting nodes reach agreement as to whether it will be the new block to be appended to the Blockchain (Step 7) following the consensus protocol. If block creation is successful, it is placed on the chain (Step 8) and now the effects of the smart contract are broadcasted into all nodes [26, 66].

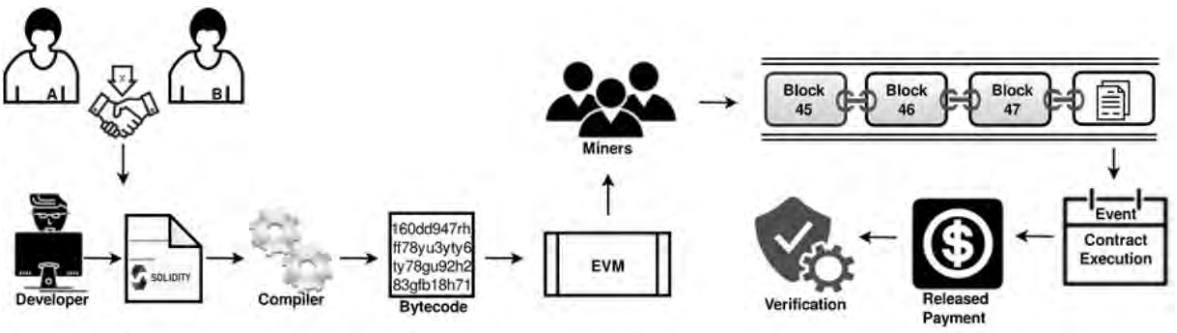

Figure 4.2: Smart Contract Execution Cycle [66]

4.6 Smart Contract Strengths

By eliminating the need for a centralized third party, smart contracts ensure uninterrupted service and facilitating peer-to-peer execution. By autonomously following predefined conditions, smart contracts ensure accurate operations, free from human error or bias as well as guaranteed service avaliability [26, 66]. Consequently, smart contracts are a promising alternative for applications that require trustless solutions without intermediaries .

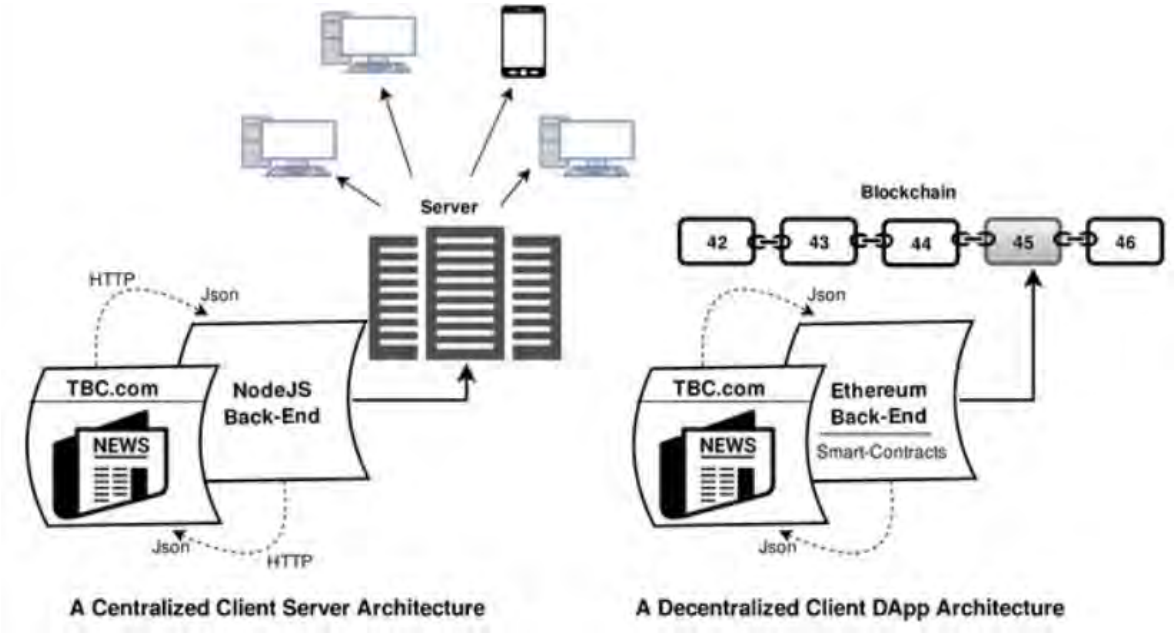

Figure 4.3: Centralized Client-Server vs dApp Architecture[66]

4.6.1 Tamper-proofness and Code immutability

As the integrity of transaction records in a distributed ledger is ensured through digital signatures and smart contract deployment and operation relies on transaction, smart contracts are rendered immutable and tamper-proof both in terms of their code as well as their execution. Individual alteration of transactions is impossible, as they are immutable once placed in the Blockchain. Thus the smart contract code deployed on the Blockchain and its execution results are also immutable, making it tamper-evident. Finally while tampered smart contracts cannot be executed, they can be updated if necessary, with the agreement of the Blockchain network's nodes.

4.6.2 Transparency

Transparency is a feature of major significance that Blockchains provide to smart contracts. The transparency of smart contracts has two aspects. Firstly, the code written in smart contracts is publicly visible to all participating nodes in the network. Secondly, the collection of transactions that contain smart contract code execution results are also transparent to the public. As a result, participants within the Blockchain network can trust the code and the results of code execution. This concept is very powerfull as it ensures that auditability for all code run in dApps which itself is a very desirable feature not found in traditional cloud applications which are closed source.

4.7 Challenges

One key challenge is the issue of security vulnerabilities [26, 66, 12]. Due to their complexity and the potential for coding errors, smart contracts can be susceptible to exploitation by malicious actors. Even small mistakes in the code can lead to significant financial losses. Additionally, the immutability of smart contracts poses a challenge when errors are discovered or updates are required. Unlike traditional software, once a smart contract is deployed on the Blockchain, it becomes difficult to modify or correct without the consensus of the network participants. Furthermore, the scalability of smart contracts is an ongoing concern. As more transactions and applications are added to the Blockchain, the increased computational and storage demands can hinder the efficient execution of smart contracts. These challenges highlight the need for thorough code auditing, robust security practices, and innovative solutions to address the limitations of smart contracts in order to realize their full potential.

The increasing adoption of dApps has shifted the focus of Blockchains as transaction ledgers to them as smart contract platforms. Smart contract have reached the point of requiring a significant portion of the platforms transaction both in terms of throughput as well as processing. This has lead to developers seeking faster platform for contract development and deployment that will be able to handle such workloads.

Chapter 5

Solana platform

5.1 Introduction

As Blockchains evolve as an ecosystem, new platforms are being introduced to mitigate challenges that Bitcoin and Ethereum have faced related to their energy consumption, slow confirmation times, and limitations in scaling. Since Bitcoin, two important paradigm shifts have emerged in the public Blockchain world. The first major paradigm shift is the adoption of Smart Contract support in most of the Blockchains that were introduced after the Smart Contract pioneer Ethereum and the second one was the shift from PoW based to PoS based Blockchains. Almost all modern public Blockchains employ a flavour of Proof of Stake for their consensus, while Smart Contract support has been a prominant feature in all types of Blockchains.

Still effors are continued in order to reduce transaction and finality times even further. As such times are being reduced, transaction throughput is increased which results in higher performance and reliability in Smart Contract execution. This has proven to be essential as Smart Contracts have evolved to the point of requiring substantial amount of code execution. Simultaneously new platforms aim to compete against traditional centralized ledger systems such as Visa.

Simultanoursly Smart Contract development requires modern tools and flexibility for developers. New tools as well as techniques are constantly being developed with dApp adoption rising and expanding in new segments. DApps utilize as much throughput they can leverage in a platform so that their performance is comparable to traditional cloud applications.

5.2 Solana structure and components

Solana is a public Blockchain introduced in 2017 [104]. Solana evolves upon the established Proof of Stake algorihm by realizing its consensus protocol combined with Proof of History (PoH) [104, 103, 105] which is a proof of verifying time deltas between events that is used to maintain passage of time and thus order. PoH greatly reduces messaging overhead in a Blockchain resulting in second level finality times in the Solana Blockchain.

The Solana Blockchain platform presents a compelling solution to the scalability and speed challenges faced by many Blockchain networks. Its innovative architecture, incorporating Proof of History, and sharding, enables Solana to achieve high throughput, low latency, and cost-effective transactions [104, 103, 105]. It also provides low overhead Smart Contract support by the use of the Rust programming language. These features position Solana as a promising Blockchain platform for applications requiring scalability and speed, such as decentralized finance, gaming, and decentralized exchanges [104, 103, 105]. As Solana continues to evolve and gain adoption, its impact on the Blockchain industry is likely to be significant, fostering innovation and enabling new possibilities for decentralized applications.

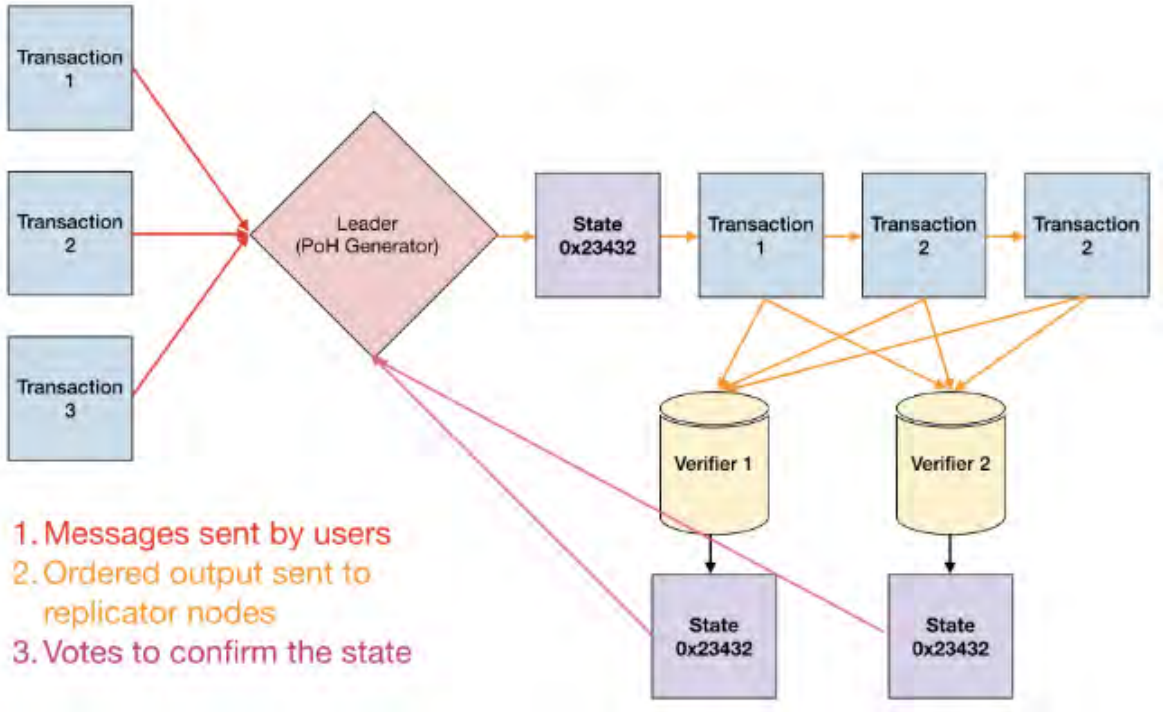

Figure 5.1: Solana structure [104]

Existing public Blockchains currently do not have a reliable time mechanism or make assumptions about participants' ability to maintain accurate time. Each node typically relies

on its own local clock, unaware of the timekeeping of other participants in the network. This lack of a trusted time source introduces uncertainty when using message timestamps to accept or reject messages since there is no guarantee that all participants will make the same decision. The Proof of History (PoH) introduced here aims to establish a ledger that provides verifiable time passage, including the duration between events and the order of messages [103]. It is expected that each and every single node in the network be able to rely on the recorded time passage in the ledger without relying on trust in other participants.

At any given moment, a system node is designated as the Leader, responsible for generating the sequence known as Proof of History [104, 103, 105]. This sequence ensures global read consistency in the network and establishes a verifiable passage of time. The Leader organizes user messages in a specific order that optimizes processing efficiency for other nodes, thereby maximizing throughput. It executes transactions based on the current state stored in RAM and shares the transactions along with a signature of the final state with replication nodes called Verifiers. Verifiers [104, 103, 105] independently execute the same transactions using their own copies of the state and publish their computed signatures as confirmations. These published confirmations serve as votes for the consensus algorithm.

5.3 Proof of History

Proof of History [104, 103, 105] is a computational sequence that enables cryptographic verification of time passage between two events. It employs a cryptographically secure function designed in such a way that the output cannot be predicted from the input and must be fully executed to generate the output. The function runs sequentially on a single core, using the previous output as the current input, periodically recording the current output and the number of iterations. External computers can then parallelly recompute and verify the output by checking each segment of the sequence on separate cores. To include data in the sequence, the data itself or a hash of the data can be appended to the function's state. By recording the state, index, and data as they are appended to the sequence, a timestamp is established, guaranteeing that the data was created before the subsequent hash in the sequence. This design also supports horizontal scaling for participants as multiple generators can synchronize by merging their states into each other's sequences.

The operational concept of the system is as follows: utilizing a collision resistant crypto-

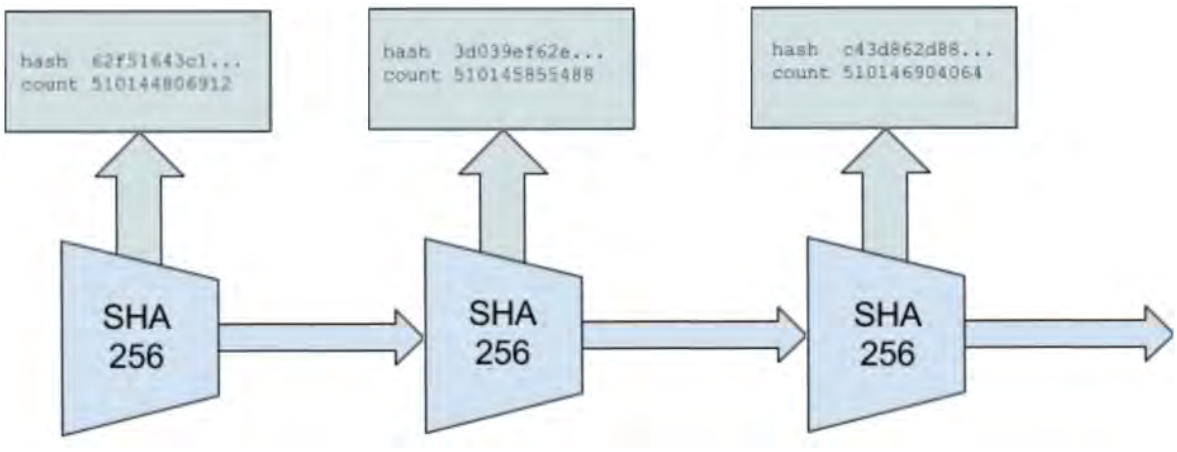

Figure 5.2: Proof of History [104]

graphic hash function (such as sha256 or ripemd) with the property that its output cannot be determined without executing the function, initiate the function from a random initial value. Take the output generated by the function and feed it back as the input for the subsequent execution of the same function [103]. Maintain a record of the count of function calls and the output produced at each call. Now it is only required to for a small subset of hashes and indices to be published in each interval.

Assuming the selected hash function is resistant to collisions, the series of hashes can only be computed sequentially by a single thread on a computer. This is as there is no way to predict the hash value at a specific index, such as index 300, without actually executing the algorithm starting from the initial value 300 times. Therefore, based on the properties of this data structure, we can conclude that real-time has elapsed between index 0 and index 300.

In the example in Figure 5.2, hash 62f51643c1 was outputed on execution number 510144806912 and hash c43d862d88 was outputed on execution number 510146904064. By trusting the preciously defined properties of the PoH algorithm, we can deduce between count 510144806912 and count 510146904064 actual real time has elapsed.

Note that the accurate amount of time elapsed is of no concern here. Essentially the repeated hash calculation constructs a notion of time by event sequencing. This technique is meant to replace the timestamp block header field hash employed in the bitcoin Blockchain in order to maintain event sequencing.

The series of hashes can be utilized to indicate that a certain piece of data was generated prior to a specific hash index [103]. This is achieved by employing a 'combine' function that merges the data with the current hash at the present index. The data can be represented as

a unique cryptographic hash of any event-related information. The combine function can be a straightforward append operation or any other operation that ensures collision resistance. Subsequently, the next generated hash serves as a timestamp for the data since it could have only been generated after the particular data was inserted [103].

5.4 PoH sequence instance with events

Let a hash sequence instance in the PoH setting. We start with index 1 and jump to index 200 and then 300 5.3.

PoH Sequence

Index	Operation	Output Hash
1	sha256("any random starting value")	hash1
200	sha256(hash199)	hash200
300	sha256(hash299)	hash300

Figure 5.3: Proof of History hash sequence [104]

When PoH records some event like for example taking a photograph, or the creation of any type of digital data such as a transaction then its hash can be appended to the hash of the previous hash of the sequence in order for the next hash to contain the fingerprint about digital data.

PoH Sequence With Data

Index	Operation	Output Hash
1	sha256("any random starting value")	hash1
200	sha256(hash199)	hash200
300	sha256(hash299)	hash300
336	sha256(append(hash335, photograph_sha256))	hash336

Figure 5.4: Proof of History hash sequence with event [104]

Hash336 is generated by combining the binary data of hash335 and the sha256 of the photograph 5.4. The sequence output includes the index and the sha256 of the photograph, allowing anyone who verifies the sequence to reproduce this modification. As the initial process remains sequential, we can deduce that entries added to the sequence must have occurred before the computation of the subsequent hashed value in the future.

POH Sequence

Index	Operation	Output Hash
1	sha256(”any random starting value”)	hash1
200	sha256(hash199)	hash200
300	sha256(hash299)	hash300
336	sha256(append(hash335, photograph1_sha256))	hash336
400	sha256(hash399)	hash400
500	sha256(hash499)	hash500
600	sha256(append(hash599, photograph2_sha256))	hash600
700	sha256(hash699)	hash700

Figure 5.5: Proof of History hash sequence with event [104]

In the sequence depicted in Figure 5.5, photograph2 precedes hash600, and photograph1 precedes hash336. When data is inserted into the sequence of hashes, it alters all subsequent values in the sequence. As long as the chosen hash function is resistant to collisions and the data is appended, it should be computationally infeasible to pre-compute any future sequences based on prior knowledge of the data that will be included in the sequence [103].

In the illustration provided in Figure 5.6, the input ”cfd40df8...” was added to the Proof of History sequence. It was inserted at count 510145855488, and the state at that point was ”3d039eef3.” This modification to the sequence affects all subsequent generated hashes, as indicated by the change in color in the figure.

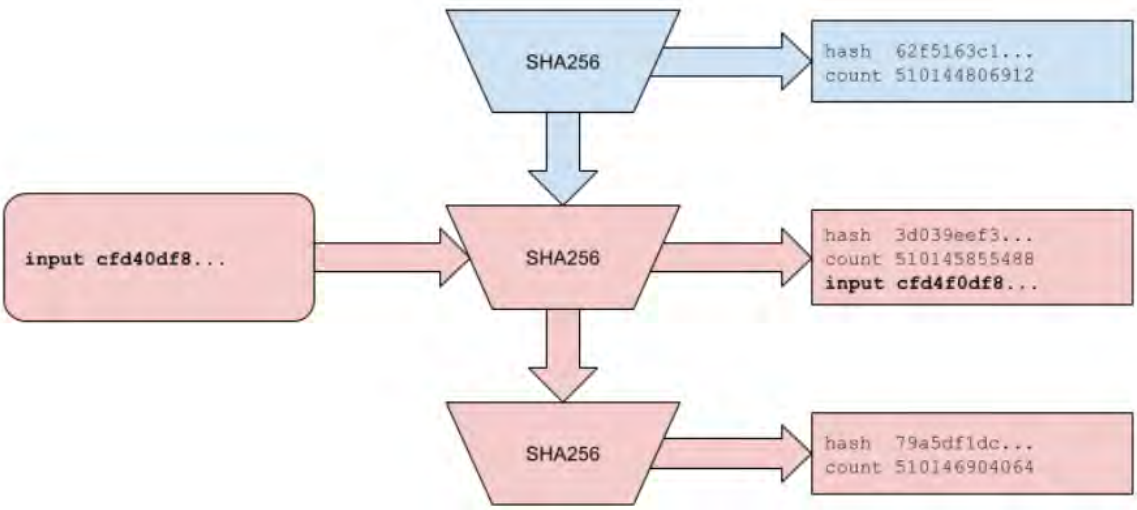

Figure 5.6: Proof of History with data entry [104]

This results in a publicly verifiable sequence where observing nodes can determine the ordering of events and by the count of hashes between events estimate the time elapsed.

5.5 Verification

All this effort to devise such an intricate mechanism for event sequence proof was driven by its ease of verification. The protocol discussed above is highly parallelizable in nature which means that GPUs can be employed in the verficiation process accelerating the verification procedure and minimizing execution time in verfying validators [104].

When provided with a certain number of cores, such as a modern GPU with 4000 cores, the verifier has the capability to divide the sequence of hashes and their corresponding indexes into 4000 segments. The verifier can then simultaneously validate each segment, ensuring the correctness of the sequence from the initial hash to the final hash within each segment. This parallel verification process allows for efficient verification of the entire sequence [104, 103, 105].

As all input strings are documented in the output, along with the corresponding counter and state to which they are appended, the verifiers can replicate each segment in parallel. This parallel replication process enables efficient and simultaneous verification of all slices of the sequence.

4000 hps would result in generating 160 kilobytes of data, which would take a GPU with 4000 cores rouhgly 0.25-0.75 milliseconds of time to verify [104, 103, 105].

5.6 Solana Proof of Stake

Generally in Proof of Stake, verifiers commit a certain amount of coin as a stake, which enables them to participate in the voting process for specific sets of transactions. The act of staking coins as collateral is considered a transaction itself and is recorded in the PoH stream, just like any other event. This is the one of the main reasons for employing the PoH sequencing in Solana's consensus mechanism [105, 103].

To cast a vote, a Proof of Stake verifier must sign the hash of the state, which represents the state after processing all the transactions up to a specific position in the PoH ledger. This vote is also recorded as a transaction in the PoH stream. By examining the PoH ledger, one can deduce the duration between each vote, as well as identify the duration for which each verifier was unavailable in the event of a partition. Essentially, the PoH ledger provides proof of the passage of time between votes and the periods of unavailability for each verifier during a partition.

This particular implementation of Proof of Stake is specifically designed for rapidly confirming the current sequence generated by the Proof of History generator. It is also used for voting and selecting the next Proof of History generator, as well as penalizing any validators that engage in malicious behavior. The algorithm relies on the timely delivery of messages to all participating nodes within a specified timeout period.

5.6.1 Staking

In this context the staked collateral coin while validators are validating transactions in the network is known as a bond. An amount of coin is taken from verifying nodes to an account which holds bonds under the staking user's identity. Such collateral coin has a predetermined timeout period which occurs after the majority of stakers have confirmed the current sequence [105].

The expectation is that the Proof of History generator will periodically publish a signature of the state. Each bonded identity is required to verify and confirm this signature by publishing their own signed signature of the state [105]. The voting process is straightforward, with only a ”yes” vote option available, without a ”no” option. If a supermajority of the bonded identities submit their votes within the specified timeout period, then this branch or proposal would be considered valid and accepted.

5.6.2 Elections

Elections for a new PoH generator take place when a failure of the current PoH generator is detected. Then for the new PoH generator election stakers are sorted in terms of their staked collateral and the one with highest voting power validator is selected [105]. To establish the new sequence, a supermajority of confirmations from the validators is required.

In order to switch votes, a validator is required to vote at a higher PoH sequence counter and include the votes it wishes to switch in the new vote. Failure to meet these conditions would make the second vote susceptible to penalty. Vote switching is designed to occur only at a height where a supermajority has not been reached.

Once a PoH generator is established, another validating node is chosen to process transactions If a Secondary exists, it will be considered as the next leader in the event of a Primary failure.The platform is structured in a way that allows the Secondary to become the Primary,

and lower-ranked generators to be promoted, either when an exception is detected or according to a predefined schedule [105].

PoH generators are equipped with an identity that signs the generated sequence [105]. A fork in the Blockchain can only happen if the identity of the PoH generator has been compromised. A fork is identified when two distinct historical records are published under the same PoH identity. This serves as an indication that the integrity of the PoH generator's identity has been compromised, leading to the creation of multiple conflicting versions of the Blockchain.

5.6.3 Failure and slashing

Errors or failures in the PoH generator would be apparent in this case in the generation of an invalid state, causing the published signature of the state to not align with the local validators' computations [105]. In such a scenario, validators will disseminate the correct signature through gossip communication, which would initiate a new round of elections.

Validators who accept and validate an invalid state will face penalties in the form of having their bonded funds reduced i.e., their bonds will be slashed [105]. This incentivizes validators to carefully verify the correctness of the state and take necessary actions to prevent the acceptance of invalid or maliciously generated states.

Penalties here are executed through the compromise of the staked collateral by the protocol. This is a process known as slashing. When a malicious vote is detected its proof will take the malicious node's collateral and place it in the reward pool for the block creator. Additionally slashing can occur when a voting node votes on an invalid hash. However two sequential votes where the first one has successfully been in the PoH sequence are not deemed as malicious but the second one is simply removed from the voting process.

5.6.4 Finality

PoH enables network verifiers to gain insights into past events with a certain level of confidence regarding the timing of those events. As the PoH generator generates a continuous stream of messages, all verifiers are expected to submit their signatures of the state within a specified timeframe, such as 500ms. This timeout duration can be further shortened based on the network conditions [104].

Since each verification is recorded in the stream, every participant in the network can verify that all verifiers have submitted their votes within the prescribed timeout, even without

directly observing the voting process. This allows for validation of timely participation by verifiers without the need for direct observation of the voting activity.

5.7 Solana Challenges

Solana has experienced multiple instances of network outages, with a total of 11 major outages and 3 minor outages in the year 2022. The duration of these outages varied, lasting from 1 hour 15 minutes to 17 hours 7 minutes. The most recent outage was particularly significant, marking the longest period of downtime in over a year.

Scalability concerns: While Solana is known for its high throughput and fast transaction processing, concerns have been raised about its ability to maintain scalability as the network grows. As the number of nodes and transactions increases, it may face challenges in sustaining its performance [104].

Network centralization: Some critics argue that Solana's consensus mechanism, which relies on a small set of validators known as "the Council," introduces a level of centralization. This has raised concerns about the platform's decentralization and censorship resistance.

Reliance on validators: Solana's security depends on the honesty and proper functioning of the network validators. If a significant number of validators behave maliciously or experience technical issues, it could impact the overall security and reliability of the platform.

Complexity and learning curve: Solana's architecture and development ecosystem can be complex, requiring a deeper understanding of its unique features and programming models. This may pose a challenge for developers and hinder wider adoption by the developer community.

Limited Smart Contract language support: At the time of writing, Solana's flagship language for Smart Contracts is the Rust programming language. This limitation may restrict the accessibility of the platform for developers who prefer other programming languages, potentially reducing the pool of talent and available tooling.

It's important to note that Blockchain platforms constantly evolve and address these weaknesses over time, and the Solana team may already be actively working on mitigating these concerns.

Chapter 6

Solana Smart Contracts

6.1 Introduction

Smart Contracts have become increasingly popular in the realm of Blockchain technology due to their ability to enable distributed, transparent, and trustless computing. These contracts have gained significant attention, considering the market capitalization of cryptocurrencies driven by Smart Contracts exceeds $1TB. However, their appeal to attackers is also notable, as the open-source nature of Smart Contract code presents opportunities for exploiting vulnerabilities and potentially stealing substantial amounts of funds. For instance, the Wormhole network suffered a loss of over $ 320 million due to a missing key check, while the Poly Network experienced a vulnerability in its Smart Contract that resulted in a loss of $ 611 million [66]. These incidents highlight the importance of ensuring robust security measures in Smart Contracts to protect against potential exploits [26, 24, 40].

Ethereum (ETH) is widely recognized as one of the leading platforms for Smart Contracts, with a vast number of applications deployed and a market capitalization exceeding $374 billion. However, Ethereum faces limitations in terms of processing speed, currently operating at a range of 10-15 transactions per second (TPS). This constraint is attributable to several factors, including the use of Proof of Work (PoW) consensus mechanism. Additionally, the design of Ethereum Smart Contracts involves maintaining a single copy of states, resulting in a sequential execution requirement for updating these states within the same contract. These factors contribute to the scalability challenges faced by Ethereum in terms of transaction throughput.

Solana Smart Contract platform aim to overcome Ethereum's speed limitations. It is

renowned as the fastest Blockchain globally, boasting transaction speeds surpassing 2,000 transactions per second (TPS) [27]. With Solana's scalability, transaction fees remain low, amounting to less than $0.01, and the platform achieves rapid transaction processing rates of around 400 milliseconds per block. This enhanced efficiency enables previously underserved users to access the decentralized ecosystem, including areas such as Decentralized Finance (DeFi), Non-Fungible Tokens (NFTs), and other Web 3.0 applications. Solana's capabilities provide an efficient platform for various emerging use cases in the Blockchain space.

6.2 Definition

Solana introduces a distinct Smart Contract model compared to traditional EVM-based Blockchains [90]. In traditional EVM-based chains, a contract combines both code/logic and state, and the entire contract is deployed on the Blockchain. In contrast, Solana's approach involves separating the Smart Contract into two components: the Smart Contract itself, which consists solely of program logic and is read-only or stateless, and the state, which is maintained separately. When deployed on Solana, the Smart Contract can be interacted with by external accounts, allowing for more flexibility and efficiency in contract execution. This design choice in Solana offers a different approach to managing and interacting with Smart Contracts compared to traditional EVM-based platforms.

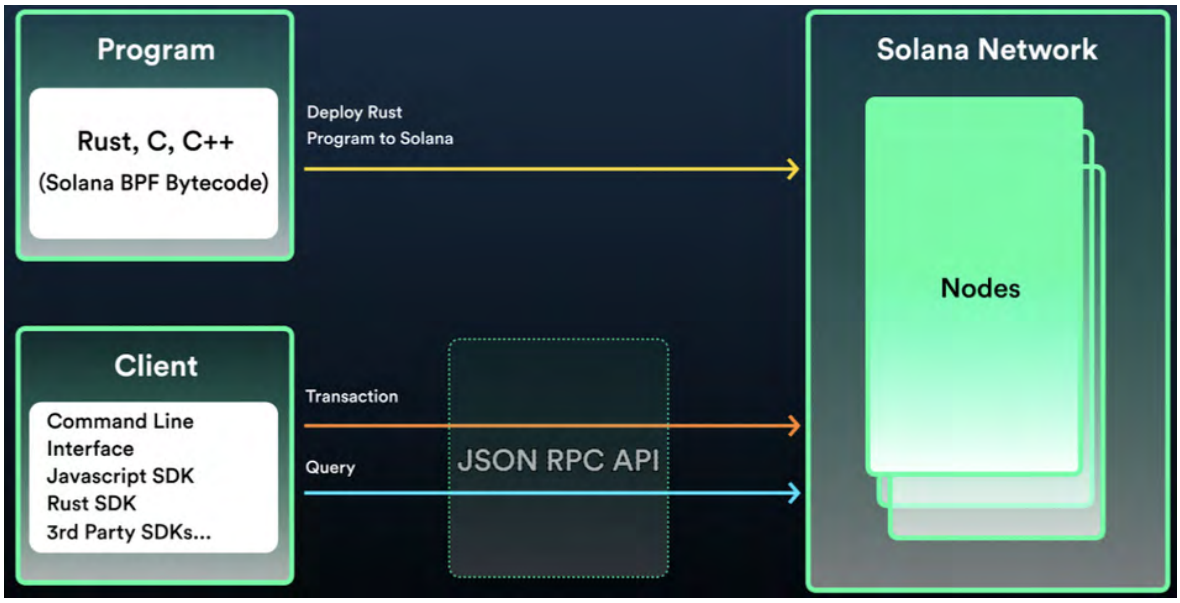

Figure 6.1: Solana Smart Contract [90]

The accounts that interact with the programs serve as storage for data associated with

program interaction. This design choice establishes a clear distinction between the state, represented by these accounts, and the contract logic contained within the programs. This differentiation is a fundamental divergence between Solana and EVM-based Smart Contracts. In Ethereum, accounts primarily serve as references to individuals' wallets, whereas Solana accounts have the capability to store data, including wallet information. This key difference allows Solana accounts to have a broader functionality in terms of data storage and handling compared to Ethereum accounts.

Solana introduces Sealevel [87], a feature that enables parallel execution of Smart Contracts. Unlike many other Blockchain networks where only one Smart Contract can influence and modify the Blockchain state at a time, Solana's runtime has the capability to process numerous contracts simultaneously. This parallel processing enables concurrent execution of non-overlapping transactions and allows transactions that only read the same state to execute concurrently as well. By leveraging Sealevel, Solana achieves greater scalability and efficiency in executing Smart Contracts.

Additionally Solana offers its CLI [81] as well as its own JSON RPC API [79] that can be employed by dApp developers in order to interact with the Solana Blockchain. There are also available SDKs in order to communicate with Solana programs.

6.3 Smart Contract - Client structure

Although Solana Smart Contracts can be written in any of the C , C++ and Rust programming languages the most widely adopted one is Rust [84].

Rust is a powerful and modern programming language that combines the performance and low-level control of languages like C and C++ with the safety and simplicity of higher-level languages. Rust enforces strict compile-time checks and prevents common programming errors such as null pointer dereferences, buffer overflows, and data races. Rust thus delivers memory safety and unlike C/C++ explicit low-level memory management is not necessary [15].

Clients in the Solana platform are untrusted [85], similar to userspace in Operating Systems terms but have the flexibility to create a program using their preferred front-end language such as C/C++/Rust. This program is then compiled locally by the client using LLVM. The compilation process involves converting the higher-level language program into a stan-

dard ELF object file. This ELF file contains the specific bytecode optimized for efficient verification and conversion to the local machine instruction set supported by Solana.

On the kernel equivalent side of the Smart Contract [80], the ELF file undergoes a series of steps for verification, loading, and execution. First, a verifier checks the validity and safety of the bytecode, ensuring it is suitable for execution. Subsequently, a loader prepares the necessary memory to load the code and marks the corresponding segment as executable. Finally, the Solana runtime takes over and initiates the execution of the program, managing the runtime environment and handling any modifications to the virtual machine.

By employing this approach, Solana enables users to develop programs [83] in their preferred front-end languages and compile them into a bytecode format that can be quickly verified and executed on the Solana Blockchain [80]. This design allows for a more efficient and secure execution of user-created programs within the Solana ecosystem.

As the program is loaded in the validator [80] and the bytecode is checked for instruction compliance with the supported instructions and memory as well as runtime [86] constraints, the times associated with these procedure make up for the most performance draining procedures when executing. Thus the target goal in this use case is the easiest fastest and simplest way to verify the instruction set.

6.4 Memory management

Memory management is the most important aspect that significantly impacts performance in the Solana engine. The engine's performance is directly influenced by the concurrent execution of contracts without any data dependencies. This implies that if all scheduled contracts are independent, they can be executed simultaneously, leveraging the available cores. As a result, the engine's throughput will scale with the increasing number of cores, following Moore's law with a doubling rate every two years [105].

The management of memory begins with the ELF itself, where contracts are initially limited to a single read-only code and data segment. This composition consists of executable code and read-only data, without any mutable globals or static variables. While this requirement may be relaxed in the future, it currently offers a straightforward solution for the aforementioned fourth requirement.

6.5 Solana's Basic Smart Contract concepts

The account model is a fundamental component of the Solana Smart Contracts [85, 91]. Accounts are the primary units for storing data and state. Each account is identified by a unique address, which is derived from a public key. Accounts can hold various types of data, including tokens, program instructions, and custom data structures defined by Smart Contracts. Solana's accounts are similar to classes in Object Oriented Programming and define the information that will be stored on chain.

The most important aspect of Solana's accounts is the fact that they are statically allocated. Once an account is allocated no reallocations are possible and thus once a Smart Contract is on-chain no alterations can be made. The same applies for data stored on chain which is a feature of Blockchains in general but troublesome for development.

Account States: Solana accounts can have different states based on their usage and permissions. The most common states include:

Read/Write Accounts: These accounts allow read and write access, meaning they can be modified by authorized parties such as Smart Contracts or external entities.

Read-Only Accounts: These accounts provide read-only access, allowing users or Smart Contracts to retrieve information from the account without modifying its state.

Program Accounts: Program accounts are specialized accounts associated with Smart Contracts. They store the program code and relevant data for executing the logic defined by the Smart Contract.

A token account [91] may store token balances and metadata, while a program account may store the program instructions and state variables. The ownership of an account is determined by the associated private key, which is used to sign transactions and authorize changes to the account's state.

As Solana accounts can interact with each other through transactions [88], transactions specify the operations to be performed on one or more accounts such as transferring tokens, updating data or invoking Smart Contract methods. Accounts can be both the source and destination of these transactions. In this case the clients invoke Smart Contract functions which target the Smart Contracts program id as the transaction receipient [83].

6.6 Solana, Anchor and Seahorse

Solana's platform provides three main alternatives when developing Solana contracts [91]. This reflects a very different featureset than other Smart Contract enabled platforms. Such alternatives are different in terms of the level of abstraction provided by the underlying framework.

6.6.1 Native Rust

Native Rust development [84] where the JSON RPC communication must be performed explicitly as messages are serialised on communication with the server. The main library that Smart Contract functionality is based on is the Solana program library (SPL). SPL provides a collection of pre-built programs and tools that developers can leverage to build Smart Contracts. Developers can also create their own custom programs using the Solana SDK [91].

6.6.2 Solana Anchor

Solana Anchor [72] is a Rust framework [77] which attempts to remove some of the boilerplate code surrounding JSON RPC communication and remote function calls.

This boilerplate code is regarding serialization and deserialization of data which is essential in Solana programming as all data and remote function calls must be serialized and then converted to the JSON RPC format in order to be sent to the validator.

6.6.3 Seahorse Solana

Searhorse [95] framework is a framework where source code is written in a very constrained subset of python like syntax [94] but with explicit Rust types [93]. This initial source code is later transpiled to a Rust Anchor program that is then compiled to Solana bytecode.

Seahorse is very early stage software introduced in September of 2022 with many unsupported features. It makes prototyping faster as it automatically creates the safe underlying rust code. However when running into unsupported features the generated rust code is provided and the programmer can implement such features in the Rust source.

6.7 Creating a First Program and deploying it on chain

Solana program compilation is different for each of the Frameworks used. What remains the same in all cases is that compiled contracts are deployed using the Solana-cli [81]. There are three networks where Smart Contracts can be deployed i.e. the Solana mainnet, Devnet as well as a local Solana network [89].

The Solana mainnet is Solana's main network where Smart Contracts run on real validators and the Solana coin used for transaction has real monetary value. Once Smart Contracts have completed development they are deployed there to enable their dApp functionality. Note here that once deployed in the mainnet no alterations can be made in the Smart Contract code.

The Devnet is a Solana network which uses real validating nodes and attempts to simulate the mainnet but coins here have no monetary value since they can be obtained using the Solana's devnet faucets that provide coin on request for developers seeking to test their Smart Contract transactions.

Local hosting in this scenario is when attempting to create a local Solana network using the Solana-test-validator cli tool [89, 81]. This tool creates a verifying node locally which can support transactions as well as Smart Contract and deployment execution. This is important as unlike mainnet and devnet this network can be trashed and created again which is crucial when trying to apply changes to Smart Contract code.

6.8 Deployment

After compilation of the Smart Contract code, dedicated Solana account [85] is created to hold the program and serve as the entry point for executing the contract's logic. The account can be funded with SOL tokens to cover transaction fees [88].

The program is deployed to the Solana Blockchain by submitting a set of transactions that include the program's bytecode and the necessary instructions to create the account. This transaction is broadcasted to the Solana network for processing.

Once the transaction is confirmed and included in a block by the Solana network validators, the program's account becomes active and ready for interaction.

Users [92] can now interact with the deployed Smart Contract by invoking its methods or sending transactions to the associated program account. These interactions trigger the execution of the contract's logic on the Solana network.

Deployed Smart Contracts can be viewed in Solana's block explorer [91] which allows users to see all transactions in the chain including Smart Contract deployment and execution transactions.

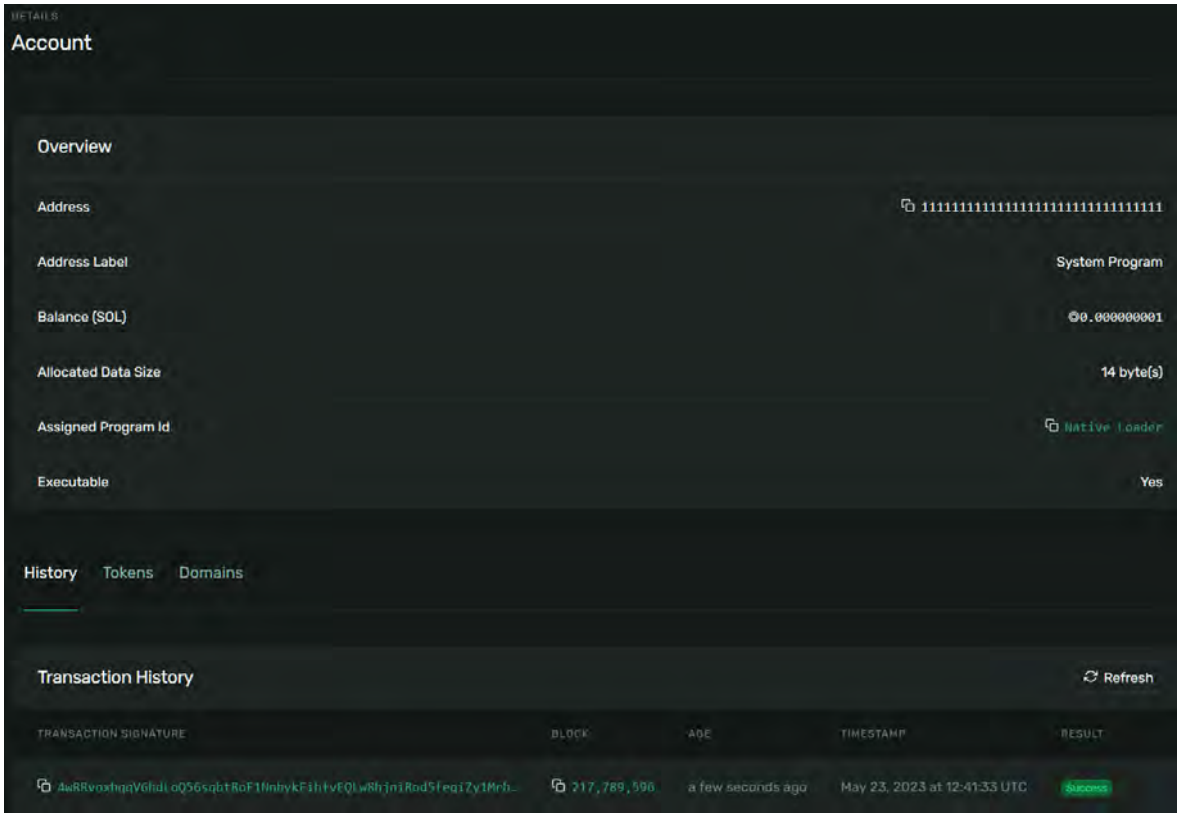

Figure 6.2: Solana explorer devnet

Chapter 7

Blockchain based Federated learning

7.1 Introduction

As discussed, Federated learning (Fl) is an innovative approach to distributed machine learning that enables model training without exposing individual data and ensuring privacy. It leverages collaborative learning principles to create privacy-preserving models. However, Fl faces challenges including privacy risks, unreliable parameter uploading, high communication costs, and more. Blockchain, being a decentralized technology, attempts to provide solution for the challenges that Fl environments face and enhance Fl performance by eliminating the need for a centralized server and replacing it with a Blockchain distributed system.

So far we have discussed the the motivating factors that drive the need for Federated learning as well as its challenges. We have also discussed the use of dedicated Blockchains or Blockchain powered Smart Contracts in the implementation of decentralized systems. Efforts have attempted to combine these two domains in Blockchain based Federated learning where Blockchain Smart Contracts implement the aggregating server that stores models and coordinates model collection in Federated learning environments.

7.2 Motivation

Various Federated learning challenges and problems can be compensated by the employment of Blockchain platforms and Smart Contracts.

7.2.1 Communication costs

Although Fl environments optimize communication costs by exchanging models and not data there is still room for improvement as communications can be altered to be even more distributed among client nodes. Efficient strategies and protocols need to be employed to further minimize communication costs and optimize the overall performance of Federated learning systems and this can be done with the use of peer to peer networks.

7.2.2 Single point of Failure

In Federated learning scenarios where a single centralized aggregator is responsible for collecting and aggregating models from multiple participants, there are inherent vulnerabilities that can lead to Single Point of Failure (SPoF) and Distributed Denial of Service (DDoS) attacks [109, 64]. The reliance on a single aggregator creates a critical dependency, as any failure or disruption in its operation can halt the entire Federated learning process. This makes the system susceptible to SPoF, where the failure of the aggregator would render the entire network non-functional. Additionally, the centralized nature of the aggregator makes it an attractive target for malicious actors to launch DDoS attacks, overwhelming the aggregator's resources and causing service disruptions. Distributed or decentralized aggregators can mitigate such risks.

7.2.3 Code and Weight redundancy

In traditional centralized or hierarchical Federated learning final total model weights are gathered in a single server and no other copies of the model are present outside the central aggregating server. Without code redundancy, there is a risk of clients running different versions of the code, leading to inconsistencies in the learning process and potentially impacting the accuracy of the aggregated model [64]. As code and data are stored in transactions in Blockchain environments they are replicated in all validating nodes as part of the chain.

7.2.4 Code and weight transparency

Without code transparency, participants have limited visibility into the algorithms and processes used for model storage and inference in the aggregating server, making it difficult to assess the fairness, bias, or potential vulnerabilities in the system. The lack of weight

transparency prevents participants from verifying the integrity and accuracy of the aggregated model's parameters. This lack of transparency can lead to concerns about trust, accountability, and the potential for malicious manipulation of the learning process or model outputs [109]. Clients that use the deployed Fl model to be able to cross reference the deployed model's weights. o address these downsides, it is essential to establish mechanisms for code and weight transparency, allowing participants to inspect, audit, and validate the processes and parameters involved in centralized Federated learning.In Blockchain environments this issue is resolved as both the operating logic as well as weights are stored in publicly audible transactions.

7.2.5 Code and weight immutability

Without code immutability, there is a risk of unauthorized modifications or tampering with the algorithms and processes used for model training. This can undermine the integrity and reliability of the learning system, potentially leading to biased or inaccurate model outputs. Similarly, the absence of weight immutability raises concerns about the trustworthiness of the aggregated model's parameters. If the weights can be easily modified or manipulated, it becomes difficult to ensure the consistency and reproducibility of the learning process. Mechanisms against of the model mutations that are unauthorised must be in place in order for Fl models to be trusted by clients who want to deploy them in applications or systems. There can be no discrepencies between deployed models unless explicitly requested by clients. This is resolved as storage in Blockchain systems is immutable [101].

7.2.6 Incentives to clients for good behaviour

Aggregating servers must provide protection against model poisoning, where malicious actors may force training nodes to train models with false data. Such faulty models will therefore propagate to the aggregated model unless detected by the aggregated server and rejected. If the model is averaged into the global model then it is impossible to undo the damage done. Parallel to that Fl aggregating systems must be able to detect the quality of service provided by clients in such environmnents. Mechanisms must be in place that recognize misbehaving lazy as well as malicious participants in Fl environments. When using Blockchain for Fl coin rewards can be donated to well behaved and willing participants.

7.3 Blockchain Based Federated learning

Blockchain Based Federated learning (BCFL) is a Federated learning paradigm where the Fl employs Blockchain in order to implement the aggregating server. Model training is carried out in Blockchain client peer nodes that communicate via peer to peer networking protocols while models are stored on-chain and replicated across participating nodes. By leveraging Blockchain, model updates can be securely stored and accessed in a distributed manner thus addressing the aforementioned Federated learning challenges [101].

7.4 Blockchain based Federated learning characteristics

BCFL's use and strength is shown through its unique characteristics [109, 64]:

7.4.1 Decentralization

The decentralized nature of Blockchain based Federated learning ensures that model updates are stored across multiple servers, providing resistance against attacks and data redundancy issues that plague traditional Federated learning systems. Here multiple servers are leveraged distributed across the network, making it more resilient to such failures. Additionally greater availability is ensured for the Federated learning system's operation as Blockchain is known to be fault tolerant.

7.4.2 Immutability

Furthermore, the use of decentralized storage mechanisms, such as Blockchain, guarantees the immutability of the models as well as executing logic as they are both stored in transactions, making them resistant to tampering or unauthorized modifications. This immutability is guaranteed in the Blockchain system itself and does not rely on third parties. Simply put in order to mutate said models or the executing code one would have to attack the entire Blockchain network and compromise it.

7.4.3 Traceability

The inherent transparency ensures that participants cannot deny their authorship of model updates. Additionally, the use of Blockchain technology enables the detection and prevention

of tampering with the records. Each block of transactions in the chain is used to generate a unique hash value, making it permanent and immutable. This means that any attempt to modify or tamper with the records can be easily detected by the server. The Blockchain-based approach provides a robust and secure framework for Federated learning, ensuring the integrity and traceability of model updates. Finally malicious agents can be identified by their cryptographic keys and excluded from future training.

7.4.4 Incentives

Participants in Blockchain-based Federated learning are incentivized through rewards or incentive mechanisms. These incentives motivate participants to contribute high-quality data and model updates, ultimately leading to the development of an accurate global model. This can be quantified by the measurement of the participants' contribution ratio. By rewarding motivated with meaningful contributions with coin, the interests of participants will be with the success of the Federated learning system.

7.4.5 Integrity and Reliability

The cryptographic connection between blocks ensures that any alteration or tampering of data can be readily detected within the Blockchain. Specifically this will result in what we descibed as a fork which will be rejected in some blocks' time and will not be finalized in the network. This inherent feature of Blockchain technology provides a high level of security and reliability.

7.4.6 Trust

In Blockchain-based Federated learning, a consensus algorithm is employed to establish trust among the participants without third parties incolved. Participants who agree to the terms of the contract are granted permission to participate in the training rounds. This consensus mechanism helps ensure the integrity and fairness of the Federated learning process as well as designing penalties for the foul participants in the system.

7.5 BCFL system design overview

The architecture of Blockchain-based Federated learning consists of several key components. First, there are Fl participants who contribute their data and model updates. Second, there is the integration of Fl with Blockchain technology. Third, miners are involved in the Blockchain network, performing computational tasks. Fourth, a Smart Contracts governs the rules and interactions within the Fl system. Fifth, a consensus algorithm is used to establish agreement among participants. Lastly, the Blockchain network itself provides the underlying infrastructure for secure and transparent transactions. Together, these components form the foundation of a Blockchain-based Fl system.

Federated learning participants in a Blockchain-based system operate similarly to traditional Fl environments. These participants can be entities or devices that contribute to model training. They are actively involved in the process of sending local model updates for verification and aggregation. Initially, the initial model is distributed to all participants in the Fl system. Each participant generates local model updates based on their respective raw datasets. In the Blockchain-based Fl system, participants and miners establish direct communication with each other, facilitating the exchange of information and coordination throughout the training process.

The integration of Federated learning with Blockchain serves as a middleware layer that enables interaction between Fl participants and the Blockchain. In previous studies, researchers have employed various methods to achieve this integration. For instance, Martinez et al. [56] utilized the REST-API to interact with the Hyperledger Fabric Blockchain. This approach allowed for the recording and incentivization of gradient uploads. Additionally, the gRPC API has been used to facilitate data transfer between Fl clients and the Ethereum Blockchain network. These APIs enable efficient communication and data exchange, enhancing the integration of Federated learning with Blockchain technologies.

The miners in the Federated learning system can encompass personal computers, standby servers, or cloud-based nodes that voluntarily download the mining software. During this stage, Fl participants transmit their local model updates to the miners. Each Fl participant or data holder establishes a direct connection with a miner to enable continuous communication. The role of the miners is to receive the local model updates from the participating Fl devices or participants. Subsequently, the aggregation process takes place, utilizing the consensus algorithm to reach a consensus on the updated global model. Once the aggregation is

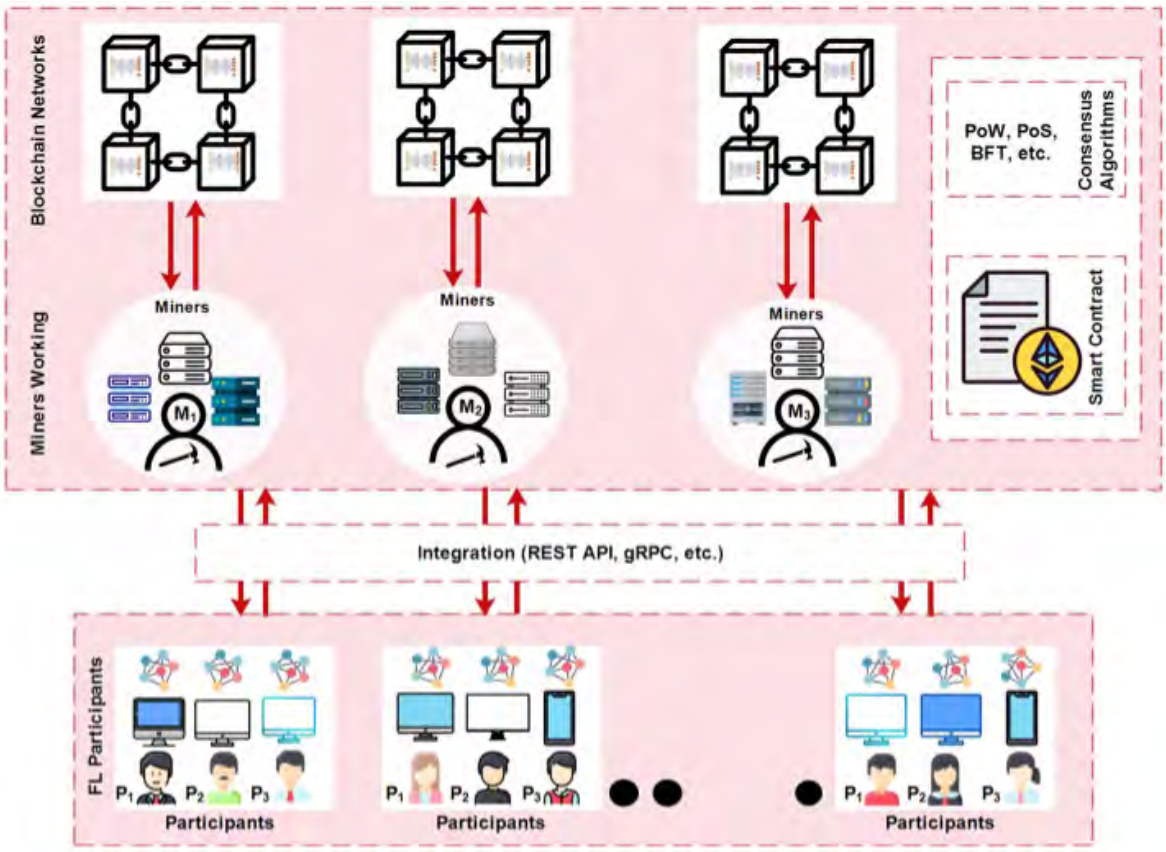

Figure 7.1: BCFL Architecture [64]

completed, a block containing the aggregated model is uploaded to the Blockchain network, ensuring the integrity and transparency of the Federated learning process.

The Smart Contracts play a vital role in the Blockchain-based system, enabling decentralized applications and automating the execution of program logic based on predefined conditions. These conditions are transparent and immutable to all participating Fl clients, and they must agree to them before joining the Fl model training process. By leveraging Smart Contracts, clients can establish agreements without the need for a trusted third party. In the context of Federated learning, researchers have utilized Smart Contracts for various purposes, including participant registration, coordination of model training, aggregation of local model updates, evaluation of participant contributions, and rewarding participants. In the architecture depicted in Figure 7.1, the Smart Contracts acts as an intermediary between the Fl participants and the miners, facilitating their interactions and enforcing the agreed-upon rules and incentives.

The Smart Contracts plays a vital role in the Blockchain-based system, enabling decentralized applications and automating the execution of program logic based on predefined

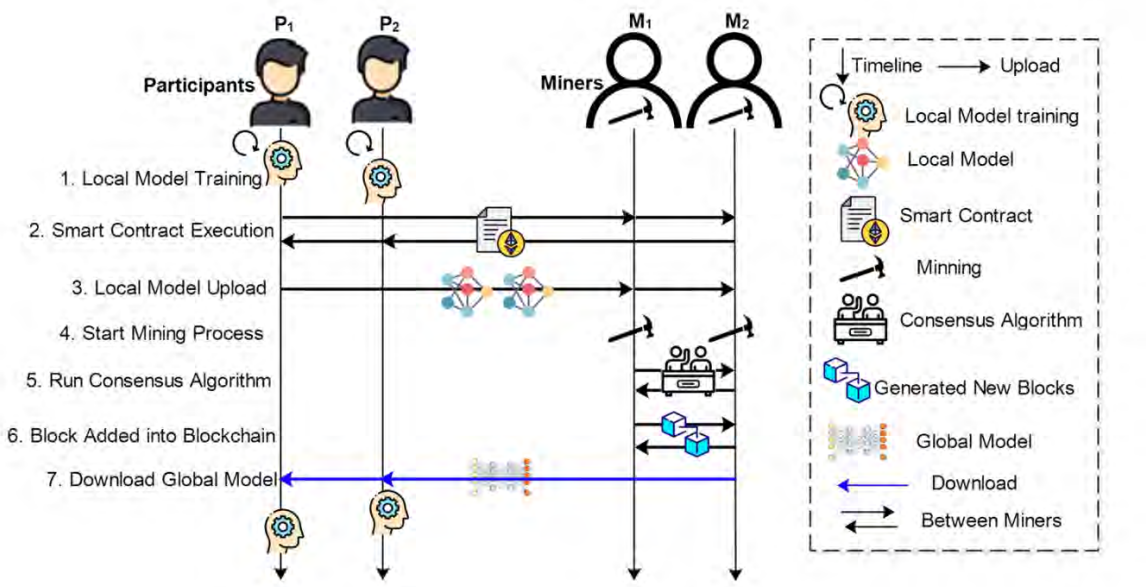

Figure 7.2: Smart Contracts based Federated learning [64]

conditions. These conditions are transparent and immutable to all participating Fl clients, and they must agree to them before joining the Fl model training process. By leveraging Smart Contracts, clients can establish agreements without the need for a trusted third party. In the context of Federated learning, researchers have utilized Smart Contracts for various purposes, including participant registration, coordination of model training, aggregation of local model updates, evaluation of participant contributions, and rewarding participants. In the architecture depicted in Figure 7.2, the Smart Contracts acts as an intermediary between the Fl participants and the miners, facilitating their interactions and enforcing the agreed-upon rules and incentives.

Finally once the local model updates have been verified and aggregated, new blocks containing these updates are added to the Blockchain network. This process ensures the integrity and immutability of the Fl model. The Fl model training continues until it reaches the desired learning rate or convergence criteria. Once this is achieved, Fl clients or other participants have the option to request and download the global model for their specific purposes. The miners, who have received the updated global model, can then distribute it to the Fl participants who need access to the finalized model. This way, the global model becomes available for download and utilization by the participants involved in the Federated learning process.

Chapter 8

Related Work

We present here the related implementations of Blockchain based Federated learning and investigate them in three different categories. Approaches that focus on providing security and privacy, approaches that focus on recording and rewarding participants as well as efforts that employ verification and accountability. New efforts are constantly being introduced and this assembly of efforts is valid through June 2023.

8.1 Blockchain based approaches to security and privacy in Federated learning

Decentralized approaches based on Blockchain technology offer effective solutions to address security and privacy issues in the Federated learning (Fl) environment [47, 48]. These approaches aim to tackle challenges such as Single Point of Failure (SPoF), poisoning attacks, free-riding, and Distributed Denial of Service (DDoS) attacks. Several studies have proposed Blockchain-based Fl frameworks and implementations to enhance security and privacy. For example, the BytoChain framework, introduced by Li et al. [48], utilizes Blockchain technology to secure Fl systems. BytoChain involves different entities such as data owners, verifiers, miners, and task publishers. Verifiers play a crucial role in reducing the verification overhead for miners by working in parallel. Moreover, a consensus algorithm called Proof of Accuracy (PoA) is employed to detect privacy loss effectively. These Blockchain-based approaches contribute to enhancing the security and privacy of Fl systems, and they are discussed in Figure 8.1, including the key contributions, Blockchain implementation frameworks, consensus algorithms, and block structures utilized in these studies [67, 50].

Approaches	Major contribution	Blockchain type	Block structure	Block storage	Consensus algorithm	Blockchain
BytoChain (Li et al.)	Byzantine resistant consensus Proof of Accuracy (PoA) Detected the random and reverse poisoning, overfitting poisoning, DoS, and free-riding attacks	Private	Merkle Tree	–	PoA	–
Chainsfl (Yuan et al.)	Raft and DAG-based blockchain consensus algorithm Synchronous and asynchronous learning combined to dismiss the drag down of stragglers	Private	Merkle Tree	Off chain	Raft and DAG	Hyperlegdger
BLADE-FL (Ma)	Prevented from the Single point of failure (SPoF) attack Misbehaved and lazy participants are recognized	Public	–	–	PoW	–
BFEL (Kang et al.)	Proof of Verifying (PoV) consensus algorithm to filter out poisoning updates A gradient compression scheme with PoV	Public and Consortium	Merkle Tree	–	PoV, DPoS, and PBFT	EOS.IO
(Short et al.)	Based on the accuracy improvement, model updates are evaluated Traceability function of blockchain for the detection of malicious users	Private	–	–	–	Hyperlegdger
BFLC (Li et al.)	Committee consensus algorithm to reduce model poisoning attacks Storage optimization, scalability of BFLC, and incentives	Consortium	–	On-chain	Committee	FISCO
(Kumar et al.)	Differential privacy (DP) and homomorphic encryption (HE) to improve the security in FL Incentive scheme	Public	–	Off-chain	–	Ethereum
Biscotti (Shayan et al.)	Prevent Sybil and poisoning attacks using VRF and PoF, and multi-krum, respectively Implemented the secret sharing scheme for secure model aggregation	Private	Merkle Tree	Off-chain	PoF	Hyperlegdger
Fed-BC (Wu et al.)	Fully decentralized system avoids SPoF attack and privacy leakage	Private	–	Off-chain	–	Hyperlegdger

Figure 8.1: BCFL efforts for security and privacy [64]

BytoChain has demonstrated its effectiveness in countering various security attacks such as reverse model poisoning as well as Distributed Denial of Service (DoS). It maintains comparable accuracy to Fl systems operating without any attacks, even under attack settings. Another notable framework, ChainsFL proposed by Yuan et al. [107], combines Blockchain and Federated learning in a two-layer architecture. The main-chain, based on the Raft consensus algorithm, coordinates devices to perform model training tasks with significant computational and storage capabilities. Additionally, a sub-chain composed of a Directed Acyclic Graph (DAG) or tangle consensus handles interactions at the sub-chain layer. ChainsFL [107] successfully detects fake model updates and addresses issues related to lazy clients. Comparative experiments involving convergence and robustness metrics were conducted against FedAvg [58] and Asynchronous Fl (AsynFL) by Cong Xie et al. [102]. The results demonstrate that ChainsFL effectively detects and eliminates malicious devices and model updates, showcasing its performance and resilience.

In the BLADE-Fl framework proposed by Ma [55], Blockchain is leveraged to enhance decentralized Federated learning (Fl) by addressing malicious learning updates and single point of failure (SPoF) attacks. The framework consists of three layers: the network layer handles task publishing and node training, the Blockchain layer tracks and aggregates model updates, and the application layer executes Fl events using Smart Contracts. Incentives are provided to participants and miners for their contributions to the training round and successful aggregation and broadcasting of the model. BLADE-FL also tackles privacy concerns, resource allocation, and lazy participant issues. Another approach, introduced by Kang et al. [33], is the Blockchain-enabled federated edge learning (BFEL) method. It utilizes a consortium Blockchain with a Proof of Verifying (PoV) consensus algorithm to identify poisoning model updates and verify their quality. Miners, selected based on their computational and storage resources, implement the consensus algorithm, while those with insufficient resources are eliminated in real-time. BFEL incorporates a gradient compression scheme to reduce gradient leaks from inference attacks. The method offers flexibility in model training, detects malicious model updates, and mitigates computation overhead.

Short et al. [70] implemented Blockchain technology to address security issues in Federated learning. Their algorithm, implemented in a Smart Contract, ensures privacy of client datasets and can integrate external tools. Hyperledger Fabric, a private Blockchain tool, was utilized for the experiment. The proposed algorithm demonstrated effectiveness against poisoning attacks. In the work by Zhao et al. [110], the Blockchain-based Fl committee (BFLC) consensus algorithm was introduced to defend against malicious attacks and reduce computation overhead. The BFLC framework consists of three steps: Blockchain storage, committee consensus algorithm, and model training. Local and global model updates are stored in separate blocks, and the consensus algorithm verifies and assigns scores to gradient updates before adding them to the Blockchain. Model training involves aggregating verified local model updates into a global model. A profit sharing scheme based on contribution is implemented to incentivize participants in the model updates process. BFLC performs well under malicious attacks and minimizes transmission costs. Kumar et al. [41] proposed decentralized training for Fl with Blockchain, incorporating security measures such as Differential Privacy and Homomorphic Encryption. Additionally, Elastic Weight Consolidation was employed to enhance the operation of the global model.

8.2 Blockchain based Federated learning record and reward approaches

Financial compensation in the form of coin can be used to provide incentives for self-interested workers or data holder devices that may be hesitant to participate elsehow. However, previous studies have shown that devices can contribute their resources effectively in Federated learning, but the cost of model training becomes a concern [41, 110]. Additionally, in Fl, there is a risk of untrusted participants performing malicious actions by sending malicious model updates, leading to model poisoning attacks. To address this, tracing or recording model updates can help detect such malicious actions, allowing for the identification and punishment of the involved participants. Incentivizing reliable participants through rewards for sending benign model updates becomes crucial. Therefore, well-designed approaches are needed to measure the beneficial contributions of participants and provide appropriate rewards. Table 8.2 provides a summary of Blockchain-based Federated learning approaches that incorporate record and reward schemes for motivating participating workers in model training rounds.

Approaches	Major contribution	Blockchain type	Block structure	Block storage	Consensus algorithm	Blockchain
FedCoin (Liu et al.)	PoSap consensus protocol for fair payment distribution between clients Record of all payments	Public	Merkle Tree	–	PoSap	–
(Martinez et al.)	Class-Sampled Validation-Error Scheme (CSVES) for rewarding and validating the model updates Record model training updates	Private	–	Off-chain	–	EOS
(Kang et al.)	Reputation metric to measure the fairness of model updates Workers reputation is calculated and managed Encouraged the high reputation workers with effective incentives	Consortium	–	On-chain	PBFT	Corda V4.0
(Behera et al.)	Record contributions of clients through smart contract and then rewarded A decentralized communication scheme for FL	Consortium blockchain setup	Merkle Tree	Off-chain	–	Ethereum
FL-MAB (Batool et al.)	Measured the relative contribution of every client by Shapley value, and allocate rewards accordingly	Public	–	Off-chain	–	Ethereum

Figure 8.2: BCFL efforts for recording and rewarding [64]

Fedcoin [52] is a Blockchain-based approach that incentivizes Fl participants to update the model. Previous studies have used Shapley Values (SVs) for profit distribution, but the calculation process for SVs can be time-consuming and computationally complex. In Fedcoin, SVs are defined as proof of Shapley (PoSap) protocol with a Blockchain consensus algorithm, providing Fl participants with non-repudiation and incentives. The authors have also developed a demonstration system that performs Fl tasks in real-time and rewards participants based on their performance.

Martinez et al. [56] proposed a record and reward approach by evaluating the contributions of participants in the model training process. Through the cryptographic signatures of transactions in Blockchain, the model update contributions are identitfied, recorded, and rewarded based on the computation power cost utilized by Fl participants. A Class-Sampled Validation-Error Scheme (CSVES) is introduced to validate valuable model updates for rewarding through a Smart Contract. As a result, participants receive incentives for their model updates, leading to more robust Fl models.

Kang et al. [34] introduced reputation as a fair metric for evaluating the robustness and trustworthiness of participants in Fl systems. They designed a reputation-aware participant selection scheme using Blockchain technology. By leveraging the properties of Blockchain, such as non-repudiation and resilience, honest reputation management of workers in updating Fl models is enabled. Additionally, the incentive approach is combined with reputation metrics to encourage devices to contribute high-quality data for model training. The researchers conducted experiments using real datasets and successfully achieved accurate reputation calculation of devices, resulting in significant improvements in model accuracy.

Behera et al. [9] utilized a Smart Contract based on the Ethereum Blockchain to incentivize Fl participants. The participants' contributions to the model training process are measured and associated with the rewards they receive. Similarly, in the work of Batool et al. [8], a monetization scheme based on Blockchain is introduced for Fl clients, along with a multi-dimensional auction called FL-MAB. Clients are selected based on their resources, such as data size, bandwidth, and relative rewards, which they submit as bids. Blockchain-based Federated learning offers non-repudiation, integrity, and incentivizes clients with cryptocurrency rewards.

8.3 Blockchain based Federated learning verification and accountable approaches

Verification and accountability are crucial in preventing attackers from sending malicious model updates in Blockchain-based Fl approaches. Smart Contracts are used to detect and financially penalize attackers. Lazy clients, who send malicious or fake model updates to save computational cost, also need to be addressed. Verification procedures are implemented to ensure the integrity and authenticity of model updates during the training process, mitigating

malicious attacks. The immutable nature of Blockchain enables data provenance and traceability in the Fl training procedure. Various Blockchain-based Fl verification schemes are presented in Table 8.3, aiming to establish trust and enhance security.

Approaches	Major contribution	Blockchain type	Block structure	Block storage	Consensus algorithm	Blockchain
Vfchain (Peng et al.)	A VFChain to verify and audit the updates Aggregated models and proofs recorded by committee selection	Private	Dual Skip Chain	–	–	Hyperlegdger
BC-based PPFL (Awan et al.)	An accountable method to record local and global model updates Tracking of data flows in FL system provides the trust and verification	Private	–	Off-chain	PoW, PoS	Hyperlegdger
BlockFLA (Desai et al.)	Through accountability protects against adversarial attacks Discouraged the backdoor attacks and applied the transparency	Hybrid	–	Off-chain	PoW, PBFT	Hyperlegdger, Ethereum
(Lo et al.)	A trustworthy system to enable accountability in FL. For auditing purposes track the local model and global model. To improve the fairness of data and models a weighted fair training was introduced	Parity consortium blockchain	–	Off-chain	Proof-of-Authority (PoA)	Galaxy FL framework (Ethereum)
Blockflow (Vaikkunth Mugunthan)	A unique accountability mechanism for model contribution Resultant auditing scores reflect the quality of the honest and malicious clients	Public	–	Off-chain	–	Ethereum

Figure 8.3: BCFL efforts for verification and accountability [64]

VFChain [63] is a verifiable and auditable Fl approach that utilizes Blockchain technology. It introduces a committee selection scheme for aggregating and verifying model updates, which are recorded in the Blockchain. Auditability is supported through a data structure called Dual Skip Chain (DSC), enabling authenticated and secure committee search and rotation. VFChain also includes an optimization method for multiple model training tasks. Experimental results demonstrate the effectiveness of VFChain in achieving verifiability and auditability in Fl using Blockchain.

Awan et al. [6] propose a privacy-preserving Fl approach using Blockchain. It involves a server, clients, and aggregators, with a distributed immutable ledger used to record local and global model updates for tamper resistance. By tracking model transactions on the Blockchain, trust and verification mechanisms are provided in the Federated learning process. The tracking process also measures each client's contribution to model updates, facilitating rewards schemes.

Desai et al. [17] develop an accountable Fl method called BlockFLA, which addresses attacks through accountability using hybrid Blockchain technology. Public and private Blockchain

tools, such as Ethereum and Hyperledger Fabric, are used. The public Blockchain architecture executes intensive algorithms and allows anyone to retrieve data, while the private Blockchain ensures communication efficiency and handles sensitive data to prevent data leakage. BlockFLA is evaluated using FedAvg and SignSGD algorithms, incorporating various features including parallelism.

Lo et al. [53] propose a trustworthy Federated learning framework that incorporates Blockchain to enhance accountability and equality in Fl systems. They introduce a Smart Contract and a weighted fair data algorithm in the data model registry to enable accountability and fairness, respectively. The framework is evaluated using a COVID-19 X-ray dataset and achieves improved accuracy compared to traditional Federated learning settings.

BlockFLow [60] focuses on accountability and privacy in decentralized Federated learning systems. It includes a model auditing process to evaluate the behavior of model contributors, distinguishing between good and malicious behavior. Contributors are rewarded with cryptocurrencies based on the public Ethereum Blockchain after the auditing process. Evaluation results demonstrate that auditing scores effectively reflect the quality of honest and malicious participants.

8.4 Open Issues

In Federated learning systems based on Blockchain, miners play a crucial role in model aggregation and reaching a consensus algorithm to receive rewards. However, some miners with malicious intent exploit vulnerabilities in the incentive distribution mechanisms. This behavior negatively impacts the revenue of honest miners and poses a significant threat to the mining pool, resulting in pool mining attacks. While previous studies have discussed this type of attack [19, 106] the specific implications of malicious miners in Blockchain-based Federated learning systems have not been thoroughly explored.

The selection of miners is an important consideration in the architecture of Blockchain-based Federated learning. The integrity and trustworthiness of miners ensure the security and privacy of the models. Two types of miners, static and dynamic (or moving), have been identified by Alladi et al. [1]. Static miners utilize fiber-optic networks to communicate with end devices for model update transactions, while dynamic miners rely on wireless networks for interaction and sharing model parameters. Careful planning is necessary for the selection of

miners, considering factors such as network resource consumption and secure design, which should be addressed in future research [33].

The immutability feature of Blockchain ensures that transactions in Blockchain-based Federated learning systems, including model updates, are permanently stored and tamper-proof. While this is advantageous, it also has drawbacks. Errors in transactions cannot be rectified, and Smart Contract assignments are unchangeable even if both parties agree to the changes. Additionally, Smart Contracts themselves are irreversible once implemented. Furthermore, any attempts to hack or access the model for legal or illegal purposes are permanently recorded in the Blockchain.

Smart Contracts, which execute predetermined logic and store the final state immutably, are susceptible to exploitation if not implemented properly [66]. Vulnerabilities and security issues are found in existing Smart Contracts. Common vulnerabilities include the execution of unknown code indirectly and incorrect handling of exceptions. For example, the activation fallback function in Smart Contracts can lead to parameter type confusion when invoked by a developer. Additionally, in Solidity (a programming language for Smart Contracts), exceptions are thrown and must be handled collaboratively between contracts. If exceptions are not resolved correctly, adversaries can exploit the contracts, resulting in transaction rollbacks.

Blockchain frameworks used in Federated learning, such as the EOS.IO Blockchain, have demonstrated higher performance throughput and efficiency compared to Ethereum. However, these frameworks have also been vulnerable to security attacks, resulting in significant financial losses. Bugs in Ethereum Smart Contracts and copy-paste vulnerabilities have been identified and analyzed by researchers.

In Federated learning, various end devices participate in the model training process and contribute local model updates. However, malicious devices can inject poisoned or tampered model data, compromising the integrity of the global model. This can lead to inaccurate aggregation results and increased resource consumption. To ensure secure and reliable Fl model convergence, it is essential to have trusted and authenticated end devices [39, 31].

The asynchrony of end devices in Federated learning can impact the efficiency and accuracy of the global model. Devices may enter or exit the training process at different times, leading to an unbalanced distribution of rewards and affecting the accuracy of the global model. Various factors, such as network issues or device limitations, can cause devices to drop out from the training process.

Synchronization is an issue in Federated learning where the system runs in a synchronous manner, waiting for all local model updates before proceeding to the next training round and aggregation. This can lead to slower model convergence due to the presence of lazy participants who take longer to complete a training iteration [15].

Blockchain forking occurs when multiple miners simultaneously mine a block, potentially leading to multiple versions of the Blockchain. Higher scalability in Blockchain systems can increase the chances of forking. Misconducted miners exploiting the system's insufficient computing power can also contribute to Blockchain forks. Customized probabilistic verification schemes can be employed to counter and mitigate forking issues in Blockchain-based Federated learning systems.

8.5 Future Directions

The integration of Blockchain technology into Federated learning shows promise in enhancing security and privacy models. It enables the implementation of recording and reward mechanisms with accountability. However, there are still open issues that need to be addressed.

Authentication scheme for Blockchain-based Federated learning: To ensure the recognition of end devices in the Federated learning system, an authentication scheme should be implemented. Devices need to be registered and assigned unique IDs before participating in model training. Developing frameworks to select devices that provide reliable and authentic model updates is crucial for the success of Blockchain-based Federated learning systems.

Vulnerabilities in Smart Contracts: Prior to implementation in Blockchain-based Federated learning systems, Smart Contracts should undergo static analysis to detect vulnerabilities. It is essential to ensure the security of Smart Contracts through code auditing, analysis, and review. Testing Smart Contracts against vulnerabilities is also important [12]. Automated tools for static analysis can provide detailed insights and help verify and fix issues. Additionally, frameworks like ZEUS [32] can be utilized for Smart Contract verification and the development of robust security tools.

Selection and Verification Mechanism for Miners: Miners play a crucial role in adding new blocks to a Blockchain network. However, there is a risk of malicious miners introducing falsified results and gaining incentives from honest miners. To address this issue, it is

suggested to propose mechanisms for selecting and verifying miners. One approach could involve choosing a leader among miners based on their performance and participation in the Blockchain-based Federated learning system. The leader would assume additional responsibilities such as miner selection, registration, verification, authentication, and other related tasks. Once miners are selected, their model updates are verified, and the models are downloaded and aggregated.

Enhancing Privacy in Ethereum Blockchain-based Federated learning: To introduce privacy to the Ethereum Blockchain, zero-knowledge proofs (ZKPs) technologies can be employed. In particular, the authors Ben-Sasson et al. [11] introduced ZKPs through scalable transparent argument of knowledge (STARKs). For future research in Blockchain-based Federated learning, it is recommended to implement ZKPs using STARKs to enhance proof creation performance, ensure post-quantum security, and eliminate the need for a trusted setup.

Contract Management Life Cycle: The utilization of contract management tools can address challenges related to immutability and irreversibility. By effectively managing the life cycle of contracts, these limitations can be overcome. An example of a contract management solution is Fabasoft contracts Fabasoft [20], which is widely used in Europe. This solution enables contracts to be stored in an audit-proof format and offers pre-defined contract management schemes, automated rights modeling, and verification capabilities.

Generally in this section we have investigated works in the field of Blockchain based Federated learning and the targets aims implementations and shortcomings of such works.

Part II

Implementation and Testing

Chapter 9

System Design

The objective for this thesis is a Federated learning environment consisting of the aggregating server logic, the client logic and model training in order to conduct experiments that access the feasability and practicality of Solana as a robust Federated learning environment.

9.1 High Level Overview

The proposed system's aim is that it must achieve the following:

- Gather data for training in participating nodes and prepare them for training
- Train a defined and shared machine learning model in each computational node
- Extract and serialize the trained model's weights in a weight vector
- Perform post training model pruning in the extracted weight vector
- Perform post training model quantization in the pruned weight vector
- Submit model update vector transactions to the Smart Contract
- Server gathers model updates from various number of participants
- Server performs the Federated Averaging
- Server makes the aggregated model available
- Clients can download model for inference

This Federated learning system is model agnostic meaning that it can be used with any model so long as it fits in the spatial constraints imposed by Solana's Smart Contracts.

To this end, multiple technologies (programming languages, frameworks etc) were employed as they are fit for different purposes (model training client code and server code) as well as multiple communication protocols and components.

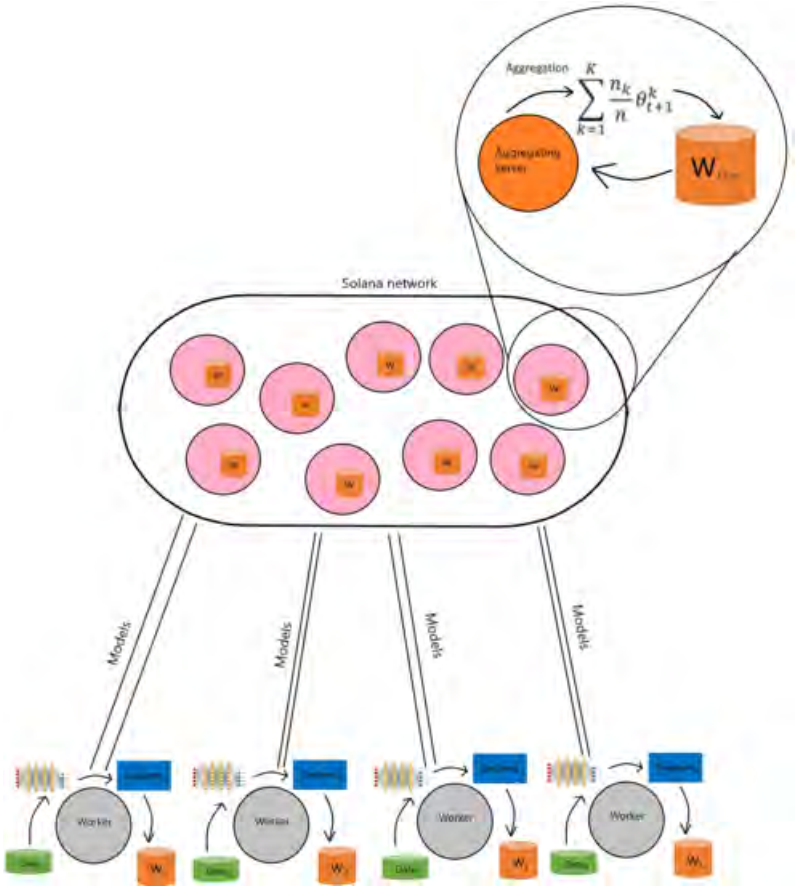

Figure 9.1: Solana BCFL Architecture

9.1.1 Breakdown of components

The proposed system must include the following components

- Aggregating server : Aggregating logic implemented as a Solana Smart Contract
- Models : Creating and training a deep neural network model in multiple variants using post training model pruning
- Training nodes : Training models and client side code for communication to the Solana network

9.2 Smart Contract aggregator

9.2.1 Aims and requirements

The implementation of the Aggregator in this environment is achieved using Solana Smart Contracts, which must at implement four key functions.

- Inintialization of the Smart Contract
- Reset of the Smart Contract
- Receive gradients
- Federated Average

In Solana Smart Contracts [77], initialization is a mandatory step that must be performed by an account before any other functions within the contract can be invoked. It is important to note here that the initialization function can be only invoked by the Smart Contract owner which is the wallet that originally deployed the Smart Contract on chain.

The reset function is also essential as once Smart Contracts are deployed to the chain they cannot be deployed again even with no changes to the code. The reset function allows the system to be easily reconfigured with different models and participants enabling the reuse of the Federated learning system with various configurations

The initialize and reset functions here take as arguements the signing authorities as well as the number of participants in the Federated learning environment which will be used in the final average calculation [72].

9.2.2 Limitations in Solana Smart Contracts

Solana promises that it can hold 10 Mb of data on chain [78, 73]. This however is not realistic as transactions are also bounded in terms of the code execution [74] that each transaction can provide. Through thorough testing it was established that a total of 2008 bytes can be used for on chain model aggregation. As some bytes are required for the Smart Contract functions 2000 ended up being used for the models. Smart Contracts catering to all possible variants were implemented and experiments were conducted in order to measure the limitations of each implementation and the number of participants that can be supported. Such 2000 bytes can be allocated as follows:

- 2000 variables with 1 byte per weight
- 1000 variables with 2 byte per weight
- 500 variables with 4 byte per weight
- 250 variables with 8 byte per weight

9.2.3 Deployment

During the development stages, the deployment of the Federated learning system was facilitated using Solana's Solana-test-validator CLI [89]. This CLI tool enabled the initiation of a Solana node within the user's local network, which served as a processing unit for transactions within the local environment. This approach allowed for convenient and efficient testing and development of the Federated learning system on the Solana Blockchain.

In contrast, when working with the Ethereum network during the Proof of Work era, establishing a local network for testing purposes necessitated the use of tools like Ganache. Ganache enabled the creation of a simulated network with multiple nodes, mimicking a realistic scenario where nodes would compete for block creation. This approach was necessary to assess the performance and behavior of the system under varying network conditions.

However, Solana differs in its consensus mechanism by utilizing a Proof of Stake system. In Solana, even when there is only a single validator present in the network, it is always selected as the leader for block creation. This unique characteristic ensures that the block creation process remains efficient and streamlined, even in scenarios with fewer network participants. This design choice simplifies the development and deployment process, as the need for setting up a complex network with multiple nodes is not required.

It is crucial to highlight that when deploying Smart Contracts on the Solana Blockchain, each Smart Contract and wallet combination can only be deployed once. This limitation arises from the fact that the program ID associated with the deployed Smart Contract is generated during the Solana deployment process [76]. Once the Smart Contract is deployed with a specific wallet, it becomes uniquely identified by the program ID [76], preventing the deployment of the same Smart Contract with the same wallet combination again. This constraint ensures the integrity of Smart Contracts on the Solana Blockchain, maintaining a clear mapping between deployed contracts and their corresponding wallet addresses [75, 73].

9.2.4 Transaction batches

Solana resistrictions also impose an upper limit of 1000 bytes per transaction [88, 74]. This is important as the uploaded models must be sent to the Smart Contract through transactions. In parallel to that there are also restrictions in terms of the number of values that the Smart Contract can handle per transaction as Solana also imposes a computational limit to code execution.

In any case model updates are uploaded to the Smart Contract in large batches of 64-bit hexadecimal values which are decoded at the Smart Contract into single byte, two byte four byte or eight byte words. These batches can either be in 250 or 125 variants in size.

9.2.5 2000 Model Smart Contract

This version of the Smart Contract allocates the 2000 available bytes into 2000 1 byte weights which are stored as unsigned 8-bit integers in the chain. They are uploaded using 2 250 64-bit value batches and decoded in the Smart Contract to receive the 8-bit weights.

9.2.6 1000 Model Smart Contract

This version of the Smart Contract allocates the 2000 available bytes into 1000 2 byte weights which are stored as unsigned 16-bit integers in the chain. They are uploaded using 2 250 64-bit value batches and decoded in the Smart Contract to receive the 8-bit weights.

9.2.7 500 Model Smart Contract

This version of the Smart Contract allocates the 2000 available bytes into 500 4 byte weights which are stored as unsigned 32-bit integers in the chain. They are uploaded using 2 250 64-bit value batches and decoded in the Smart Contract to receive the 8-bit weights.

9.2.8 250 Model Smart Contract

This version of the Smart Contract allocates the 2000 available bytes into 250 8 byte weights which are stored as unsigned 64-bit double precision floats in the chain. They are uploaded using 2 125 64-bit value batches but in this case require no decoding as the values can be stored directly.

9.3 Fl client

Clients here refer to the participating nodes in the Solana BCFL environment. Clients have three major components in terms of functionality.

- Data retrieval
- Model training
- Solana RPC

9.4 Model extraction

All participating nodes within the network must agree on the specific model architecture that is to be trained. Since our Fl environment supports horizontal Federated learning then so long as the data have the same attributes or features among nodes then there are no restrictions in terms of the number of examples that nodes are able to train models with.

The required functions in term of the model aspect of the participating nodes is the ability to define a model to be trained and after the model's training to extract its weights and convert them to the selected format for upload to the Smart Contract aggregator.

9.5 Solana communication

Each participating node takes the encoded weights and uploads them to the Smart Contract aggregator through the implemented commit functions of the Smart Contract which gather the model weights through transactions [88]. These can either be done in a single or dual transactions per participant.

9.5.1 Number of parcticipants

In typical Federated learning setups, the number of participants usually ranges from 10 to 100. However, during our testing, we conducted experiments with varying participant counts, starting from a single participant as a proof of concept to a maximum of 100 participants. This broad range allowed us to assess the functionality and scalability of the Smart Contract across different participant scenarios.

Chapter 10

Implementation

10.1 Solana Smart Contract

For conducting the experiments, the Smart Contract code was developed by combining high-level code from the Seahorse framework and making necessary adjustments to the generated Rust code. By leveraging the high-level code in the Seahorse framework [95], the development process was streamlined, allowing for efficient implementation of the desired Smart Contract functionalities. The framework provided pre-defined structures, functions, and utilities that simplified the coding process and ensured adherence to the Solana Blockchain's programming standards. It also automatically handles mutexes, packages, derived attributes as well as transaction signing and more. Tweaks to the underlying Rust code could involve modifying variables, adding or removing certain operations, or integrating additional functionality as necessary [93, 94, 73, 77].

It is important to note that while the code in the Seahorse framework may resemble Python, the underlying types used in the codebase are inherited from Rust. Therefore, the code primarily utilizes specific types such as u8, u16, u32, and f64 [77]. In this context, the data types in the code align with their Rust counterparts. However, it is worth mentioning that Seahorse only includes a 64-bit floating-point type (f64) and does not support other floating-point types such as f32. To accommodate the representation of 8-bit, 16-bit, and 32-bit floating-point values, the code utilizes the u8, u16, and u32 types respectively, repurposing them to store floating-point data. This approach allows the framework to work within the constraints of the available data types while still providing functionality for floating-point operations and calculations.

10.1.1 2000 u8 Model Smart Contract

In this context, the available storage of 2000 bytes, represented as u8 values, is divided into two distinct 1000-byte models: model A and model B. To upload these models to the Solana Blockchain, a total of two transactions would be required, as the constraint on the size of each transaction is 1000 bytes.

Thus, given the 1000-byte transaction limit our initial effort was to upload each model in two transactions. However, it was discovered that the process of decoding and storing 1000 weights in the Smart Contract exceeded the computational quota enforced by the Solana Blockchain. Consequently, the sending transactions were further split into 500-byte chunks, enabling the upload of 500 weights per transaction.

Once uploaded onto the Smart Contract, the Smart Contract code takes the received 8-byte values and utilizes the ”int_bytes” function provided by the Solana prelude library. This function is responsible for converting the 8-byte values into 8 separate single-byte integers. These single-byte integers are then added to the existing stored values within the defined model A and model B.

10.1.2 1000 u16 Model Smart Contract

Here, the provided space of 1000 bytes which are u16 ints in this case, is divided into 500-byte models. To upload these models to the Solana Blockchain, a total of two transactions would be required, as the constraint on the size of each transaction is 1000 bytes.

So this was our initial made to upload each model in two transactions, given the 1000-byte transaction limit. However, it was discovered that the process of decoding and storing 1000 weights in the Smart Contract exceeded the computational quota enforced by the Solana Blockchain. Consequently, the sending transactions were further split into 500-byte chunks, enabling the upload of 500 weights per transaction.

In this scenario, as there are no additional functions available, such as the ”int_bytes” function, the decoding of the sent values is done differently. The decoding process utilizes bit masks spanning two bytes. Each of the four encoded values is extracted by applying the appropriate bit mask, and then these extracted values are appended to the previously accumulated values in their respective models. This can all be done with two transactions per node.

10.1.3 500 u32 Model Smart Contract

In this particular case, the available storage of 500 units, represented as u32 values, is divided into two distinct 250-unit 4-byte models, namely model A and model B. To upload these models onto the Smart Contract, the process is carried out in two separate transactions. During the upload process, four-byte bitmasks are employed to extract the encoded values within each 8-byte chunk. Since in this case as well we can afford to computationally digest all 1000 bytes at once there is no need for utilizing more than two transactions per participant for model upload.

10.1.4 250 f64 Model Smart Contract

Contrary to the previous cases discussed, the implementation of this system was straightforward and required no additional decoding of values. The transactions involved in this system directly send double-precision floating-point numbers to the Smart Contract. These numbers are then directly added to the previously stored values, and the resulting values are stored on the Blockchain.

For each participant, the transactions in this system send a total of 125 f64 values to the Smart Contract. This process allows for the seamless integration of the participant's model updates into the existing model stored on the Blockchain. Two transactions are required for each participant's model upload, ensuring the accurate representation of their contributions to the Federated learning process.

10.1.5 Federated Averaging

For the FedAvg part it is important to note that while in every case as the aggregated values are integers integer division is employed to compute the average whereas in the f64 case floating point division is used to compute the resulting model.

More specifically in every case the average is calculated for all created models in the Smart Contract and client side logic can be used to download the model to the client as downloading an account's data in the Solana platform is implemented through the Solana library.

10.2 RPC Client

The method that Solana uses for Smart Contract Remote procedure calls is called JSON RPC [79]. JSON RPC packages requests in a JSON file which is later serialized and sent to the awaiting platform. While many languages and frameworks can utilize JSON RPC communication in our environment we implemented the Client side code using typescript which is a statically typed JavaScript and can therefore utilize JavaScript's libraries and frameworks. There are two key JavaScript frameworks used in this scenario, web3 as well as anchor.

10.2.1 web3.js

Web3.js [82] is a JavaScript library that provides developers with a comprehensive set of tools and functionalities to interact with Ethereum and similar Blockchain Smart Contracts such as the Solana's smart conrtacts. With Web3.js, developers can create, deploy, and interact with Smart Contracts, retrieve and manipulate Blockchain data, and execute transactions on the Solana network. It offers a user-friendly interface, extensive documentation, and a wide range of features, making it a popular choice among developers for building decentralized applications.

10.2.2 anchor.js

Anchor.js [72] is a JavaScript library designed specifically for developers working with Solana Blockchain applications. It serves as a powerful toolkit that simplifies the process of building decentralized applications on the Solana network. With Anchor.js, developers can easily interact with Solana's Smart Contracts, deploy and manage programs, and handle transactions seamlessly. Anchor.js abstracts away the complexity of Solana's low-level programming, allowing developers to focus on their application's logic and functionality. It offers various utilities and tools, such as state management, event handling, and program testing, to streamline the development process.

10.2.3 Types from Solana Smart Contracts

Finally a file with type definitions stemming from the Smart Contract compilation is imported which is used when downloading the values from the Solana Account in order to unpack received data from the chain.

10.3 Model training

To conduct our experiments, we employed Tensorflow's Keras high-level API [37] in Python to create models within the participating nodes. Tensorflow's Keras provides a user-friendly and intuitive interface for building and training neural networks, enabling efficient and streamlined model development. In our specific experiments, we chose to focus on testing network architectures for the MNIST-DIGIT dataset [45, 36].

By utilizing Tensorflow's Keras API, we were able to create and configure the neural network models within the nodes. This involved defining the layers, specifying activation functions, configuring loss functions, and selecting optimization algorithms. Tensorflow's Keras abstracts away the low-level details, allowing us to focus on the high-level design of the models and their training.

This provided a solid foundation for evaluating the efficacy and practicality of our Federated learning environment. The combination of a powerful machine learning framework and a well-established dataset allowed us to analyze the performance and convergence of the models within our Federated learning setup.

10.3.1 MNIST Digit

The MNIST-DIGIT dataset [45] is a widely used benchmark dataset in the field of machine learning and computer vision. It consists of a large collection of handwritten digit images, each labeled with the corresponding digit it represents. By selecting this dataset for our experiments, we aimed to evaluate the performance and effectiveness of our Federated learning setup in training models on a well-known and standardized dataset.

It is also realatively demaning in terms of the neurons required for this classification task.

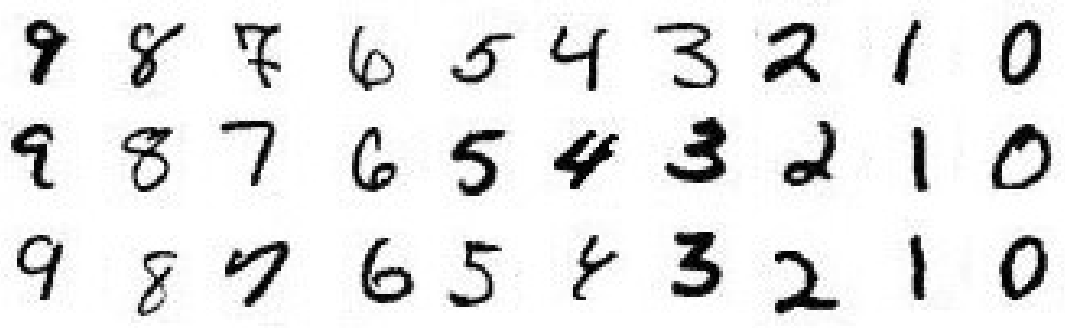

Figure 10.1: MNIST DIGIT [45]

10.4 Baseline 2000 Weight Deep Convolutional neural network

10.4.1 Neural net architecture

For our testing purposes, we employed a hybrid convolutional [59] deep learning approach as the model architecture. This approach incorporates both convolutional layers [62] and deep neural network layers, allowing for effective feature extraction and classification in the context of the MNIST-DIGIT dataset. It is important to note here that the test set made up of the 33% percent of the original data.

10.4.2 Convolutional network

The initial stage of the model architecture comprises two convolutional layers. The first convolutional layer consists of four filters, each with a kernel size of 7x7. This layer is designed to receive the 28x28 greyscale images as input. Following the convolution operation, a Rectified Linear Unit (ReLU) activation function is applied, enhancing the non-linear representation of the extracted features. The output of this first convolutional layer is then passed through the second convolutional layer, which comprises two 2x2 filters, each also utilizing a ReLU activation function.

After the second convolutional layer, the model proceeds with a maxpooling layer, which reduces the spatial dimensions of the features while retaining their essential information. Subsequently, a flatten layer is introduced to transform the pooled features into a vector representation, facilitating the transition from the convolutional layers to the fully connected layers of the deep neural network.

10.4.3 Deep network

The deep neural network component of the model consists of four fully connected layers. The first three layers contain 8, 8, and 4 ReLU neurons, respectively. These layers contribute to the abstraction and combination of features learned from the earlier convolutional layers. Finally, the output layer comprises ten softmax neurons, allowing for the classification of the input images into the ten different digits.

It is important to note here that the use of the softmax activation function is responsible

for much of the deep network's connections. Contrary to a regression neural network or even binary classification, multi-class classifications require more neurons in the final layers of the network.

```
model_2000 = keras.Sequential([
  layers.Conv2D(filters=4, kernel_size = (7,7), activation="relu", input_shape=(28,28,1)),
  layers.Conv2D(filters=2, kernel_size = (2,2), activation="relu"),
  layers.MaxPooling2D(pool_size=(2,2)),
  layers.Flatten(),
  layers.Dense(8, activation='relu'),
  layers.Dense(8, activation='relu'),
  layers.Dense(4, activation='relu'),
  layers.Dense(10, activation='softmax'),
])
```

Figure 10.2: KERAS MODEL

10.4.4 Loss function

During the training process, the categorical crossentropy loss function was used, which is well-suited for multi-class classification tasks like the MNIST-DIGIT dataset. This loss function measures the dissimilarity between the predicted probability distribution and the true labels of the input images, encouraging the model to accurately classify each digit [30].

10.4.5 Optimizer selection

To optimize the model's parameters, we utilized the ADAM optimization algorithm. ADAM combines the advantages of both adaptive gradient descent and root mean square propagation. It dynamically adjusts the learning rate for each parameter based on its past gradients, allowing for efficient and effective convergence during training [38].

10.4.6 Batch size and training epochs

To balance the trade-off between computational efficiency and model performance, we selected a batch size of 128. This means that during each training iteration, the model processed 128 images before updating its weights. Additionally, we allocated 30% of the available data as the validation set, which allowed us to monitor the model's performance on unseen data and make informed decisions regarding its generalization capabilities.

10.4.7 Evaluation

Throughout the training process, the model iterated through approximately 200 epochs. Eventually, the model achieved an accuracy of 93.37% on the test set, demonstrating its ability to accurately classify the MNIST-DIGIT dataset.

```
model_2000.evaluate(x_test,y_test,verbose=1)
619/619 [==============================] - 2s 2ms/step - loss: 0.2400 - accuracy: 0.9373
[0.2400197982788086, 0.9373232126235962]
```

Figure 10.3: KERAS Evalutation

The selection of this specific architecture was driven by the need to balance model performance with the limitations of Federated learning environments. In such Federated learning environments, it is common to encounter resource-constrained or low-powered devices that participate in the training process.

By opting for a model architecture that utilizes only 2000 weights, we aimed to lower that the computational and storage requirements of the participating devices. This smaller model size allows for more efficient training and inference processes on these devices, while still maintaining satisfactory performance.

10.4.8 Model extraction

Once the model training process is complete, the next step involves extracting the weights of the trained model from the Tensorflow framework. To prepare the weights for uploading to the Smart Contract, a crucial step is to linearize them. This process involves transforming the multi-dimensional weight tensors into a one-dimensional vector.

This is done using the provided TensorFlow instructions but is important to be performed identically in each computing node as if this is not the case then missmatched weights will be aggregated resulting in invalid models.

After the aggregated model has been trained and downloaded the inverse procedure must deserialize the weights properly in order to place them in the correct weight locations as to reproduce the trained model. This means that all participating nodes must share the serialization algorithm.

10.5 Model pruning

With the 2000 Weight model as a baseline we proceeded to implement the rest of the models through model pruning. Neural network pruning involves the removal of unnecessary weights or neurons from a trained model. The concept of pruning can be likened to pruning in agriculture, where unnecessary branches or stems are cut off from a plant to enhance its growth and efficiency. Neural network pruning offers several methods to reduce the size of a model [7].

10.5.1 Weight pruning

One approach is weight pruning, which involves setting individual parameters to zero. This process leads to the creation of a sparse network where unnecessary weights are eliminated. Consequently, the overall parameter count of the model decreases, resulting in smaller model sizes. By eliminating unnecessary weights, the total number of parameters in the model decreases, while the overall architecture remains unchanged [7]. This reduction in parameters can lead to smaller model sizes and improved computational efficiency without compromising the original network structure.

10.5.2 Neuron pruning

Another method of pruning involves removing entire nodes from the network. This approach aims to shrink the overall architecture of the network while striving to maintain the accuracy achieved by the initial larger network. By selectively removing nodes that contribute less to the overall performance, the pruned network can achieve a smaller size and potentially better efficiency while retaining its predictive capabilities [7].

10.5.3 Approaches used in our models

For pruning our models we tested both cases of model pruning in varying percentages of pruning in order to seek the optimal accuracy per lower weight count as well as neuron count.

We tested all possible sparcities starting from $25 - 99\%$ sparcities and then performed an in depth analysis in the $0 - 10\%$ range.

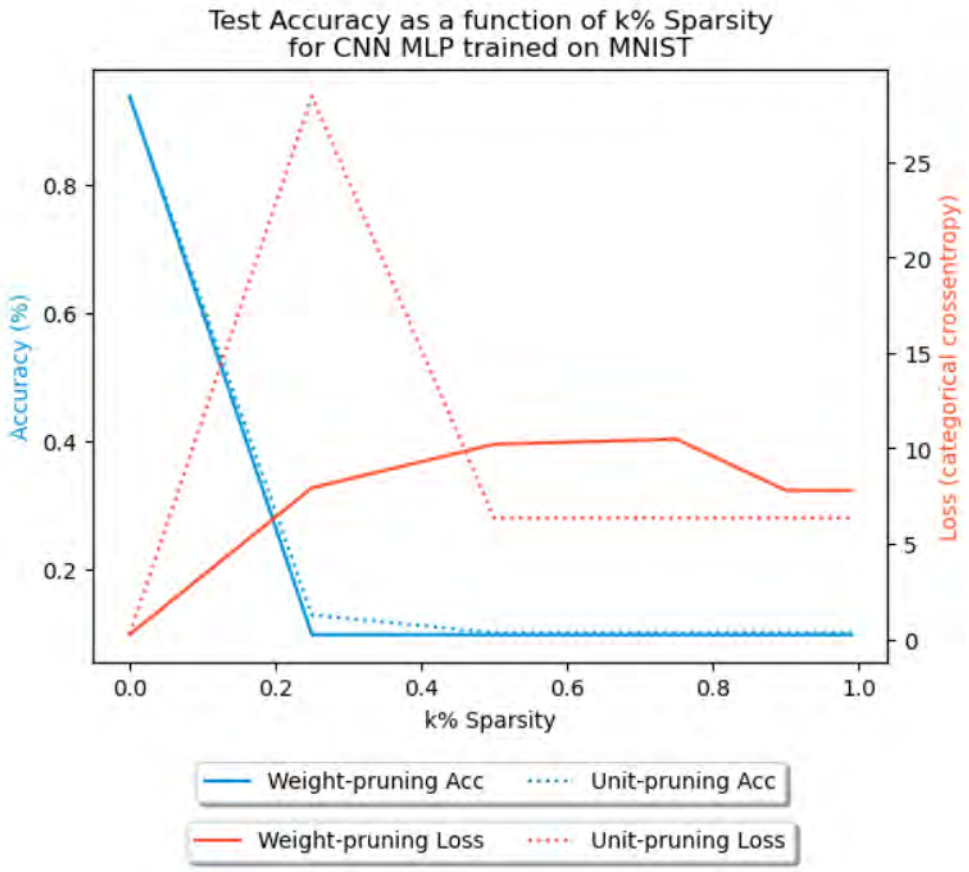

Figure 10.4: 25-100% sparcities

In our analysis, we noticed a significant and abrupt decline in model accuracy beyond the 25% pruning threshold. This decline was characterized by a sharp drop, resembling an elbow shape, in the accuracy curve. Concurrently, we observed a corresponding increase in model loss, indicating a deteriorating performance as the sparsity level exceeded 25% 10.4.

In contrast, when considering neuron pruning, we discovered that no pruning could be effectively performed until the sparsity level reached 25%. This is because lower sparsity percentages did not result in the removal of any neurons, as they were not substantial enough to warrant the cutoff of a single neuron. Consequently, the effects of neuron pruning on model performance were not evident until a certain threshold of sparsity was reached.

These observations highlight the delicate balance between sparsity and model performance. While higher sparsity percentages can lead to reductions in model size, it is crucial to consider the impact on usability and accuracy. Our findings indicate that maintaining a sparsity level below 5% for weight pruning and considering sparsity levels above 25% for neuron pruning can help strike a balance between model size reduction and preserving model functionality and accuracy.

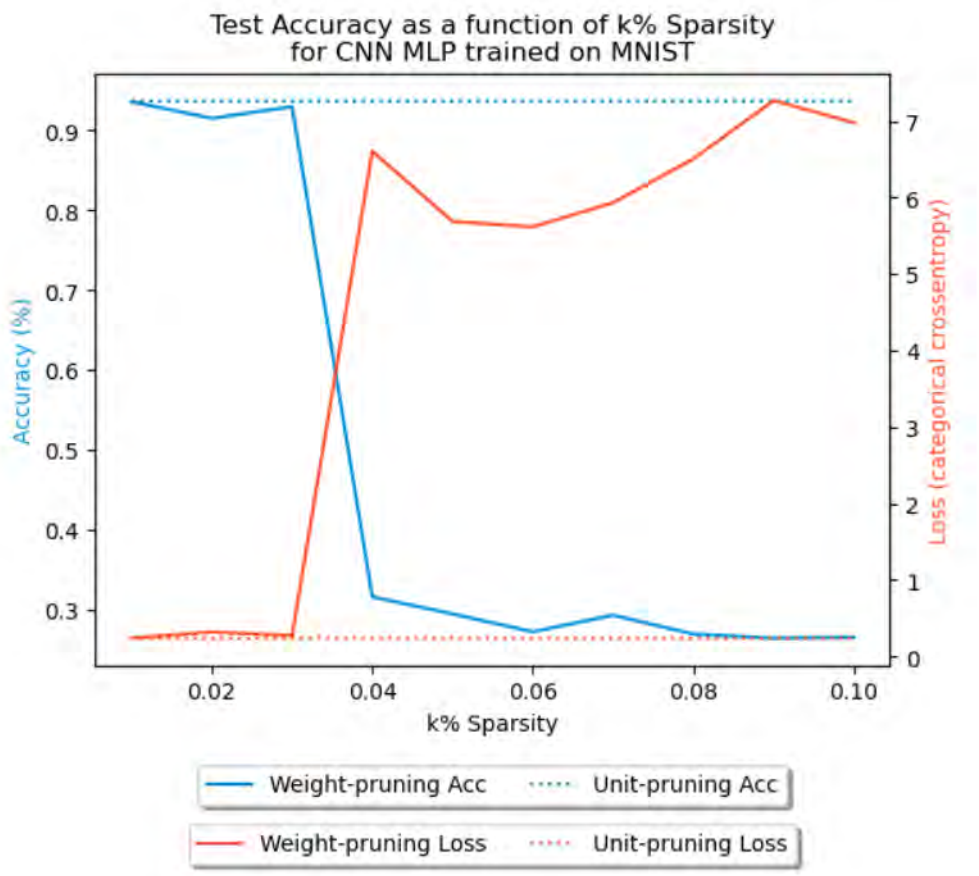

Figure 10.5: 0-10% sparcities

During our in-depth analysis in the 0-10% range, we noticed an interesting trend regarding the impact of sparsity on model performance. We observed that as the sparsity of the models increased beyond 5%, their usability began to decline gradually. Despite this decrease in usability, we found that the accuracies of the models remained relatively stable, ranging between 90% and 93%, when weight sparsity was introduced 10.5.

The trade-off between sparsity and accuracy is critical when applying pruning techniques. While higher sparsity can lead to reduced model size, there is a limit beyond which accuracy suffers significantly. Striking the right balance between sparsity and accuracy is crucial to ensure the model remains functional and performs well.

This testing indicated that in such small models neuron pruning is especially difficult as it imposes the risk of destroying the model. On the other hand weight pruning can be beneficial in small percentages as in such contraint scenarios even cutting off 3% of the 2000 weights discussed is major savings.

```
 MNIST Weight-pruning

k% weight sparsity:  0.0       Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.25      Test loss: 7.93247      Test accuracy: 09.69 %%
k% weight sparsity:  0.5       Test loss: 10.21363     Test accuracy: 09.69 %%
k% weight sparsity:  0.75      Test loss: 10.48046     Test accuracy: 09.69 %%
k% weight sparsity:  0.9       Test loss: 7.79420      Test accuracy: 09.69 %%
k% weight sparsity:  0.95      Test loss: 7.79420      Test accuracy: 09.69 %%
k% weight sparsity:  0.97      Test loss: 7.79420      Test accuracy: 09.69 %%
k% weight sparsity:  0.99      Test loss: 7.79420      Test accuracy: 09.69 %%

 MNIST Unit-pruning

k% weight sparsity:  0.0       Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.25      Test loss: 28.44310     Test accuracy: 12.76 %%
k% weight sparsity:  0.5       Test loss: 6.34623      Test accuracy: 10.00 %%
k% weight sparsity:  0.75      Test loss: 6.35078      Test accuracy: 10.00 %%
k% weight sparsity:  0.9       Test loss: 6.35078      Test accuracy: 10.00 %%
k% weight sparsity:  0.95      Test loss: 6.35078      Test accuracy: 10.00 %%
k% weight sparsity:  0.97      Test loss: 6.35078      Test accuracy: 10.00 %%
k% weight sparsity:  0.99      Test loss: 6.35078      Test accuracy: 10.00 %%
```

Figure 10.6: Pruning results 25-100%

```
 MNIST Weight-pruning

k% weight sparsity:  0.01      Test loss: 0.24248      Test accuracy: 93.58 %%
k% weight sparsity:  0.02      Test loss: 0.31916      Test accuracy: 91.51 %%
k% weight sparsity:  0.03      Test loss: 0.27478      Test accuracy: 92.97 %%
k% weight sparsity:  0.04      Test loss: 6.60784      Test accuracy: 31.60 %%
k% weight sparsity:  0.05      Test loss: 5.68814      Test accuracy: 29.46 %%
k% weight sparsity:  0.06      Test loss: 5.61678      Test accuracy: 27.23 %%
k% weight sparsity:  0.07      Test loss: 5.93384      Test accuracy: 29.31 %%
k% weight sparsity:  0.08      Test loss: 6.50550      Test accuracy: 26.94 %%
k% weight sparsity:  0.09      Test loss: 7.26891      Test accuracy: 26.43 %%
k% weight sparsity:  0.1       Test loss: 6.97960      Test accuracy: 26.55 %%

 MNIST Unit-pruning

k% weight sparsity:  0.01      Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.02      Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.03      Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.04      Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.05      Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.06      Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.07      Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.08      Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.09      Test loss: 0.24002      Test accuracy: 93.73 %%
k% weight sparsity:  0.1       Test loss: 0.24002      Test accuracy: 93.73 %%
```

Figure 10.7: Pruning results 0-10%

Chapter 11

Evaluation

In our evaluation of the proposed Federated learning system, our primary objective is to determine the optimal balance within the Smart Contract environment. This balance encompasses two key aspects: the number of participants involved and the capacity to store weights efficiently 11.1.

By varying the number of participants, we aim to understand the scalability and performance implications of the Federated learning system. We will examine scenarios with different participant sizes, ranging from a small proof-of-concept setting with a single participant to larger experiments involving a considerable number of participants.

Simultaneously, we will explore the capacity of the Smart Contract to store weights effectively. This entails analyzing the limitations and capabilities of the underlying Blockchain infrastructure in handling and managing a varying number of weights [111].

11.1 Methodology

Initially, we focused on testing four different models with different numbers of weights: 250 weights 11.3, 500 weights 11.4, 1000 weights **??**, and 2000 weights 11.6. By examining these models individually, we aimed to understand how the system performs with varying model complexities and sizes.

Then we explored the impact of participant numbers on the system's behavior and scalability, by starting with a single participant and gradually increased the participant count up to 1000 participants 11.3. This approach allowed us to assess the system's performance under different participant densities.

11.2 Testing

Table 11.1 represents the search space explored. The tests were conducted in terms of the Weight Vector T $\times$ num, the models and the number of participants k.

Weight Vector $T \times num$	250 $\times$ f64, 500 $\times$ u32, 1000 $\times$ u16, 2000 $\times$ u8
Models	Model 250, Model 500, Model 1000, Model 2000
Number of Participants k	1, 5, 10, 50, 100 ,1000

Table 11.1: Search Space

This table represents the max number of participants supported by each Smart Contract per model. The capacity of each Smart Contract was found to be influenced by the data type used to store the model.

To provide a comprehensive overview of the results obtained from all the experiments conducted, the following table summarizes the findings. This consolidated view allows for a quick comparison and analysis of the system's performance across various model and Smart Contract configurations 11.2.

Note that this evaluation is carried out in terms of the capacity of a single Smart Contract. This means that should multiple deployed Smart Contracts be introduced, the number of weights that are supported can be increased. This comparison is relevant as it seeks the optimum storage configuration in a single Smart Contract.

The subsequent tables provide a detailed breakdown of the individual experiments conducted for each model and Smart Contract configuration.

The experiments conducted revealed that the primary reason for experiment failure was the occurrence of an overflow in the aggregating values. This overflow led to the halt of further transaction processing and invalidated any subsequent model uploads. While the specific behavior varied depending on the model being tested, the general conclusions drawn from these results are applicable to any model [71].

Although smaller numerical values might allow for the addition of a few participants in the federated environment, such changes were deemed negligible. The focus of the experiments was primarily on assessing the order of magnitude in terms of the number of participants in the Federated learning system.

Maximum participants per Model				
	250 × f64	500 × u32	1000 × u16	2000 × u8
250 × f64	1000	5	1	1
500 × u32	-	5	1	1
1000 × u16	-	-	1	1
2000 × u8	-	-	-	1

Table 11.2: Maximum participants per model

Model 250				
	250 × f64	500 × u32	1000 × u16	2000 × u8
1	✓	✓	✓	✓
5	✓	✓	X	X
10	✓	X	X	X
50	✓	X	X	X
100	✓	X	X	X
1000	✓	X	X	X

Table 11.3: Tests conducted in 250 f64 model

Model 500				
	250 × f64	500 × u32	1000 × u16	2000 × u8
1	X	✓	✓	✓
5	X	✓	✓	X
10	X	X	X	X
50	X	X	X	X
100	X	X	X	X
1000	X	X	X	X

Table 11.4: Tests conducted in 500 u32 model

Model 1000				
	250 × f64	500 × u32	1000 × u16	2000 × u8
1	X	X	✓	✓
5	X	X	X	X
10	X	X	X	X
50	X	X	X	X
100	X	X	X	X
1000	X	X	X	X

Table 11.5: Tests conducted in 1000 u16 model

Model 2000				
	250 × f64	500 × u32	1000 × u16	2000 × u8
1	X	X	X	✓
5	X	X	X	X
10	X	X	X	X
50	X	X	X	X
100	X	X	X	X
1000	X	X	X	X

Table 11.6: Tests conducted in 2000 u8 model

Chapter 12

Discussion of Results

12.1 Results per model

12.1.1 250 Model

Among the model and Smart Contract configurations tested, the experiment using the 250 weights and 250 f64 model in the corresponding Smart Contract was the only one that encompassed a wide range of participant numbers, extending up to the thousands. This particular experiment warrants further discussion, which will be addressed later.

While the 250 f64 model fits within the other Smart Contracts, it is noteworthy that the 250 f64 Smart Contract, designed specifically for double precision floating-point numbers, provides greater capacity for aggregation. As a result, it is recommended to use the 250 f64 Smart Contract in scenarios involving this particular model 11.3.

These observations highlight the importance of selecting the appropriate Smart Contract configuration based on the specific model requirements, particularly considering the potential for overflow and the capacity for aggregation.

12.1.2 500 Model

This case is recommended for environments with low participant counts as it can only accomodate 1 to 5 participants for model aggregation. Even with 32-bit integers the system experienced overflows and failed to accommodate all transactions. It is important to note here that 1 to 5 participants can also be serviced with a private Blockchain approach 11.4.

12.1.3 1000 Model

Similar to the findings with the 250 f64 model, the experiments conducted with the 1000 u16 model revealed that it can only be stored on the Blockchain by a single participant. Although this outcome showcases the potential benefits of model auditability, immutability, and transparency, it does not create a practical Federated learning environment 11.5.

12.1.4 2000 Model

In the case of the 2000 u8 model, it also faced limitations in terms of storage within the Smart Contract. Even with the presence of two participants, it was not feasible to train the model simultaneously. Consequently, the 2000 u8 model cannot effectively utilize the Smart Contract for Federated learning purposes 11.6.

These observations highlight the importance of considering the model size and the limitations of the Smart Contract storage capacity when designing a Federated learning environment. While the implementation of certain models and Smart Contract configurations may offer advantages in terms of auditability and transparency, it is crucial to ensure that the chosen setup is capable of supporting multiple participants and facilitating collaborative model training effectively.

12.2 General Discussion

So taking everything into consideration trying the only viable Smart Contract execution is the 250 f64 model 11.2. This does not mean that we can only store 250 weights on the aggregating server. To accomodate larger models multiple Solana Smart Contract deployments can be employed. This while expensive and troublesome when deploying in the Solana devnet will increase throughput and further decentralize the federated model. Further expansion options will be discussed in part three though.

In parallel, our pruning analysis demonstrated that when employing Smart Contracts for model aggregation even the smaller model reduction can be important for storing on chain. In the MNIST model demonstrated through cutting of more than 3% of the model's weights rendered it useless. The pruning analysis for deployment in the Smart Contract must be done ad hoc per model trained as it will provide different results in various use cases.

Part III

Conclusions

Chapter 13

Discussion

The central question that this thesis aimed to address is Is Solana suitable for BCHFL? The short answer is that Solana is a very fast platform but its ecosystem is very immature for efficient development. It also deals with serious limitations in terms of the computational budget allowed per transaction which further constraints development efforts but is a common occurence in Blockchain platforms.

13.1 Challenges

The main challenges surrounding this thesis revolved arround the technical challenges in writing the Smart Contract itself and establishing communication to the Solana Blockchain. Neural network model development has seen tremendous adoption and iterative design to the point where it is trivial to create various architectures in seconds. Simultaneously technologies like JSON RPC and TypeScript web clients are mature enough where documentation is ample and accurate. However this is not the case with developing for Solana.

During the development process documentation for the Solana platform were severly limited with only the docs and a handful of blogs and repositories available for referencing. In terms of academic adoption only a single work from Duffy et al. [18] we found was accomodated with code with most papers about Solana referencing it in terms of its potential as a platform and not actually implementing code for it [5].

Another challenge was the limited available space which forced us to consider pruning methods in order to accomodate for our model storage. This may not be an issue when dealing with small scale models such as linear models but such testing would not be realistic.

13.2 Solana as a platform

As Solana evolves and gains wider adoption, efforts will be made to address and overcome the challenges it has faced. The development and deployment processes will be streamlined, making it more accessible for developers interested in web3 development. With time, Solana's ecosystem is expected to mature, similar to Ethereum's, and there will be a focus on implementing projects that are native to the Solana Blockchain.

However, it's important to acknowledge that Solana has encountered intermittent network disruptions and blackouts in the past, primarily driven by high transaction volumes and network congestion. These events have temporarily impacted the availability and reliability of the Solana Blockchain, raising concerns about its stability and potential limitations during periods of peak usage. It will be crucial for the Solana team to address these scalability and performance challenges to ensure a robust and resilient network infrastructure.

Simultaneously, Solana's high throughput and low latency capabilities make it well-suited for handling a large number of transactions and supporting real-time applications. Its unique architecture, leveraging a combination of Proof of History (PoH) and Proof of Stake (PoS) as discussed, enables fast transaction finality and scalability.

These promising features will incentivise developers with building decentralized applications (dApps) and interact with the Solana Blockchain.

13.3 Future Directions

In the context of Solana as a platform for Blockchain-based Collaborative Federated learning (BCHFL), there are several additional features that can be implemented to enhance security, privacy, and overall functionality.

One important aspect is the implementation of advanced security measures to identify and handle lazy or misbehaved participants. By incorporating mechanisms to detect participants who are not actively contributing or are behaving maliciously, the integrity and effectiveness of the Federated learning process can be preserved.

Furthermore, the integration of collateral mechanisms can add an extra layer of security and trust in the BCHFL system on Solana. Collateral requirements can ensure that participants have a vested interest in the accuracy and success of the collaborative learning process, discouraging any potential malicious actions or data manipulation.

To address privacy concerns, techniques like differential privacy and homomorphic encryption can be employed in Solana-based BCHFL. Differential privacy techniques can help protect sensitive participant data by adding controlled noise to the aggregated model updates, preserving individual privacy while still enabling meaningful learning. Homomorphic encryption can enable secure computation on encrypted data, allowing participants to securely share model updates without revealing their raw data.

To mitigate poisoning attacks, which involve injecting malicious or biased data into the Federated learning process, Solana can incorporate mechanisms that verify the integrity and authenticity of participant contributions. Techniques like anomaly detection and reputation systems can help identify and exclude participants who attempt to manipulate or poison the learning process.

The utilization of secret sharing schemes can further enhance privacy and security in Solana's BCHFL. By dividing sensitive model parameters or updates into multiple shares distributed among participants, no single participant has access to the complete information, protecting against data breaches or unauthorized access.

Tracking both local and global models can provide transparency and accountability in the Federated learning process on Solana. By monitoring the evolution of individual participants' local models and aggregating them into a global model, it becomes possible to ensure consistency, detect any deviations, and evaluate the overall progress of the learning process.

In a Proof of stake Blockchain like Solana incentives can be employed in order to reward diligence of participating nodes. Since Smart Contracts in Solana can hold coin and coin transactions are native to the platform, this can be done elegantly with Solana's native coin SOL.

Lastly, integrating the use of Shap values, which quantify the impact of individual features on model predictions, can enable tracking of effective changes made by participants' model updates. By rewarding participants based on their contributions and the effectiveness of their updates, Solana's BCHFL can incentivize active and valuable participation while maintaining fairness and accuracy.

Implementing these additional features in Solana's BCHFL ecosystem can enhance the security, privacy, and performance of the Federated learning process, making it a more robust and efficient platform for collaborative machine learning.

13.4 Future Research

n addition to implementing extra features within the Solana Blockchain, there are alternative approaches that can be employed to address the limited on-chain storage capacity and enhance the functionality of the Blockchain-based Collaborative Federated learning (BCHFL) system.

One approach is to leverage the integration of Solana with Arweave, a decentralized storage network. By utilizing Arweave's permanent and scalable data storage capabilities, the BCHFL system can offload the storage of large models or training data to the Arweave network. This allows for efficient utilization of on-chain storage within Solana while ensuring the availability and accessibility of the necessary data for collaborative learning.

Another approach is to integrate Solana with decentralized cloud platforms. By leveraging decentralized cloud solutions, such as platforms utilizing Blockchain or peer-to-peer networks, the BCHFL system can utilize distributed storage resources that are not limited by the on-chain storage capacity of Solana. This enables the system to efficiently store and retrieve large models or training data while benefiting from the scalability and fault-tolerance offered by decentralized cloud environments.

Furthermore, Solana's BCHFL can be effectively utilized in edge cloud environments. Edge cloud refers to the deployment of cloud resources closer to the edge devices or endpoints, reducing latency and enabling real-time processing. By leveraging Solana's high throughput and low latency capabilities, the BCHFL system can be deployed in edge cloud environments, allowing for efficient collaboration and learning directly at the edge devices. This enables Federated learning scenarios in resource-constrained environments, such as Internet of Things (IoT) devices or edge computing devices, without requiring extensive on-chain storage.

Finally Solana's cross-chain compatibility enables interoperability with other Blockchain networks and must be explored in the context of Federated learning. Through the implementation of bridging mechanisms, such as cross-chain bridges or protocols like Wormhole, Solana enables the seamless transfer of assets and data between different Blockchain ecosystems. This cross-chain compatibility expands the scope of Solana's capabilities, facilitating the integration of decentralized applications (DApps) and services from other Blockchains into the Solana network.

13.5 Concluding Remarks

This thesis aimed to investigate the development of a Federated learning aggregator in the Solana platform. Besides development it is in fact one of the most promising candidates for real time dApps and therefore Blockchain based Federated learning. Even though Ethereum has a more mature ecosystem, BCHFL efforts there remain proofs of concept as they are not high throughput enough for real deployment. Other efforts based on Hyperledger are not as fair as Hyperledger is a private Blockchain which brings significant performance improvements in itself but must be deployed in a trusted environment. While this can be the case and private Blockchains are applicable to Federated learning environments we focused on a public Blockchain as its security relies on the security of the Blockchain platform it is deployed on.

Bibliography

[1] Alladi, T., Chamola, V., Sahu, N., and Guizani, M. Applications of blockchain in unmanned aerial vehicles: A review. *Vehicular Communications 23* (2020), 100249.

[2] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A. D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S. W., and Yellick, J. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. Association for Computing Machinery.

[3] Antal, C., Cioara, T., Anghel, I., Antal, M., and Salomie, I. Distributed Ledger Technology Review and Decentralized Applications Development Guidelines. *Future Internet 13* (2 2021), 62.

[4] Asante, M., Epiphaniou, G., Maple, C., Al-Khateeb, H., Bottarelli, M., and Ghafoor, K. Z. Distributed Ledger Technologies in Supply Chain Security Management: A Comprehensive Survey. *IEEE Transactions on Engineering Management 70* (2 2023), 713–739.

[5] Ashraf, M., and Heavey, C. A Prototype of Supply Chain Traceability using Solana as blockchain and IoT. *Procedia Computer Science 217* (2023), 948–959.

[6] Awan, S., Li, F., Luo, B., and Liu, M. Poster: A Reliable and Accountable Privacy-Preserving Federated Learning Framework Using the Blockchain. Association for Computing Machinery, pp. 2561–2563.

[7] Bandaru, R. Neural network pruning, 9 2020. Date Accessed 08/06/2023.

[8] Batool, Z., Zhang, K., and Toews, M. FL-MAB: Client Selection and Monetization for Blockchain-Based Federated Learning. Association for Computing Machinery, pp. 299–307.

[9] Behera, M. R., Upadhyay, S., and Shetty, S. Federated Learning using Smart Contracts on Blockchains, based on Reward Driven Approach. *arXiv* (7 2021).

[10] Ben, W., Z. V., Kai, C., Liu, Y. Q. Y., and Tan. Federated Recommendation Systems, 2020.

[11] Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.* (2018), 46.

[12] Blaize. We secure defi smart contracts, 2021. Date Accessed 08/06/2023.

[13] BnB-Smart-Chain. BnB Smart Chain, 4 2020. Date Accessed 08/06/2023.

[14] Cheng, Y., Liu, Y., Chen, T., and Yang, Q. Federated Learning for Privacy-Preserving AI. *Commun. ACM 63* (11 2020), 33–36.

[15] Cui, S., Zhao, G., Gao, Y., Tavu, T., and Huang, J. VRust: Automated Vulnerability Detection for Solana Smart Contracts. ACM, pp. 639–652.

[16] Daniel Laine, S. B., and Sekniqi, E. G. S. K. Avalance Platform, 6 2020.

[17] Desai, H. B., Ozdayi, M. S., and Kantarcioglu, M. BlockFLA: Accountable Federated Learning via Hybrid Blockchain Architecture. Association for Computing Machinery, pp. 101–112.

[18] Duffy, F., Bendechache, M., and Tal, I. Can Solana's high throughput be an enabler for IoT? IEEE, pp. 615–621.

[19] Eyal, I., and Sirer, E. G. Majority is Not Enough: Bitcoin Mining is Vulnerable. *Commun. ACM 61* (6 2018), 95–102.

[20] Fabasoft. Digital contract management made easy., 1 2021. Date Accessed 08/06/2023.

[21] Fischer, M. J., Lynch, N. A., and Paterson, M. S. Impossibility of Distributed Consensus with One Faulty Process, 1985.

[22] Goodman, L. Tezos — a self-amending crypto-ledger White paper, 9 2014.

[23] Grigg, I. EOS - An Introduction, 7 2017.

[24] Guo, H., and Yu, X. A survey on blockchain technology and its security. *Blockchain: Research and Applications 3* (6 2022), 100067.

[25] Guru, A., Mohanta, B. K., Mohapatra, H., Al-Turjman, F., Altrjman, C., and Yadav, A. A Survey on Consensus Protocols and Attacks on Blockchain Technology. *Applied Sciences 13* (2 2023), 2604.

[26] Hewa, T. M., Hu, Y., Liyanage, M., Kanhare, S. S., and Ylianttila, M. Survey on Blockchain-Based Smart Contracts: Technical Aspects and Future Research. *IEEE Access 9* (2021), 87643–87662.

[27] Holovský, M. Comparison of public Blockchain platforms, 1 2021. https://medium.com/coinmonks/unhyped-comparison-of-blockchain-platforms-679e122947c1.

[28] Huang, X., Ding, Y., Jiang, Z. L., Qi, S., Wang, X., and Liao, Q. DP-FL: a novel differentially private federated learning framework for the unbalanced data. *World Wide Web 23* (7 2020), 2529–2545.

[29] Ioini, N. E., and Pahl, C. A Review of Distributed Ledger Technologies, 2018.

[30] Janocha, K., and Czarnecki, W. M. On Loss Functions for Deep Neural Networks in Classification. *arXiv* (2 2017).

[31] Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Eichner, H., Rouayheb, S. E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konecný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Qi, H., Ramage, D., Raskar, R., Raykova, M., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. Advances and Open Problems in Federated Learning. *Foundations and Trends® in Machine Learning 14* (2021), 1–210.

[32] Kalra, S., Goel, S., Dhawan, M., and Sharma, S. ZEUS: Analyzing Safety of Smart Contracts. Internet Society.

[33] Kang, J., Xiong, Z., Jiang, C., Liu, Y., Guo, S., Zhang, Y., Niyato, D., Leung, C., and Miao, C. Scalable and Communication-Efficient Decentralized Federated Edge Learning with Multi-blockchain Framework. Z. Zheng, H.-N. Dai, X. Fu, and B. Chen, Eds., Springer Singapore, pp. 152–165.

[34] Kang, J., Xiong, Z., Niyato, D., Xie, S., and Zhang, J. Incentive Mechanism for Reliable Federated Learning: A Joint Optimization Approach to Combining Reputation and Contract Theory. *IEEE Internet of Things Journal 6* (2019), 10700–10714.

[35] Karthika, V., and Jaganathan, S. A quick synopsis of blockchain technology. *International Journal of Blockchains and Cryptocurrencies 1* (2019), 54–66.

[36] Keras. Keras Datasets. Date Accessed 08/06/2023.

[37] Keras. Keras API. Date Accessed 08/06/2023.

[38] Kingma, D. P., and Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* (12 2014).

[39] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv* (10 2016).

[40] Krichen, M., Ammi, M., Mihoub, A., and Almutiq, M. Blockchain for Modern Applications: A Survey. *Sensors 22* (7 2022), 5274.

[41] Kumar, S., Dutta, S., Chatturvedi, S., and Bhatia, M. Strategies for Enhancing Training and Privacy in Blockchain Enabled Federated Learning. IEEE, pp. 333–340.

[42] Lamport, L. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM 21* (7 1978), 558–565.

[43] Lamport, L. Paxos made simple, 11 2001.

[44] Lamport, L., Shostak, R., and Pease, M. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst. 4* (7 1982), 382–401.

[45] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 2278–2324.

[46] Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine 37* (5 2020), 50–60.

[47] Li, Y., Chen, C., Liu, N., Huang, H., Zheng, Z., and Yan, Q. A Blockchain-Based Decentralized Federated Learning Framework with Committee Consensus. *IEEE Network 35* (12 2021), 234–241.

[48] Li, Z., Yu, H., Zhou, T., Luo, L., Fan, M., Xu, Z., and Sun, G. Byzantine Resistant Secure Blockchained Federated Learning at the Edge. *IEEE Network 35* (7 2021), 295–301.

[49] Lim, M. Breaking Down the Ethereum Yellow Paper, 2021. Date Accessed 08/06/2023.

[50] Lin, S.-Y., Zhang, L., Li, J., li Ji, L., and Sun, Y. A survey of application research based on blockchain smart contract. *Wireless Networks 28* (2 2022), 635–690.

[51] Liu, J., Huang, J., Zhou, Y., Li, X., Ji, S., Xiong, H., and Dou, D. From distributed machine learning to federated learning: a survey. *Knowledge and Information Systems 64* (4 2022), 885–917.

[52] Liu, Y., Ai, Z., Sun, S., Zhang, S., Liu, Z., and Yu, H. FedCoin: A Peer-to-Peer Payment System for Federated Learning, 2020.

[53] Lo, S. K., Liu, Y., Lu, Q., Wang, C., Xu, X., Paik, H. Y., and Zhu, L. Toward Trustworthy AI: Blockchain-Based Architecture Design for Accountability and Fairness of Federated Learning Systems. *IEEE Internet of Things Journal 10* (2 2023), 3276–3284.

[54] Lyu, L., Yu, J., Nandakumar, K., Li, Y., Ma, X., Jin, J., Yu, H., and Ng, K. Towards Fair and Privacy-Preserving Federated Deep Models. *IEEE Transactions on Parallel amp; Distributed Systems 31*, 2524–2541.

[55] Ma, C., Li, J., Shi, L., Ding, M., Wang, T., Han, Z., and Poor, H. V. When Federated Learning Meets Blockchain: A New Distributed Learning Paradigm. *IEEE Computational Intelligence Magazine 17*, 26–33.

[56] Martinez, I., Francis, S., and Hafid, A. S. Record and Reward Federated Learning Contributions with Blockchain. IEEE, pp. 50–57.

[57] Mazieres, D. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus, 2014.

[58] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-Efficient Learning of Deep Networks from Decentralized Data.

[59] Mrazova, I., and Kukacka, M. Hybrid convolutional neural networks. pp. 469–474.

[60] Mugunthan, V., Rahman, R., and Kagal, L. BlockFLow: An Accountable and Privacy-Preserving Solution for Federated Learning. *arXiv* (7 2020).

[61] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[62] O'Shea, K., and Nash, R. An Introduction to Convolutional Neural Networks. *arXiv* (11 2015).

[63] Peng, Z., Xu, J., Chu, X., Gao, S., Yao, Y., Gu, R., and Tang, Y. VFChain: Enabling Verifiable and Auditable Federated Learning via Blockchain Systems. *IEEE Transactions on Network Science and Engineering 9* (1 2022), 173–186.

[64] Qammar, A., Karim, A., Ning, H., and Ding, J. Securing federated learning with blockchain: a systematic literature review. *Artificial Intelligence Review 56* (2023), 3951–3985.

[65] Raphael, M. The Bitcoin whitepaper, explained and commented, 2020. Date Accessed 08/06/2023.

[66] Sayeed, S., Marco-Gisbert, H., and Caira, T. Smart Contract: Attacks and Protections. *IEEE Access 8* (2020), 24416–24427.

[67] Shayan, M., Fung, C., Yoon, C. J. M., and Beschastnikh, I. Biscotti: A Blockchain System for Private and Secure Federated Learning. *IEEE Transactions on Parallel and Distributed Systems 32* (12 2021), 1513–1525.

[68] Sheriff, K. Mining Bitcoin with pencil and paper, 2017. Date Accessed 08/06/2023.

[69] Shi, E. *Foundations of Distributed Consensus and Blockchains*, in progress ed., vol. Preliminary Draft. 2020. Date Accessed 08/06/2023.

[70] Short, A. R., Leligou, H. C., Papoutsidakis, M., and Theocharis, E. Using Blockchain Technologies to Improve Security in Federated Learning Systems. IEEE, pp. 1183–1188.

[71] Solana-Anchor-Documentation. Anchor error reference. Date Accessed 08/06/2023.

[72] Solana-Anchor-Documentation. Anchor high-level overview. Date Accessed 08/06/2023.

[73] Solana-Anchor-Documentation. Anchor accounts. Date Accessed 08/06/2023.

[74] Solana-Anchor-Documentation. Anchor constraints. Date Accessed 08/06/2023.

[75] Solana-Anchor-Documentation. Anchor program module. Date Accessed 08/06/2023.

[76] Solana-Anchor-Documentation. Anchor CLI. Date Accessed 08/06/2023.

[77] Solana-Anchor-Documentation. Anchor Rust code reference. Date Accessed 08/06/2023.

[78] Solana-Anchor-Documentation. Anchor space reference. Date Accessed 08/06/2023.

[79] Solana-Documentation. JSON RPC API. Date Accessed 08/06/2023.

[80] Solana-Documentation. Solana deploy a program. Date Accessed 08/06/2023.

[81] Solana-Documentation. Solana CLI. Date Accessed 08/06/2023.

[82] Solana-Documentation. Solana web3.js. Date Accessed 08/06/2023.

[83] Solana-Documentation. Solana programs. Date Accessed 08/06/2023.

[84] Solana-Documentation. Solana developing with Rust. Date Accessed 08/06/2023.

[85] Solana-Documentation. Solana accounts. Date Accessed 08/06/2023.

[86] Solana-Documentation. Solana runtime compute budget. Date Accessed 08/06/2023.

[87] Solana-Documentation. Solana runtime. Date Accessed 08/06/2023.

[88] Solana-Documentation. Solana transactions. Date Accessed 08/06/2023.

[89] Solana-Documentation. Solana test validator. Date Accessed 08/06/2023.

[90] Solana-Documentation. What is Solana? Date Accessed 08/06/2023.

[91] Solana-Documentation. Solana terminology. Date Accessed 08/06/2023.

[92] Solana-Documentation. Solana wallets. Date Accessed 08/06/2023.

[93] Solana-Seahorse-Documentation. Seahorse accounts. Date Accessed 08/06/2023.

[94] Solana-Seahorse-Documentation. Seahorse instructions. Date Accessed 08/06/2023.

[95] Solana-Seahorse-Documentation. The Seahorse language. Date Accessed 08/06/2023.

[96] Theymos. Fork, 2010. Date Accessed 08/06/2023.

[97] Uddin, M. A., Stranieri, A., Gondal, I., and Balasubramanian, V. A survey on the adoption of blockchain in IoT: challenges and solutions. *Blockchain: Research and Applications 2* (6 2021), 100006.

[98] Visa. Visa Statistics. Date Accessed 08/06/2023.

[99] Walker, G. How does Bitcoin Work ?, 2020. Date Accessed 08/06/2023.

[100] Wood, G. Ethereum: A secure decentralised generalised transaction ledger, 2014.

[101] Wu, X., Wang, Z., Zhao, J., Zhang, Y., and Wu, Y. FedBC: Blockchain-based Decentralized Federated Learning. IEEE, pp. 217–221.

[102] Xie, C., Koyejo, S., and Gupta, I. Asynchronous Federated Optimization. *arXiv* (3 2019).

[103] Yakovenko, A. Proof of History: A Clock for Blockchain. Date Accessed 08/06/2023.

[104] Yakovenko, A. Solana: A new architecture for a high performance blockchain v0.8.13, 2017.

[105] Yakovenko, A. High performance memory management for smart contracts, 8 2018. Date Accessed 08/06/2023.

[106] Yonatan, Ayelet, Z. A. S., and Sompolinsky. Optimal Selfish Mining Strategies in Bitcoin. B. G. Jens and Preneel, Eds., Springer Berlin Heidelberg, pp. 515–532.

[107] Yuan, S., Cao, B., Peng, M., and Sun, Y. ChainsFL: Blockchain-driven Federated Learning from Design to Realization. pp. 1–6.

[108] Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., and Gao, Y. A survey on federated learning. *Knowledge-Based Systems 216* (2021), 106775.

[109] Zhao, J., Wu, X., Zhang, Y., Wu, Y., and Wang, Z. A Blockchain Based Decentralized Gradient Aggregation Design for Federated Learning. I. Farkaš, P. Masulli, S. Otte, and S. Wermter, Eds., Springer International Publishing, pp. 359–371.

[110] Zhu, H., Zhang, H., and Jin, Y. From federated learning to federated neural architecture search: a survey. *Complex Intelligent Systems 7* (4 2021), 639–657.

[111] Zhu Yiming, Ehsan-ul Haq, L.-H. L. G. T. P. H. A Reddit Dataset for the Russo-Ukrainian Conflict in 2022. *arXiv* (7 2022).