FEDSMARTEUM: SECURE FEDERATED MATRIX FACTORIZATION

USING SMART CONTRACTS FOR MULTI-CLOUD SUPPLY CHAIN

A Dissertation
IN
Computer Science
and
Biomedical and Health Informatics

Presented to the Faculty of the University
of Missouri–Kansas City in partial fulfillment of
the requirements for the degree

DOCTOR OF PHILOSOPHY

by
SRINI BHAGAVAN

Kansas City, Missouri
2021

© 2021
SRINI BHAGAVAN
ALL RIGHTS RESERVED

FEDSMARTEUM: SECURE FEDERATED MATRIX FACTORIZATION USING SMART CONTRACTS FOR MULTI-CLOUD SUPPLY CHAIN

Srini Bhagavan, Candidate for the Doctor of Philosophy Degree

University of Missouri–Kansas City, 2021

ABSTRACT

With increased awareness comes unprecedented expectations. We live in a digital, cloud era wherein the underlying information architectures are *expected* to be elastic, secure, resilient, and handle petabyte scaling. The *expectation* of epic proportions from the next generation of the data frameworks is to not only do all of the above but also build it on a foundation of trust and explainability across multi-organization business networks. From cloud providers to automobile industries or even vaccine manufacturers, a typical supply chain consists of complex networks of suppliers, manufacturers, distributors, retailers, auditors, and consumers. With cloud providers, even though there is an increased focus on self-service or cloud provider-managed SaaS (Software-as-a-service), a portion of sales for an enterprise customer occurs the old-fashioned way with the sales department drawing up a purchase order for the procurement process. In many cases, there could be several disjoint, not fully digitized strings of suppliers behind the scenes. This

leaves the buyer unbeknownst and unaware of the state of their order in real-time as it is challenging to build machine learning and AI-based systems for order fulfillment, time and cost predictive models, share data transparently and issue remediations when multiple organizations are involved to fulfill an order.

In this dissertation, motivated by challenges in the industry, we propose a decentralized distributed system that can be considered as a building block for supply chain infrastructures, regardless of industry. The design goal of our system is to streamline complex non-repudiated transaction workflows by efficient handling of enterprise-scale purchase orders. We present transparent alternatives in real-time to customers based on model inference to respond to prediction requests. To further support this, we build a recommendation system model (Matrix Factorization) that is trained using Federated Learning on an Ethereum blockchain network. We leverage smart contracts that allow decentralized serverless aggregation to update localized items vectors. Furthermore, we utilize Homomorphic Encryption (HE) to allow sharing the encrypted gradients over the network while maintaining their privacy. Based on our results, we argue that training a model over a serverless Blockchain network using smart contracts will provide the same accuracy as in a centralized model while maintaining our serverless model privacy and reducing the overhead communication to a central server. Finally, we assert such a system that provides transparency, audit-ready and deep insights into supply chain operations for enterprise cloud customers resulting in cost savings and higher Quality of Service (QoS).

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the Graduate Studies, have examined a dissertation titled "FedSmarteum: Secure Federated Matrix Factorization
Using Smart Contracts for Multi-Cloud Supply Chain", presented by Srini Bhagavan, candidate for the Doctor of Philosophy degree, and certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Praveen Rao, Ph.D., Committee Chair
Department of Computer Science & Electrical Engineering

Jenifer Allsworth, Ph.D., Co-Discipline Advisor
Department of Biomedical & Health Informatics

Yugyung Lee, Ph.D.
Department of Computer Science & Electrical Engineering

Ghulam Chaudhry, Ph.D.
Department of Computer Science & Electrical Engineering

Deep Medhi, Ph.D.
Department of Telecommunications & Computer Networking

CONTENTS

ILLUSTRATIONS

TABLES

LISTINGS

ALGORITHMS

ACKNOWLEDGEMENTS

It does and did take a village. There were numerous people that were involved in guiding and helping me achieve my goal. My village starts with my parents *Prof. Sree Krishna Bhagavan* and *Ananda*, wife *Harini*, sons *Druv* and *Mayan*, sister *Padmini* and brother *Yateesh*, and my entire bedrock of extended family and friends. None of this would have been remotely possible without all of them showering me with their love, sacrifice, support, and their relentless belief in my quest to fulfill my dream.

I would like to express my sincere gratitude to my inspiring advisor, **Dr. Praveen Rao**, for his guidance, support, and motivation. It was a great honor and privilege to work directly under his supervision. He is simply the best advisor and guide any aspiring student can wish to seek.

Thank you **Dr. Allsworth** for helping me navigate my interdisciplinary journey, **Dr. Lee** for being there and guiding me every step of the way for Computer Science, **Dr. Ghulam Chaudhry** and **Dr. Medhi** for your encouragement right from day one of the program. Thank you to all the extraordinary Computer Science and DBHI Professors for believing in me and the trust you bestowed upon me as I worked through all my coursework, qualifiers, comprehensive, research, and defense. I will cherish all of this deeply in my heart as I reflect fondly upon my Ph.D. journey.

I would also like to acknowledge UMKC for all the support and awarding me the Big Data Fellowship, AFRL for graduate research scholarship, and NSF CBL grant (1747751) for the support. Finally, thank you to IBM, my employer, and to all my esteemed colleagues for encouraging and supporting me throughout the course of my Ph.D.

CHAPTER 1

INTRODUCTION

1.1 Introduction to Blockchain

Access to the Internet has now become an entitlement in most countries and a critical part of our daily lives. By leveraging innovations in the Internet of Things (IoT) and scalable data systems [50, 51], a plethora of new business applications can emerge in the coming years. These applications can impact domains such as government, finance, trade, commerce, education, and online engagement. Technology experts in the field of distributed computing expect blockchain technology to make waves similar to the ones we saw at the advent of the Internet. In simple terms, a blockchain is a decentralized, distributed system wherein copies of an immutable and secure time-stamped ledger are held by multiple participating parties. With data being shared and accessed by all participating entities, it lends itself to being more trustworthy and unfettered, as there are no brokers in the middle or need for a central authority. All the top technology players have been embracing and nurturing blockchain technology and have been very careful not to marginalize it as part of the larger infrastructure. With the promise of blockchains in any reliable and auditable transaction application, there has been a lot of research and development around "smart contracts" whose use makes distributed computing possible. Essentially, a smart contract is a computer code that runs on top of a blockchain and allows parties to securely execute transactions based on certain agreements.

Gartner predicts that by using blockchains and smart contracts, *companies will increase overall data quality by 50% by 2023*. Furthermore, they expect transparency, granularity, and quality in decision making (with analytic systems) will increase tremendously as data availability will also increase by 30%. As blockchain technology has entrenched over the past few years, there seem to be a renaissance in business-to-business (B2B) scenarios. Entirely new business models have emerged forging ahead distributed ledger technologies (DLT) and data-driven digital networks. Blockchains solve several business problems that exist today in DLTs. Examples include: (a) How does one identify and inventory distributed datasets? (b) What about data governance to fulfill regulatory compliance to reduce operational risks? This is where smart contracts are useful. A smart contract facilitates transactions between interested parties in the network when specific events are triggered. Smart contracts add tremendous power to a blockchain by making it transaction-worthy and thereby, enabling its widespread adoption.

Blockchain technology can also be used to secure smart cities. In recent decades, the world has experienced unprecedented urban growth due to population increase, limited resources, and climate changes. Research shows that more people live in cities (54%) compared to people who live in rural areas (46%), and this percentage will increase up to (66%) by 2050. To cope with the huge increase in population, smart cities are aiming to implement and utilize modern technology. So far, the advancements of IoT and wireless have made it easy to interconnect remote devices. Such devices were distributed publicly in different locations, and for this reason, such devices should be able to block security attacks. There is a strong case to be made to use blockchain networks as a solution for such

systems because they contain a ledger that records all the transactions and operations and is secure since the attacker has to compromise 51% of the network to enforce the hashing power of the target network. Additionally, by utilizing smart contracts, we can manage such transactions without an intermediary [13].

1.2 Blockchain as service

Blockchain technology is a distributed database of records or shared public/private ledgers with smart contracts of all digital events that have been executed and shared among blockchain participants [11, 69]. Blockchain has gained recent popularity in supply chains due to several network-organic features, the fundamental ones being *Provenance*, *Consensus*, *Immutability* and *Finality*. By providing cryptographic proof of a set of transactions, blockchain gets us one step closer to building trust in a system providing secure audit trails. Typically complex order fulfillments are facilitated by businesses participating in a heterogeneous network of geographically dispersed suppliers, customers, regulators, lending institutions, and other supporting entities. This has the potential for disparities to quickly escalate [33] as evidenced in recent COVID-19 supplies and vaccine distribution. It exposed fragility in our vaccine supply chain with 72% of active ingredients manufactured outside the United States [15]. The enterprise cloud supply chain has similar challenges. The cost of procuring a Hybrid Cloud (a system consisting of private and public cloud services) can be in the order of several million dollars. Hence, suppliers need to be incentivized to invest in the overall health of the supply chain network in addition to just optimizing locally. Solving such a gamut of challenges are pivotal in cloud

resource procurement and deployment supply chain orchestration [15].

Large enterprises strategically deploy a multi-cloud solution to build cloud-native containers [17,68]. This means the entire process of ordering, procuring, deploying, set up cloud resources across multiple cloud vendors and platforms based on their information architecture needs is a combinatorial explosion problem at the very least as shown in Figure 1.

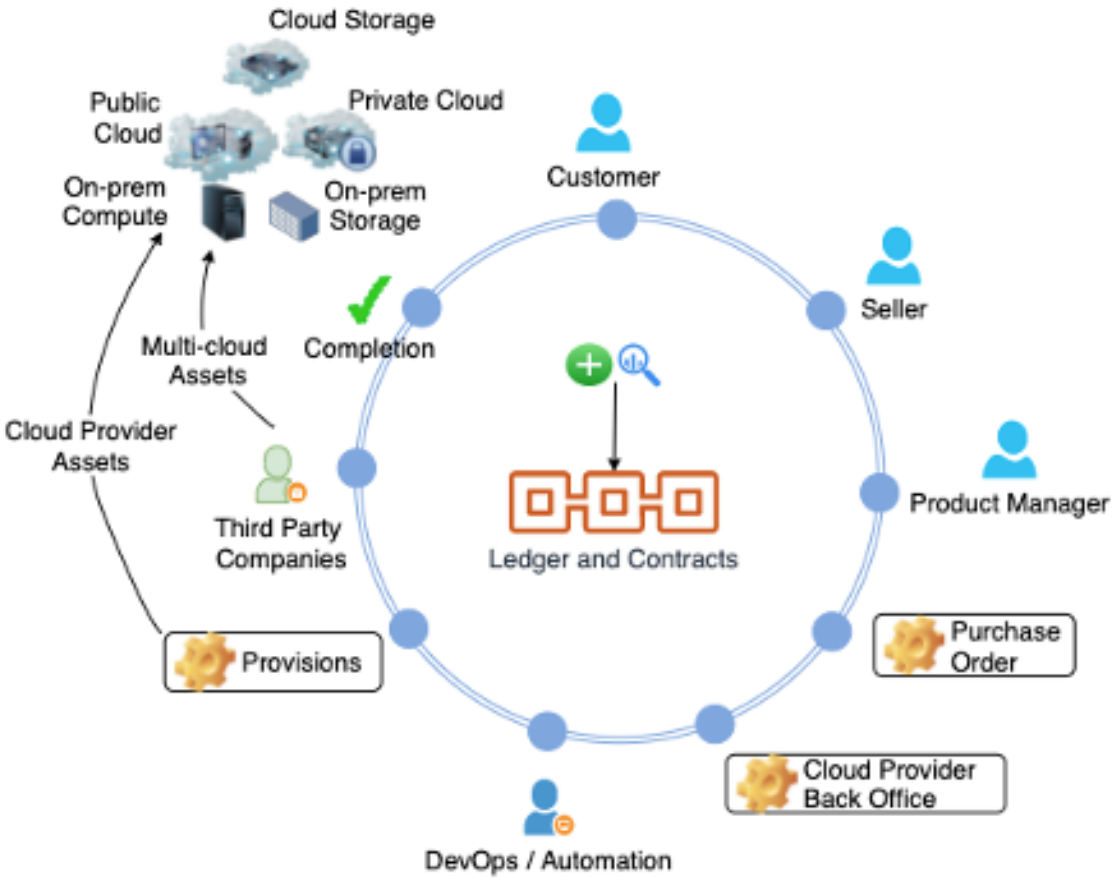


Figure 1: Hybrid cloud and multi-cloud resource procurement supply chain.

In addition to multi-cloud, hybrid deployments are prevalent [70] as customers

may be in various stages in their cloud journey. Some critical applications may never move to public clouds but will require cloud packaged software to run on their private in-house cloud environments. This is also observed as part of our practical experience dealing with such scenarios.

1.3 Blockchain and Machine Learning

Blockchain with its shared ledger represents a single system of cryptographic truth, tamper-resistant audit logs, and algorithmic trust, all of which provide the foundation that AI-based complex supply chains need. Blockchain has established itself as a disruptive technology enabling organizations to reinvent, cross borders and collaborate with their business network in previously unimaginable ways. Proof helps engender trust, and blockchains provide irrefutable cryptographic proof of transactions for secure audit trails and thereby, promote trust in the overall system [7]. Figure 2 illustrates the use of Merkle trees to encode blockchain data to be more secure as every parent node is labeled with the cryptographic hash of its child nodes. Smart contracts (autonomous contracts or chain code) are a core fabric component that enables automation across a multi-organization trusted network and its algorithms define the life cycle of one or more business objects.

Multi-organization enterprise customers looking to modernize their automation stack and gravitating toward a *consume anywhere* architecture with a hybrid cloud framework to address the dynamic nature of their businesses and in doing so, they also do want to not worry about accountability, privacy, scalability, and finality of transactions. Recent

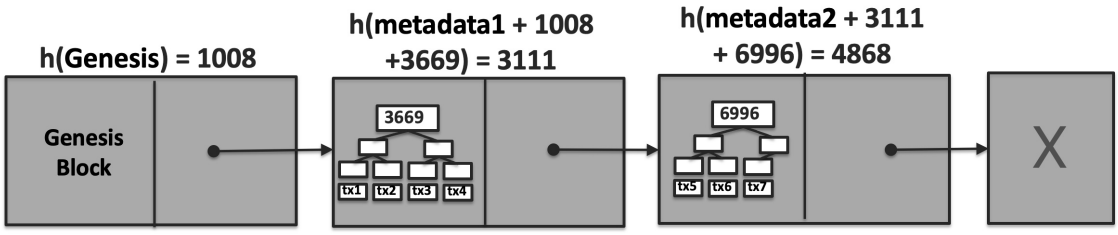


Figure 2: A Hash tree or Merkle tree combined with hash chain promotes efficient search within a block - O(log N)

advances in enterprise blockchain have not only entrenched these enterprise attributes but also proved the potential applicability of blockchain is far greater than just the ubiquitous cryptocurrency. It encompasses a vast range of industries including building a very robust supply chain where it is important to understand data provenance, how goods are sourced, how ethical is the process, how easy is to log, track and prove exceptions and thereby shorten reconciliation cycles. It is imperative for such a supply chain to have the system be fault-tolerant. With blockchain fault-tolerant consensus algorithms, the network continues to operate even in the presence of malicious or careless participants. Just like any established network, they need to be governed member access permissions that are regulated based on business needs i.e., permissioned does not mean private.

1.3.1 Why Federated Learning?

Businesses continue to burn the midnight oil to protect and preserve security and privacy [24, 40]. However, the effectiveness of AI depends on access to all the data we generate. Motivated by growing private data concerns, as well as the explosion in computing capabilities that end-users possess, a new machine learning paradigm called Federated Learning is gaining popularity. Federated Learning performs Machine Learning by first

training a model at each client-side privately and confidentially and subsequently collects and averages models from all participating clients to generate a more generalized model Figure 3 [23, 47].

There are multiple hospitals represented, with each hospital having a different number of x-ray images, perhaps representing different demographics (general vs pediatric vs geriatrics vs cancer specialties, etc.,). The hospital in the bottom right corner has only 3 x-ray images (in real life there could be a few hundred or thousands) but they may not be enough to train a model. For example, if a hospital wants to train a model to predict 2 types of cancers, but most of the cancer x-ray images they have for one type of cancer, then the model will not learn how to predict the second type of cancer accurately. Therefore, these 4 hospitals in the diagram collaborate together in order to train a powerful model that has been trained on different types of cancers. So each hospital provides the x-ray images they have. The problem is that they are not allowed to share these images with each other (privacy concerns). The solution is to add a trusted central server (which does not have access to the images either). This server's job is to create a model and send it to all the collaborated hospitals. Each hospital will then take the model and train it on their own dataset then return the model to the server. The central server will then average all of these models to create a global model. This global model is a powerful model that has been trained on all of the hospital's images while maintaining each hospital's privacy.

Since none of the clients share their confidential data, it solves multi-organization data privacy challenges [31]. Healthcare and user's edge devices are two of the main categories that use Federated Learning to train models. However, when deployed to solutions

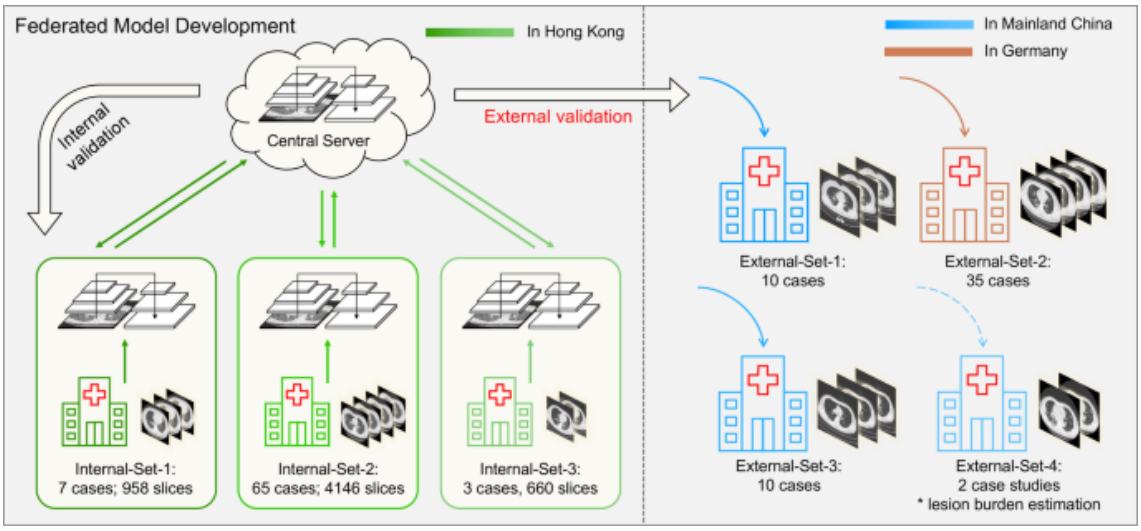


Figure 3: Federated Learning implemented between three hospitals

across multi-organizations current Federated Learning architecture has a few challenges such as excessive communications, scalability, and data leakage.

1.4 Key Contributions and Novelty

In our work, we propose a decentralized blockchain network and smart contracts to allow transparent, audit-ready, immutable, and trust-based collaboration between multiple cloud providers and vendors as part of fulfilling a complex customer purchase order. Our target was to improve the enterprise customer's Quality of Service (QoS) attributes when deploying and maintaining a multi-cloud solution. We further intend to delight their experience by providing fabric to share accurate supply chain insights from multiple parties by implementing a *decentralized* (instead of *centralized*) federated learning matrix factorization model using Ethereum blockchain.

The evolution of novelty is shown in Figure 4 and key contributions of our work

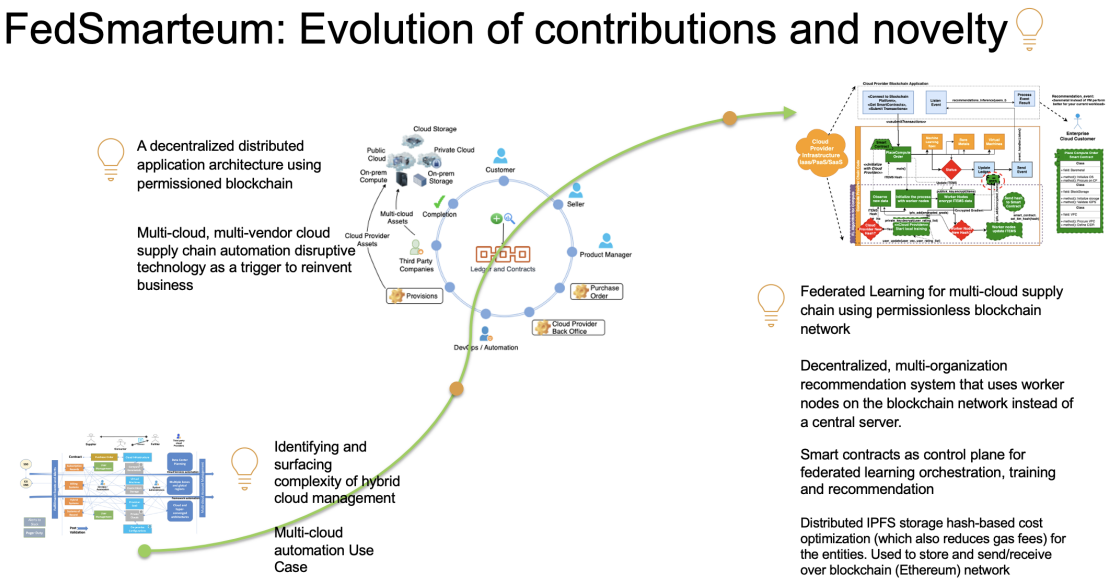


Figure 4: Evolution of Key FedSmarteum Contributions.

are as follows:

- A decentralized trusted architecture to overcome the centralized federated learning single server challenges such as the single point of failure and the excessive communications between a single server and the rest of the nodes.

- Smart contracts to orchestrate enterprise customer purchases of multi-cloud assets deployed on a permissioned blockchain network. The novelty of the solution also lies in using smart contracts as the control plane to integrate the Federated Learning algorithm to recommend cloud provider resources based on (similar) customers usage patterns.

- Decentralized architecture protects users from a trusted but curious server. The worker nodes carry out the updates using the encrypted data and gradients.

- Federated learning implementation on blockchain network allows collaboration between several parties to generate a global model even when if there is no trust between the included parties.

- Our system can be applied publicly to provide customers and third parties with insights, future predictions, and recommendations while training the model on the localized data without violating their privacy.

- Decentralized recommendation system that uses worker nodes on the blockchain network to train and update items vectors instead of a central server.

- Distributed IPFS storage hash-based cost optimization (which also reduces gas fees) for the entities. We used IPFS to store, send, and receive user's HE gradients over the blockchain (Ethereum) network.

CHAPTER 2

BACKGROUND

2.1 Blockchain

Blockchain by itself does not entirely encompass the space of what is called distributed ledger technologies which record and maintain anything of value. It is interesting to note that there are actually other ways of achieving this distributed consensus idea that is done slightly differently. In blockchain, there are mainly two types of blockchain networks, **permissioned** and **permission-less** networks [4]. In general, the difference between these two types is obvious from their names. The permission-less blockchain is the network that requires no permissions to participate in the chain which is the main type that was outlined by *Satoshi Nakamoto*. All general blockchain networks are considered to be permission-less chains which is another way to say *public*. Anyone can join the permission-less blockchain. However, it tends to be a bit slower than the permissioned ones. All transactions stored on permission-less networks are usually validated by the public without any third party. On the other hand, permissioned blockchains can be considered private blockchains as well. Joining a permission blockchain may require the permission of the owner of the chain. Moreover, validating the stored data may also be done by the owner and the responsible team. Permission and permission-less blockchain share several similarities [66] such as:

- Both permission and permission-less blockchains are immutable which means both

of the networks will not allow modifying or altering the stored data without having sufficient power over the network.

- Both are distributed ledgers which means having different versions of the same data that is stored in different places and connected through the same network.
- Both networks use a consensus mechanism which means multiple versions of the ledger to agree on what all should look like.

While the permissioned blockchains can have access control, high customizability, and better scalability, it may also have some drawbacks such as the security that depends on its members and it is less transparent from the permission-less blockchains.

2.2 Consensus in Blockchain Networks

2.2.1 Consensus Algorithms

Blockchains are known for their immutability and reliability. In blockchain networks, there is no central system of authorization, rather it is a decentralized network where all the peers in the network agree to validate the state of the distributed ledger. This type of verification where all the peers agree on the same operation can be done with the help of *consensus protocol* which are part of blockchain networks. In general, consensus algorithms help in building trust and reliability between the peers within a network. In particular, these consensus algorithms validate each block before they get added to the network. Each block has to be the one and only version of the block that has been agreed upon by all the peers [48].

Consensus algorithms work in a way that allows each node in the network to have the same priority among all the other nodes. In other words, consensus algorithms maintain some agreements and collaboration between all the peers to maintain the peers' rights in the network. A list of the most common consensus algorithms:

- *Proof of Work* (POW) relies on selecting a miner for the generation of the next block. The idea is to solve mathematical questions and give out the solution. Such questions and puzzles require a lot of computational power. Therefore, the node that solves the puzzle first gets to mine the next block. Bitcoin is an example network that uses POW.

- *Proof of Stake* (POS) Ethereum uses POS as a consensus algorithm instead of POW. In POS, the investment happens in the coins rather than the computational power. The validators invest by locking up their coins as a stake. Then validators start betting on the blocks that can be added to the network and the reward comes based on their performance.

- *Proof of Capacity* (POC) relies on the validators investing in their hard disk drive rather than investing in the computational power. The more space a validator has, the better chance they get for being selected to mine the next block.

- *Proof of Elapsed Time* is a consensus algorithm that was proposed by Intel where the miners will be randomly chosen based on random waiting time. PoET is similar to POW wherein both miners have to solve a hash problem, but in PoET it is done with significantly lower energy consumption.

There are many other consensus algorithms such as *Proof of Burn* (POB), *Proof of Importance* (POI), *Delegated Proof of Stake* (DPoS), *Practical Byzantine Fault Tolerance* (PBFT), etc., that are used by various networks to achieve specific objectives outlined by the consensus algorithms.

2.2.2 Consensus in Permissioned and Permission-less Networks

The previous section explained how the blockchain network allows members to participate in the verification process to verify blocks. However, this process in permissioned networks is a little bit different than permission-less networks. In permission-less networks (the public networks), anyone can join, participate in the consensus process, and mine for the next block. Bitcoin and Ethereum are two of the biggest permission-less network examples.

Permission-less networks, as discussed earlier, provide better decentralization than permissoned networks. Therefore, all the peers in the network have to agree upon the verification and the trust between each other. However, permissioned networks are different. Permissioned and permission-less share the same characteristics such as immutability and decentralization, but permissioned networks are owned by private sectors that may have different type of consensus. Generally, in permissioned networks, the owner verifies and decides who can join the network. Furthermore, the owner can assign a few members to help in verifying and managing the consensus. *NXXTECH* is an example of permissioned networks that use Proof of Authority (POA) as a consensus protocol [29].

2.3 Hyperledger

2.3.1 Hyperledger

Hyperledger is a hub for blockchain development. Official hyperledger defines it as *Hyperledger is an open-source collaborative effort created to advance cross-industry blockchain technologies. It is a global collaboration, hosted by The Linux Foundation, including leaders in finance, banking, Internet of Things, supply chains, manufacturing, and Technology [1].*

The Hyperledger hub or platform is mainly for blockchain technology. Here is a list of the most common Hyperledger projects:

- Hyperledger Fabric is an IBM project that was designed for the purpose of large-scale blockchain applications with some flexibility in permissions. Hyperledger IBM is provided as a service that is easy to start and implement [34, 36–38].
- Hyperledger Iroha is another project that was mainly designed to keep things simple and easy to incorporate into infrastructure, IoT devices, and other applications.
- Hyperledger Sawtooth is a project developed by Intel that uses a consensus algorithm called Proof of Elapsed Time (PoeT). Additionally, there are many other Hyperledgers such as Hyperledger Composer, Explorer, Burrow, Cello, etc.

Libra [43] is a permissioned blockchain digital currency that was proposed by Facebook. Currently, the currency is not fully released, only some of the experimental code was released. Libra, the decentralized blockchain database was designed mainly

for the volatility in cryptocurrency with the ability to serve as an exchange medium for billions of people around the world. Libra blockchain was meant to create a huge infrastructure to foster innovation and increase access to financial services. *Libra Core* was implemented to validate Libra. Since Libra is a permissioned blockchain, it will not rely on cryptocurrency mining, rather only authorized members from Libra are allowed to validate and process the transactions.

2.3.2 Blockchain-as-a-Service (BaaS) Offering

BaaS offering started recently due to the increasing adoption of blockchains in the marketplace. BaaS offering is about third-party businesses installing blockchain networks technologies to help in developing and maintaining blockchain networks on cloud service providers. BaaS has grown really fast recently and gained a lot of attention since it resolves most of the complex work around the network topology and makes it easier and faster for end-users to start building the DApps without focusing on the underlying infrastructure [20]. BaaS resolves many of the issues related to transparency, efficiency, and cost. Most companies choose to use BaaS to meet user requirements in a faster way. Moreover, using a cloud-based solution such as Blockchain-as-a-Service by governments and individual users will further limit the need to manage their data and networks and ensure all security controls are in place [46]. Currently, there are many companies that offer such services such as *DRAGONCHAIN, BLOQ, FACTOM, SYMBIONT, BLOCKSTREAM*, etc. However, since blockchain is a growing and never-ending network, it cannot be offered as Blockchain-as-a-Product (BaaP). Therefore, most of the

BaaS providers offer to make it easier for users, business owners, and companies to start using the database directly by offering different services [73] such as:

- Installing the infrastructure to simplify writing to the database.
- Installing traceable chain solutions for auditing.
- Decentralization for identities for single sign-on.

2.3.3 Bitcoin

The idea of *Bitcoin* as digital currency is also not a new idea. Satoshi Nakamoto proposed a more reliable decentralized architecture and reignited the space [53]. From a technical perspective, we can think about the Bitcoin ledger as a **State Transition Function** where the input is a state and a transaction and the output is another state. Generally, in the banking system, the input state is the current balance and the transaction is sending some amount of money ($X) from party A (Alice) to party B (Bob). When this transaction happens, the state function reduces ($X) from Alice and adds the same amount to Bob. However, it also checks if Alice has ($X) even before the transaction takes place. If Alice does not then the state returns an error. This can be easily expressed as in Algorithm 1.

In Bitcoin, the **state** is the collection of coins that a user has before spending them.

2.3.4 Ethereum

Ethereum is one of the cryptocurrencies platforms that provides a global computing platform which is known as Ethereum Virtual Machine (EVM) [12]. Programmers

Algorithm 1 Bitcoin Transaction

function TRANSFER(S, R, A)
    for Each r in R **do**
        if sender is not S **then**
            return Error
        end if
        if $\{Signature_s\}$ does not match $\{Signature_r\}$ **then**
            return Error
        end if
        if $\{Alice_{balance}\} < A$ **then**
            return Error
        end if
    end for
    return Successfully sent for all recipients
end function

use different languages in order to write JavaScript-like code such as Solidity which is one of the most common languages used for Ethereum and smart contracts. Smart contracts include different services such the computational power, online storage, network bandwidth, etc. Some current proposals are targeting the interaction in the physical world transactions as well. This is another level of smart contracts that may include smart management and ownership.

2.3.5 Corda

Corda is a permissioned decentralized ledger framework that uses the pluggable method of consensus. Corda is the only open-source blockchain/distributed ledger platform built for business [19]. Corda enables businesses to transact directly and in strict privacy using smart contracts, reducing transaction and record-keeping costs and streamlining business operations. Corda is partially decentralized and specifically designed for financial applications [64].

2.3.6 Iota

Iota is a distributed ledger that uses Directed Acyclic Graph (DAG) instead of a blockchain. This protocol is known as *Tangle*. Iota was mainly designed for IoT applications. In fact, Iota is the first cryptocurrency that was designed for IoT applications purposes. It was designed in a way to minimize the network overhead and the transaction time. Iota does not have a transaction fee and it uses cumulative weight as a reliability indicator [64].

2.3.7 Developer Tools and environments

Blockchains and smart contracts have gained a lot of attention in the last few years due to their decentralized architecture. These applications have been mushrooming in many application areas. In this section, we discuss popular tools, environments, platforms [3] and programming languages [4] used to build smart contracts in a blockchain network.

a) Platform and Tools

- Ethereum: *Vitalik Buterin* released the Ethereum whitepaper in 2013. Since then, Ethereum has grown and popular very fast. Vitalik's vision was to allow developers to code their own decentralized DApps on blockchain and Ethereum is the supercomputer that allows renting out the computational power, gas, and other resources. Ethereum started with the consensus algorithms *Proof-of-Work* (POW) following Nakamoto's mechanism. However, Ethereum plans to move to *Proof-of-Stake* (POS) where all the mining process is virtual and the miner's hash is proportional to their stake [49].

- EOS: EOS is originated from the software EOSIO that was built by *Block.one*. Ethereum is the first platform for smart contracts. However, it is very slow managing 15-20 contracts/second. Therefore, there was a need for another platform that is faster and supports more modern DApps and that is the reason behind EOS. EOS aims to support large-scale DApps by using decentralized operating systems rather than decentralized supercomputers like Ethereum. EOS users own resources in exchange for their stake share. So, users who own 0.001% of the stake, own 0.001%

of the computational power as well.

- Cardano: Cardano was found by one of the Ethereum co-founds, *Charles Hoskinson*. Cardano is somehow unique since it was originated from academic research where the team wanted to set some principles such as:
  - The Separation of accounting and computation to different layers.
  - The implementation of core components should be in highly modular code.
  - A competition between a small group of researchers and developers in peer-reviewed research.
  - The development of a funding mechanism for future use.
- RootStock (RSK): The RSK platform is connected to Bitcoin blockchain through sidechain technology. The purpose of RSK was to be a competitor with Ethereum but using Bitcoin as the underlying cryptocurrency. RSK started with a smart idea which is to provide Bitcoin with smart contract functionalities.

b) Programming Languages

- Solidity was originally developed by Gavin Wood, Christian Reitwiessner, Yoichi Hirai, and several of Ethereum's core contributors. Solidity is relatively new and it is an object-oriented Turing-complete programming language that has around 200,000 developers.
- Golang is a programming language with C-like syntax. Most of the code that is used in hyperledger Fabric for smart contracts was developed using Golang [25].

- JavaScript is a dynamic object-oriented programming language. Before JS, websites used to be static. JS plays a very important role in web technologies.

- C++ is a general-purpose programming language. It may not be the easiest language to code with, but it is one of the most reliable and used languages and that is due to its strength and ability to scale.

- Java is one of the most popular programming languages. Java has a very similar syntax to C++. It is reliable, popular, and has a lot of support online.

- SQL was developed by IBM to communicate with databases by sorting, querying, and interacting with data.

2.4 Application of Blockchain

2.4.1 Popular Use Cases of Blockchain

While it appears the industry has now a plethora of use-cases that are, in retrospect almost all of them are tailor-made for blockchain technology. Here are some broad categories that give us a perspective on the far-reaching impact of blockchain. Cryptocurrency was the first and most widely known but it did not take long for the industry to quickly realize that the underlying technology was applicable to multiple market sectors. Cryptocurrencies allow interested parties to transact digital currency with each other but that can be extended beyond just peer-to-peer transactions. Just imagine the power of businesses transacting across borders with the same ease with blockchain technology. Supply chain is being tapped in a huge way to reduce the churn in the plethora of transactions

that must occur to broker a supply chain transaction. Even having the ability to establish the validity of the vendor or supplier who are geographically dispersed is a huge plus!

Personal data management is the novel notion of an individual owning their personal data and having the ability and means to share it with whomever they want. It allows individuals to maintain and control their privacy. Individuals may choose to sell their data, donate for research, etc., basically, individuals control how their medical history data is used. Very practical and quick use is the ability to have an individual's medical history travel with the individual. Since it is verified from previous doctors, hospitals, and lab reports, it speeds up processing by avoiding mundane tasks such as filling out the same paperwork every time and minimizes interaction with intermediaries like the insurance companies. The notion of an individual owning their own medical history and data is quite enticing. Some companies are working on tracking the origins of diamonds to ensure buyers that the diamonds they purchase do not originate from parts of the world that are in conflict and do not conform to UN regulations. There are several use cases that encompass the vast expanse of finance, forensics, voting, food safety, and so on. The use of blockchain technology and building DAPs are far too many.

Smart Dubai, as an example, is a project that was issued by the Strategic Affairs Council, part of The Executive Council of Dubai to mandate the UAE pass as the only digital identity to be used by citizens and residents to access government services in Dubai. This comes as a huge jump in the context of smart cities that utilize blockchain networks in different aspects and government sectors. From 2015 to 2017, Dubai's blockchain community grew by 24%, five percentage points above the global average of 19%, which is a

huge jump for the city after they realized how important and useful blockchain networks are. Currently, 24 blockchain use cases are now being implemented across 8 industry sectors including finance, education, real estate, tourism, commerce, health, transportation, and security. Such networks make it easier for governments to manage the city network through smart contracts, and easier for users to get their work done in a faster way without the delay that occurs usually because of the intermediary level. Furthermore, this takes the trust between the user and the applied system to another level where it is almost impossible for hackers to change or alter the data in the network since it is almost impossible for hackers to compromise 51% or more of the city's network. [67]

In this section, we will discuss the potential pitfalls, challenges and advantages, and disadvantages of smart contracts. We summarize the properties of different blockchains in Table 1.

2.4.2 Pitfalls

Atzei et.al. [3] have discussed several pitfalls of smart contracts. These are presented below:

- Smart contracts are public on Ethereum. Therefore, all the code and data on smart contracts can be seen by others. The private keyword on the smart contract does not actually make the data private from the users, rather from other contracts. Therefore, the best practice while having data on smart contracts is to encrypt the data.

- While the public functions within smart contracts are callable from the world, the private functions are callable from within the contract itself. Setting some functions

Table 1: Landscape of blockchain platforms

Name	Permissioned	Permissionless	POW	POS	POA	POC	Tangle	Pluggable	Smart Contracts
Ethereum		√	√	√					√
Hyperledger Fabric	√							√	√
Libra	√			√					√
NXXTECH	√				√				√
Multichain	√					√			
Corda	√							√	√
Iota		√					√		

to public and others to private is an important thing to remember while writing these contracts.

- Each transaction requires a specific amount of gas and it cannot be done without it. Therefore, utilizing efficient data structures for different tasks is really important. Removing unnecessary processes is also important to reduce the amount of gas.

- Test your smart contract many times to make sure it does not have any vulnerabilities or broken pieces. This is one of the most important tips and extra testing will not harm your contract, but it will make it better. There are different tools where you can test your contract with such as test coverage analyzers, linkers, formal verification, Symbolic execution, etc.

2.4.3 Challenges

Smart contracts are types of executable programs that run between business parties. They are pieces of code that execute when called. Mostly, these contracts are strict and trusted. However, these contracts are still vulnerable and can be broken by hackers if they understand the contract really well. In some cases, the conditions listed in smart contracts can be met by a third party. So far, few cases were registered where smart contracts have been broken and millions of dollars have been lost. Nevertheless, this issue is not specific to smart contracts, rather it is a security issue in general. It should be noted that smart contract technology is promising and there is an ongoing effort to fix such issues and make these contracts efficient and more reliable.

2.4.4 Advantages and Disadvantages of Smart Contracts

Blockchains and smart contracts help us record transactions. All of the smart contract transactions are stored in chronological order in the blockchain network. Smart contracts allow automating the process between parties. Additionally, blockchains and smart contracts are the essential key factors towards smart cities. It is time to move to a secure network with smart contracts that people trust to store, manage, and control different systems within smart cities. In general, smart contracts make it faster for users to get their work done without the middleman in a more secure technique. Smart contracts are executable programs that provide the first party with an asset of value in case of the second party conditions are met. Therefore, there is no need for middle parties anymore. However, there are no standards or regulating bodies that govern what such technologies

should follow. In fact, being fairly new and not entrenched yet, smart contract standards are altered frequently. Moreover, these concepts are implemented in different technologies, programming languages, and platforms, making it hard to keep track of all of the advantages and disadvantages of these various implementation stacks. Hence making it very complex for a novice user interested in this technology.

Smart contracts can help in reducing fraud activities by recording all transactions in the blockchain network. However, the immutability can be considered as both an advantage and disadvantage at the same time since changing these blocks is very hard and the nodes will detect any suspicious editing of the existing blocks. At any time, if there is a change a new contract needs to be created. Otherwise, the existing contracts cannot be altered or changed. Furthermore, smart contracts can reduce failure since the decentralized architecture means no specific organization or entity is responsible for the blockchain network. Hence, if one node is down, nothing will be lost. However, writing smart contracts in an inefficient way may lead to consuming too much gas (the required fee to execute a smart contract). Moreover, having too many smart contracts can impact performance by delaying its execution, and thereby contribute to delay in the overall network. See Table 2.

2.5 Understanding Cloud Provider Supply Chain

Blockchain is a team sport and is often based on open governance whose benefits are viewed in terms of *Assets*, *Participants* and *Transactions*. This distributed peer-to-peer network comprises multiple stakeholders from different organizations requiring a shared view of (in)tangible assets and the provenance of their associated transactions. A group of

Table 2: Advantages and Disadvantages of Smart Contracts

Advantages	**Disadvantages**
Records all transactions	Lack of rules or regulatory body
Executable programs that enforce conditions	Can be challenging to create and maintain new contracts
Reduces fraud activities	Immutable
Reduces failure by recording all transactions	Inefficient smart contracts are cost prohibitive
Enforces conditions consistently	Too many contracts impacts performance

such transactions forms a block is cryptographically and chronologically linked together post validation (based on consensus algorithms).

Blockchain networks are permissioned and permission-less as their names imply [44]. Satoshi Nakamoto [54] describes the blockchain public permissionless network that powers cryptocurrency. However, business networks have specific rules around privacy and confidentiality, how transactions are endorsed, how consensus is achieved (often selective endorsement as opposed to proof of work), and who on the network is permissioned to receive them.

Hyperledger is a blockchain development hub [2]. One of the projects is Hyperledger Fabric which is an IBM project that enables the development of large-scale blockchain applications [11, 18, 61]. Cloud service providers recently started to offer Blockchain-as-a-Service (BaaS) which is quietly gaining popularity with DAP (Decentralized Application) developers [72].

It is commonplace for a cloud provider to have business systems catering to thousands of suppliers, partners, and internal stakeholders often working on siloed systems attempting to solve disputes that are innate to cloud provider supply chain network's quality of service [65] (see Figure 5). Such a business system dexterously passes the *Fit for Blockchain* test. Desirable platform attributes such as consensus, provenance, immutability, and finality [33] attract DAP developers. Audit transparency enables all supply chain entities to query the ledger to accurately determine the progress of the cloud procurement order and facilitate the prevention of breakdowns before they occur. With reasonable anonymity, for example, it can identify and recommend resurrecting a storage appliance vendor without disrupting the procurement chain flow.

Such deep levels of audit capabilities also satisfy regulatory compliance. Simultaneous peer-peer transaction settlements of geographically dispersed data center vendors can also eliminate intermediaries, which also translate to savings and promotes trust. Supply chain resiliency is vastly improved by smart contracts and human delays and errors are vastly reduced or eliminated. Automatic triggers can be put in place to notify participants to take corrective actions in near-real-time (or smart inventory predicting). For example, a particular data center is having issues with fulfilling a vSphere Hypervisor on

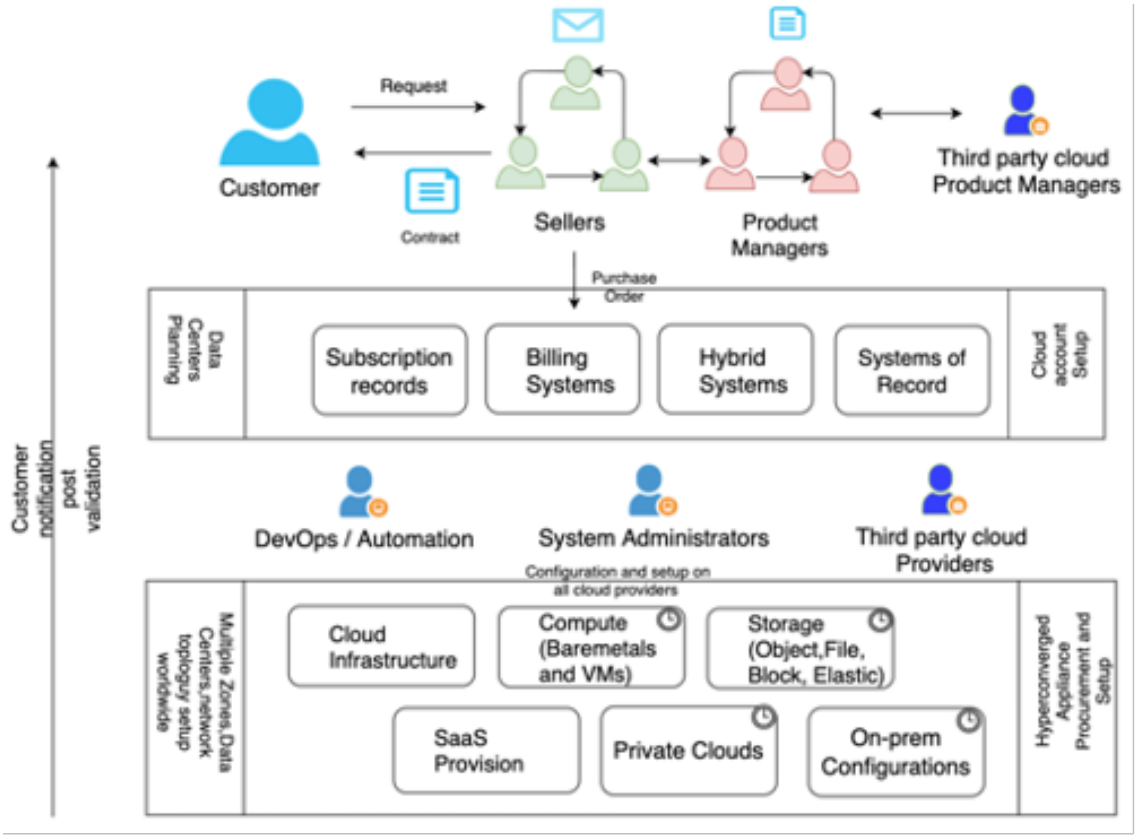


Figure 5: Actors (humans/systems) involved in the cloud provider supply chain.

Intel's Xeon (Broadwell) processor bare-metal request. In the age of demanding strong social ethics, we would like to call out a welcoming side effect of provenance. Consumers with strong socially ethical values expect the same from their suppliers. Even in a cloud procurement transaction, the consumer can verify that all parts of the cloud infrastructure are procured in socially responsible ways [17, 22, 45, 61].

CHAPTER 3

ETHEREUM AND SMART CONTRACTS

3.1 Smart Contracts

While smart contracts are the executable files [14], the ledgers are the files that hold the current and the historical information about different business operations and entities.

A common set of contracts, interests, processes, rules, and definitions should be set before different business parts start transactions. Figure 6 shows a basic contract in its executable form with simple conditions to sell a car. Blockchain networks turn these rules and contracts into executable programs that are known as smart contracts. Smart contract goal is to enforce the set of rules that were defined within while its execution [21]. For instance, we can add a condition within our smart contract to ensure that the newly purchased car will be delivered within a specific time frame. Otherwise, a funded amount of money will be released automatically. In our previous diagram, we can see how both parts contribute to the smart contract through different functions such as (setCar, update, transfer, etc.). These steps are required to establish an agreed business process such as selling a car [35].

In general, there are two states that a ledger can hold. First, the immutable records are the immutable records that were recorded using the blockchain. Second, the world state is the current state that holds the required objects' cache and value. Smart contracts

Figure 6: Smart contract classes

have access to two different pieces of the ledger, the immutable and the world state to perform different operations such as appending, updating, or deleting data (See Figure 7).

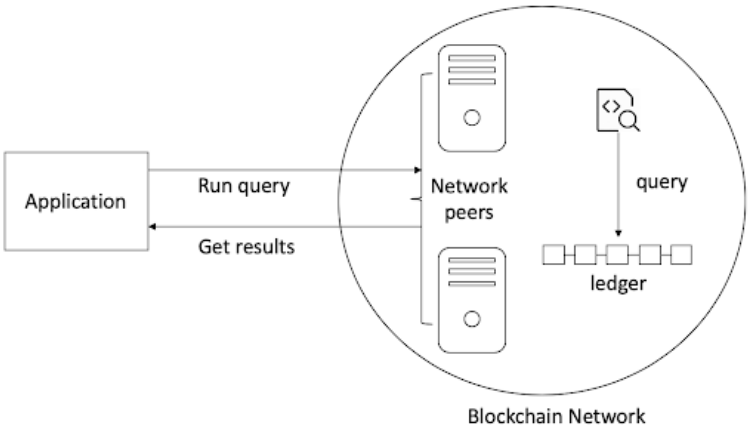


Figure 7: Blockchain

3.1.1 Endorsement Phase

Smart contracts come with different endorsement policies that approve transactions generated by smart contracts before releasing them. More than one organization

might share to endorse a smart contract before considering it valid. All of these transactions along with their final output valid or invalid are recorded using the ledger. Generally, any node in the network will have the authority to validate the smart contract. Unlike Hyperledger Fabric which requires a trusted organization to validate the contract.

3.1.2 Validation

The validation happens in two steps. First, the transaction has to be validated by a trusted organization. Second, the current value of the world state needs to be validated.

As discussed earlier, smart contracts can be used in various sectors to execute transactions without man-in-the-middle. They can be written such that when a seller wants to sell tickets for an event, a smart contract can be written to release the ticket to the buyer instantly upon meeting the specific conditions. In such scenarios, there is no need for a third party. The smart contract will release the payment to the seller and the item (ticket) to the buyer automatically. When a buyer fails to pay the required amount of credits, we can enforce the rules and conditions that need to be executed as a consequence.

There are several discussions in the marketplace to consider using smart contracts in real estate transactions. Its popularity will increase if we implement these contracts to be secure and tamper-proof for artifacts like clear title, liens, and other encumbrances so that they are transparent to the buyer. As seen in Figure 8, the smart contract will release the money to the seller, while releasing the property to the buyer at the same time. We can assume the contract also takes care of automatically notifying both parties, city and county registrars, and other stakeholders about the executed transaction for posterity and

action.

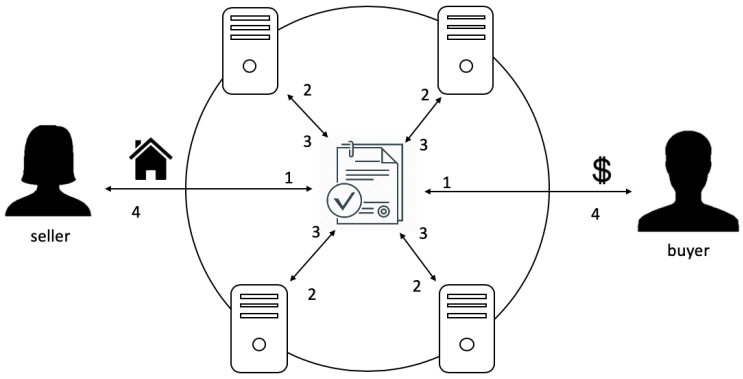


Figure 8: Transaction validation through smart contract (1-Seller and Buyer initiate; 2-Smart contract requests Blockchain; 3-Blockchain validates smart contract request; 4-Release for both parties)

3.2 Calculating the importance of a Smart Contract

Generally, it is hard to evaluate how good a smart contract is and that is due to the different criteria and standards are there. To be more specific, while evaluating a smart contract, a specific evaluation technique to the method should be followed such as the popularity of contracts. In this section, we evaluate the contract based on their popularity. In simple words, the more account addresses calling the DApp, the better the contract is (taking into consideration the importance of the calling accounts). Given Figure 9 where we have three accounts calling the DApp, the total importance for this contract is 3 units assuming that each account contributes with 1 unit.

However, the more contract a user call, the less attention these contracts get which means if a user calls only 1 contract, then the contract gets the user's full attention and

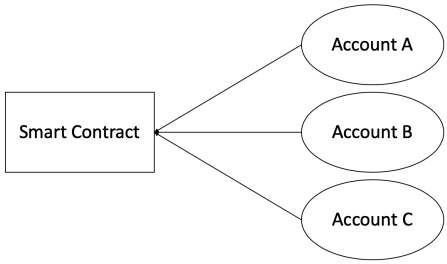


Figure 9: The function for getting all the services

get 1 unit for popularity. When a user calls many contracts, that means the user attention will be distributed over all the contracts which means each contract will get 1/n popularity units. In the following figure, three account addresses call two contracts. The amount of attention that each contract gets from accounts A and B is 1/2 since each of them calls two contracts and the attention will be divided among. Whereas the amount of attention that the second contract is getting from account C is 1 since it is the only contract that C is calling as can be seen in Figure 10.

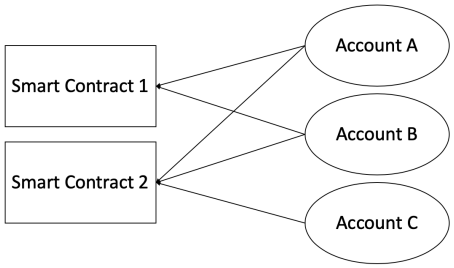


Figure 10: The function for getting all the services

Consider having one account calling different contracts and each contract is being called by a different account multiple times. In order to translate this scenario into Math,

we can think about it as a bipartite graph where one side is the accounts and the other side are the contracts. Calculating the edge between the contract u and the account v can be done using Equation 3.1 [55]:

$$W_{(u,v)} = \frac{\#(u,v)}{\sum_{(u,x)} \#(u,x)} \times P_u, \tag{3.1}$$

where $\#(u,v)$ is the number of times the address u calls the contract v. Since the accounts have different levels of importance based on different parameters within different systems, the term P_u was added to indicated different level of importance for different accounts. Computing the score of a contract v can be done using Equation 3.2.

$$Score(v) = \sum_{(u,v)} W_{u,v} \tag{3.2}$$

3.2.1 Use Cases

Real Estate: The impact of blockchain and smart contracts on real estate is akin to the impacts that Uber and Lyft had over traditional transportation methods. Ownership of apartments or houses can be transferred without the middleman. Smart contracts can release the ownership automatically when the transaction conditions are met such as the amount owed, time period, etc. With the help of blockchain and smart contracts, complex and time-consuming real estate transactions can be reduced to a few clicks on a smart device.

Dealerships: Dealerships can use smart contracts to keep track of vehicles ownerships and maintenance. For example, smart contracts may be enforced to make maintenance

every six months or so. If the condition fails then a driving license will be suspended or other action may take place.

Elections: We can use blockchains and smart contracts in government elections where we make some conditions for voters such as age, US citizen, etc. Once these conditions are met then, the voter can vote. After that, it will be hard to manipulate the votes. So this gives us a solution to keep track of votes.

Management: We can also use such technology for management purposes where the owner can release the products once he receives the money.

Medication: Applying blockchain and smart contracts in medicine helps prevent fraud by registering every step, process, medicine, and state in the blockchain after the condition of the patient is met.Using blockchain and smart contracts we can apply such networks that store health data information to be useful in critical and emergency situations such as involving experts and individuals in the crisis.

3.3 Writing Smart Contracts

Smart contracts can be written and used in many different languages and platforms. Solidity for example is one of the most common programming languages that is used for smart contracts. In our work, we will go over the basics of smart contracts using Solidity and how to implement our scenario using Solidity. Each Solidity program starts with the phrase *pragma solidity* $\hat{}$ *0.5.0*, which means this smart contract is written using Solidity version 0.5.0. The keyword "contract" is used to start a smart contract followed by the name of the contract. After that, the user may start declaring the needed variables

and data structures to hold the required values. These variables can be declared in Solidity using one of the following types:

3.3.1 Using Basic Data Types

String which is one of the most common data types and it is used to store string type values such as *Hello World*. **Int** which is another common data type that holds a value of an integer number without fractions (positive/negative). **Uint** unsigned integer which is an integer without a sign (positive by default). **Bool** which is the Boolean type to hold either *True* or *False*, could also refer to "0" or "1". **Address** which is similar to string. It holds 20-byte address which is the size of the Ethereum address.

One can use a combination of these variables to create specific data structures using a "struct" followed by the struct name. For example, *struct mySpecificStructure string name, int age, address myAddress* is a simple example on how to combine different variables.

Fixed-array which is a container that holds a value of several same-type variables. **Mapping** Is a data structure that is used to describe key: value pairs. It is known as a dictionary or hashmap or hash table in other programming languages.

We will use Remix online IDE to run Solidity to explain how to use the most common smart contracts data structures and how to implement our system the Provisioning blockchain. The screenshot below shows one of the simplest implementations for smart contracts where it contains a single string variable "systemName" and two functions that returns the value of this variable and set a new value, listing 5.

For this demo, we will use the *Solidity version 0.5.1*. After compilation, you should get a green notification which means there are no syntax errors in the smart contract and it is ready for deployment. This is the dashboard for Remix where a user can choose the account to use and deploy the smart contract using an interface. After deployment, the user will be able to see two buttons (these buttons reflect the execution mode of the functions of the smart contract). By clicking one of these buttons, you are basically calling the function that is associated with that specific button as shown in Figure 11.

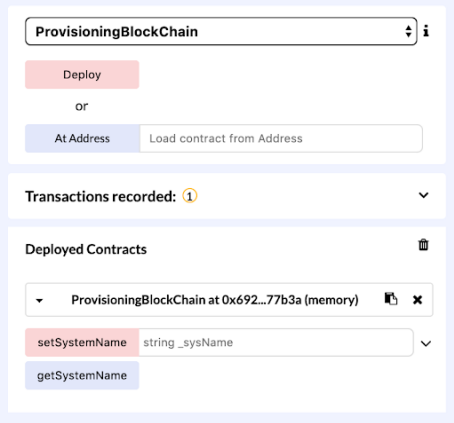


Figure 11: The deployment of Provisioning blockchain

By clicking on the first function *getSystemName*, the function will return the default value of the variable since the variable has not been set yet. The default value has been set to *IBMProvisioningBC* using the constructor. To set a new value, fill up a new value in the blank and call the function *setSystemName*. By calling the *getSystemName* again, you will notice how the value was set directly. By looking at Remix logs, you will be able to see how these blocks are generated after each operation as in Table 3.

By expanding one of the operations, you can see the block hash, where did it come

Table 3: Remix transactions logs

[call] from:0xca35b7d915458ef540ade6068dfe2f44e8fa733c to:ProvisioningBlockChain.getSystenName() data:0x499...d4983
transact to ProvisioningBlockChain.setSystemName pending ...
[vm]from:0xca3...a733cto: ProvisioningBlockChain.setSystemName(string) 0xbbf...732dbvalue:0 weidata:0xa44...00000logs: 0hash:0xe08...78214
call to ProvisioningBlockChain.getSystenName
[call]from:0xca35b7d915458ef540ade6068dfe2f44e8fa733cto: ProvisioningBlockChain.getSystenName() data:0x499...d4983

from, transaction cost, value, etc., as show in Table 4.

Table 4: Expanding a specific log shows the transaction hash

transaction hash	0xbbad6006f1b ... 01d4eb1a1c8483ac4
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	0xbbf289d846208c16edc8474705c748aff07732db
transaction cost	22692
execution cost	1420
hash	0xbbad6006f1b7794 ... 31afe29ae01d4eb1a1c8483ac4
input	0x499d4983

Notice that it is not necessary to have a *getSystemName* function, we can add the

modifier “public” in front of the variable name and that allows calling the variable to return its value without a function. Moreover, the constructor can also be removed. The value can be set directly after the variable name, listing 6.

The modifier *constant* can be added to prevent changing the variable value in future.

ENUM type enumerated list that allows us to keep track of the list items. In our scenario, we will use the enum to keep track of our PBC system states. We declared an enum *State* that has five different states. By default, the state will be set to *Waiting*. The function *pbcDev* will change the state of the state to Dev. Another function *isActive* will return if the current state is *Active* or not as seen in listing 7.

The function *isActive* will return *False* that is because the default state is “Waiting” whereas the function *pbcDev* will change the current state to Dev and the *pbcState* will return the current state of the system.

Our attention is to build a provisioning smart contract that keeps track of all the services for a current system. At any time, users should be able to add, delete, display different services. Our network with the help of our smart contract will allow us to keep track of all the services in the system currently. Moreover, at any specific time, users should be able to trace back the blocks to see the previous services in the system. The smart contract will be the executable program that users interact with to add, delete, edit a service. Using the ledger, all the transactions will be recorded to allow the owners to trace back who added the service, who removed a service, etc. This will help us monitor, control, and manage these services in a better way.

3.3.2 Smart Contract Functions

Our current smart contract contains four main functionalities.

a) Add a New Service

After listing the functions names along with their parameters, the smart contract will look as shown in listing 8.

For the purpose of filling these functions we have different options, whether we use key-value data structure which is known as *mapping* or we can use a simple array. For each data structure, there are advantages and disadvantages that we will discuss soon.

Mapping Keyword for Storing Key-Value Pairs: One of the ways to accomplish our task is to use a mapping data structure to map each service address to a bool variable. The bool variable verifies if the service exists in our system then it holds *True*, *False* otherwise. After filling up the missing pieces, the smart contract will have the same code as in listing 9.

Please note that adding and removing services is a quick and easy task that takes O(1). However, returning and displaying all the services may not be possible using this data structure since Solidity does not support iterating.

Using Arrays: Let us implement the same functionalities with an array data structure rather than a mapping as shown in listing 10.

Note that we were able to solve our previous issue which is listing the entire services that currently exist in the system. However, removing service may take O(n) in the worst case. Also, we have added another function to return the index for a specific service.

Using a Linked List: We begin with the same data structure "mapping" and for each position will store the address of the next address. Table 5 represents the mapping.

Table 5: A hash-map using a linked list

Key	Value
placeHolder	0x01
0x01	0x0a
0x0a	0xdd
0xdd	0x33
0x33	0xa1
0xa1	placeHolder

Figure 12 represents the linked list. The previous data structure can be implemented as in Figure 11. The previous code will create the mapping and set the default value *Placeholder* to point to itself which means the mapping is empty. Adding a new Service is simple as adding a new node at the top of our linked list as in Figure 13 and listing 11.

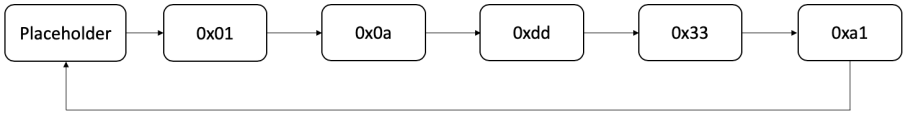


Figure 12: A link list for our services

Delete an Existing Service: Deleting a service is basically deleting a node from the linked list. In our case, it is a single link list in which all we have to do is to place

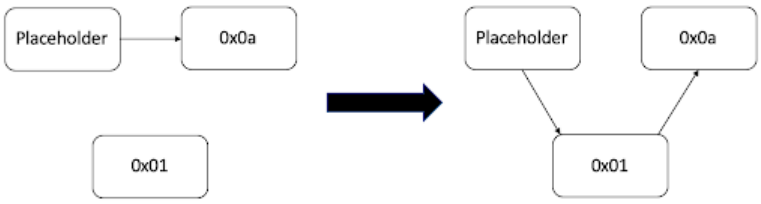


Figure 13: Illustration of adding a service

a pointer at the node that we want to delete, change the pointer from the previous node to the next node and finally delete the first pointer that points to the unwanted node. See Figures 14, 15, 16, and 13.

Figure 14: A pointer to the node to be deleted

Figure 15: Change the pointers direction

Note that we have used the function *_getPrevService* that is currently not in our code. We can implement it as shown in listing 14.

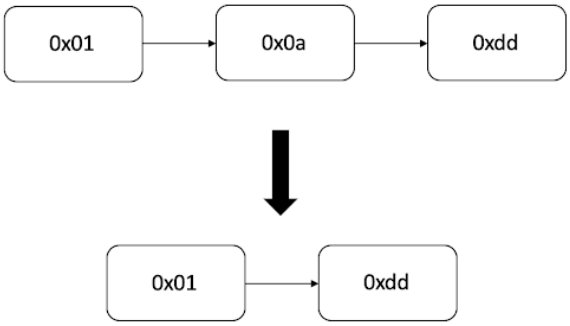


Figure 16: What is the linked list used to be

List All Existing Services: To accomplish this task we start by looping through our mapping starting from the Placeholder address. As each address points to the next address, this process will keep going until it points to the Placeholder again. See Figure 15.

CHAPTER 4

RELATED WORK

Given the complexity of the proposed system, we have a two-part approach to the related work discussion. The first part focuses on the federated learning (matrix factorization) model used in our technique. The second part evaluates similar systems that use federated learning built on blockchain.

4.1 FedMF

The authors of FedMF [16], Chai et al., introduced a secure matrix factorization framework using federated learning where the model learns using the user's gradients only that are sent to the server instead of the raw preference data. The authors proved that while the gradients seem secure enough, they still might leak some of the user's data. [60, 77] Therefore, they took the implementation of FedMF a step further by implementing the matrix factorization framework using homomorphic encryption. [27]. To this end, the implementation of FedMF is a secure framework that can be used to learn a matrix factorization model. The architecture of FedMF still relies on the traditional federated learning approach that utilizes a single centralized server. In industry, when a huge number of parties are included, a single server will not be efficient to coordinate, collect, average all of the client's models. Therefore, a decentralized version of FedMF is required to allow multiple worker nodes to carry the global computational progress.

4.2 Baffle

The authors of BAFFLE [62] created a decentralized aggregator-free blockchain-driven environment that leverages smart contracts to coordinate the federated learning settings and aggregate the user's models. BAFFLE enhances the aggregation process and boosts the computational progress by dividing the global parameters space into chunks with a score and a bid strategy. Chunking the parameters space is due to the limit of the pertaining size of the Ethereum Virtual Machine (EVM) which is 24 kB [57]. Furthermore, due to the expensive SC storage, the models need to be stored in a serialized format. Therefore, the authors use a partitioning algorithm that is used by all of the users to first chunk then serialize the parameters. Finally, the budget for each chunk allows users to decide on which chunks to contribute with, which saves time, capacity, and does not add unnecessary communication on the network. However, the machine learning model type that is used in BAFFLE allows aggregating the user's models in order to generate a more generalized global model. But that is not the case when it comes to recommendation systems models where each user data vector is different than the others since not all of the users have the same items and products. Therefore, it is not possible for matrix factorization techniques to aggregate the user's models to generate a global model. For matrix factorization, each user data vector needs to be updated individually based on that specific user's data. Hence, BAFFLE environment is not applicable for recommendation system models. Furthermore, in our implementation, we store the items data on the worker nodes and keep the user's data localized on the user's machines while leveraging the benefits of smart contracts for other federated learning tasks.

4.3 Trust-based system for Collaborative Filtering Recommendation Systems on the Blockchain

Tzu-Yu et al. [76] proposed a trust-based system for Collaborative Filtering recommendation systems on the blockchain. The proposed system provides a secure and trust-based system supported by the use of smart contracts in the main blockchain protocol. Using blockchain the authors proposed a framework to collect large volumes of data and allow users to host and share a mutual recommender model that is being used as a public resource. Using incentive mechanisms, users are able to share the model parameters, new images, movies, titles, etc. that allow updating the existing models hosted by a specific user or a group of users. In such scenarios, all users are assumed to be trusted users who will not share malicious information or update the model with false predictions. In addition to the model parameters being shared publicly on the network, the gas fees for the smart contract storage and other operations are very expensive. In our implementation, all of the gradients are stored in IPFS (distributed storage). The hash from IPFS is then shared on the Ethereum network to reduce gas fees instead of sharing the actual data. On top of that, all operations and model updates are computed using the homomorphic encrypted items and gradients vectors.

4.4 Healthmudra

Blockchain started to make in-roads into the medical fields to facilitate collaboration between patients, healthcare providers, and insurers via a secure, transparent, and

immutable network. The authors of HealthMudra Rashmi et al. [10] presented a recommender system that prevents diabetes which is the most challenging disease in the healthcare sector. [10] HealthMudra is built using a blockchain network powered by machine learning algorithms and optimization protocols that mainly use filtering techniques to provide recommendations in order to prevent diabetes. Such implementations are helpful when patients share their health data with different healthcare providers to get helpful recommendations. In our work, we consider federated learning to keep the user data localized. Even when gradients are shared, they are shared after applying homomorphic encryption to perform all of the learning and model updating using the encrypted data and not raw data.

4.5 FLChain

Several studies have discussed the area of federated learning using blockchain which lead to a new paradigm known as FLchain. This technique transfers the current Mobile Edge Computing (MEC) networks into decentralized, secure, and privacy-preserving systems. Nguyen et al. [56] proposed a FLchain network to train a shared model using mobile edge devices. FLchain allows training a shared model while keeping the user's data localized on their devices to benefit from privacy enhancement. A group of MEC servers is initialized with their associated user's devices. Each user carries out the training locally and commits its update to these servers through a transaction that is stored in a Merkle tree. Such approaches like FLchain and BAFFLE are applicable in the case

of using specific types of machine learning techniques where averaging all user's models generates a better global model. However, in recommendation systems averaging all user's updates is not the right way to achieve better accuracy since each user's updates are specific to that user. In the recommendation systems area, users with similar behaviors tend to have similar updates, but that cannot be generalized among all of the users on the network.

CHAPTER 5

APPROACH

5.1 FedSmarteum for multi-organization automation

We propose FedSmarteum, a decentralized distributed architecture as shown in Figure 17 using blockchain and smart contracts to enable cloud providers and their suppliers to navigate their supply chain and engage the customer with verifiable and immutable data. We evaluated cloud resource provisioning compute, storage, and network resources.

We evaluated the use of permissioned *Blockchain-as-a-Service* (BCaaS) platform offered by several cloud providers, including IBM. This facilitates the development and deployment of the blockchain network exploiting built-in capabilities for the governance of policies, smart contracts, security, and compliance, etc.

As part of this approach, a private permissioned blockchain network was developed with all the stakeholders as peers. As shown in Figures 18 19, we have customer, seller, biller, DevOps admin, third-party suppliers who are either systems or stakeholders participating in the blockchain. To facilitate faster deployment of the blockchain network, we suggest using a BCaaS instance from a cloud provider (in this case IBM Blockchain Platform). This will set up peer stakeholders nodes owned and administered by the impacted organizations. The peers will also host instances of ledgers and smart contracts where transactions are validated and executed.

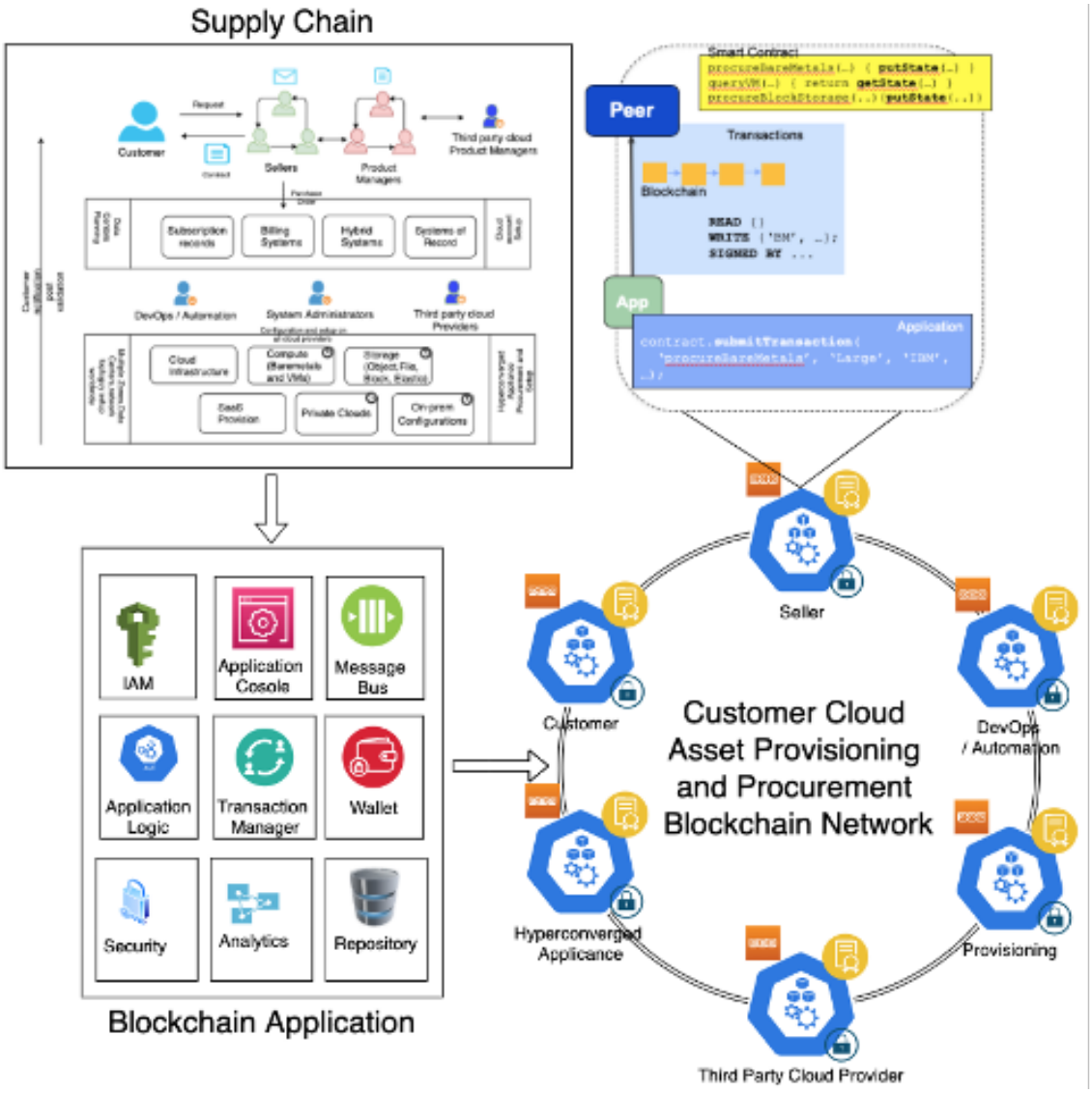


Figure 17: Actors (humans/systems) involved in the cloud provider supply chain.

Organizations were created for each of the entities and policies for managing network participants. Channels were created to facilitate information exchange between peers such as DevOps and sales organizations to protect the privacy of pricing information and prevent sharing from the broader network.

We used Visual Studio Code with the IBM Blockchain Platform extension to create and package smart contracts. From the IDE, we packaged and exported the smart contracts and we used the IBM blockchain platform console to install and instantiate the smart contracts. Upgrading smart contracts to newer versions was very simple.

We are depicting a simplified typical scenario (compute, storage and software) as handled by the permissioned blockchain version of FedSmarteum in Figure 18. This represents supply chain interactions for a hyperconverged appliance purchase order. However, in reality, several other business processes that are part of the purchase fulfillment that also have the potential to go wrong are not depicted for brevity. Smart contracts help to automate some of the notification processes so that actions are taken immediately. During the process, the status of the purchase order and all transactions initiated by *submitTransaction* are stored in the blockchain. All stakeholders in the supply chain can see exactly where a given order is in the process, problems encountered, and who is currently holding up the fulfillment. For example, in the permissioned blockchain version of FedSmarteum, smart contracts are implemented to automatically trigger additional business workflows. When the system is back-ordered in a data center and the customer needs to be solicited to pick other options in case wait time is unacceptable, the application invokes *processNotification* in the customer's smart contract to notify the customer about

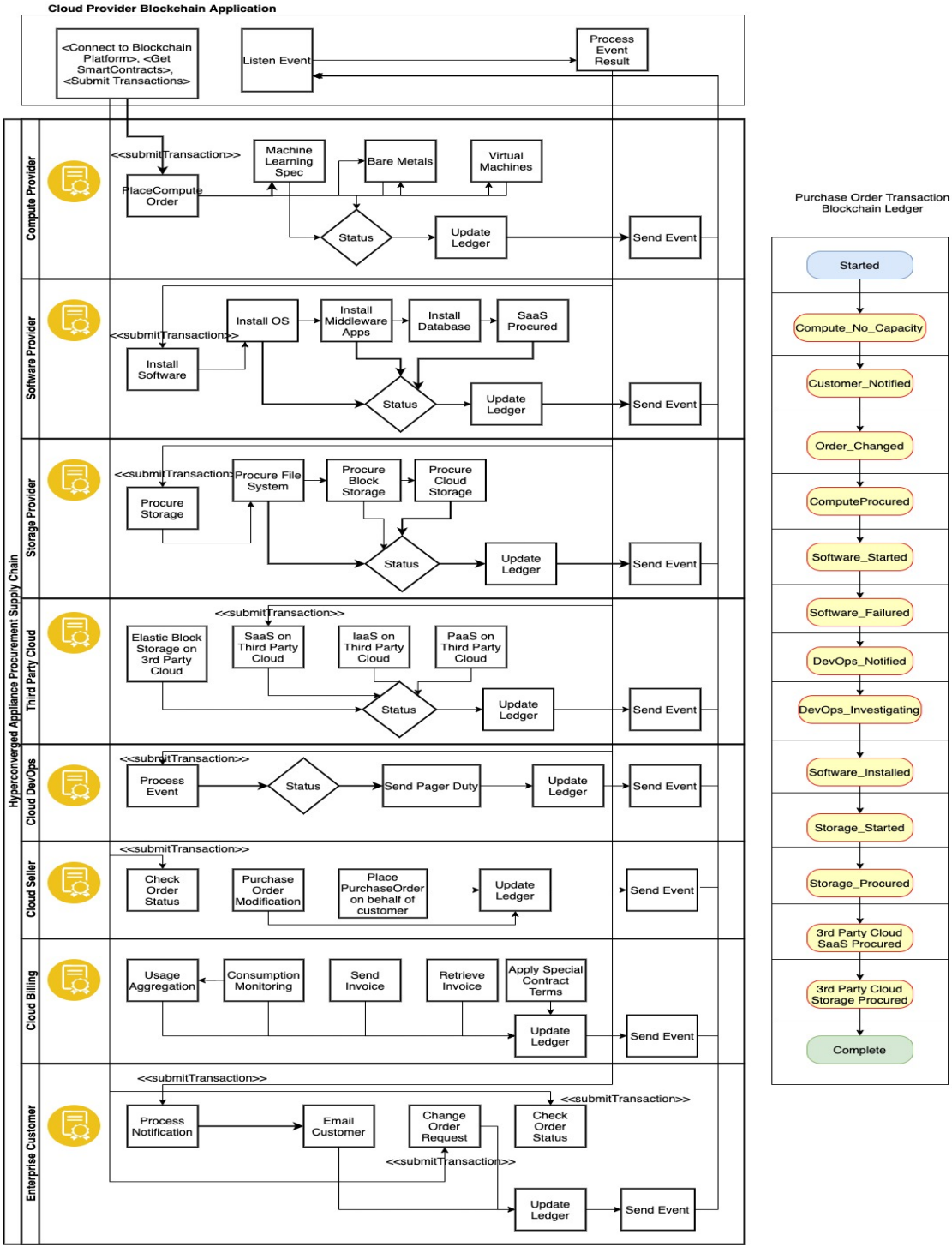


Figure 18: Smart contract enabled hyperconverged appliance supply chain.

the situation. The customer changes the order by specifying a different datacenter with confirmed availability or chipset or storage performance attributes, and the procurement process automatically continues with updated terms and possible cost implications. The DevOps engineer may also be notified by the smart contract to fix the problem.

Several smart contracts were created to simulate our use cases. The smart contracts follow the fabric-contract-api interface. For example, *async PlaceComputeOrder(ctx, dataCenterId, computeType, orderId, customerEmail)* and the application submits the "PlaceComputeOrder" transaction by *contract.submitTransaction('PlaceComputeOrder', "DAL01", "vm.small", orderId, customerEmail);*.

The transparency that the framework provides to the customer along with access to the provenance of data is unmatched. Furthermore, there is nothing stopping the platform to take this a step further wherein the sales leader (who is another peer in the network) is also notified by the smart contract which determines that there was a change to the original purchase contract and needs to be modified. The speed with which this exchange can transpire, and the resultant non-repudiated transaction that meets all legal requirements as well that can be put in place was previously unimaginable in existing supply chain workflows. All peers are rightly incentivized to facilitate the earliest fulfillment of the order. It also helps them to keep their inventories current by optimizing with their downstream suppliers and vendors.

We then built an application to execute the business logic in smart contracts to perform operations such as creating, transferring, or updating cloud provider assets on the blockchain ledger. For example, updating the procurement status of bare metal and

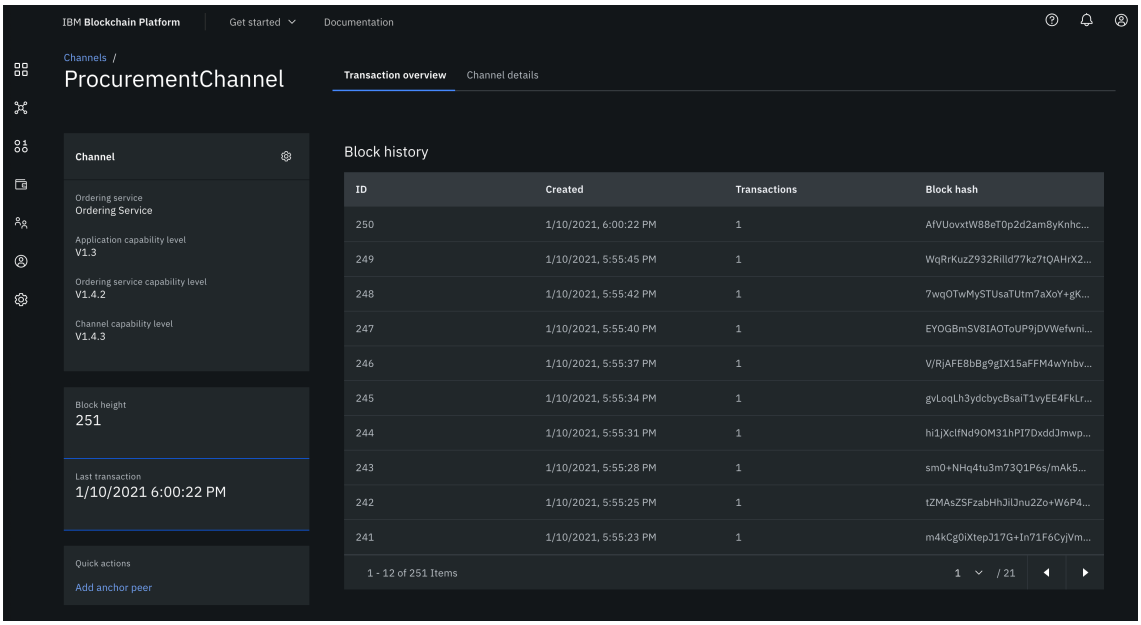


Figure 19: Transaction block history in IBM Blockchain platform console

storage. Attributes such as Identity and Access Management (IAM) to authenticate users and systems were set up, analytics to alert on problematic third-party vendors, etc. We observed that as expected all the transactions are recorded in the blocks. This is visible in the IBM Blockchain console as shown in Figure 5, thereby promoting trust and transparency among the stakeholders.

5.2 Extending FedSmarteum to incorporate Federated Learning

We further enhanced FedSmarteum with the ability for multiple organizations to share an aggregated recommendation system that allows enterprise customers the ability to make decisions based on latest supply chain and resource availability metrics. With

transparency built into the system, organizations can train models privately on their private data and facilitate the presentation of collective predictive analytics to an enterprise customer.

The system is built using an Ethereum blockchain in order to provide customers with transparency while keeping the data private. Ethereum leverages the use of smart contracts that coordinates the federated learning process between all the included nodes and cloud parties. In our work, we train a Matrix Factorization model in a federated way where all the customers' data remain localized on data owner's devices. The items data will be placed on one of the worker nodes (could be one of the nodes that are included in training the model). Sending and receiving the training data, weights, and gradients on the Ethereum network is too expensive and requires a lot of gas fees, other than the network transfer size limitations. Therefore, we use a distributed database (IPFS) [39] that provides hash-based storage. When storing an object in IPFS, the distributed storage will provide a hash to be used when reading the object from the storage. Consequently, instead of sharing a model, dataset, or gradients set on the blockchain, we store these objects on IPFS and then send that hash to the smart contract which is going to share it with the participating nodes. In this way, all of the nodes will be able to share data on the blockchain using IPFS without incurring expensive fees. The remaining question is, should all the nodes trust IPFS or the worker nodes in order to share their weights or gradients? The answer is no. In our work, we have enabled homomorphic encryption where each node encrypts its data before sharing it on IPFS or on the chain. Using HE, nodes do not need to trust IPFS or any of the worker nodes. HE allows operations on

the encrypted data such as addition, multiplication, and subtraction. In our scenario, we are using the encrypted customer's data to update the encrypted items data. Therefore, all the participants keep their data private and share their encrypted gradients only. The smart contract coordinates the federated learning process between the included parties. The purpose of the blockchain network is to keep all the parties included and updated during the process. Blockchain provides transparency between the customers and the cloud providers. All customers will be up to date with their procurement and services while the cloud providers will be able to get insights using the customer's data without violating their privacy as shown in Figure 20.

There are 2 main parts depicted in Figure 20. The first one is the multi-organization supply chain blockchain application and the second part below is the federated learning-based recommendation system. All organizations are nodes in the blockchain and so are the worker nodes of the recommendation system. It should be noted that given the complexity of the architecture represented Figure 20, it only represents some of the key connections and flows involved in the execution of the system. Initially, we integrate into the supply chain ordering pipeline, as in this case, the *PlaceComputeOrder* smart contract, which may be initiated by a company's DevOps automation that is in place based on the purchase order. The definition of the smart contract that for *PlaceComputeOrder* is also depicted (to the right). It is part of the Cloud provider fabric and has privileges to procure any compute resource that is part of the purchase order interacting with the APIs in the hardware services layer of the cloud provider. This smart contact orchestrates the training phase of the federated learning model by controlling the traffic between the worker

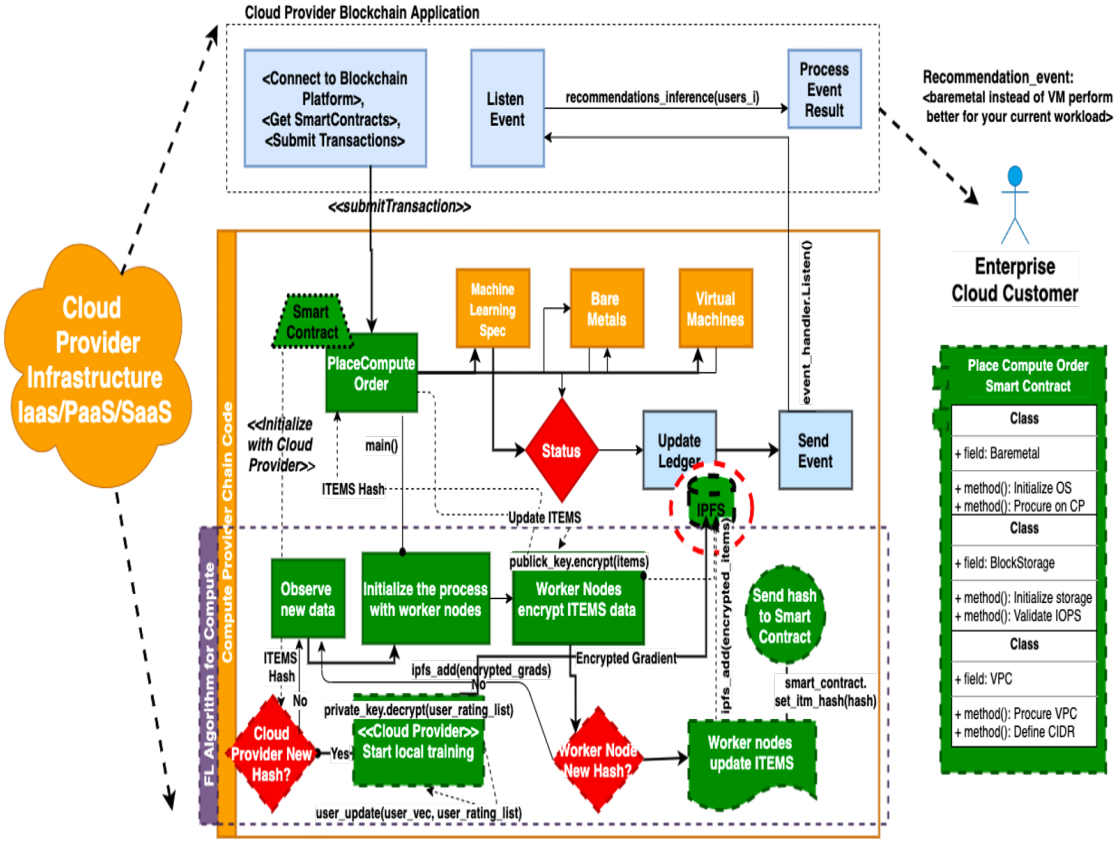


Figure 20: Architecture of FedSmarteum

nodes and the cloud providers. The smart contract also facilitates recommendations to the user via events and orchestrates all the rounds and epochs as part of the recommendation system using items and users data.

5.3 FedSmarteum Methodology

As discussed earlier, federated learning is an approach to allow users to train a shared model on their joint data without exposing users' local data. Federated learning comes under the umbrella of privacy-preserving techniques and can be categorized based on the distribution characteristics of the data [74]. Horizontal federated learning is one of the categories of federated learning where users' data share the same feature space while different users have different samples. Therefore, this matrix factorization model can be considered as an example of horizontal federated learning since the rating data shared with each user has the same feature space, but different users have different samples. In our implementation, we assume all of the users in our scenario are cloud providers. Our technique secures the users further from the central server since this server might be a trusted, but curious server.

5.3.1 Objective Function - Stochastic Gradient Descent for user-level matrix factorization

In this section, we introduce the optimization method that is used in the matrix factorization model [16, 41] which is stochastic gradient descent. We design a *serverless decentralized network* to update the model locally using worker nodes in a decentralized way instead of relying on a central server.

Assuming we have a m number of items and n number of users where each user rated a number of items (a subset of m). Given $[n] := \{1, 2, ..., n\}$ which is the set of users and $[m] := \{1, 2, ..., m\}$ is the set of items, we can denote $\mathcal{M} \in [n]X[m]$ for the user-item rating pairs and M is the total number of ratings $M = |\mathcal{M}|$. We denote the user i that rated item j by r_{ij}. Assuming the value r_{ij} is given, then the recommendation system is expected to predict all of the items for all of the users by fitting a binary model on the existing ratings. That is computed using user matrix $U \in R^{n \times d}$ and item matrix $V \in R^{m \times d}$ and the output matrix is then used to predict the user i's rating on items j which can be described as $\langle u_i, v_j \rangle$. The authors of FedMF [16] shows how to compute U and V by solving the regularized least squares minimization as in Equation 1.

$$\min_{U,V} \frac{1}{M}(r_{i,j} - \langle u_i, v_j \rangle)^2 + \lambda ||U||_2^2 + \mu ||V||_2^2 \tag{5.1}$$

λ and μ are small values that were added to rescale the penalizer. U and V will be updated using the Stochastic Gradient Descent as in Equations 2 and 3.

$$u_i^t = u_i^{t-1} - \gamma \nabla_{u_i} F(U^{t-1}, V^{t-1}) \tag{5.2}$$

$$v_i^t = v_i^{t-1} - \gamma \nabla_{v_i} F(U^{t-1}, V^{t-1}) \tag{5.3}$$

where

$$\nabla_{u_i} F(U, V) = -2 \sum_{j:(i,j)} v_j (r_{i,j} - \langle u_i, v_j \rangle) + 2\lambda u_i \tag{5.4}$$

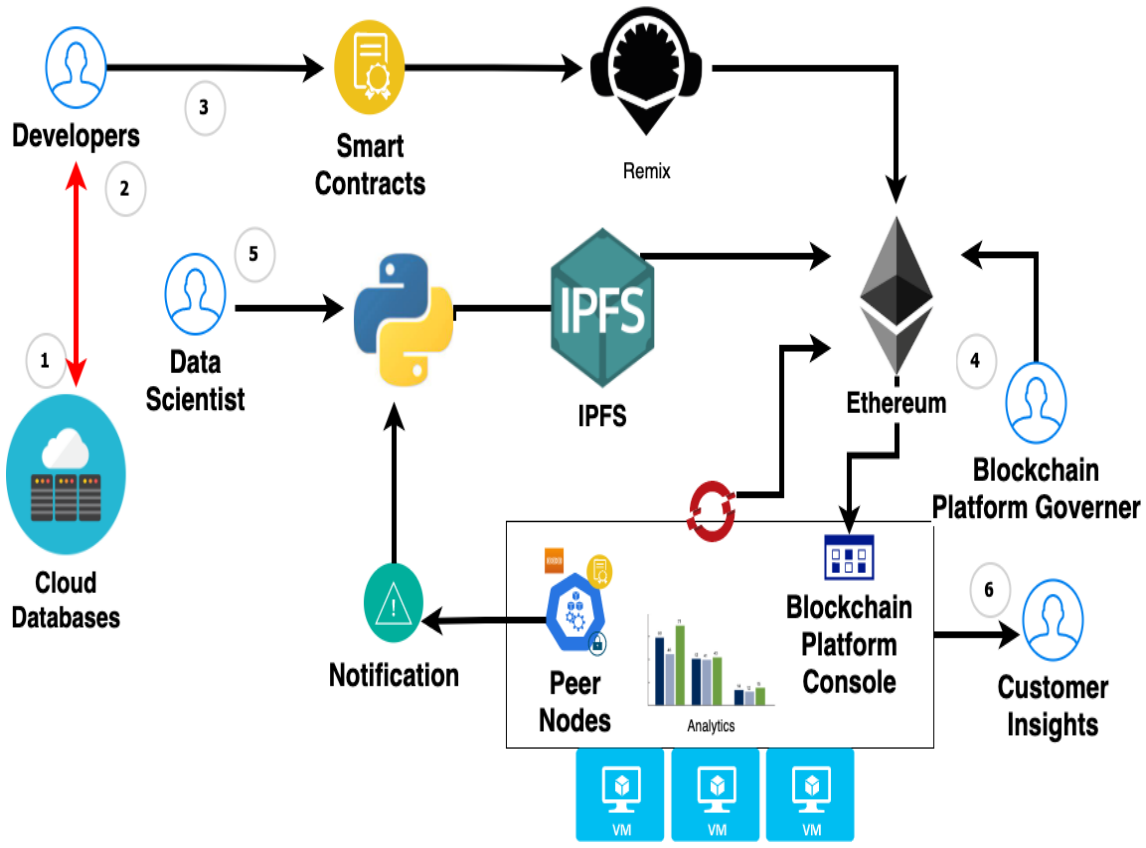


Figure 21: Deployment flow depiction of FedSmarteum.

$$\nabla_{v_i} F(U, V) = -2 \sum_{i:(i,j)} u_i (r_{i,j} - \langle u_i, v_j \rangle) + 2\lambda v_j \tag{5.5}$$

This update takes place iteratively until the number of rounds are met.

5.3.2 The Decentralized Matrix Factorization

In our implementation, all users keep their rating data localized without sharing it with anyone. Then, the model is trained on the user's joint data. In order to achieve this goal, we leverage the use of the decentralized matrix factorization approach. This

approach decomposes the updating of algorithms into two parts where the first part is performed on the user's device locally and the second part is computed using the worker nodes in a decentralized way. Equation number 2 is computed on the user i's device whereas equation 3 computes in a decentralized way using the worker nodes. The decentralization part is because of two main reasons: (i) to keep the user's rating data localized, and (ii) to prevent a trusted but curious server from recovering insights from the model.

Algorithm 2 Decentralized User-level matrix factorization

Init: Worker nodes initialize item profile matrix V
Init: User initializes user profile matrix U
Output: Converged U and V
    IPFS and worker nodes keeps latest item-profile for all users
    User local update:
        Smart Contract shares the hash with users to obtain V, perform local update:
            $u_i^t = u_i^{t-1} - \gamma \nabla_{u_i} F(U^{t-1}, V^{t-1})$
            $Gradient_i = \gamma \nabla_{v_i} F(U^{t-1}, V^{t-1})$
        Worker nodes update:
            Smart Contract shares IPFS $Gradient_i$ block with worker nodes for user-i
            Perform update: $v_i^t = v_i^{t-1} - Gradient_i$

The general user-level matrix factorization method allows users to keep their data localized on their devices while they share the generated gradients as plain text with the server [41]. The authors of FedMF [16] proved that while the users keep their data localized, the server is still able to decode users' ratings through the gradients. Therefore, the authors have implemented the recommendation system using HE in order to secure the gradients. The server is still able to perform the same updates using the encrypted

gradients while maintaining the model accuracy. The encrypted gradients will secure the users from the curious server and other attacks. However, the server will have an overhead updating the encrypted items data using the encrypted gradients. We replaced the server with worker nodes in order to carry out the items update in a decentralized fashion.

Our focus was on the efficiency of a decentralized framework while maintaining the model accuracy. We observed our generated model had similar predictions when testing the recommendation system on the MovieLens [52] rating dataset and our cloud dataset. However, since our algorithm was implemented in a decentralized way, there is a difference in the execution time as in our proposed approach the worker nodes are carrying out the items updates instead of the central server. Algorithm 2 shows the updating procedures.

Using the cloud provider illustration, we examine Algorithm 2 to depict the training algorithm for updating/training the USERS data in a cloud provider. To train, we need both ITEMS (V) and USERS (U) data. We begin by initializing the ITEMS matrix V in the worker nodes and the users matrix U in the cloud provider. The output describes trained U and V in Step 3. In step 4, we use IPFS to keep track of V. We then start with local updates (training) for all cloud providers. The smart contract shares the hash of the ITEMS vector V with the cloud providers to perform a local update. At this point, each cloud provider has both parts as shown in step 6. Steps 7 and 8 represent how to update user and gradients, after which we perform worker node updates, update the shared items matrix, and specific item record as shown in steps 9-11.

5.4 Passing Gradients Over Blockchain

Most of the Blockchains have an upper size limit on the transactions. Ethereum Virtual Machine (EVM) has the limit of 24 kB [57]. Any random smart contract that contains many functions, too much code, with multiple events will hit the size limit instantly. For example, ERC1400 Security Token Standard requires 27 functions and 13 events [57]. With additional application functions and specific code to implement these standards, the limit will easily exceed 24 kB. Therefore, storing a large volume of amount of data on the smart contract is not feasible.

On the other hand, gas fees paid for transactions is expensive. As of August 25, 2021, the gas fee for a single transaction is 0.0013 Ether/transaction [75], with the Ether price today is $3,247.73, each Ethereum transaction costs almost $4. For example, the average size of a photo that was taken using an iPhone-6 is 2-3MB. Hence, depending on the Ether price, buying a car might be cheaper than adding one photo on Ethereum blockchain [58]. Ethereum network can be used due to its security, immutability, and transparency. However, for distributed storage purposes, we have used Interplanetary File System (IPFS) [39]. Using Ethereum and IPFS creates a simple, yet powerful, system of immutable content. This way, all data, and models can be stored on IPFS, while transactions and communications can take place using the Ethereum blockchain. Using IPFS, we are able to timestamp much larger data to be used over the blockchain than the pure blockchain networks. When users add data to IPFS, the protocol returns a hash for the data. This hash is cryptographically guaranteed to be unique to the content so no two sections of data will have the same hash. When the same section of data is added again

to IPFS for the second time, the same unique hash will be returned. To retrieve data from IPFS, we use the reverse way. The same cryptographic hash returned from the storage process is then returned for the IPFS in order to retrieve the original data that was stored. By using the IPFS mechanism, we guarantee that our data has not been tampered with. Furthermore, all of the data stored on IPFS during our implementation is homomorphic encrypted. Therefore, the data cannot be read or tampered with by others. The receiver node can then download the encrypted data from IPFS and use the gradients for updating the items data. The smart contract is responsible for this process and directing the hash values between the user's local devices, worker nodes, and training nodes.

After the model is trained, the items data can be used with customer data to provide specific recommendations for that customer. The updated items data will be available in IPFS and worker nodes. The participating cloud providers and other privileged entities in the supply chain can use this data with new customer's data to generate predictions and recommendations. The reason for leaving the encrypted items data on worker nodes is due to the fact that training is perpetual and iterative. As additional customers and data are generated, so are the training iterations. The accuracy of the model has a strict relationship with time and the number of users. All participating cloud providers have equal access to the model.

Figure 21 represents the deployment architecture for FedSmarteum. We have multiple stakeholders of the system including developers who are responsible for maintaining smart contracts, the data scientists who build and maintain ML algorithms, the blockchain governor who is responsible for setting up and maintaining the network connection and

protocols, and finally, the customer who relies on the system for insights to help make business decisions on cloud resource utilization across a multi-organization platform deployment. For example, a customer may be recommended by the model that a specific machine type on data center_X has taken longer to deploy due to unavailability of the desired chipset, and other customers have either chosen to deploy a different machine type of chipset or pick an entirely different region previously. Having this recommendation enables customers to avoid unnecessary delays and costs. Also note that in our system, the customer is a training data provider for data scientists to train the ML models in a federated learning approach.

As alluded to earlier, smart contracts are developed to orchestrate the supply chain flow and federated learning technique and they constitute the backbone of our system. We deploy these smart contracts using the Remix platform on the Ethereum network. Machine Learning engineers and data scientists can then build their models and start the training process. The worker nodes connected to IPFS will then pull the encrypted items data from IPFS, decrypt it, generate gradients, and update the customer's data, and finally encrypt the gradients and push them back to IPFS. IPFS contains the latest items data that can be used to get insights by using the Analytics platform supplied by cloud providers. For example, we can deploy on an IBM Red Hat OpenShift cluster in a containerized environment which gives us ready access to analytic tools. As a cloud provider, we can rely on these insights to provide recommendations and predictions for future customers based on the previous customer's behavior.

5.5 Recommendation System in Play

For purposes of illustration, we begin with an enterprise customer making an order request to procure several resources. Prior to the order being placed, it is directed to the recommendation system to validate the availability or price range of the order or time to deploy (based on similar recent orders) with similar resources, workloads or perhaps a high-performance storage (GB/sec throughput). The recommendation system may predict to the customer that there is cost-effective and faster option on another cloud provider based on model memorization (intuitively, we say that a model unintentionally memorizes some value if the model assigns that value a significantly higher likelihood than would be expected by random chance). The customer then has an option to change the order to whatever was recommended or to keep the initial order. Deployment changes and billing information changes are made automatically based on customer input.

This drastically saved the customer time, reduced cost, possibility of dealing with future incompatibilities, clarity in billing, and most important is entrenching trust and transparency in dealing with their enterprise orders with their supplier. Addressing some or all of these dimensions ensures customer delight, satisfaction, and repeat business which are intangibles that drive any business.

CHAPTER 6

IMPLEMENTATION AND EVALUATION

6.1 Implementation

We deployed an Ethereum blockchain connected to multiple participating nodes that represent cloud providers and other stakeholders in the multi-organization supply chain. We conducted all of the smart contract executed federated learning evaluation using a well known MovieLens dataset [32] and our private cloud dataset. We ran the evaluations using a different number of users, items, and rounds. Our implementation relies on decentralized architecture and smart contracts that coordinate the training process between cloud providers and worker nodes. Owing to this, we observed a few extra seconds of delay for each round of execution. This extra time is attributed to the time it takes for the smart contract to initialize the process with the worker nodes and the rest of the additional time is consumed during the process of pushing and pulling the encrypted data from IPFS. It is our intent to demonstrate that for industry applications, we can replace the central server with decentralized distributed worker nodes using blockchain and smart contracts while maintaining the same model accuracy. Table 6 shows the observed difference in time between the distributed FedSmarteum and the centralized implementation. Figure 6 compares the execution time between FedSmarteum and the centralized implementation. In this experiment, we ran both models on the cloud dataset including 20 customers with 40 items (multi-cloud resources and services). The difference in time

between the models in negligible which validates the decentralized approach applicability of FedSmarteum Figures 22, 23, 24.

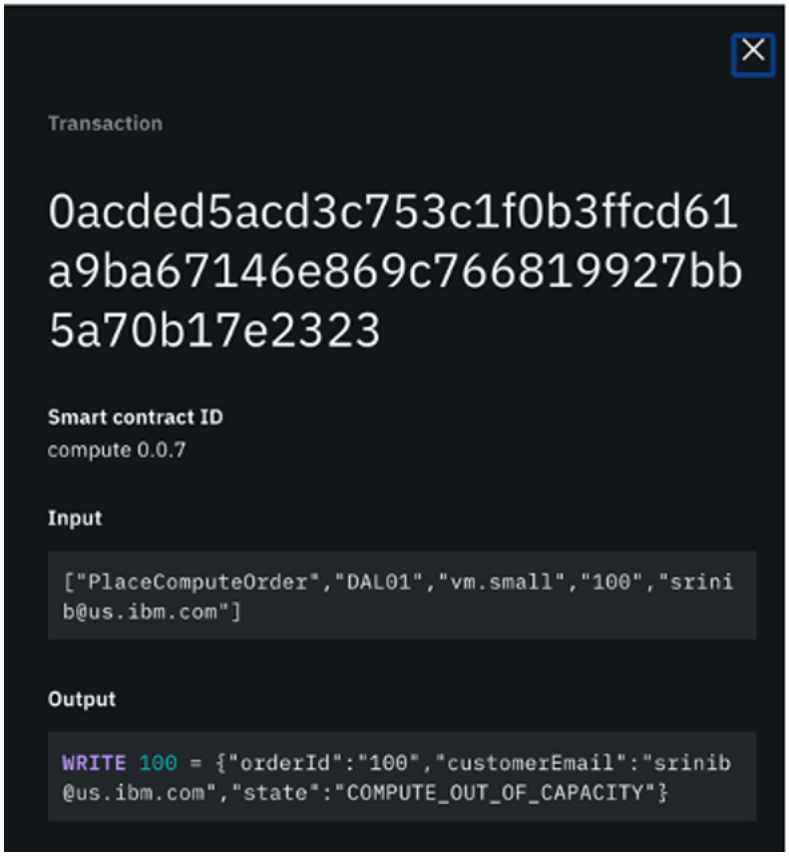


Figure 22: Place compute order transaction block in the ledger.

We deployed an Ethereum blockchain connected to multiple participating nodes that represent cloud providers and other stakeholders in the multi-organization supply chain as shown in Figure 25. We conducted all of the smart contract executed federated learning evaluation using a well-known MovieLens dataset [32] and our private cloud dataset. We ran the evaluations using a different number of users, items, and rounds.

```
// send event
let orderEvent = {
    type: "PlaceComputeOrder",
    customerEmail: customerEmail,
    orderId: orderId,
    status: order.state,
    dataCenterId: dataCenterId,
    applianceId: customerEmail+"_"+computeId
};

await ctx.stub.setEvent('OrderEvent', Buffer.from(JSON.stringify(orderEvent)));

// return compute order object
return JSON.stringify(order);
```

Figure 23: Place compute order order event.

Our implementation relies on decentralized architecture and smart contracts that coordinate the training process between cloud providers and worker nodes. Owing to this, we observed a few extra seconds of delay for each round of execution. This extra time is attributed to the time it takes for the smart contract to initialize the process with the worker nodes and the rest of the additional time is consumed during the process of pushing and pulling the encrypted data from IPFS. It is our intent to demonstrate that for industry applications, we can replace the central server with decentralized distributed worker nodes using blockchain and smart contracts while maintaining the same model accuracy. Table 6 shows the observed difference in time between the distributed FedSmarteum and centralized FedMF federated learning architectures. Figures 26, 27, 28, 29 compare the execution time between FedSmarteum and FedMF. In figure 26, we ran both models on the cloud dataset including 20 customers with 40 items (multi-cloud resources and services), Figure 27 we used 50 customers with 45 items, Figure 28 we used 75 customers

```
// The process of procuring a compute in a given data center by a customer
sync PlaceComputeOrder(ctx, dataCenterId, computeType, orderId,  customerEmail) {
    console.log("PlaceComputeOrder");

    // Validate: get datacenter
    let dataCenterData = await ctx.stub.getState(dataCenterId);
    let dataCenter;
    if (dataCenterData) {
        dataCenter = JSON.parse(dataCenterData.toString());
    } else {
        throw new Error('datacenter not found');
    }

    // get computes from the purchase order
    let computeId = dataCenterId+"_"+computeType;
    let computeData =  await ctx.stub.getState(computeId);
    let compute;
    if (computeData) {
        compute = JSON.parse(computeData.toString());

    } else {
        throw new Error ("compute not found")
    }

    // get the order from world state if it exists
    let orderData;

    try {
        orderData = await ctx.stub.getState(orderId);
    } catch (error) {
        console.log(error);
        orderData = false;
    }


    let order = {
        orderId: orderId,
        customerEmail: customerEmail,
        state: ComputeProcurementState.ComputeRequested
    };

    // get capacity of the machine type in the datacenter
    let quantity  = parseInt(compute.quantity)
    if (quantity == 0 ) {
        // not enough capacity
        order.state = ComputeProcurementState.ComputeNoCapacity
```

Figure 24: Smart Contract for place compute order.

with 45 items, and Figure 29 we used 100 customers with 45 items. As can be seen in Figure 26, the difference in time between the models is negligible which validates the decentralized approach applicability of FedSmarteum.

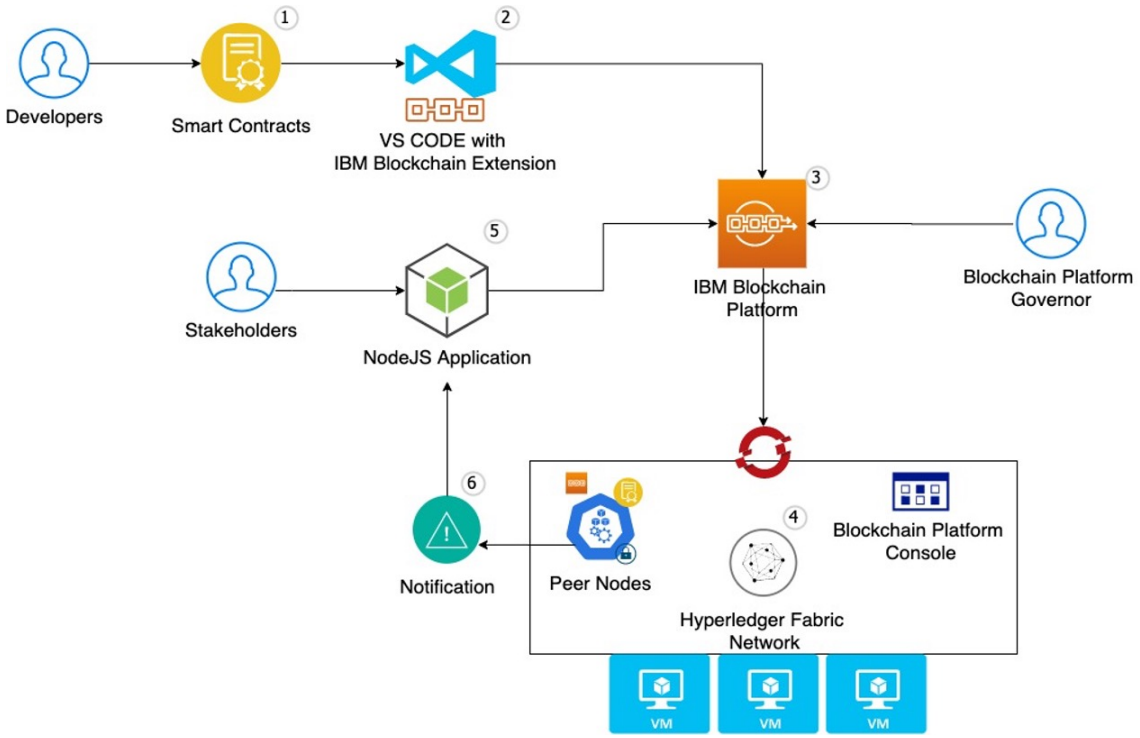


Figure 25: FedSmarteum Deployment framework.

6.2 Statistical Analysis

We look at the bar graphs Figures 26 - 29 and Table 6 to compare the time taken to train the model in two different modes distributed and centralized. This data indicates that it takes longer to train the model in distributed mode. However, we want to determine whether the difference is statistically significant.

In order to do that, we perform statistical analysis between distributed training

Table 6: Execution time for movielens dataset on a distributed and a centralized federated learning topologies.

		Distributed	Centralized
3 Users - 10 Items	R-1	0:00:42	0:00:40
	R-5	0:02:41	0:02:35
	R-10	0:05:34	0:05:08
	R-15	0:08:33	0:07:50
	R-20	0:10:54	0:09:27
	R-25	0:14:31	0:12:59
5 Users - 20 Items	R-1	0:01:55	0:01:48
	R-5	0:08:50	0:07:51
	R-10	0:16:39	0:15:31
	R-15	0:24:41	0:23:41
	R-20	0:31:41	0:28:31
	R-25	0:44:31	0:37:42
10 Users - 40 Items	R-1	0:06:25	0:06:09
	R-5	0:29:30	0:27:52
	R-10	1:02:54	0:57:02
	R-15	1:29:30	1:20:59
	R-20	1:56:15	1:50:39
	R-25	2:26:37	2:19:47

"R" represents the number of rounds. The time is expressed in Hours:Minutes:Seconds.

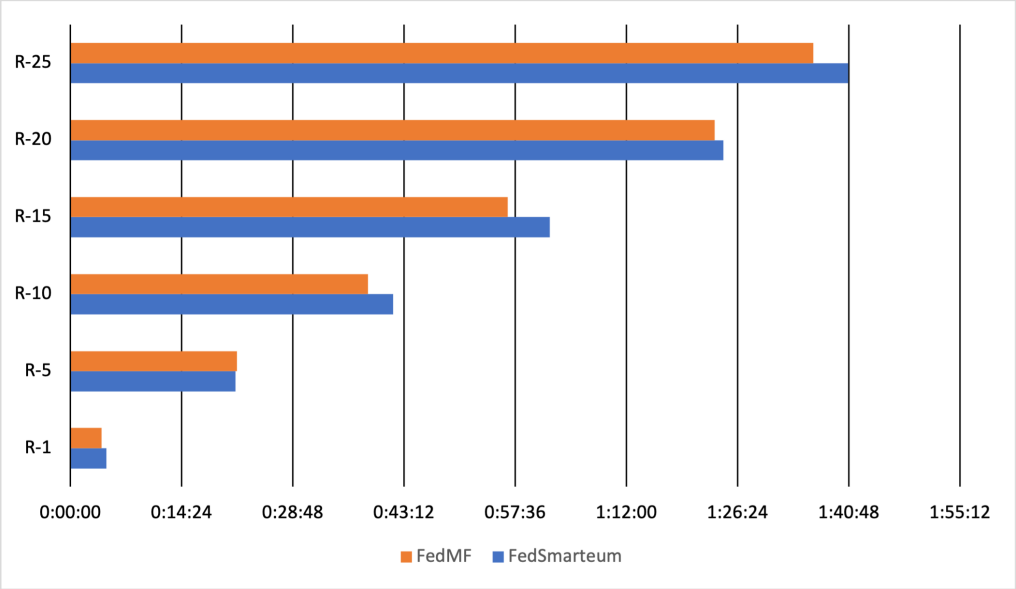


Figure 26: The execution time for centralized vs. distributed using movielens dataset. The experiment includes 20 customers with 40 items.

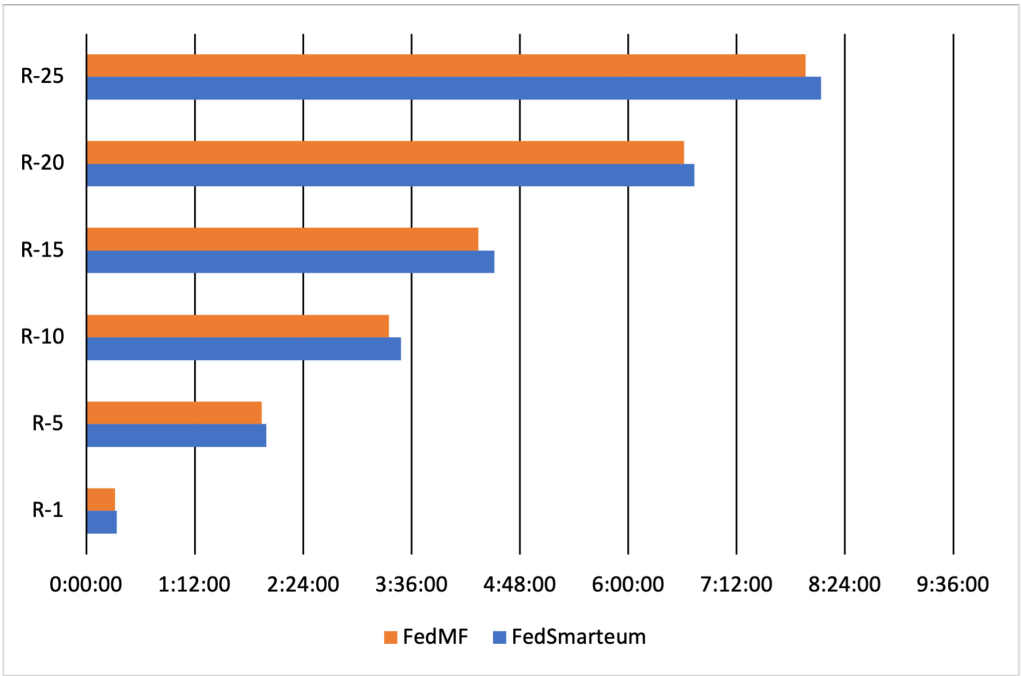


Figure 27: The execution time for centralized vs. distributed using our cloud dataset. The experiment includes 50 customers with 45 items.

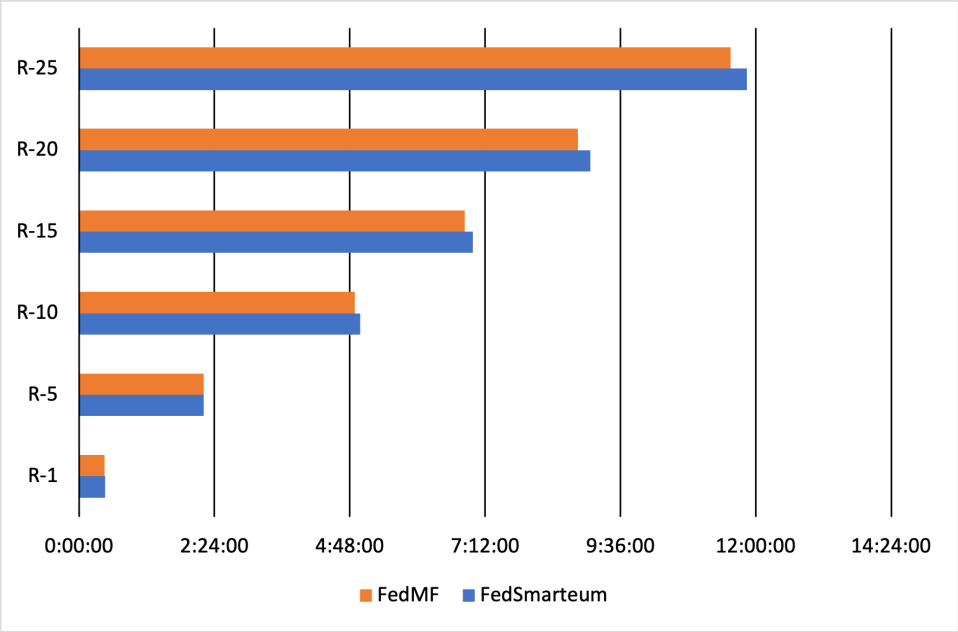


Figure 28: The execution time for centralized vs. distributed using our cloud dataset. The experiment includes 75 customers with 45 items.

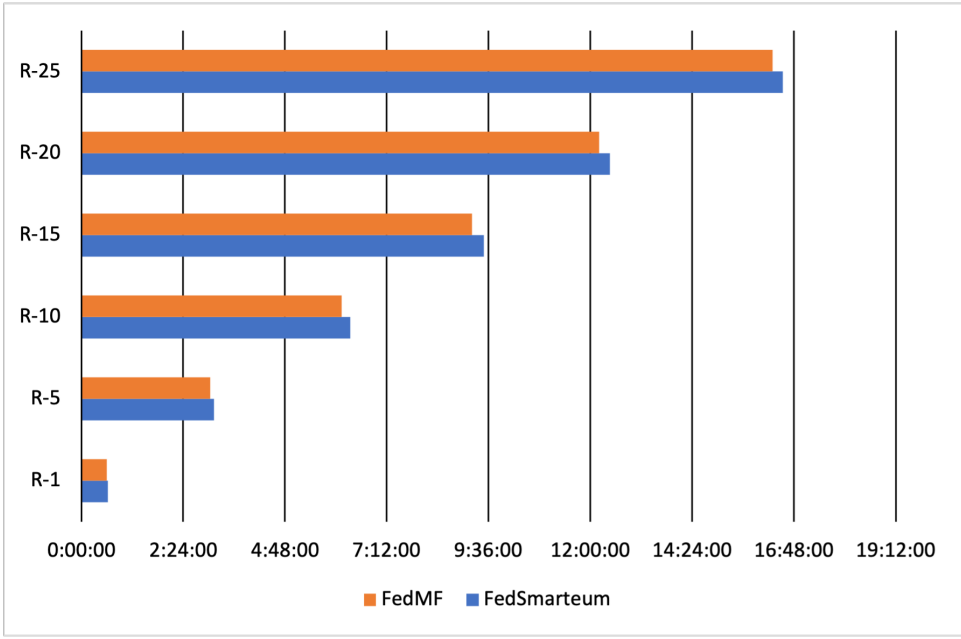


Figure 29: The execution time for centralized vs. distributed using our cloud dataset. The experiment includes 100 customers with 45 items.

mode and centralized training mode on two different data sets by comparing the means of the two groups. For purposes of brevity, we will select the following datasets for our statistical analysis.

Case 1: 10 customers, 40 items, Table 6

Case 2: 100 customers, 45 items, Figure 29

6.2.1 Hypothesis

Research Hypothesis: There is a *significant difference* in the means model-training time between distributed mode and centralized mode.

Null Hypothesis: There is *no significant difference* in the means model-training time between distributed mode and centralized mode.

6.2.2 Choosing the right statistical analysis model

The right **test to choose** depends on the hypothesis (what we want to analyze) and the data set.

(i) The hypothesis compares the means between the two groups (distributed vs centralized).

(ii) The data type is time duration, which is a ratio data type.

We think the two-tailed independent-samples t-test is the appropriate test since it is used to compare the means between two groups. Furthermore, the basic assumptions of the t-test is met:

1 The data consists of one dependent variable (the time duration) which is a ratio data type.

2 The data consists of one dependent variable that consists of two categorical, independent groups distributed training mode and centralized training mode.

We will run other statistical tests to verify the assumptions of the t-test before performing the t-test analysis. These are the statistical assumptions that must be met:

1 The data is independent.

2 The data is normally distributed.

3 The data is homogenous.

As discussed above, we will run the t-test twice, one for each pair of data set.

6.2.3 Case 1: 10 customers, 40 items

Test for data Independence: Yes, since the two groups do not have any relationship or influence between observations.

Testing for Normality: The data is normally distributed because Shapiro-Wilk test (Figure 30) indicates that $p > 0.05$. Furthermore, the box plots (Figure 33) and the Q-Q Plots (Figures 31 32) indicate the data is normally distributed.

Test for homogeneity of Variances: Met. Variances are the same since (p=0.882) > 0.05 as shown in Figure 34

The t-statistics is 0.161, p=0.876. Because p is > 0.05, this is not statistically significant.

Tests of Normality

	Training_mode	Kolmogorov-Smirnov[a] Statistic	df	Sig.	Shapiro-Wilk Statistic	df	Sig.
Time	distributed	.140	6	.200*	.978	6	.941
	centralized	.134	6	.200*	.978	6	.941

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Figure 30: Tests for Normality.

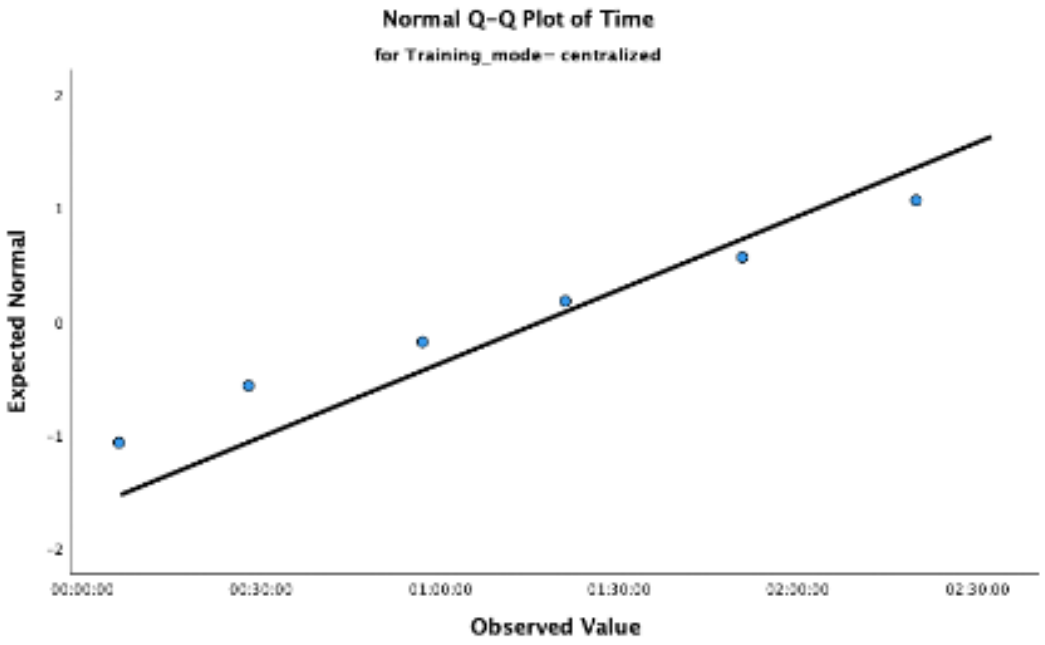


Figure 31: Normal Q-Q plot of time for centralized.

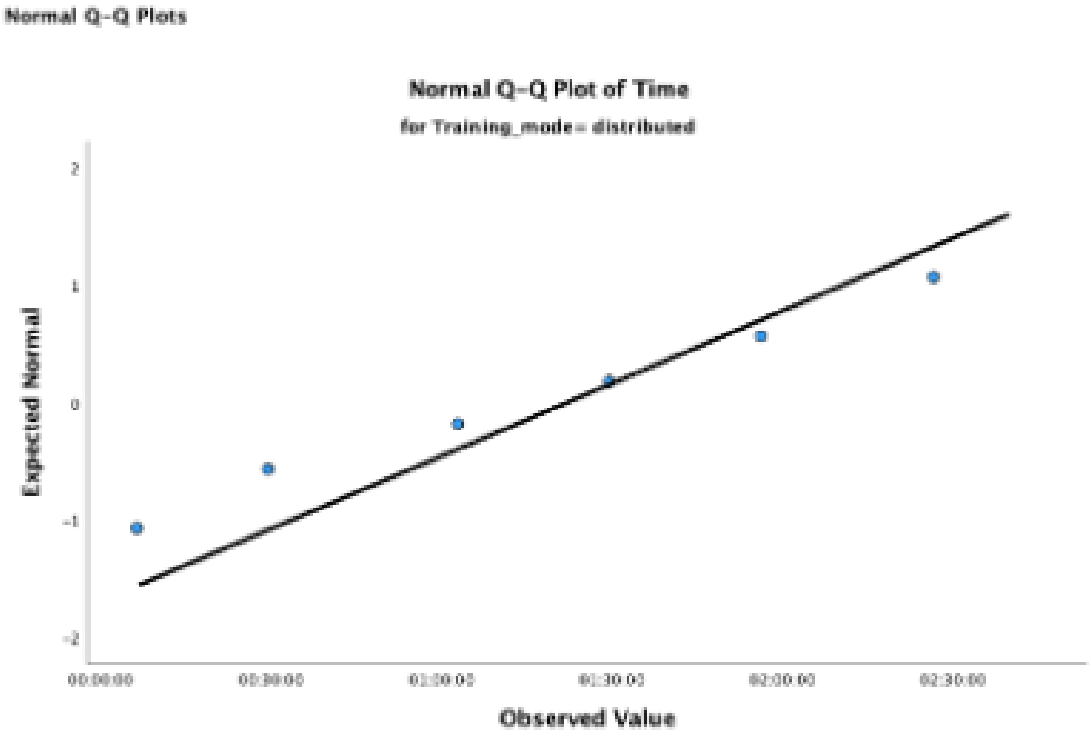


Figure 32: Normal Q-Q plot of time for distributed.

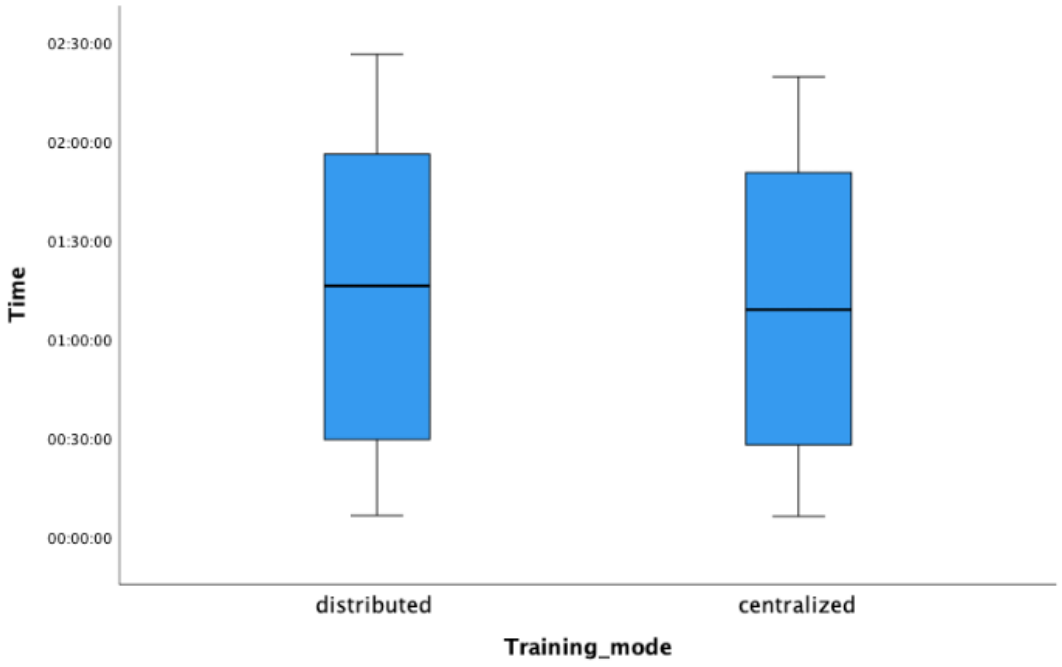


Figure 33: Box plot depicting normally distributed and centralized.

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means							
						Significance		Mean	Std. Error	95% Confidence Interval of the Difference	
		F	Sig.	t	df	One-Sided p	Two-Sided p	Difference	Difference	Lower	Upper
Time	Equal variances assumed	.023	.882	.161	10	.438	.876	0:04:47	0:29:47	-1:01:35	1:11:09
	Equal variances not assumed			.161	9.976	.438	.876	0:04:47	0:29:47	-1:01:36	1:11:10

Figure 34: Levene's test for homogeneity of variances.

Group Statistics

	Training_mode	N	Mean	Std. Deviation	Std. Error Mean
Time	distributed	6	1:15:11	0:52:50	0:21:34
	centralized	6	1:10:24	0:50:19	0:20:32

Figure 35: Group Statistics.

Effect Size: Although we have discovered that the difference is not statistically significant, we still need to calculate the effect size. As shown in Figure 37, Coden's d = 0.093, which is considered a small effect.

Conclusion: We wanted to know whether the difference in the means of the time required to train the model between distributed mode and centralized mode is significant. The research hypothesis is "There is a *significant difference* in the means model-training time between distributed mode and centralized mode". The Null Hypothesis is "There is *no significant difference* in the means model-training time between distributed mode and centralized mode".

We chose a two-tailed independent samples t-test and tested the assumptions. The data is normally distributed based on the plots and Shapiro-Wilk's test (Sig (p) $>$ 0.05).

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means							
						Significance				95% Confidence Interval of the Difference	
		F	Sig.	t	df	One-Sided p	Two-Sided p	Mean Difference	Std. Error Difference	Lower	Upper
Time	Equal variances assumed	.023	.882	.161	10	.438	.876	0:04:47	0:29:47	-1:01:35	1:11:09
	Equal variances not assumed			.161	9.976	.438	.876	0:04:47	0:29:47	-1:01:36	1:11:10

Figure 36: Independence Samples test.

Independent Samples Effect Sizes

		Standardizer[a]	Point Estimate	95% Confidence Interval	
				Lower	Upper
Time	Cohen's d	0:51:35	.093	-1.042	1.223
	Hedges' correction	0:55:55	.086	-.961	1.128
	Glass's delta	0:50:19	.095	-1.043	1.224

a. The denominator used in estimating the effect sizes.
Cohen's d uses the pooled standard deviation.
Hedges' correction uses the pooled standard deviation, plus a correction factor.
Glass's delta uses the sample standard deviation of the control group.

Figure 37: Effect size.

The data was found to be homogenous. We found the effect size to be 0.093, which is small. The means from the two groups were 1:15:11 +- 0:52:50 and 1:10:24 +- 0:50:19 respectively. The t-statistic value is 0.161, (Sig (p)=0.876 $>$ 0.05). **Therefore, the null hypothesis was accepted**.

6.2.4 Case 2: 100 customers, 45 items

Test for data Independence: Yes, since the two groups do not have any relationship or influence between observations.

Testing for Normality: The data is normally distributed because Shapiro-Wilk test (Figure 38) indicates that p $>$ 0.05. Furthermore, the box plots (Figure 41) and the Q-Q Plots (Figures 39 40) indicate the data is normally distributed.

Test for homogeneity of Variances: Met. Variances are the same since (p=0.963) $>$ 0.05 as shown in figure 42

The t-statistics is 0.052, p=0.959. Because p is $>$ 0.05, this is not statistically significant.

Effect Size: Although we have discovered that the difference is not statistically significant, we still need to calculate the effect size. As shown in Figure 45, Coden's d = 0.030, which is considered a small effect.

Conclusion: We wanted to know whether the difference in the means of the time required to train the model between distributed mode and centralized mode is significant. The research hypothesis is "There is a *significant difference* in the means model-training time between distributed mode and centralized mode". The Null Hypothesis is "There is

Tests of Normality

	Training_mode	Kolmogorov-Smirnov[a] Statistic	df	Sig.	Shapiro-Wilk Statistic	df	Sig.
Time	distributed	.132	6	.200*	.980	6	.953
	centralized	.131	6	.200*	.980	6	.951

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Figure 38: Tests for Normality.

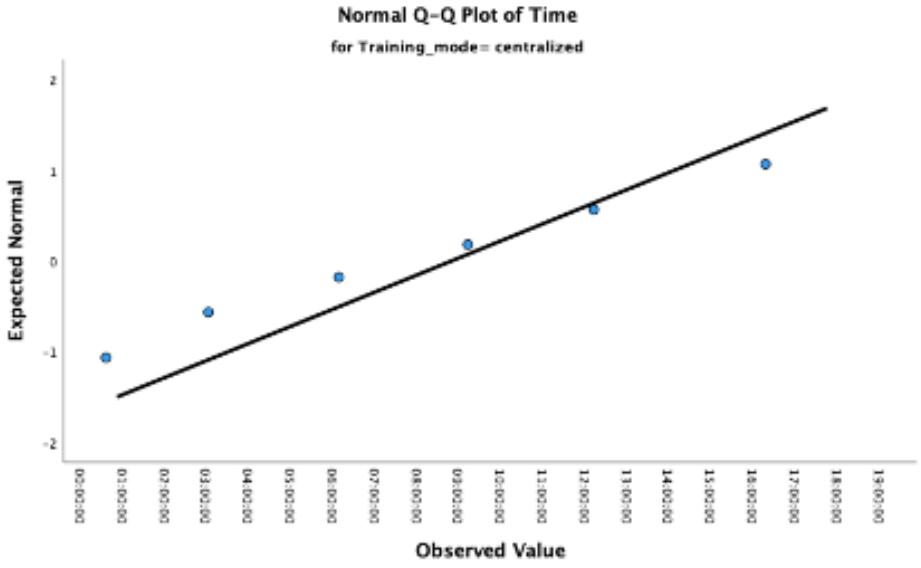


Figure 39: Normal Q-Q plot of time for centralized.

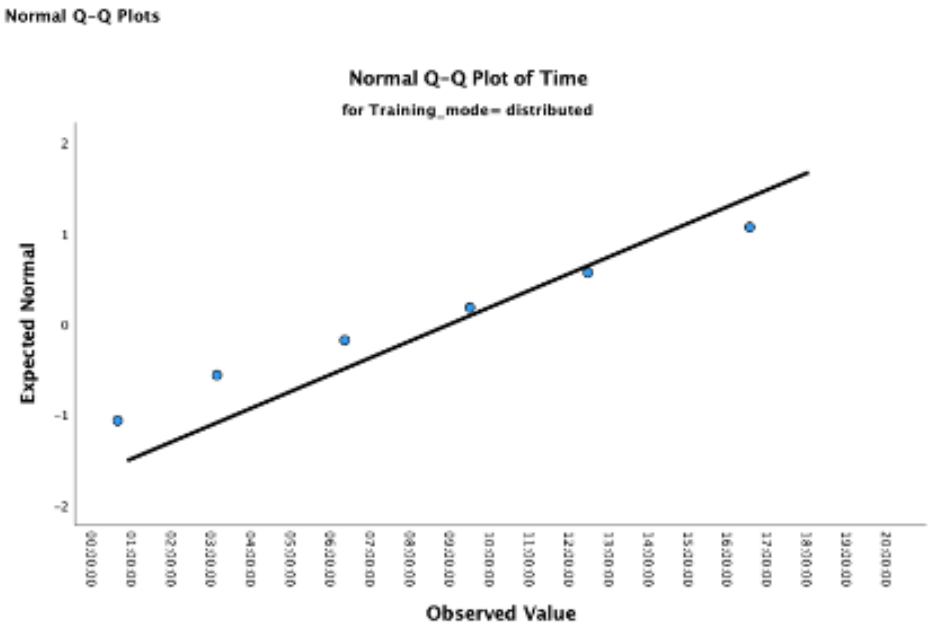


Figure 40: Normal Q-Q plot of time for distributed.

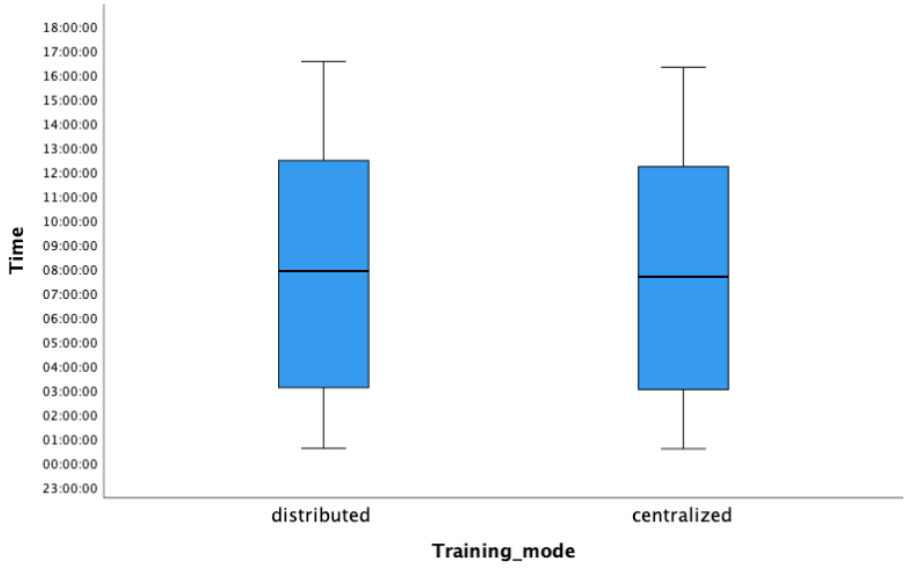
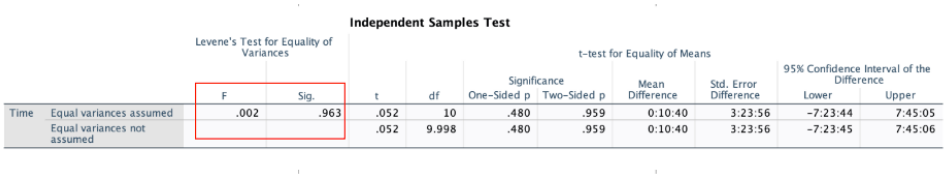


Figure 41: Box plot depicting normally distributed and centralized.



Independent Samples Test

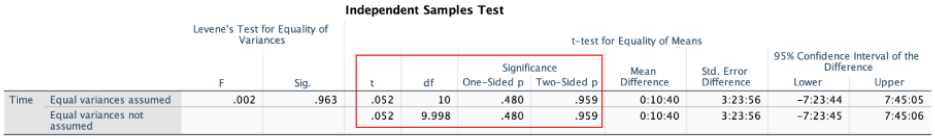
		Levene's Test for Equality of Variances		t-test for Equality of Means							
						Significance		Mean	Std. Error	95% Confidence Interval of the Difference	
		F	Sig.	t	df	One-Sided p	Two-Sided p	Difference	Difference	Lower	Upper
Time	Equal variances assumed	.002	.963	.052	10	.480	.959	0:10:40	3:23:56	-7:23:44	7:45:05
	Equal variances not assumed			.052	9.998	.480	.959	0:10:40	3:23:56	-7:23:45	7:45:06

Figure 42: Levene's test for homogeneity of variances.

Group Statistics

	Training_mode	N	Mean	Std. Deviation	Std. Error Mean
Time	distributed	6	8:05:31	5:55:53	2:25:17
	centralized	6	7:54:50	5:50:34	2:23:07

Figure 43: Group Statistics.



Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means							
						Significance		Mean	Std. Error	95% Confidence Interval of the Difference	
		F	Sig.	t	df	One-Sided p	Two-Sided p	Difference	Difference	Lower	Upper
Time	Equal variances assumed	.002	.963	.052	10	.480	.959	0:10:40	3:23:56	-7:23:44	7:45:05
	Equal variances not assumed			.052	9.998	.480	.959	0:10:40	3:23:56	-7:23:45	7:45:06

Figure 44: Independence Samples test.

Independent Samples Effect Sizes

		Standardizer[a]	Point Estimate	95% Confidence Interval Lower	Upper
Time	Cohen's d	5:53:14	.030	-1.102	1.161
	Hedges' correction	6:22:49	.028	-1.017	1.071
	Glass's delta	5:50:34	.030	-1.103	1.161

a. The denominator used in estimating the effect sizes.
Cohen's d uses the pooled standard deviation.
Hedges' correction uses the pooled standard deviation, plus a correction factor.
Glass's delta uses the sample standard deviation of the control group.

Figure 45: Effect size.

no significant difference in the means model-training time between distributed mode and centralized mode".

We chose a two-tailed independent samples t-test and tested the assumptions. The data is normally distributed based on the plots and Shapiro-Wilk's test (Sig (p) $>$ 0.05). The data was found to be homogenous. We found the effect size to be 0.030, which is small. The means from the two groups were 8:05:31 +- 5:55:53 and 7:54:50 +- 5:50:34 respectively. The t-statistic value is 0.052, (Sig (p)=0.959 $>$ 0.05). **Therefore, the null hypothesis was accepted**.

6.3 Benchmark Description and Workload

We conducted our experiments using two datasets. The first one is the MovieLens datasets [52] is a well-known benchmark that has been collected and made available by the GroupLens Research. [30] The dataset contains 100,000 movie ratings and 3,600 tag applications applied to 9724 movies by 610 users. The dataset is available at the following link: https://grouplens.org/datasets/movielens/.

The second dataset is our private cloud dataset. The premise for this dataset was to represent cloud resources (compute, storage, network, etc.,) procured by enterprise customers across multiple cloud providers in order to deploy their application workloads Figures 46, 47. This cloud dataset includes data for 50 different cloud deployable resources provisioned across multiple cloud providers and third-party vendors along with the count for each resource applied. We ran our evaluation using 50, 75, and 100 customers with 45 items.

Each user in the cloud dataset is represented with a specific id to keep track of all user's deployments and resources. When a user deploys and provisions a resource such as an x-type server, the server id will then be added to that specific user record. Our cloud dataset is represented using records where the user id is the key for each record. Resources deployed by a specific user will be added to his specific record to create a relationship between the user id and the resource id. The count is also included in this process, when a resource id is added to a user record, the number of resources will also be added to the record. This dataset can be represented using a graph where the main node is the user id, and the objects connected to the main nodes are the resources deployed by the user. Each resource node will have its own objects and entities such as the count, price, date, etc.,

6.4 Blockchain Implementation and Encryption

To implement the homomorphic encryption, we used the Python programming language. Using Paillier encryption, [59] we kept the length of the public key 1024. We used Truffle Suite [71] in order to deploy a synthetic Ethereum blockchain for development and testing purposes. In particular, we used Ganache version 2.5.4 [26] which allows interacting with smart contracts using Python. The smart contracts were deployed using Remix IDE. [42] All smart contracts were written in Solidity programming language using the compiler version ^0.4.21.

IPFS version 0.15.0 was used to allow the distributed encrypted items data to be shared. IPFS powers the Distributed Web using a peer-to-peer hypermedia protocol

	Profile name	Family	vCPUs	RAM	Bandwidth	Instance storage
○	mx2d-2x16	Memory	2	16 GiB	4 Gbps	1 x 75 GB
○	mx2-2x16	Memory	2	16 GiB	4 Gbps	–
○	mx2d-4x32	Memory	4	32 GiB	8 Gbps	1 x 150 GB
○	mx2-4x32	Memory	4	32 GiB	8 Gbps	–
○	mx2d-8x64	Memory	8	64 GiB	16 Gbps	1 x 300 GB
○	mx2-8x64	Memory	8	64 GiB	16 Gbps	–
○	mx2d-16x128	Memory	16	128 GiB	32 Gbps	1 x 600 GB
○	mx2-16x128	Memory	16	128 GiB	32 Gbps	–
○	mx2-32x256	Memory	32	256 GiB	64 Gbps	–
○	mx2d-32x256	Memory	32	256 GiB	64 Gbps	2 x 600 GB
○	mx2-48x384	Memory	48	384 GiB	80 Gbps	–
○	mx2d-48x384	Memory	48	384 GiB	80 Gbps	2 x 900 GB
○	mx2d-64x512	Memory	64	512 GiB	80 Gbps	2 x 1200 GB
○	mx2-64x512	Memory	64	512 GiB	80 Gbps	–
○	mx2-96x768	Memory	96	768 GiB	80 Gbps	–
○	mx2d-96x768	Memory	96	768 GiB	80 Gbps	2 x 1800 GB

Figure 46: Compute options for cloud provider machine types.

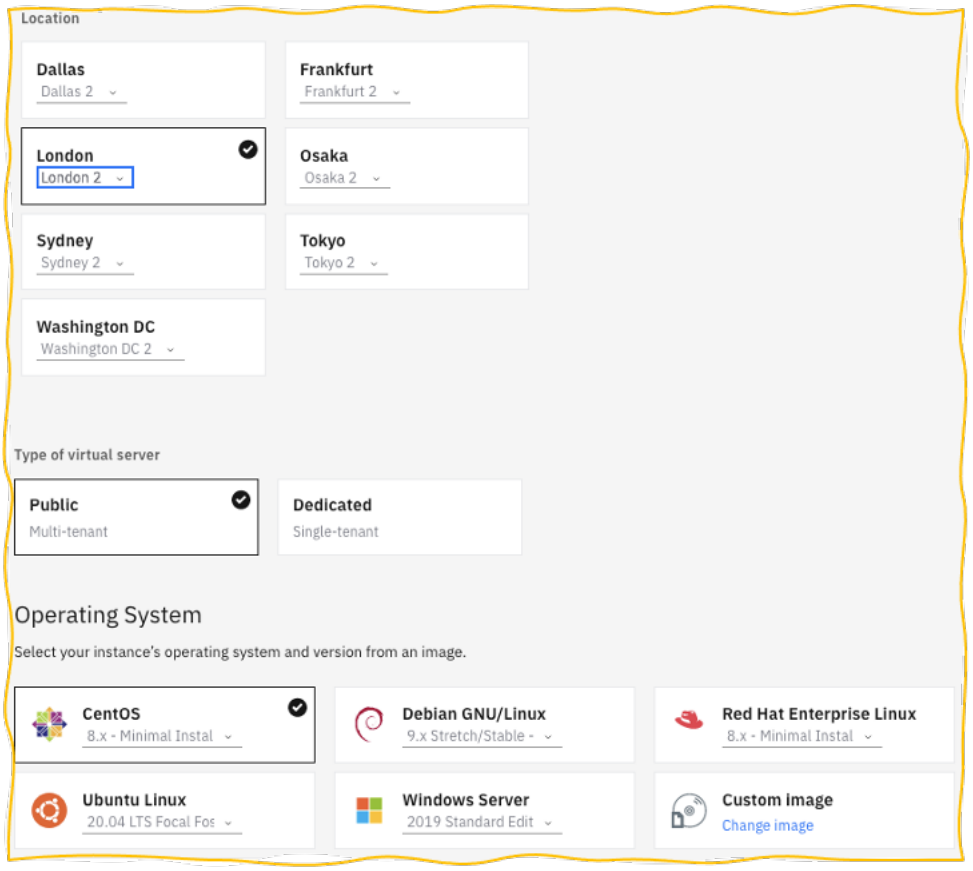


Figure 47: Multi-zone data centers, multi-tenant hosts, Operating system options.

designed to preserve and grow humanity's knowledge by making the web upgradeable, resilient, and more open. [39]

6.5 FedSmarteum APIs

As mentioned before, we injected our smart contracts in Python through different APIs that interact with the smart contracts interface. In listing 1 is the APIs from the user that interact with smart contracts directly in order to update their gradients. Listing 2 shows how the loss is being calculated. Listing 3 shows how worker nodes iterate over the items and interact with smart contracts to update the items for different users. Listing 4 shows the actual wrapper for Python APIs that call the smart contract APIs.

Listing 1: User update on worker nodes.

```
def user_update(single_user_vector, user_rating_list, pos):

    gen_hash = smart_contract.getItemsVec()
    subprocess.run(['ipfs', 'get', gen_hash], stdout=subprocess.
        PIPE)

    f=open('items_v.p', 'rb')
    encrypted_item_vector = pickle.load(f)
    f.close()

    item_vector = np.array([[private_key.decrypt(e) for e in
        vector] for vector in encrypted_item_vector], dtype=np.
        float32)

    gradient = {}
    for item_id, rate, _ in user_rating_list:
```

```
        error = rate - np.dot(single_user_vector, item_vector[
            item_id])
        single_user_vector = single_user_vector - lr * (-2 *
            error * item_vector[item_id] + 2 * reg_u *
            single_user_vector)
        gradient[item_id] = lr * (-2 * error *
            single_user_vector + 2 * reg_v * item_vector[item_id
            ])

    encrypted_gradient = {vector: [public_key.encrypt(e,
        precision=1e-5) for e in gradient[vector]] for vector in
        gradient}

    f=open('user_out.p', 'wb')
    pickle.dump({'single_user_vector':single_user_vector, '
        encrypted_gradient':encrypted_gradient}, f)
    f.close()

    user_hash = subprocess.run(['ipfs', 'add', 'user_out.p'],
        stdout=subprocess.PIPE)
    user_hash = user_hash.stdout.decode('utf-8').split()[1]

    smart_contract.setGradHash(user_hash, pos)
```

Listing 2: Calculating loss.

```
def loss():
    loss = []
    # User updates
    for i in range(len(user_id_list)):
        for r in range(len(train_data[user_id_list[i]])):
            item_id, rate, _ = train_data[user_id_list[i]][r]
```

```
            error = (rate - np.dot(user_vector[i], item_vector[
                item_id])) ** 2
            loss.append(error)
    return np.mean(loss)
```

Listing 3: Calling smart contracts and passing gradients.

```
encrypted_item_vector = [[public_key.encrypt(e, precision=1e-5)
   for e in vector] for vector in item_vector]
    f=open('items_v.p', 'wb')
    pickle.dump(encrypted_item_vector, f)
    f.close()

    gen_hash = subprocess.run(['ipfs', 'add', 'items_v.p'],
       stdout=subprocess.PIPE)
    gen_hash = gen_hash.stdout.decode('utf-8').split()[1]

    smart_contract.setItemsVec(gen_hash)

    for iteration in range(max_iteration):

        encrypted_gradient_from_user = []
        user_time_list = []
        for i in range(len(user_id_list)):

            user_update(user_vector[i], train_data[user_id_list[
                i]], i)
            returned_hash = smart_contract.getGradHash(i)

            subprocess.run(['ipfs', 'get', returned_hash],
                stdout=subprocess.PIPE)
```

```
            f=open('user_out.p', 'rb')
            pobj = pickle.load(f)
            f.close()
            user_vector[i] = pobj['single_user_vector']
            gradient = pobj['encrypted_gradient']
            encrypted_gradient_from_user.append(gradient)

        for g in encrypted_gradient_from_user:
            for item_id in g:
                for j in range(len(encrypted_item_vector[item_id
                    ])):
                    encrypted_item_vector[item_id][j] =
                        encrypted_item_vector[item_id][j] - g[
                        item_id][j]
```

Listing 4: Python based Smart contract APIs

```
def setItemsVec(new_hash):
        tx_hash = contract.functions.setItemsVector(new_hash).
            transact()
        web3.eth.waitForTransactionReceipt(tx_hash)
        print('The new hash for Items Vector has been set to: {}
            '.format(
            contract.functions.getItemsVectorHash().call()
        ))

def getItemsVec():
        return contract.functions.getItemsVectorHash().call()

def getWorkerAdd():
        return contract.functions.getWorkerAddress().call()
```

```
def setGradHash(new_hash, pos):
        tx_hash = contract.functions.setGradientsHash(new_hash,
            pos).transact()
        web3.eth.waitForTransactionReceipt(tx_hash)
        print('Done updating the grad hash array')

def getGradHash(pos):
        return contract.functions.getGradientsHash(pos).call()
```

In this section thus far we demonstrated performance and scalability characteristics among other things of critical elements of novelty in the FedSmarteum system such as IPFS, federated learning, HE, etc., It is important to note that executing an enterprise customer purchase order across multiple cloud provider's supply chains to massive scale is both cost and time prohibitive. It is critical to call out that in addition to multi-organization automation, there are several other intangibles that need to be considered by both consumers and suppliers to orchestrate and reap benefits from multi-organization automation. Table 7 attempts to summarize some of the key dimensions of implementing FedSmarteum and deploying it as a plug and play system compared to several existing supply chains that may have pockets of automation and optimization.

Furthermore, with such a system, we inch closer to building multi-organization systems across any industry that is willing to share. For example pharmaceuticals and drug delivery companies. Such organizations with research funding can share data across a blockchain network that trains on its local privacy-preserved data and shares the model outcomes and training results. Recommendation systems can be built to reduce or avoid

Table 7: Tangible and intangible perspectives for multi-org organization supply chain

Area	**Existing Supply Chains**	**FedSmarteum**
Novelty	No similar systems	FedSmarteum
Automation	Partial and confined to company's firewalls, SLAs are hard to meet	Makes multi-org automaton possible
Transparency	Siloed data visibility, minimal transparency	Will drive incentivize increased transparency
Elapsed time	Days/weeks to handle non-digital	Hours (potentially), with stakeholder agreement
Incentive to improve	Lack of clear E2E accountability	Hold supply chain blocking entities accountable
Future work	Peephole optimizations, incremental at best	AI to improve predictions and preempt issues, increased stakeholder collaboration

unnecessary research, be highly accurate saving both time and money by alerting participating entities of training results. The smart contracts orchestrate the learning so all epochs and rounds are calculated and immutable. This can easily lend itself to *attribution* of who contributed what and how much to the model accuracy. Such a transparent system allows for revenue sharing models as well as build trusts across the board.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Democratization via a decentralized architecture provides equal rights for clients to share unbiased data. The effective use of blockchain to drive the underpinnings of the supply chain in general and specifically for cloud provider supply chain procurement and deployment will vastly change the landscape of customer, cloud provider, and their vendors. This will help cloud providers focus their attention on improving their internal processes and automation and servicing their customers even more efficiently with trust in resiliency in their engagement. AI has cemented its place as the tool of choice for analytic workloads and also helps build predictive models that are able to assist and automate workflows across multi-organizations business networks. Current implementations lean toward federated learning for training AI models on private data. However, federated learning relies on a central server to perform the global computation progress which can lead to several issues related to handling data heterogeneity between parties, coordination of learning process, bias due to lack of verification datasets, single point of failure, and above all communication overhead between parties involved. We proposed a decentralized federated learning matrix factorization technique implemented using a blockchain. Our implementation leverages the use of smart contracts to orchestrate and automate the asset procurement process. This immutable, decentralized architecture reduces the overhead communication between the server and all the nodes while retaining data ownership

and privacy with participants. Additionally, we eliminated the single point of failure and proved that decentralized recommendation systems can be implemented using blockchain while maintaining the model accuracy with marginal overhead in computational time. We hope this paper will inspire cloud providers and their supplier community to consider a decentralized framework as they engineer their next generation of supply chain automation.

We believe the watershed moment for transparent and trustworthy multi-organization supply chain orchestration has arrived. To further our research in this area, we plan to focus on reducing the time required to complete a single iteration and worker node updates using our decentralized federated learning technique. We also hope to design more accurate prediction models for resource procurement and deployment through the effective use of reinforced learning. We hope to explore similar techniques to homomorphic encryption that requires less computing and updating time. Reducing the communication between worker nodes, IPFS, and smart contracts will reduce the execution time in each iteration. We are also investigating model governance and how cloud providers can have access to the model based on the provided training data and accuracy before and after including each cloud provider. Finally, we want to further improve this framework's plug-and-play delivery model to solve various other industry supply chains challenges.

CHAPTER 8

APPENDIX

Listing 5: A simple contract

```
pragma solidity ^0.5.1;
contract ProvisioningBlockChain {
    string systemName;
    consructor() public {
        SystemName = "ProvisioningBC";
    }
    function getSystenName() public view returns (string memory)
        {
        return systemName;
    }
    function setSystemName(string memory _sysName) public {
        systemName = _sysName;
    }
}
```

Listing 6: Using the public keyword

```
pragma solidity ^0.5.1;
contract ProvisioningBlockChain {
    string public systemName = "ProvisioningBC";
    function setSystemName(string memory _sysName) public {
        systemName = _sysName;
    }
}
```

Listing 7: Using enum in a smart contract

```
pragma solidity ^0.5.1;
contract ProvisioningBlockChain {
    enum State{Waiting, Dev, Testing, Activate}
    State public pbcState = State.Waiting;
    function pbcDev() public {
        pbcState = State.Dev;
    }
    function isActivate() public view returns(bool){
        return pbcState == State.Active;
    }
}
```

Listing 8: Four services in the smart contract

```
pragma solidity ^0.5.1;
contract ProvisioningBlockChain {
    // A data structure will be added here
    function addService(address service) public{
        require(!isCurrentService(service));
        // TODO
    }
    funcion delService(address service) public{
        (require(!isCurrentService(service));
        // TODO
    }
    function isCurrentService(address service) public view
       returns (bool){
        // TODO
    }
    function getAllServices() public view returns (address[]
       memory){
```

```
        // TODO
    }
}
```

Listing 9: Using mapping in the smart contract

```
pragma solidity ^0.5.1;
contract ProvisioningBlockChain {
    mapping (address => bool) services;
    // A data structure will be added here
    function addService(address service) public{
        require(!isCurrentService(service));
        services[service] = true;
    }
    funcion delService(address service) public{
        (require(!isCurrentService(service));
        services[service] = false;
    }
    function isCurrentService(address service) public view
       returns (bool){
        return services[service];
    }
    function getAllServices() public view returns (address[]
       memory){
        // TODO
    }
}
```

Listing 10: Using arrays

```
pragma solidity ^0.5.1;

contract ProvisioningBlockChain{
```

```
address[] services;
function addService(address service) public{
    require(!isCurrentService(service));
    services.push(service);
}
funcion delService(address service) public{
    (bool found, uint256 index) = _getServiceIndex(service);
    require(found);
    for(uint256 i = index; i< services.length; ++i){
        services[i-1] = services[i];
    }
    services.pop();
}
function isCurrentService(address service) public view
   returns (bool){
    (bool found, ) = _getServiceIndex(service);
    return found;
}
function getAllServices() public view returns (address[]
   memory){
    return services;
}

function _getServiceIndex(address service) internal view
   returns(bool, uint256){
    for(uint256 i = 0; i<services.length; ++i){
        if(service == services[i]){
            return(true, i)
        }
    }
    return(false, 0);
```

```
    }
}
```

Listing 11: Mapping address to another address

```
mapping(address => address) _nextService;
unit256 public listSize;
address constant placeHolder = address(1);
constructor() public {
    _nextService[placeHolder] = placeHolder;
}
```

Listing 12: Adding a service

```
function addService(address service) public{
    require(!isCurrentService(service));
    _nextServices[service] = _nextServices[placeHolder];
    _nextServices[placeHolder] = service;
    listSize++;
}
```

Listing 13: Deleting a service

```
function delService(address service) public{
    require(isCurrentService(service));
    address prevService = _getPrevService(service);
    _nextServices[prevService] = _nextServices[service];
    _nextServices[service] = address(0);
    listSize--;
}
```

Listing 14: Get the previous service

```
function _getPrevService(address service) internal view returns(
    address){
     address curAddress = placeHolder;
     while(_nextServices[curAddress] != placeHolder){
          if(_nextServices[curAddress] == service){
               return curAddress;
          }
          curAddress = _nextServices[curAddress];
     }
     return address(0);
}
```

Listing 15: List all existing services

```
function getAllServices() public view returns(address[] memory){
     address[] memory services = new address[](listSize);
     address curAddress = _nextServices[placeHolder];
     for(uint256 i = 0; curAddress != placeHolder; ++i){
          services[i] = curAddress;
          curAddress = _nextServices[curAddress];
     }
     return services;
}
```

In this section, we provide few code snippets to help with results reproduction. For example, listing 16

Listing 16: User update on worker nodes.

```
def user_update(single_user_vector, user_rating_list, pos):

     gen_hash = smart_contract.getItemsVec()
```

```
subprocess.run(['ipfs', 'get', gen_hash], stdout=subprocess.
    PIPE)

f=open('items_v.p', 'rb')
encrypted_item_vector = pickle.load(f)
f.close()

item_vector = np.array([[private_key.decrypt(e) for e in
    vector] for vector in encrypted_item_vector],
                       dtype=np.float32)

gradient = {}
for item_id, rate, _ in user_rating_list:
    error = rate - np.dot(single_user_vector, item_vector[
        item_id])
    single_user_vector = single_user_vector - lr * (-2 *
        error * item_vector[item_id] + 2 * reg_u *
        single_user_vector)
    gradient[item_id] = lr * (-2 * error *
        single_user_vector + 2 * reg_v * item_vector[item_id
        ])

encrypted_gradient = {vector: [public_key.encrypt(e,
    precision=1e-5) for e in gradient[vector]] for vector in
    gradient}


f=open('user_out.p', 'wb')
pickle.dump({'single_user_vector':single_user_vector, '
    encrypted_gradient':encrypted_gradient}, f)
f.close()
```

```
    user_hash = subprocess.run(['ipfs', 'add', 'user_out.p'],
        stdout=subprocess.PIPE)
    user_hash = user_hash.stdout.decode('utf-8').split()[1]

    smart_contract.setGradHash(user_hash, pos)
```

Listing 17: Calculate user model loss.

```
def loss():
    loss = []
    # User updates
    for i in range(len(user_id_list)):
        for r in range(len(train_data[user_id_list[i]])):
            item_id, rate, _ = train_data[user_id_list[i]][r]
            error = (rate - np.dot(user_vector[i], item_vector[
                item_id])) ** 2
            loss.append(error)
    return np.mean(loss)
```

Listing 18: Update the items model distributed

```
timeor = time.time()
    t = time.time()
    encrypted_item_vector = [[public_key.encrypt(e, precision=1e
        -5) for e in vector] for vector in item_vector]
    f=open('items_v.p', 'wb')
    pickle.dump(encrypted_item_vector, f)
    f.close()

    gen_hash = subprocess.run(['ipfs', 'add', 'items_v.p'],
        stdout=subprocess.PIPE)
    gen_hash = gen_hash.stdout.decode('utf-8').split()[1]
```

```
smart_contract.setItemsVec(gen_hash)

print('Item profile encrypt using', time.time() - t, '
    seconds')

for iteration in range(max_iteration):

    print('Iteration', iteration)

    # Step 2 User updates
    encrypted_gradient_from_user = []
    user_time_list = []
    for i in range(len(user_id_list)):

        user_update(user_vector[i], train_data[user_id_list[
            i]], i)
        returned_hash = smart_contract.getGradHash(i)

        subprocess.run(['ipfs', 'get', returned_hash],
            stdout=subprocess.PIPE)

        f=open('user_out.p', 'rb')
        pobj = pickle.load(f)
        f.close()
        user_vector[i] = pobj['single_user_vector']
        gradient = pobj['encrypted_gradient']
        encrypted_gradient_from_user.append(gradient)

    # Step 3 Server update
    for g in encrypted_gradient_from_user:
```

```
            for item_id in g:
                for j in range(len(encrypted_item_vector[item_id
                    ])):
                    encrypted_item_vector[item_id][j] =
                        encrypted_item_vector[item_id][j] - g[
                        item_id][j]

        # for computing loss
        item_vector = np.array([[private_key.decrypt(e) for e in
            vector] for vector in encrypted_item_vector])
        print('loss', loss())

        m, s = divmod(int(time.time() - timeor), 60)
        h, m = divmod(m, 60)
        print(f'{h:d}:{m:02d}:{s:02d}')
```

Listing 19: Calculate user accuracy distributed.

```
prediction = []
    real_label = []

    # testing
    for i in range(len(user_id_list)):
        p = np.dot(user_vector[i:i + 1], np.transpose(
            item_vector))[0]

        r = test_data[user_id_list[i]]

        real_label.append([e[1] for e in r])
        prediction.append([p[e[0]] for e in r])

    prediction = np.array(prediction, dtype=np.float32)
```

```
    real_label = np.array(real_label, dtype=np.float32)

    print('rmse', np.sqrt(np.mean(np.square(real_label -
        prediction))))
```

Listing 20: Smart Contract API calls.

```
import json
from web3 import Web3

# Set up web3 connection with Ganache
ganache_url = "http://127.0.0.1:7545"
web3 = Web3(Web3.HTTPProvider(ganache_url, request_kwargs={'
    timeout': 50}))

# Set a default account to sign transactions - this account is
    unlocked with Ganache
web3.eth.defaultAccount = web3.eth.accounts[0]
# Greeter contract ABI

abi = json.loads('')

address = web3.toChecksumAddress('') # FILL ME IN

# Initialize contract
contract = web3.eth.contract(address=address, abi=abi)



def setItemsVec(new_hash):
    tx_hash = contract.functions.setItemsVector(new_hash).
        transact()
    web3.eth.waitForTransactionReceipt(tx_hash)
```

```
    print('The new hash for Items Vector has been set to: {}'.
        format(
          contract.functions.getItemsVectorHash().call()
    ))

def getItemsVec():
    return contract.functions.getItemsVectorHash().call()

def getWorkerAdd():
    return contract.functions.getWorkerAddress().call()

def setGradHash(new_hash, pos):
    tx_hash = contract.functions.setGradientsHash(new_hash, pos)
        .transact()
    web3.eth.waitForTransactionReceipt(tx_hash)
    print('Done updating the grad hash array')

def getGradHash(pos):
    return contract.functions.getGradientsHash(pos).call()
```

REFERENCE LIST

[1] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal, 2018), ACM, Eurosys '18, pp. 1–15.

[2] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference* (2018), pp. 1–15.

[3] Atzei, N., Bartoletti, M., and Cimoli, T. A survey of attacks on Ethereum smart contracts. In *International conference on financial cryptography and data security* (New York, NY, 2016), IACR Cryptology ePrint archive, IACR.org, p. 1007.

[4] Beedham, Matthew. The difference between permissioned and permissionless blockchains. `https://thenextweb.com/hardfork/2018/11/05/permissioned-permissionless-blockchains`, 2018. Accessed: 2020-04-28.

[5] Bhagavan, S., Alsultan, K., and Rao, P. The Case for Designing Data-Intensive Cloud-Based Healthcare Applications. In *SWH@ ISWC* (2018).

[6] Bhagavan, S., Gharibi, M., and Rao, P. FEDSMARTEUM: Secure Federated Matrix Factorization Using Smart Contracts for Multi-cloud Supply Chain. In *IEEE Big Data, in the 1st International Workshop on Serverless Machine Learning for Intelligent and Scalable AI Workflow.* (2021).

[7] Bhagavan, S., Rao, P., and Ngo, T. C3HSB: A Transparent Supply Chain for Multi-cloud and Hybrid Cloud Assets Powered by Blockchain. In *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)* (2021), IEEE, pp. 100–103.

[8] Bhagavan, S., Rao, P., and Ngo, T. C3HSB: A Transparent Supply Chain for Multi-cloud and Hybrid Cloud Assets Powered by Blockchain. In *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)* (2021), IEEE, pp. 100–103.

[9] Bhagavan, S., Rao, P., and Njilla, L. A Primer on Smart Contracts and Blockchains for Smart Cities. *Handbook of Smart Cities* (2020), 1–31.

[10] Bhardwaj, R., and Datta, D. Development of a Recommender System HealthMudra Using Blockchain for Prevention of Diabetes. *Recommender System with Machine Learning and Artificial Intelligence: Practical Tools and Applications in Medical, Agricultural and Other Industries* (2020), 313–327.

[11] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., et al. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM*

workshop on programming languages and analysis for security (2016), pp. 91–96.

[12] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., et al. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security* (New York, NY, 2016), ACM, ACM Workshop, pp. 91–96.

[13] Biswas, K., and Muthukkumarasamy, V. Securing smart cities using blockchain technology. In *2016 IEEE 18th international conference on high performance computing and communications; IEEE 14th international conference on smart city; IEEE 2nd international conference on data science and systems (HPCC/SmartCity/DSS)* (Sydney, Australia, 2016), IEEE, pp. 1392–1393.

[14] Buterin, V., et al. A next-generation smart contract and decentralized application platform. Tech. rep., Ethereum.org, July 2013.

[15] Calvert, D. How to keep supply chains reliable when the world's upended. `https://phys.org/news/2020-05-chains-reliable-world-upended.html?linkId=89703811`. Accessed: 2021-11-11.

[16] Chai, D., Wang, L., Chen, K., and Yang, Q. Secure federated matrix factorization. *IEEE Intelligent Systems* (2020).

[17] Charles McLellan. Multicloud: Everything you need to know about the biggest trend in cloud computing. `https://www.`

zdnet.com/article/multicloud-everything-you-need-\
to-know-about-the-biggest-trend-in-cloud-computing/.
Accessed: 2021-11-11.

[18] Corda. Open Source Blockchain Platform for Business. www.corda.net/. Accessed: 2021-11-11.

[19] Corda. Corda: Open-source blockchain platform for business. https://www.corda.net, 2020. Accessed: 2020-04-01.

[20] Daley, Sam. 19 Blockchain-as-a-service companies making the DLT more accessible. https://builtin.com/blockchain/blockchain-as-a-service-companies, 2020. Accessed: 2020-04-06.

[21] Delmolino, K., Arnett, M., Kosba, A., Miller, A., and Shi, E. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *International conference on financial cryptography and data security* (New York, NY, 2016), Springer, Springer, pp. 79–94.

[22] Disruptor Daily. Ultimate Guide To Blockchain In Supply Chain. www.disruptordaily.com/ultimate-guide-to-blockchain-in-supply-chain/. Accessed: 2021-11-11.

[23] Dou, Q., So, T. Y., Jiang, M., Liu, Q., Vardhanabhuti, V., Kaissis, G., Li, Z., Si, W., Lee, H. H., Yu, K., et al. Federated deep learning for detecting COVID-19 lung abnormalities in CT: a privacy-preserving multinational validation study. *NPJ digital medicine 4*, 1 (2021), 1–11.

[24] Felix Xu. Privacy-preserving computation on blockchains could prevent breaches. https://cointelegraph.com/news/privacy-preserving-computation-on-blockchains-could/-prevent-breaches. Accessed: 2021-11-11.

[25] Fleta. 6 Popular Blockchain Programming Languages Used for Building Smart Contracts — And FLETA will support them all. https://medium.com/fleta-first-chain/6-popular-blockchain-programming-languages-used-for/-building-smart-contracts-and-fleta-will-/7b310f1a9e2, 2019. Accessed: 2020-02-15.

[26] Ganache. Ganache one click blockchain. https://www.trufflesuite.com/ganache/. Accessed: 2021-11-11.

[27] Gentry, C. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing* (2009), pp. 169–178.

[28] Gharibi, Mohamed, B. S., and Rao, P. FederatedTree: A Secure Serverless Algorithm for Federated Learning to Reduce Data Leakage. In *IEEE Big Data, in*

the 1st International Workshop on Serverless Machine Learning for Intelligent and Scalable AI Workflow. (2021).

[29] Group, Netis. The difference between permissionless and permissioned networks. https://medium.com/netis-group-blog/the-difference-between-permissionless-and-permissioned/-networks-5acd05578676, 2020. Accessed: 2020-03-01.

[30] GroupLens. MovieLens datasets collected by GroupLens. https://grouplens.org/datasets/movielens/. Accessed: 2021-11-11.

[31] Hao, M., Li, H., Luo, X., Xu, G., Yang, H., and Liu, S. Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics 16*, 10 (2019), 6532–6542.

[32] Harper, F. M., and Konstan, J. A. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis) 5*, 4 (2015), 1–19.

[33] IBM. Blockchain for Supply Chain. https://www.ibm.com/blockchain/industries/supply-chain. Accessed: 2021-11-11.

[34] IBM. IBM Blockchain Hyperledger. Where blockchain for business comes to life. https://www.ibm.com/blockchain/hyperledger, 2019. Accessed: 2020-04-01.

[35] IBM. IBM Use Case: Vehicle Manufacture. https://www.ibm.com/blockchain/hyperledger, 2019. Accessed: 2019-10-10.

[36] IBM. IBM Blockchain Cloud docs. https://cloud.ibm.com/docs/blockchain-sw?topic=blockchain-sw-get-started-console-ocp, 2020. Accessed: 2020-04-01.

[37] IBM. IBM Blockchain solutions. https://www.ibm.com/blockchain/solution, 2020. Accessed: 2020-04-01.

[38] IBM. IBM Code Pattern. https://developer.ibm.com/patterns/category/blockchain, 2020. Accessed: 2020-04-01.

[39] IPFS. Interplanetary File System (IPFS). https://ipfs.io/. Accessed: 2021-11-11.

[40] Jayasinghe, U., Lee, G. M., MacDermott, Á., and Rhee, W. S. Trustchain: A privacy preserving blockchain with edge computing. *Wireless Communications and Mobile Computing 2019* (2019).

[41] Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer 42*, 8 (2009), 30–37.

[42] Latif, R. M. A., Hussain, K., Jhanjhi, N., Nayyar, A., and Rizwan, O. A remix IDE: smart contract-based framework for the healthcare sector by using Blockchain technology. *Multimedia Tools and Applications* (2020), 1–24.

[43] Libra. The Libra Blockchain. Tech. rep., Libra.org, June 2019.

[44] Matthew Beedham. Here's the Difference between Permissioned and Permissionless Blockchains. https://thenextweb.com/hardfork/2018/11/05/permissioned-permissionless-blockchains/. Accessed: 2021-11-11.

[45] Matthew Tillman. Human Error Takes Toll on Global Supply Chain. https://transportpro.blog/2018/07/24/human-error-takes-toll-on-global-supply-chain/. Accessed: 2021-11-11.

[46] McFarlane, Chrissa. Are Smart Cities The Pathway To Blockchain And Cryptocurrency Adoption https://www.forbes.com/sites/chrissamcfarlane/2019/10/18/are-smart-cities-the-pathway-to-blockchain-and/-cryptocurrency-adoption/#1db185004609, 2019. Accessed: 2020-04-20.

[47] McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics* (2017), PMLR, pp. 1273–1282.

[48] Mingxiao, D., Ma, X., Zhang, Z., and Qijun, C. Consensus Algorithms in Blockchain. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (Banff, Canada, 2017), IEEE, IEEE Explore.

[49] Mitra, Rajarshi. Top Four Smart Contracts Platforms: An In-Depth Analysis. https://blockgeeks.com/guides/smart-contract-platforms-comparison-rsk-vs-ethereum/-vs-eos-vs-cardano, 2020. Accessed: 2020-03-10.

[50] Mondal, A., Rao, P., and Madria, S. K. 17th IEEE International Conference on Mobile Data Management (MDM). In *Mobile Computing, Internet of Things, and Big Data for Urban Informatics* (Porto, Portugal, 2016), Volume 32, Issue 5, IEEE, pp. 8–11.

[51] Mondal, A., Rao, P., and Madria, S. K. Mobile Computing, IoT and Big Data for Urban Informatics: Challenges and Opportunities. In *Handbook of Smart Cities: Software Services and Cyber Infrastructure* (2018), Volume 32, Issue 5, Springer International Publishing, pp. 81–113.

[52] MovieLens. The MovieLens Dataset. https://movielens.org/. Accessed: 2021-11-11.

[53] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. Tech. rep., Bitcoin.org, Dec. 2008.

[54] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[55] Nebulas. Details on the Smart Contract Ranking Algorithm. https://medium.com/nebulasio/

details-on-the-smart-contract-ranking-algorithm-part/-1-723143c689c3, 2018. Accessed: 2020-02-01.

[56] Nguyen, D. C., Ding, M., Pham, Q.-V., Pathirana, P. N., Le, L. B., Seneviratne, A., Li, J., Niyato, D., and Poor, H. V. Federated learning meets blockchain in edge computing: Opportunities and challenges. *IEEE Internet of Things Journal* (2021).

[57] Nick Mudge. Ethereum's Maximum Contract Size Limit is Solved with the Diamond Standard. https://dev.to/mudgen/ethereum-s-maximum-contract-size-limit-is-solved-with/-the-diamond-standard-2189. Accessed: 2021-11-11.

[58] Ober, M. The Decentralized Power of Ethereum and IPFS. https://medium.com/pinata/ethereum-and-ipfs-e816e12a3c59. Accessed: 2021-11-11.

[59] Paillier. A Python 3 library implementing the Paillier Homomorphic Encryption. https://github.com/data61/python-paillier. Accessed: 2021-11-11.

[60] Pasquini, D., Ateniese, G., and Bernaschi, M. Unleashing the Tiger: Inference Attacks on Split Learning. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (2021), pp. 2113–2129.

[61] Rajarshi Mitra. Blockchain And Supply Chain: A Dynamic Duo. www.blockgeeks.com/guides/blockchain-and-supply-chain/. Accessed: 2021-11-11.

[62] Ramanan, P., and Nakayama, K. Baffle: Blockchain based aggregator free federated learning. In *2020 IEEE International Conference on Blockchain (Blockchain)* (2020), IEEE, pp. 72–81.

[63] Rao, P., Bhagavan, S., and Njilla, L. A new tool for detecting tampering of big data programs. In *Cyber Sensing 2020* (2020), vol. 11417, International Society for Optics and Photonics, p. 114170J.

[64] Salimitari, M., and Chatterjee, M. A survey on consensus protocols in blockchain for IOT networks. Tech. rep., Cornell University, Ithaca, NY, Sept. 2018.

[65] Sam Mire. Blockchain In Supply Chain Management: 13 Possible Use Cases. www.disruptordaily.com/blockchain-use-cases-supply-chain-management. Accessed: 2021-11-11.

[66] Sharma, Toshendra Kumar. Permissioned and Permissionless Blockchains: A Comprehensive Guide. Tech. rep., Blockchain Council, Walnut, CA, Nov. 2019.

[67] SmartDubai. Smart Dubai announces achievements of Dubai Blockchain Strategy. https://www.smartdubai.ae/newsroom/

news/smart-dubai-announces-achievements-of-dubai/-blockchain-strategy-2020, 2020. Accessed: 2020-02-20.

[68] Srini Bhagavan, K. A., and Rao, P. The Case for Designing Data-Intensive Cloud-Based Healthcare Applications. In *SWH ISWC* (2018).

[69] Srini Bhagavan, P. R., and Njilla, L. A Primer on Smart Contracts and Blockchains for Smart Cities. *Handbook of Smart Cities* (2020), 1–31.

[70] TOSHENDRA KUMAR SHARMA. Permissioned and Permissionless Blockchains: A Comprehensive Guide. www.blockchain-council.org/blockchain/permissioned-and-permissionless-blockchains-/a-comprehensive-guide/. Accessed: 2021-11-11.

[71] Truffle. Truffle Suite. https://www.trufflesuite.com/. Accessed: 2021-11-11.

[72] Utsav Jaiswal. A Guide to BaaS - for People Who Do Not Understand SaaS. https://hackernoon.com/a-guide-to-baas-for-people-who-do-not-understand-/saas-bf4990d0377d. Accessed: 2021-11-11.

[73] Utsav, Jaiswal. A Guide to BaaS for people who do not understand SaaS. https://hackernoon.com/

a-guide-to-baas-for-people-who-do-not-understand-/saas-bf4990d0377d, 2018. Accessed: 2020-04-28.

[74] Yang, Q., Liu, Y., Chen, T., and Tong, Y. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST) 10*, 2 (2019), 1–19.

[75] Ycharts. Ethereum Average Transaction Fee. https://ycharts.com/indicators/ethereum_average_transaction_fee_eth. Accessed: 2021-11-11.

[76] Yeh, T.-Y., and Kashef, R. Trust-Based Collaborative Filtering Recommendation Systems on the Blockchain. *Advances in Internet of Things 10*, 4 (2020), 37–56.

[77] Zhu, L., and Han, S. Deep leakage from gradients. In *Federated learning*. Springer, 2020, pp. 17–31.

VITA

Srini Bhagavan joined UMKC in the fall of 2017 to pursue an interdesciplinary Ph.D. in Computer Science and Medical and Health BioInformatics . His research interests include Data Science, Blockchain, Federated Learning, Supply chain, and Bioinformatics. Srini worked under the supervision of Dr. Praveen Rao and he is currently an STSM (Sr Technical Staff Member), Multi-cloud Infrastructure Engineering Lead at IBM's Data and AI division and has more than 20 years experience in Software Engineering and Design.

Srini published the following papers:

- Bhagavan, S., Gharibi, M., Rao, P., FedSmarteum: Secure Federated Matrix Factorization Using Smart Contracts for Multi-cloud Supply Chain, IEEE Big Data, in the 1st International Workshop on Serverless Machine Learning for Intelligent and Scalable AI Workflow, 2021 [6]

- Gharibi, M., Bhagavan, S., Rao, P., FederatedTree: A Secure Serverless Algorithm for Federated Learning to Reduce Data Leakage, IEEE Big Data, in the 1st International Workshop on Serverless Machine Learning for Intelligent and Scalable AI Workflow, 2021 [28]

- Bhagavan, S., Rao, P., and T. Ngo., C3HSB: A Transparent Supply Chain for Multicloud and Hybrid Cloud Assets Powered by Blockchain, IEEE 37th International Conference on Data Engineering (ICDEW), 2021 [8]

- Rao, P., Bhagavan, S., Laurent N., A New Tool for Detecting Tampering of Big Data Programs,International Society for Optics and Photonics, 2020 [63]

- Bhagavan, S., Rao, P., Njilla, L., A Primer on Smart Contracts and Blockchains for Smart Cities, Handbook of Smart Cities, 1-31, 2020 [9]

- Bhagavan, S., Alsultan, K., Rao, P., The Case for Designing Data-Intensive Cloud-Based Healthcare Applications., SWH@ ISWC, 2020 [5]