Analysis and Design of Open Decentralized Serverless Computing Platforms

by

Sara Ghaemi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

The distributed ledger technology (DLT) has been envisioned to be a disruptive technology with applications in various industries. For the DLTs to be effectively used in practice, it is crucial to assess their performance in different use cases and scenarios. In this thesis, we first conduct a systematic survey on the performance evaluation of distributed ledgers. Then we identify and present the performance evaluation techniques, the most important performance metrics, and the bottlenecks of various well-known DLTs. Finally, we present a list of possible directions for future research.

Our survey identifies the lack of a detailed performance evaluation for DAG-based distributed ledgers presented so far. Compared to blockchains, DAG-based DLTs offer better performance and scalability by design. In the second part of this thesis, we focus on the performance evaluation of IOTA, which is the prominent implementation of DAG-based distributed ledgers. In this work, we investigate the impact of different design parameters on the performance of an IOTA network. We then propose a layered model to help the users determine the optimal waiting time to resend an unconfirmed transaction. The results can be used by both system designers and users to support their decision making.

In the third part of this thesis, we use the findings from the previous two parts to design an open blockchain-based serverless computing platform called ChainFaaS that runs on personal computers. To better understand the capacity of personal computers, we conducted a survey that aims to find their unused computational power. The results indicate that the typical CPU utilization of a personal computer is only 24.5% and, on average, a personal computer is only used 4.5 hours per day. This shows a significant

computational potential that can be used towards distributed computing. In this work, we introduce ChainFaaS with the motivation to use the computational capacity of personal computers as well as to improve developers' experience of internet-based computing services by reducing their costs, enabling transparency, and providing reliability. We propose the design of ChainFaaS and then implement and evaluate a prototype of this platform to show the feasibility of this paradigm.

The current implementation of ChainFaaS provides payment using a monetary smart contract on the blockchain network. For this platform to be used in practice, ChainFaaS needs to support payment in established cryptocurrencies. As a result, the blockchain network in ChainFaaS needs to interoperate with other distributed ledger technologies to enable such payments. In the last part of this thesis, we investigate a possible solution to enable interoperability in blockchains. We propose a blockchain interoperability solution for permissioned blockchains based on the publish/subscribe architecture. We then implement a prototype of the proposed solution and evaluate its performance. The result of this research not only enables ChainFaaS to support payments in established cryptocurrencies, but it also allows any other blockchain-based application to interoperate with other blockchain networks and use the data and information available on them.

Preface

The research of this thesis has been conducted in the Performant and Available Computing Systems (PACS) Lab, led by Dr. Hamzeh Khazaei. Chapter 2 of this thesis has been published as C. Fan, S. Ghaemi, H. Khazaei and P. Musilek, "Performance Evaluation of Blockchain Systems: A Systematic Survey," in IEEE Access, vol. 8, pp. 126927-126950, 2020, doi: 10.1109/ACCESS.2020.3006078. Caixiang Fan and I shared the responsibility of gathering and selecting related articles and forming summaries of the articles for the survey paper.

Chapter 3 of this thesis has been submitted as C. Fan, S. Ghaemi, H. Khazaei, Y. Chen and P. Musilek, "Performance Analysis of the IOTA DAG-based Distributed Ledger," in ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS). I was responsible for extending the simulation tool for DAG-based distributed ledgers and conducting the simulations. I also contributed to the formation of the analytical layered model and to the manuscript edits. Caixiang Fan was responsible for the analytical layered model and for the manuscript composition.

Chapter 4 of this thesis has been published as S. Ghaemi, H. Khazaei and P. Musilek, "ChainFaaS: An Open Blockchain-Based Serverless Platform," in IEEE Access, vol. 8, pp. 131760-131778, 2020, doi: 10.1109/ACCESS.2020.3010119. I was responsible for the design, implementation, and evaluation of the system as well as the manuscript composition.

Chapter 5 of this thesis is part of research conducted through an internship at the Linux Foundation. This project was a collaboration with Dr. Rui S. Cruz and Rafael Belchior at Instituto Superior Técnico, Universidade de Lisboa, and Sara Rouhani at

the University of Saskatchewan. I was responsible for the design, implementation, and evaluation of the system as well as the majority of the manuscript composition. Sara Rouhani and Rafael Belchior assisted with the concept formation, system evaluation, and manuscript composition. Dr. Rui S. Cruz, Dr. Hamzeh Khazaei, and Dr. Petr Musilek were the supervisory authors and were involved with concept formation and manuscript edits.

Acknowledgements

I would like to thank everyone who has supported and guided me throughout my research. First, I highly appreciate the support of the Future Energy Systems Research Initiative under the Canada First Research Excellence Fund (CFREF) at the University of Alberta. I would like to express my deep and sincere gratitude to my supervisors, Dr. Hamzeh Khazaei and Dr. Petr Musilek, for providing support and invaluable guidance throughout my research.

My sincere thanks also go to the team that I worked with during my internship at the Linux Foundation, Sara Rouhani, Rafael Belchior, and Dr. Rui S. Cruz, for providing me with the opportunity to further my research and for their professional guidance.

Thanks to all my friends and colleagues in the PACS lab who helped me throughout my studies. I would particularly like to thank Caixiang Fan, who was like a mentor to me, for his assistance with this research.

In addition, I am extremely grateful to my husband, Nima, for his continuing support and his encouragements when the times got rough. I am also very much thankful to my parents, Maryam and Zabihollah, and my sister, Mahsa, for their spiritual support and guidance throughout my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

1.1 Introduction

Since the introduction of Bitcoin, distributed ledgers have attracted massive attention from both academia and industry. Distributed ledger technologies (DLTs) are mainly known for their application in cryptocurrency. However, many other interesting applications for this technology are also being studied by different research groups. DLTs can be thought of as a distributed system and database that is transparent and immutable with enhanced security and fault tolerance compared to centralized solutions. The participating parties keep a record of the ledger, and they are guaranteed to reach a consensus on the order and validity of the transactions. This technology enables parties who do not necessarily trust each other to reach an agreement without the need for a trusted third party middleman.

Despite all of the mentioned advantages, the decentralized nature of distributed ledgers significantly limits their performance. As a result, many researchers have shifted their focus to the performance evaluation and performance enhancement of DLTs. In the first part of this thesis, we present a comprehensive and systematic survey on the performance evaluation of distributed ledgers. In this work, we present the research done on the performance evaluation of the mainstream DLTs and compare the advantages and disadvantages of these solutions. We identify the techniques, evaluation metrics, and mostly used benchmark workloads for evaluating the performance of

DLTs. We also summarize the identified bottlenecks of various well-known blockchain systems. We conclude this work by gathering the challenges and opportunities for further research on the performance of DLTs. The results of this research are later used in the rest of this thesis.

Our survey identified a research gap in the literature which was a detailed performance evaluation of distributed ledgers that leverage directed acyclic graphs (DAGs). This type of DLT has a different data structure compared to blockchain, which is the most popular form of distributed ledger. In blockchain, transactions are bundled into blocks of data that are linearly chained, whereas in DAG, transactions are linked to each other in a graph without the need to be wrapped in a block. These fundamental differences enable DAG-based ledgers to offer lower latency, higher throughput, and better scalability compared to most traditional blockchains. As one of the first and most prominent DAG-based ledgers, IOTA [1] has drawn a lot of attention. In the second part of this thesis, we focus on the performance analysis of IOTA as a representative of the DAG-based distributed ledgers. We aim to find the most influential design factors in the throughput of an IOTA network from the system designer's point of view as well as the optimal waiting time for confirmation before resubmitting the transaction from the user's point of view. To achieve this goal, we extend the DAGsim [2] simulator to support the currently running consensus protocol on the IOTA network, and we study the network under different circumstances using the extended simulator. We also conducted experiments on an IOTA network to verify our results from the simulation. Finally, we proposed a layered analytical model that users can leverage to derive the optimal waiting time for resubmitting their transactions.

In the third part of this thesis, we use the findings from the previous two parts to design an open blockchain-based serverless platform called ChainFaaS. The idea in this work is to use the unused computational power of personal computers towards an open serverless computing platform. To better understand the capacity of personal computers, we conducted a survey that aims to find their unused computational power.

The results indicate that personal computers run on an average CPU utilization of 24.5%, and on average, a personal computer is only used 2.5 hours per day. In this work, we leverage blockchain to run a serverless computing platform on personal computers that is reliable, transparent, and lowers the developers' costs. Also, personal computer users can rent out their unused computational capacity to have some income. We propose the design of ChainFaaS, and we implement a prototype of the platform as a proof-of-concept to show the feasibility of our design. Finally, we evaluate the implementation to analyze its non-functional properties.

Similar to our proposed serverless application, many other applications for blockchain and distributed ledger technologies have been proposed by academia and industry. As a result, there are many permissioned and permissionless heterogeneous blockchain networks running all around the world. While this surge is promising for the blockchain technology, it has led to data, asset, and network silos. This limits the blockchain technology to reach its full potential as the blockchain networks cannot interoperate. In the last part of this thesis, we propose a blockchain interoperability solution for permissioned blockchains that works based on the publish/subscribe architecture. We then implement a prototype of our proposed solution. Finally, we evaluate the implementation to analyze its performance. The results of this research can be used to allow ChainFaaS to interoperate with other blockchains. ChainFaaS can allow payments on other cryptocurrencies or even use the assets and information available on other blockchain networks.

1.2 Distributed Ledger Technologies

Any ledger that is stored in a distributed fashion and shared among a set of nodes or participants can be referred to as a distributed ledger. For new information to be added to this ledger, all participating nodes must reach a consensus on whether the information is legitimate or not. The algorithm which determines how this decision is made, called *consensus algorithm*, is an important part of the distributed ledger

technology (DLT). Blockchain is a type of distributed ledger technology that has recently become extremely popular. It is mainly known for its use in cryptocurrencies such as Bitcoin [3], Ethereum [4], and Ripple [5]. However, blockchains, and distributed ledger technologies in general, are more than just cryptocurrencies. In a recent study [6], CB Insights has reported 55 industries that can be transformed by the distributed ledger technology. Some examples are biomedical and health care [7–9], energy [10], Internet of Things (IoT) [11, 12], supply chain [13, 14], and cloud computing [15].

1.2.1 Categorization of DLTs

A possible taxonomy of distributed ledger technologies is shown in Figure 1.1. Distributed ledgers can be categorized as permissioned and permissionless, based on whether the identities of the participants are known. If the identity of everyone is known, the ledger is permissioned, and if everyone participates in the network anonymously, the ledger is permissionless. Also, based on who can participate in the network, distributed ledgers can be categorized as private and public. In private ledgers, only those who are approved can participate in the network. On the other hand, in public ledgers, anyone can participate without the need to be approved [16]. For example, Bitcoin runs on a public permissionless blockchain. Therefore, based on the permissions and accessibility of the ledger, DLTs can be divided into four groups, as shown in Figure 1.1. Public permissionless ledgers, such as Bitcoin, Ethereum, and Litecoin, have no restriction on the participating parties. In public permissioned ledgers, the identity of participants should be known but anyone can read and validate the ledger. EOS, Ripple, and Sovrin are examples of this type. In private permissionless blockchains, the identities of the participants are not known but only pre-approved nodes validate the data. Examples of this category include LTO, Holochain, and Monet. Finally, in private permissioned ledgers, such as Hyperledger and Corda, access is restricted to pre-approved participants and the identities of the participants are known.

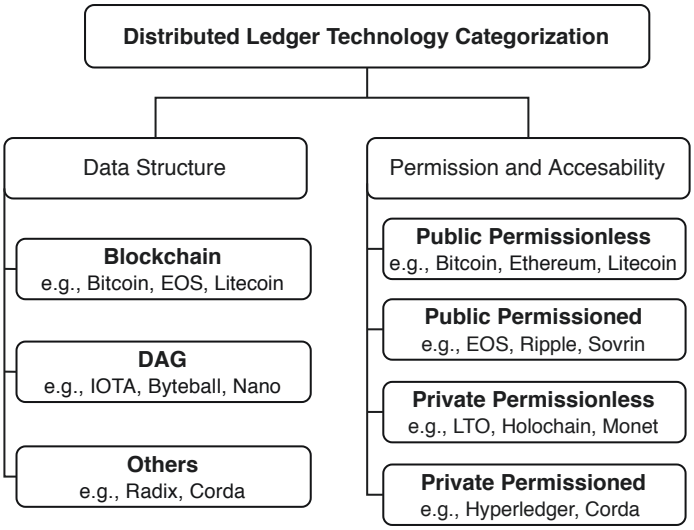


Figure 1.1: Categories of distributed ledger technologies.

In terms of the data structure, there are two main types of distributed ledger technologies: blockchain and directed acyclic graph (DAG). In a blockchain, transactions are bundled in blocks of data that are linearly chained to each other, just like a linked list. The blocks are connected in a chronological order, which is unalterable. Examples of this category are Bitcoin, Ethereum, EOS, and Litecoin. On the other hand, in DAG, transactions are linked to each other in a graph. Usually, there is no block in DAG, and transactions are the components of the DAG [17]. This category includes DLTs such as IOTA, Byteball, and Nano. In addition, there are distributed ledgers that have their unique data structure and do not fall into either of these categories, such as Radix and Corda.

Blockchain is a distributed database that is immutable, transparent, verifiable, and managed by a set of participants who do not necessarily trust each other. At any point in time, each transaction can be executed and verified by the participants based on a consensus algorithm. In public permissionless blockchains, this consensus algorithm is usually a computationally expensive task that takes a lot of time and energy. This type of consensus algorithm is called Proof-of-Work (PoW), which limits the speed of the transactions. Although other solutions for public blockchains have

been introduced, most of them are slow in terms of the number of transactions that they can verify per second. On the other hand, in permissioned blockchains, since the identities are known, the consensus algorithm can be less costly, which makes them dramatically faster. Usually, these blockchains are used between participants who have the same goal but do not necessarily trust each other. In a blockchain, each block consists of many transactions bundled together and a header, which consists of a hash. For each block, the hash is a unique value that is created based on the data inside that block and the previous block's hash. The hash is the element that makes blockchain an immutable chain of data. No one can change the transactions in the blocks that have already been verified because of this hash.

In blockchain, there is a concept called smart contract, which allows customization of the contract between the entities in the blockchain. Szabo first introduced the term smart contract in 1994 as a computerized transaction protocol that executes the terms of a contract [18]. Within blockchain, smart contracts are the pieces of code that contain the contract between entities and are stored on the blockchain. Participants can trigger these contracts to use their functionalities [19]. Using smart contracts on blockchains, trusted distributed applications can be created.

DAG-based distributed ledger, also called DAG distributed ledger or DAG ledger, is a promising DLT that stores transaction data as vertices of a directed acyclic graph. Different new transactions can be appended to different vertices in a DAG at the same time. Compared to blockchain, which bundles transactions in blocks and stores blocks one by one to a chain, DAG ledger naturally obtains better concurrency. Therefore, it has many advantages over blockchain on transaction throughput, network scalability and resource efficiency.

According to the graph vertex granularity, DAG ledger can be further divided into two categories: block-based (blockDAG) and transaction-based (TDAG) [20]. The former first wraps transactions into blocks, and then appends the blocks onto a DAG. Some examples are CDAG [21], PHANTOM [22] and SPECTRE [23]. The

latter attaches transactions to a DAG immediately and directly, without waiting for block composition. Three notable examples are IOTA [1], Byteball [24] and Nano [25]. Since blockDAG is still in the conceptual stage, we focus our research on TDAG. For simplicity, DAG refers to TDAG throughout this thesis without further specification.

In the design of DAG ledgers, each end user issuing a transaction also needs to validate the previous transactions. All participants act as validators and contribute to maintain the network. In other words, no miners are needed in DAG distributed ledger systems. For example, IOTA requires the network participant or node to approve two previous transactions in order to issue a new transaction, while Byteball encourages referencing all unapproved transactions. In addition, multiple transactions can be added to a DAG simultaneously and concurrently to increase system throughput and scalability.

1.2.2 DLT Abstraction Layers

Dinh *et al.* [26] introduced a blockchain design comprised of four identified abstraction layers, namely application, execution engine, data model and consensus. In the Oracle blockchain guidance book [27], the authors defined five layers to display the general architecture of blockchain, including the application and presentation layer, consensus layer, network layer, data layer and hardware/infrastructure layer. To better describe the architecture of DLT for the purpose of performance evaluation, we formulate an abstraction layer architecture following mainly Dinh's model [26], but extend it to five layers shown in Fig. 1.2.

Application Layer

As the top presentation of DLT's technology stack, this layer contains the applications that are mainly used by the end users. Up to date, the most popular one is still cryptocurrency. As the first published digital currency, Bitcoin has controlled most of the marketplace and developed many variants. Ethereum has its own currency called

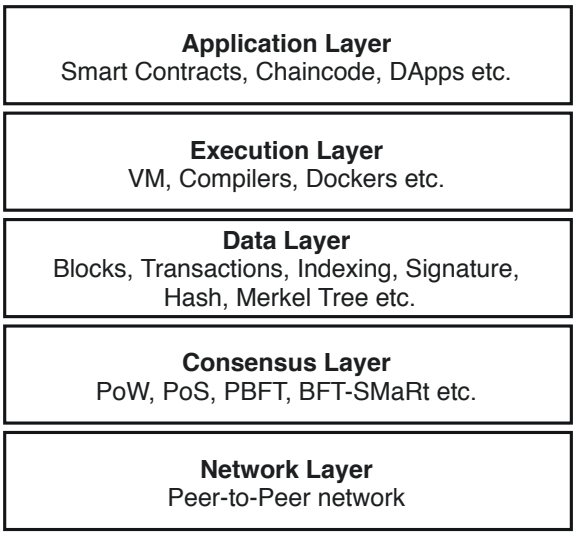


Figure 1.2: Abstraction layer model for DLT.

Ether. IOTA also has its currency with the same name as the network, *IOTA* [1]. Other examples include the wallet to manage cryptocurrency, smart contracts, and all kinds of decentralized applications (*DApps*). In a DLT system, a smart contract is a piece of code designed to digitally facilitate, verify, or enforce the execution of a contract. For Ethereum, the smart contract is running on a dedicated virtual machine (called EVM); and most contracts on the system are related to cryptocurrency. While HLF's smart contract is running in a container such as Docker. One of the best-known DApps is the *decentralized autonomous organization (DAO)* in Ethereum, which creates communities to raise funding for exchange and investment.

Because this layer is in charge of presenting the final results executed from the distributed ledger system, it is supported and impacted by all the lower layers. Therefore, the performance evaluation results of the application layer reflect the overall performance of tested DLTs.

Execution Layer

The execution layer is in charge of executing contract or low-level machine code (bytecode) in a runtime environment installed on DLT network nodes. Ethereum has its own machine language and a virtual machine (EVM) developed to run the smart

contracts code. Unlike Java virtual machine (JVM), the EVM reads and executes a low-level representation of smart contracts called the Ethereum *bytecode*. The smart contracts are programmed in a dedicated high-level language named *Solidity*, which is first compiled to bytecode by Solidity compiler. The Ethereum bytecode is an assembly language made up of multiple opcodes. Each *opcode* performs certain action on the Ethereum blockchain. In contrast, HLF does not take the semantics of language into consideration. It runs the compiled machine codes (from chaincode) inside Docker images. In addition, HLF's smart contract (chaincode) supports multiple general high-level programming languages such as Go, node.js, and Java rather than a dedicated language like Solidity of Ethereum. IOTA does not support smart contracts up to date. It adopts Java as the main development language and runs its reference implementation (IRI) in JVM. IOTA also has a version running in Docker image.

The runtime environment used to execute contracts or transactions needs to be efficient. And the execution result should be deterministic to avoid the uncertainty and inconsistency of transactions on all nodes. Any transaction abortion caused by inconsistent execution would result in computation resource waste and further decrease the performance. Additionally, the resource configurations (e.g., CPU and RAM) may impact the execution performance.

Data Layer

In the data layer, a wide range of data-related topics are defined, including transaction models, data structure, Merkel trees, hash function, encryption algorithms, etc. There are two popular transaction models: *unspent transaction output (UTXO)* and *account*. For UTXO, one owner completes value transfers by signing a transaction transferring the ownership of the UTXO to the receiver's public key. It involves an extra step of searching for ownership of the transaction from the sender's side. The account-based model is more efficient as it atomically updates two accounts in one transaction. A smart contract (chaincode for HLF) is a special type of account.

For blockchain, blocks containing transactions and contract execution states are chained together in a linked list by putting the hashed result of its previous block's content into the header of the current block. Ethereum and HLF employ a two-layer data structure to organize the block's content. All states are stored in a key-value database on a disk and indexed in a hash tree. The hash tree root is contained in the block's header. With a similar design, different DLTs have their own storage solutions for each level. For example, Ethereum uses LevelDB, and HLF uses CouchDB to store the states; Ethereum and Parity employ Patricia-Merkle (key-value store) tree, while HLF implements Bucket-Merkle tree to store the indices [26]. For IOTA, transactions are directly appended to the DAG structure called *tangle* in a hashed manner. The IRI uses RocksDB database to store the snapshot, a pruned ledger to prevent the tangle from expanding too large in size.

Besides the factors mentioned above, there are other design parameters in the data layer, such as hash functions (e.g., SHA 256 v.s. SHA 128), encryption algorithms (RSA v.s. ECC), and block size. All these factors may influence the performance of a blockchain system.

Consensus Layer

The consensus protocol is the core of a DLT system. It sets the rules and forces all nodes to follow them to reach an agreement (e.g., transaction confirmation) on blockchain content. Generally, there are two basic types of consensus mechanisms, which are proof-based and vote-based consensuses [28]. The most popular proof-based consensus is proof-of-work (PoW), which has been employed by many blockchain systems. PoW is a very computation intensive consensus. It requires the nodes to solve a difficult puzzle to compete for the right of recording the ledger. Only the first node (called *winner*) solving the puzzle can append the block to the ledger and gets incentives accordingly. Since PoW provides high security, integrity and decentralization in an untrusted environment, it is very popular in public blockchains [29]. However,

the classic PoW protocol has a poor efficiency on processing transactions. To tackle this problem, many variations have been proposed to keep the same safety while achieving a better performance [30]. Examples include greedy heaviest observed subtree (GHOST), proof of authority (PoA), proof of stake (PoS) and proof of elapsed time (PoET).

The vote-based consensuses are communication intensive. Different from PoW, vote-based solutions always give a deterministic execution result and usually achieve a relatively high performance. They rely on frequent message transitions among different roles in a network to ensure that all nodes reach an agreement on the block order. It is very popular in permissioned blockchains. Raft and Byzantine fault tolerance (BFT)-based, (e.g., PBFT and BFT-SMaRt) algorithms are two representatives of this consensus type. Raft has only crash fault tolerance (CFT), while PBFT and BFT-SMaRt can address Byzantine fault.

There are also some hybrid DLTs [31] that combine different types of consensuses. For example, Tendermint combines PoS and PBFT; EOS takes a hybrid design combining PBFT and DPoS. Both target on improved performance and enhanced security. Interested readers may refer to the published surveys of blockchain consensus. Because of the deterministic properties, BFT-based consensus algorithms have a much lower transaction delay than PoW. But the expensive communication cost makes it difficult to scale, especially in a large network. Therefore, consensus design, evaluation and optimization in DLTs still remain an active research topic.

Network Layer

A peer-to-peer (P2P) network is the foundation of a DLT system. It takes care of peer discovery, transactions, and block propagation. In a public blockchain such as Bitcoin, this network is very large, with thousands of nodes working together to reach consensus. For private blockchain systems, the scale varies from several entities to over a hundred. Either way, a basic requirement for the P2P network is to provide

speed and stablility. When a new participant wants to join, this layer ensures that nodes can discover each other. Then, all connected nodes communicate, propagate and synchronize with each other to maintain the current state of the blockchain network. Specifically, transaction broadcast, validation and transaction commit are all completed via this layer, as well as the world state propagation. In the P2P network, there are two basic types of nodes: full nodes and light nodes. Full nodes take care of mining, transaction validation and execution of consensus rules, while light nodes only keep the header of the blockchain (keys) and act as clients to issue transactions.

Therefore, the network layer is critical, especially for communication-intensive DLTs. Peer discovery and ledger synchronization among neighbours directly rely on the network, so that the speed determines the efficiency. And some detailed metrics, such as the number of transactions per network data are also related to this layer. Moreover, the package loss rate and network delay may have an impact on the performance of DLT.

1.3 Serverless Computing

To understand serverless computing, we should first define cloud computing. Cloud computing is defined as a model in which a shared pool of configurable computing resources can be accessed conveniently and on-demand through the internet. These resources can be networks, servers, storage, applications, or services that are provisioned and released with minimal effort and service provider interaction [32].

Based on the developer's control level over the cloud infrastructure, a few service models are available for cloud computing. In the infrastructure-as-a-service (IaaS) model, the developer has control over both the application and the cloud infrastructure. The developer should manage the virtual machine provided by the cloud provider and the deployment of the application on top of the underlying virtual machine. In platform-as-a-service (PaaS), the developer manages the application code, and the provider maintains the infrastructure. In software-as-a-service, full applications are

delivered over the internet to end-users via providers. Both the infrastructure and the application are managed by the provider [33].

Serverless computing (also known as function-as-a-service or FaaS for short) is the latest paradigm in cloud computing in which the developer deploys functions to the cloud and delegates the operational and management tasks of servers to the provider [34]. The developer can focus on designing and implementing the application instead of spending time on the management, operation, and maintenance of the infrastructure. This characteristic makes serverless similar to PaaS solutions. However, in serverless, costs are truly event-based, unlike in PaaS. This means that the developers only pay for the execution time of their program. In other cloud computing solutions, the developers are billed for the resource allocated to their tasks, even if they are not used.

Another feature of serverless is autoscaling. When the developer deploys a function to serverless computing, they do not need to worry about how their code should scale. No matter how many times concurrent events to the function are triggered, the serverless provider will serve them by running new instances.

Moreover, in serverless computing, everything is based on containers or similar concepts, i.e., lightweight, isolated environments. Since containers are usually small in size and fast to start, every time an event is triggered in serverless computing, the cloud provider can spin up a new container on a virtual machine. In a serverless platform, it can take a few hundred milliseconds to a few seconds to serve a request. For some use cases, such as consumer-facing applications, this time is not acceptable. For instance, imagine a website that wants to use serverless to serve requests. For each request, the users should wait at least a few hundred milliseconds for the container to spin up and then a few more for the function to run. Based on these characteristics, serverless computing cannot currently be used for all types of workloads. Serverless providers advise developers not to submit functions that are time-sensitive since they cannot guarantee a fast response.

The above-mentioned features make serverless computing a great solution for tasks that are not time-critical. Currently, cloud giants are the main providers of serverless computing. Some examples of public serverless platforms are AWS Lambda [35], Azure Functions [36], Google Cloud Functions [37], and IBM Cloud Functions [38].

1.4 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 summarizes our systematic survey on the performance evaluation of blockchain systems. Chapter 3 contains our study on the performance of IOTA DAG-based distributed ledger. Chapter 4 presents our work on an open blockchain-based serverless computing platform called ChainFaaS. Chapter 5 outlines our work on blockchain interoperability for permissioned blockchains. Finally, chapter 6 concludes this thesis and proposes some potential future work in this area.

Chapter 2

A Systematic Survey on Performance Evaluation of Blockchain Systems

Blockchain has been envisioned to be a disruptive technology with potential for applications in various industries. As more and more different blockchain platforms have emerged, it is essential to assess their performance in different use cases and scenarios. In this work, we conduct a systematic survey on the blockchain performance evaluation by categorizing all reviewed solutions into two general categories, namely, empirical analysis and analytical modelling. In the empirical analysis, we comparatively review the current empirical blockchain evaluation methodologies, including benchmarking, monitoring, experimental analysis and simulation. In analytical modelling, we investigate the stochastic models applied to performance evaluation of mainstream blockchain consensus algorithms. Through contrasting, comparison and grouping different methods together, we extract important criteria that can be used for selecting the most suitable evaluation technique for optimizing the performance of blockchain systems based on their identified bottlenecks. Finally, we conclude the survey by presenting a list of possible directions for future research.

2.1 Introduction

Since its first introduction in Bitcoin by Nakamoto [39] in 2008, blockchain has been recognized as a disruptive technology in various industries beyond cryptocurrency, including finance [40, 41], Internet of Things (IoT) [42, 43], health care [44, 45], energy [46–48] and logistics [49, 50]. In 2019, *CB Insights* identified 55 industries that can be transformed by this technology [6]. Compared to conventional, centralized solutions, blockchain has some significant advantages such as immutability, enhanced security, fault tolerance and transparency. However, the decentralized nature of blockchain dramatically limits its performance (e.g., throughput and latency). For example, Bitcoin can only achieve a low throughput of 7 transactions per second (TPS), and it takes around 10 minutes for a transaction to get confirmed [51]. In contrast, current centralized payment systems such as VisaNet and MasterCard can reach thousands of TPS and almost real-time payments. By taking a similar consensus algorithm, proof-of-work (PoW), other blockchain platforms such as Ethereum [4] and Litecoin [52] also inherit the performance flaws of Bitcoin. Without doubt, the performance issue has become the major constraint of blockchain's applications in production. This is especially true for systems demanding high performance such as the online transaction processing (OLTP) and real-time payment systems.

To overcome this problem, many blockchains put efforts on improving their performance, e.g., by modifying the system structure and designing new consensus algorithms. These solutions include, but are not limited to, off-chain [53–56], side-chain [57–60], concurrent execution (smart contract) [61–63] sharding [64–68], and directed acyclic graph (DAG) [1, 22–25, 69–71].

Existing and new solutions should be comparatively evaluated in a meaningful manner to show their efficiency and effectiveness. For example, different versions of Hyperledger Fabric (HLF), e.g., HLF v0.6 and HLF v1.0, should be compared in the same evaluation framework to demonstrate the performance advantages/disadvan-

tages of the new release. In addition, through performance evaluation and analysis, bottlenecks can be identified and used to inspire further optimization ideas. Therefore, performance evaluation plays an important role in the area of blockchain research.

To this end, it would be useful to summarize, classify and survey the existing efforts on blockchain performance evaluation and to identify future directions in this area. However, most existing related surveys only focus on improvement (scalability) solutions or a specific evaluation topic of blockchain performance. A representative list of existing surveys, shown in Table 2.1, clearly identifies the need for a systematic survey on blockchain performance evaluation.

Table 2.1: Research scope of existing blockchain performance related surveys

Year	**Survey**	**Research scope**
2018	Kim *et al.* [72]	scalability solutions
2019	Rouhani and Deters [73]	security, performance, and applications of smart contract
2019	Zheng *et al.* [74]	challenges of performance and security
2019	Wang *et al.* [75]	benchmarking tools and performance optimization methods
2020	Zhou *et al.* [76]	scaling solutions to blockchain
2020	Yu *et al.* [77]	sharding for blockchain scalability

In this contribution, we present a comprehensive, systematic survey on blockchain performance evaluation. The survey covers existing studies on evaluating the performance of various mainstream blockchains, and compares their advantages and disadvantages. It addresses the following research questions:

RQ1. What are the current mainstream techniques, main evaluation metrics and benchmark workloads for blockchain performance evaluation?

RQ2. How to comparatively evaluate the performance of two blockchain systems with different consensuses?

RQ3. What are the significant bottlenecks identified in various blockchain systems?

RQ4. What are the main challenges and opportunities in blockchain performance evaluation?

To answer these questions, the authors have searched and reviewed the latest papers published since 2015. The papers have been retrieved from major scientific databases, including *ACMDL, IEEEXplore, Elsevier, MPDI* and *SpringerLink*. In addition, closely related papers cited by the selected communications have also been taken into consideration. Note that this survey focuses only on blockchain performance evaluation, and solutions for blockchain performance or scalability improvement are not discussed. Interested readers may refer to the published surveys of performance improvement solutions for blockchain [72–77] listed in Table 2.1.

To the best knowledge of the authors, this is the first survey that systematically reviews the state-of-the-art on the blockchain performance evaluation from several different perspectives. The reviewed evaluation approaches can be classified into two high-level groups: empirical evaluation and analytical modelling, as shown in Figure 2.1.

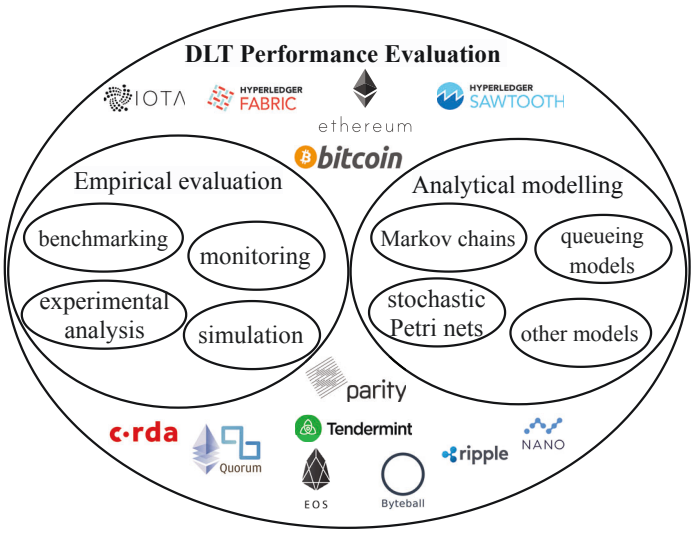


Figure 2.1: A landscape of DLT performance evaluation approaches and evaluated ledgers.

Empirical evaluation includes benchmarking, monitoring, experimental analysis and simulation. Analytical modelling mainly covers three types of stochastic models: Markov chains, queueing models and stochastic Petri nets (SPNs). Through this classification, we aim to depict a big picture of the performance evaluation landscape, identify current challenges in this area, and provide suggestions for future research. The contributions of this survey can be summarized as follows:

- It provides a systematic survey on the blockchain performance evaluation, covering all existing evaluation (empirical and analytical) approaches for evaluating the mainstream blockchain systems.
- It introduces existing popular models for analytical performance evaluation of prominent blockchain platforms, categorizes them and performs a comparative analysis of their advantages and disadvantages.
- It identifies the current challenges in this area, and subsequently provides suggestions for future research.

The background information needed for this work can be found in Section 1.2. The remainder of this chapter is organized as follows. Section 2.2 introduces the blockchain performance evaluation solutions from the perspective of empirical analysis, including benchmarking, monitoring, experimental analysis and simulation. Section 2.3 focuses on the technical and mathematical introduction of existing commonly used performance modelling solutions including Markov chains, queueing models and stochastic Petri nets. The following Section 2.4 summarizes the major findings and points out potential opportunities in this area for future research according to the identified open issues. Finally, the survey is concluded in Section 2.5.

2.2 Empirical Analysis in Blockchain Performance Evaluation

In this section, we investigate existing approaches to blockchain performance evaluation from the perspective of empirical analysis. Specifically, different solutions, including benchmarking, monitoring measurements, self-designed experiments and simulation, are reviewed and compared. In practice, these approaches are usually used together to provide more evidence for blockchain performance evaluation.

2.2.1 Blockchain Benchmarking Tools

The performance benchmarking has been well studied and documented for the cloud (e.g., Hadoop, Mapreduce and Spark) and database (e.g., relational and NoSQL) systems. Some proposed benchmark frameworks such as TPC-C [78], YCSB [79] and SmallBank [80] are well-established and have essentially formed the industrial standards. For example, YCSB is widely used for benchmarking NoSQL databases such as Cassandra [81], MongoDB [82] and HBase [83]; and SmallBank is a popular benchmark for OLTP workload.

However, these frameworks cannot be directly applied to benchmark distributed ledger systems due to the diversity of consensus mechanisms and APIs. As more and more blockchain systems emerge striving to improve DTL performance, it becomes imperative to devise a solution for comparing different platforms in a meaningful manner.

Up to date (June 2020), there are three popular performance benchmarks dedicated to evaluating blockchain systems, as listed in Table. 2.2.

Blockbench [26] is the first benchmark framework designed for evaluating private blockchains in terms of performance metrics on throughput, latency, scalability and fault-tolerance. Presently, it supports measurement on four major private blockchain platforms, namely Ethereum, Parity, HLF and Quorum. However, it claims to support

Table 2.2: A comparison of three popular blockchain benchmarks

Frameworks	**Supported DLTs**	**Workloads Used**	**Evaluated Metrics**	**Pros & Cons**
Blockbench [26]	Ethereum, Hyperledger Fabric (HLF), Parity and Quorum.	a) macro: YCSB(k-v store), Smallbank(OLTP), EtherId, Doubler, and WavesPresale b) micro: DoNothing, Analytics, IOHeavy, and CPUHeavy	throughput, latency, scalability and fault-tolerance.	adaptor-based framework, scalable; carefully designed workloads; but they are constant.
DAGbench [84]	IOTA, Nano and Byteball	a) value/data transfer b) transaction query: 1) input/output transaction numbers and 2) balance for a given account	throughput, latency, scalability, success indicator, resource consumption, transaction data size and transaction fee	adaptor-based framework, scalable; specific for DAG DLT; workloads are not representatives.
Hyperledger Caliper [85]	Hyperledger blockchains (Fabric, Sawtooth, Iroha, Burrow and Besu), Ethereum, FISCO BCOS	Self-defined in the configuration file	throughput, latency, resource consumption, success rate	adaptor-based framework, scalable; no pre-defined workload design, but support more DLT systems.

the evaluation of any private blockchain by accordingly extending the workload and blockchain adaptors.

In the design of Blockbench, four abstraction layers in blockchain are identified: consensus, data model, execution engine and application, from the bottom (low level) to the top (high level). The consensus layer is in charge of setting the rule of agreement and getting all network participants to agree on the block content so that it can be appended to the blockchain. The data model defines the data structure, content and operations on the blockchain data. The execution engine contains resources of the runtime environment such as the EVM and Docker, which support the execution operations of blockchain codes. The application layer includes all kinds of blockchain applications such as smart contracts and different types of DApps. It is worth noting that Blockbench adopts and designs various workloads to test the performance of different layers, as shown in Table 2.3.

Hyperledger Caliper [85] is a performance evaluation framework mainly focusing on benchmarking Hyperledger blockchains such as Hyperledger Fabric, Sawtooth, Iroha,

Table 2.3: Blockbench workloads for evaluating each layer of blockchain

Layer	**Benchmark Workload**		**Workload Description**	**Measurement Identifier**
Application	Macro Workloads	YCSB	Key-value store	throughput and latency
		Smallbank	OLTP workload	throughput and latency
		EtherId	Name registrar contract	throughput and latency
		Doubler	Ponzi (pyramid) scheme	throughput and latency
		WavesPresale	Crowd sale	throughput and latency
Execution Engine	Micro Workloads	CPUHeavy	Sort a large array	latency
Data Model		VersionKVStore	Keep state's versions (HLF only)	latency
		IOHeavy	Read and write a lot of data	latency
Consensus		DoNothing	Simple contract, do nothing	latency

Burrow and Besu. In the system architecture, there are two main components: Caliper core and Caliper adaptors. The former defines system workflow, while the latter are used to extend evaluation for other blockchains such as Ethereum and FISCO BCOS. Before running a test, benchmark workloads and necessary information interfacing adaptor to the system under test (SUT) need to be predefined in configuration files. During the test, a resource monitor runs to collect resource utilization information (e.g., CPU, RAM, network and IO) and all clients publish their transaction rate control information to a performance analyzer. When a test is finished, a detailed test report is generated by a report generator.

DAGbench [84] is a relatively recent framework dedicated to benchmarking DAG distributed ledgers such as IOTA, Nano and Byteball. The currently supported indicators are throughput, latency, scalability, success indicator, resource consumption, transaction data size and transaction fee. From the system design perspective, DAGbench shares the same approach with Blockbench and Caliper which adopt a modular adaptor-based architecture. Users just need to choose (or develop if not available) associated adaptors for different workloads and blockchain systems under evaluation.

Besides the general performance metrics evaluation, there are also studies focusing on specific metrics for particular blockchain. For example, OpBench [86] and another benchmark framework [87] are proposed to evaluate if a miner's award is proportional against to the CPU execution time or consumed computation power for Ethereum smart contracts.

2.2.2 Blockchain Performance Monitoring

Blockchain benchmarking usually requires a standardized environment and a well-documented workload as input. However, for public blockchain systems, it is impossible to have a good control against the real workload and consensus participants, which makes the benchmarking more challenging. In terms of evaluating public blockchains, there are two potential solutions.

The first solution is to build a private version of the associated test network and leverage the existing benchmarks mentioned above to evaluate blockchain under artificially designed workloads. This may require new adapter development for either workload or blockchain network. In addition, this approach should take into consideration the scalability problem of blockchain because the tested private version of blockchain may encounter scaling issues when implemented publicly. Therefore, the tested result may show better values of performance metrics compared to the real public network.

The second solution is to monitor and evaluate the live public system's performance under realistic workload [88]. Zheng *et al.* [89] proposed a detailed, real-time performance monitoring framework using a log-based approach. It has lower overhead, more details, and better scalability compared to its counterpart solution via remote procedure call (RPC). The high-level system framework is shown in Fig. 2.2.

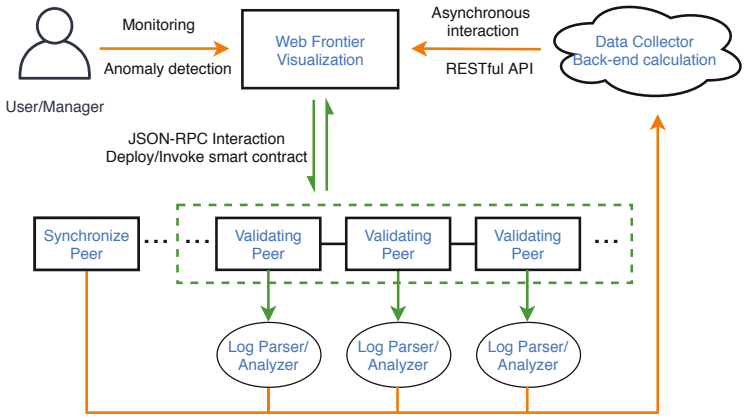


Figure 2.2: Blockchain performance monitoring framework [89]

2.2.3 Experimental Analysis of Blockchain Systems

In this section, we look at DLT performance evaluation from the perspective of empirical analysis based on self-designed experiments. Even though empirical analysis can hardly provide standardized test results like benchmarking, this approach is very flexible in parameterization. It can be used to identify potential bottlenecks and pave the way to further performance improvements.

Experiment-based approaches have been widely employed to evaluate distributed ledger systems such as Hyperledger, Ethereum and DAG-based ledger. Various private blockchain platforms and different versions of a certain blockchain can be compared on performance by running tests under a well-controlled test environment. In addition, some studies examined the detailed performance, for example, the performance of different encryption and hash algorithms, from the data layer in the blockchain abstraction model.

Hyperledger Performance Analysis

Nasir *et al.* [90] conducted an experimental performance analysis of two versions of HLF (v0.6 and v1.0) on their execution time, latency, throughput and scalability by varying the workloads and node scales. The overall results indicate that HLF v1.0 consistently outperforms HLF v0.6 across all evaluated performance metrics.

Baliga *et al.* [91] took an experimental approach to study throughput and latency of HLF v1.0. Using Caliper as the benchmarking tool, the authors configured different transaction and chaincode parameters to explore how they impact transaction latency and throughput under micro-workloads. Fabric's performance characteristics were also studied by varying the number of chaincodes, channels and peers. The results show that the throughput of HLF v1.0 is sensitive to the orderer settings, and it is a significant drawback for the commiter in this version that it does not process transactions in parallel, incapable of taking advantage of multiple vCPUs.

Another comprehensive empirical study was conducted by Thakkar *et al.* [92] who explored the performance bottlenecks of the HLF v1.0 under different block sizes, endorsement policies, number of channels, resource allocation and state database choices (GoLevelDB vs. CouchDB). The experimental results indicated that endorsement policy verification, sequential policy validation of transactions in a block, and state validation and commit (with CouchDB) were the three major bottlenecks. Accordingly, the authors suggested three optimization solutions, including parallel VSCC validation, cache for membership service provider (MSP), and bulk read/write for CouchDB. All these optimizations have been implemented in release HFL v1.1.

A study completed at IBM by Androulaki *et al.* [93] focused on HLF v1.1 to explore the impact of block size, peer CPU, and SSD vs. RAM disk on blockchain latency, throughput and network scalability under different numbers of peers. The results show that HLF v1.1 achieves end-to-end throughput of 3500+ TPS in certain popular deployment configurations, with the latency of a few hundred *ms*, scaling well to 100+ peers.

Nguyen *et al.* [94] conducted an experimental study to explore the impact of large network delays on the performance of Fabric by deploying HLF v1.2.1 over an area network between France and Germany. The results reveal that an obvious network delay (3.5s) brings 134 seconds offset after the 100th block between two clouds, which indicates that the tested version of Fabric can not provide sufficient consistency

guaranties. Therefore, HLF v1.2.1 cannot be used in critical environments such as banking or trading. This was the first work that experimentally demonstrated the negative impact of network delays on a PBFT-based blockchain.

Another HLF performance evaluation work focusing on the underlying communication network was conducted by Geyer *et al.* [95] using Caliper [85] and a dedicated testbed on which network parameters, such as latency or packet loss, can be configured. In the experiments, the influence of transaction rate, chaincode, network properties, local network impairment, and block size have been separately examined and quantitatively analyzed. The experiment results identified the validation of the transactions as the major contributor to the transaction latency in HLF.

As the first long-term support release, HLF v1.4 caught the attention of several blockchain researchers. Kuzlu *et al.* [96] investigated the impact of network workloads on the performance of a blockchain platform in terms of transaction throughput, latency, and scalability (i.e., the number of participants serviceable by the platform). Following network load parameters were varied in the experiment: number of transactions, transaction rate and transaction type.

Although the practical Byzantine fault tolerance (PBFT) algorithm has been adopted as the consensus protocol since its version 0.6, dishonest participants and their attacks such as intentionally delaying messages, sending inconsistent messages and distributed-denial-of-service (DDoS) never stop. Malicious behaviour may significantly undermine the system in terms of both security and performance. To explore the performance of HLF with malicious behaviour, Wang [97] designed multiple malicious behaviour patterns and experimentally tested the transaction throughput and latency on HLF. The results show that delay attacks, along with keeping some replicas out of working, dramatically decrease the system performance.

Apart from Fabric, Z. Shi *et al.* [98] empirically studied the performance of Sawtooth, another well-known permissioned blockchain platform from Hyperledger. The examined performance metrics include consistency (i.e., whether the platform's per-

formance behaves consistently each time with the same workload and cloud VM configuration), stability (i.e., whether the platform's performance remains stable with the same workload, but different cloud VM configurations) and scalability (i.e., how the platform performance achieves scalability with different workloads and configuration parameters). The adjustable configuration parameters identified for optimizing the performance of Sawtooth are scheduler and maximum batches per block.

From the results of empirical performance analysis summarized above, it is obvious that Hyperledger needs improvement on both *geographical scalability* (limited by the network latency) [94] and *size scalability* (the platform fails scaling beyond 16 nodes [26]). The bottleneck is the adopted PBFT consensus, which is a communication bound mechanism as opposed to the computation intensive PoW [99] consensus.

Ethereum Performance Analysis

Rouhani and Deters [100] studied the performance of Ethereum on a private blockchain by analyzing two most popular Ethereum clients: PoW-based Geth and PoA-based Parity. The results indicate that, compared to Geth, Parity is 89.82% faster in terms of transaction processing, on average, under different workloads.

Yasaweerasinghelage *et al.* [101] introduced an approach to predict the latency of blockchain-based systems using software architectural modelling tool Palladio workbench [102] and simulation. They leveraged the proposed method to test latency on a private Ethereum (Geth) experimental environment. The results show a low relative error of response time, mostly under 10%.

Bez *et al.* [103] conducted an initial quantitative analysis on the scalability of Ethereum. The transaction throughput was evaluated under an extensible test environment with synthetic benchmarks. The results indicate that Ethereum follows the scalability trilemma, which claims that a blockchain platform can hardly reach decentralization, scalability and security simultaneously.

DAG DLT Performance Analysis

In traditional blockchain systems, transactions are stored in blocks that are then organized as a ledger in a single chain data structure. This structure makes it incapable of concurrently generating blocks, and thus limiting the transaction throughput. In DAG-based distributed ledgers, transactions or blocks are organized in different vertices of the directed graph, which allows parallel block generation and inclusion. Based on this idea, many distributed ledgers have been proposed with their own consensus mechanisms. For example, IOTA [1] employs a cumulative weight approach for transaction confirmation and Markov chain Monte Carlo (*MCMC*) sampling algorithm for random *tip selection*; Byteball [24] achieves consensus by relying on 12 selected reputable *Witnesses*; and Nano [25] adopts a balance-weighted vote mechanism to reach agreement on transaction confirmation.

Even though DAG-based ledgers are designed to theoretically have faster transaction speed than blockchain, it is necessary to evaluate the performance of existing DAG distributed ledger implementations and identify their potential bottlenecks. Fan *et al.* [104] demonstrated the scalability of IOTA under IoT scenarios in a private network with 40 nodes. The experiment results indicated that transaction throughput (TPS) has good linear scalability against the transaction arrival rate. Three representatives of DAG-based distributed ledgers, namely IOTA, Nano and Byteball, were comparatively evaluated using the proposed DAGbench in [84]. From the experimental results, some useful observations, such as the advantages and disadvantages of the three tested DAG implementations, can be obtained.

Comparative Analysis

Before developing a blockchain-enabled application, decision makers should first assess the suitability [105] of blockchain implementation. Then, a comparative performance analysis is often necessary to select a blockchain platform that will perform well in the target application environment.

After developing Blockbench, Dinh *et al.* [26] used this tool to conduct a comparative performance analysis on three mainstream private blockchains, namely Ethereum (geth v1.4.18), Parity (v1.6.0) and HLF (v0.6.0-preview). Their findings can be summarized as follows: 1) HLF performs consistently better than Ethereum and Parity across all macro (e.g., throughput and latency) and micro (e.g., IOHeavy) benchmarks, but it fails to scale up to more than 16 nodes; 2) The consensus protocols are identified as major bottlenecks for HLF and Ethereum, while transaction signing is a bottleneck for Parity. The authors further compared the performance of two different versions of HLF v0.6.0 and v1.0.0 against IOHeavy workload in their more recent work [99].

Because of the lack of interface standards, evaluating different blockchains remains difficult. To overcome this problem, a generic workload performing the same functions on different blockchain interfaces was designed in [106] to comparatively evaluate three prominent consortium blockchain platforms for IoT. They were HLF v0.6 with the PBFT consensus, HLF v1.0 with the Byzantine fault-tolerant state machine replication (BFT-SMaRt) consensus, and Ripple with the Ripple consensus. Results confirmed that the evaluated blockchains could offer reasonable throughput but with very limited scalability.

Pongnumkul *et al.* [107] conducted a preliminary performance analysis of two popular private blockchain platforms HLF (v0.6) and Ethereum (geth 1.5.8, private deployment) under varying workloads. The experimental results demonstrated that HLF outperforms Ethereum in terms of all evaluated metrics (execution time, latency and throughput). However, this study also pointed out that the performances of both platforms are still not competitive with current mainstream database systems, especially under high workloads. This conclusion was tested and confirmed by another, more recent study [108], in which Ethereum and MySQL were compared.

Comparative analysis can also be conducted on consensus algorithms of different blockchains. For example, Hao *et al.* [109] compared the performance between Hyperledger (PBFT) and private Ethereum (PoW) via their proposed benchmark

framework constructed with four modules: workload configuration module, consensus smart contract module, data collector module and the target blockchain platforms. The evaluation results show that HLF consistently outperforms Ethereum in terms of average throughput (TPS) and latency. This study also points out that the consensus mechanism induces performance bottleneck in private blockchains. Another example is the performance analysis conducted on PoW and the Proof-of-Collatz Conjecture (PCC) [110]. PCC [111] is a recently introduced number-based theoretic PoW using a new metric called *Collatz orbits*, which are defined in the Collatz Conjecture algorithm. Authors evaluated these two consensus algorithms with respect to the execution time, the deployment time and the latency on a private blockchain network. The experiment results demonstrate that PCC-based blockchain consistently outperforms PoW-based blockchain in terms of all tested metrics and even steadily achieves 1000× faster execution speed than of PoW.

To provide system designers suggestions on smart contract platform selection, Benahmed *et al.* [112] conducted a comparative performance analysis of Hyperledger Sawtooth, EOS and Ethereum. Following the workloads used in Blockbench [26], the authors modified and defined three types of workloads, namely CPUHeavy, KVStore (Key-Value Store), and SmallBank, to comparatively test CPU consumption, memory consumption, load scalability and network scalability in distributed ledgers. The results reveal that the third generation platform EOS outperforms the other two in both resource consumption and speed, but with some shortcomings such as centralization. In addition, according to their performance in the test, Sawtooth was suggested for use in the Internet of Things and Ethereum's PoA implementation for the fast development of web-oriented applications.

To explore whether existing blockchain solutions can scale to large IoT networks, Han *et al.* [113] comparatively evaluated the performance of five selected prominent distributed ledgers using classic consensus protocols: Ripple, Tendermint, Corda and v0.6 and v1.0 of HLF. A series of exclusive tests were run to evaluate the throughput

and latency with different numbers of nodes (ranging from 2 to 32) for each of the ledgers. The results show that even though these systems can sometimes provide thousands of TPS throughput, their networks usually do not scale to tens of devices as the performance drops dramatically when the number of nodes increases. Table 2.4 lists an overview of various DLTs' performance extracted from the reviewed experimental analysis studies.

Table 2.4: Overview of different DLT performance (throughput and latency) under various evaluation environments

DLT	**Consensus**	**Throughput (TPS)**	**Latency (Secs)**	**Network (Size)**	**Node Configuration**
HLF v0.6 [26]	PBFT	1273	38	8 nodes	E5-1650 3.5GHz CPU, 32GB RAM, 2TB HD
	PBFT	1122	51	8 nodes	
Ethereum geth v1.4.18 [26]	PoW	284	92	8 nodes	
	PoW	255	114	8 nodes	
Parity v1.6.0 [26]	PoA	45	3	8 nodes	
	PoA	46	4	8 nodes	
Quorum 2.0 [91]	Raft	2,000+	1.5	3 nodes	8 vCPUs 4 cores 3.6 GHz, 16GB RAM
	IBFT	1,900	3.2	4 nodes	
	IBFT	1,800	3.5	4 nodes	
HL Sawtooth v1.1.2 [112]	PoET	3	-	6 nodes	Dockers share VM on Intel Xeon X7350 CPU 16 Cores, 2.93GHz, 64GB RAM
EOS v1.5.3 [112]	DPoS	21	-	6 nodes	
Ethereum Geth v1.8.21 [112]	PoW	10	-	6 nodes	
HLF v1.0 [106]	BFT-SMaRt	1,700	-	16 nodes	E5-2630 CPU 8 cores 2.4GHz, 64GB RAM
HLF v0.6 [106]	PBFT	2600	1.8	16 nodes	
Ripple v0.60.0 [106]	XRP	1450	6	16 nodes	
Tendermint v0.22.4 [113]	PBFT and Casper	6,000	0.15	16 nodes	E5-2630 CPU 4 cores 2.4GHz, 12GB RAM
Tendermint v0.22.4 [113]	PBFT and Casper	5,600	0.05	16 nodes	
R3 Corda v3.2 [113]	BFT-SMaRt	50	8	4 nodes	
Geth v1.7.3 [109]	PoW	130	1,297	4 nodes	8GB RAM, 128GB SSD
Geth v1.7.3 [109]	-	235	569	4 nodes	
HLF v1.0 [109]	BFT-SMaRt	535	78	4 nodes	
HLF v1.0 [109]	-	1,033	40	4 nodes	
Geth [100]	PoW	-	0.199	1 node	Core i7-6700 CPU, 24GB RAM
Parity [100]	PoW	-	0.105	1 node	
Geth 1.5.8 [107]	-	21	361	1 node	AWS EC2 Intel E5-1650 8 core CPU, 15GB RAM, 128GB SSD
HLF v0.6 [107]	-	160	4	1 node	

Encryption Performance Analysis

In addition to the end-to-end performance metrics, there are also some evaluation works focusing on the detailed performance of a certain step or subprocess such as the efficiency of encryption and hash function. According to Park *et al.* [114], the transaction processing time equation is

$$T = t_i + t_c = (t_v + t_{pow} + t_n + t_e) + t_c, \tag{2.1}$$

where t_i refers to the issuance time, t_c to the confirmation time, t_v to the validation time, t_{pow} to the PoW time, t_n to the network overhead, and t_e to the processing overheads. The processing overheads include encryption/decryption, hashing and authentication. Efficient encryption and hashing algorithms contribute to transaction issuance speed in DLT. Chandel *et al.* [115] analyzed and compared the performance of the two most commonly used encryption algorithms in blockchain, Rivest-Shamir-Adleman (RSA) and elliptic-curve cryptography (ECC). Their comprehensive analysis results based on the key size, key generation performance and signature verification performance show that the ECC algorithm (adopted by Bitcoin and Ethereum) outperforms RSA in general. This study also points out that ECC satisfies the security needs of blockchain better than RSA.

More recently, Ferreira *et al.* [116] conducted a study on Blockchain-based IoT (BIoT) [11] to explore the performance of hash function in blockchains. Particularly, authors developed a blockchain in an IoT scenario to evaluate the performance of different cryptographic hash functions such as MD5, SHA-1, SHA-224, SHA-384 and SHA-512. The test results show that SHA-224 and SHA-384 are the best hash functions for blockchain due to their lack of collision attacks. In hashing ciphers, a collision attack is the problem that there exist two different messages m_1 and m_2, such that $\text{hash}(m_1) = \text{hash}(m_2)$. In addition, these two hash functions are more time-efficient than others to process blockchain functions with the advantage of producing a smaller

average block size.

2.2.4 Simulation

All the evaluation solutions mentioned above (i.e., benchmarking, monitoring and experimental analysis) require the availability of the systems, no matter private or public blockchains. However, the system under evaluation is not always available. For instance, when a company needs to make a selection between two blockchain platforms under development according to their performance, none of the previously discussed solutions is feasible. Moreover, it is usually costly on both time and resources to construct a real blockchain network for testing. This brings us to explore another evaluation approach, namely, *simulation*. A blockchain simulator can mimic the behaviours of network nodes in reaching the consensus, providing performance similar to a real system. Besides, a blockchain simulator usually provides a convenient way for users to tune the system parameters to run different settings for the sake of comparison. In this subsection, we will take a look at the role of simulation in the blockchain world.

BlockSIM

In 2019, there were three similar simulators with the same name, *BlockSim* (or *BlockSIM*), proposed for simulating blockchain systems. Alharby and Moorsel [117] proposed and implemented a framework called BlockSim to build discrete-event dynamic system models for PoW-based blockchain systems. This framework was organized in three layers: incentive layer, connector layer and system layer. Using the proposed simulation tool, the authors explored the block creation performance under the PoW consensus algorithm. This simulator helped to understand the details of the block generation process in PoW. The predefined test cases were validated and verified in their extension study, where the simulation outcomes were compared with results of real-life systems such as Bitcoin and Ethereum to show the feasibility of this approach. However, the extensibility of this simulator is still a problem for future

research.

To help architects better understand, evaluate and plan for the system performance, Pandey *et al.* [118] proposed and developed a comprehensive open-source simulation tool called BlockSIM for simulating private blockchain systems. This tool is designed to evaluate system stability and transaction throughput (TPS) for private blockchain networks by running scenarios, and then decide on the optimal system parameters suited for the purposes of architects. The comparison results between BlockSIM and a real-world Ethereum private network running PoA consensus show that BlockSIM can be used effectively.

More recently, Faria and Correia [119] presented a flexible discrete-event simulator (also called BlockSim) to evaluate different blockchain implementations. With a good design of APIs, new blockchains can be easily modelled and simulated by extending the models. Running this simulator for Bitcoin and Ethereum, the authors got some interesting performance conclusions. For instance, doubling the block size (number of transactions) had a small impact on the block propagation delay (10ms), while encrypting communication had a higher impact on that delay (more than 25%).

DAGsim

Similarly, Zander *et al.* [120] presented a continuous-time, multi-agent simulation framework called *DAGsim*, for DAG-based distributed ledgers. Specifically, the performance of IOTA in terms of the transaction attachment probability was analyzed using this tool. The results indicate that agents with low latency and high connection degrees have a higher probability of having their transactions accepted in the network. Another multi-agent tangle simulator [121] built with NetLogo simulates both random uniform and MCMC tip selection in a visualized and interactive way.

In addition to pure simulators, some other studies leverage simulations combined with analytical results to conduct validation or exploration. Park *et al.* [114] proposed and implemented a general DAG-based cryptocurrency simulator using Python. This

simulator was used to validate the proposed analytical performance model, through which they found that by issuing a transaction with a smaller average number of parents n in DAG, the transaction speed (TPS) can be increased. Kusmierz *et al.* [122] ran IOTA tangle simulations in a continuous-time model to explore how different tip selection algorithms, i.e., uniform random tip selection (URTS) and unbiased random walk (URW), affect the growth of the tangle. Simulations under varying transaction arrival rates were used to analyze the performance of the tangle.

2.2.5 Comparison of Different Evaluation Solutions

In the previous subsections, we introduced four types of empirical evaluation solutions and surveyed existing studies which adopted the associated approaches. In this subsection, we comparatively discuss the advantages and disadvantages of the above-mentioned solutions. This comparison is based on both the general characteristics of the individual approaches and their suitability in evaluating different types of blockchains. The compared items are divided into two categories: solution requirements and solution efficiency, see Table 2.5. Solution requirements describe the network specifications for evaluating blockchain systems in terms of the node, network and workload. Solution efficiency provides three dimensions, namely parameterization, extensibility and difficulty, to compare the efficiency and effectiveness of different solutions.

Table 2.5: A comparison on different empirical performance evaluation solutions for blockchain system

Solutions	**Characteristics**			**Efficiency**		
	Node	**Network**	**Workload**	**Parameterization**	**Extensibility**	**Difficulty**
Monitoring	Real	Real	Real	Low	Low	Easy
Benchmarking	Real	Test	Artificial	Low	High	Easy
Experimental Analysis	Real	Test	Artificial	High	Low	Medium
Simulation	Virtual	Virtual	Virtual	Very High	Very High	Hard

Monitoring the performance of a blockchain system requires a realistic deployment

of the system in production with real workloads. Even though this approach can also be used to evaluate a private blockchain in an experimental setup, we argue that it is more suitable to evaluate public blockchain when compared with benchmarking. In the context of evaluating a public blockchain, it becomes difficult to change any parameters for multiple tests. The challenge of the extensibility lies in the development of adaptable log parser for various blockchains. But, it is easy to deploy for certain blockchains using the existing solutions [89].

Benchmarking requires a well-controlled evaluation environment with a test network and artificial workloads. Once a benchmark tool is selected, the supported workloads and test metrics are limited, as well as the parameters which can be tuned. For example, Blockbench doesn't support tuning the network layer parameters such as network delay and, up to date, it only supports evaluating four types of blockchain platforms, i.e., Ethereum, HyperledgerFabric (HLF), Parity and Quorum. However, the well-designed APIs allow users to develop their own adaptors and extend its feasibility to evaluate any private blockchains. So, the extensibility of benchmarking is relatively higher than monitoring. In addition, this solution is easy to deploy since there have been several popular and well-documented benchmark tools, see Table 2.2.

Experimental analysis refers to the evaluation solution based on self-designed experiments. This is a very general solution that is commonly used. It is very similar to benchmarking but different in two main aspects. First, self-design gives more flexibility in evaluating impact factors, providing a high capability of parameterization. For example, the impact of network delay on HLF performance can be evaluated in a self-designed experiment, which is not supported by benchmarking. Second, the test is usually dedicated to a specific blockchain and is not as standardized as benchmarking, which limits the extensibility. So, the deployment difficulty partly depends on the complexity of the SUT and what to evaluate.

Simulations have a relatively greater difficulty in the stage of simulator design and development. But, once it is completed, the simulator usually provides a number

of advantages in comparison to other approaches. The simulation solution is very extensible and can be used to quickly test different configuration parameters at a low cost. As mentioned in subsection 2.2.4, another obvious advantage of simulation is that it does not require the availability of the blockchain. However, as for the evaluation results, there may be a relatively large difference (e.g., 10%) between simulation and experiment, which induces concerns about the accuracy of this solution. Moreover, some metrics cannot be evaluated in simulators such as the transactions per CPU, transactions per memory second, transactions per disk IO, and transactions per network data for a blockchain system.

2.3 Analytical Modelling in Blockchain Performance Evaluation

Analytical modelling of performance leverages mathematical tools to formalize blockchain system in an abstract way and to solve ensuing models with rigor. The model output (e.g., average transaction latency being expressed as a function of network indicators) provides analytical evidence for blockchain performance evaluation. In this section, we survey the performance evaluation solutions of distributed ledger systems based on analytical modelling. We aim to summarize the mainstream techniques, explore how and why these models are employed for certain distributed ledgers, and then identify the current challenges in blockchain performance modelling. In particular, we focus on surveying the stochastic models, which have been used to successfully model many cloud systems.

In Table. 2.6, we classify the existing popular solutions of performance modelling for distributed ledgers into four categories: Markov chains, queueing models, stochastic Petri nets and other models.

Table 2.6: A summary of blockchain performance modelling studies

Model Types	Models	Consensus	DLs	Model Outputs
Markov Chains	Absorbing Discrete Time Markov chain (DTMC) [123]	Raft	private blockchains	network split probability as a function of packet loss rate, election timeout, and network size
	Discrete Time Markov chain (DTMC) [124]	the tangle	IOTA	cumulative weight and transaction confirmation delay
Queueing Models	$M/G^B/1$ queue variant [125]	PoW	Bitcoin	tx confirmation time
	CTMC GI/M/1 queue [126]	PoW	Bitcoin	mean number of txs in the queue and in a block; average tx-confirmation time.
	CTMC GI/M/1 queue [127]	PoW	Bitcoin	mean stationary number of txs in the queue and in the block
	M/G/1 queue variant [128]	PoW	Bitcoin	confirmation time and tx delay
	non-exhaustive queue [129]	PoW	NA	mean number of txs and mean confirmation time of txs in the system
	Discrete-time $GI/GI^N/1$ queue [130]	Proof-of-Authority	Ethereum	system queue size and tx waiting time
	$M/M^B/1$ queue [95]	BFT-SMaRt	HLF	tx latency
	M/G/1 and M/M/1 queue [131]	PBFT	NA	system delay
	(n,k) fork-join queue [132]	vote-based consensus	permissioned blockchain	blocks commitment delay, block validation response time and synchronization processes among mining nodes.
	Fluid queue [133]	the tangle	IOTA	conflicting txs cannot coexist when a random tip-selection algorithm is employed
Stochastic Petri Nets	Generalized stochastic Petri nets (GSPN) [134]	PBFT	HLF v1.2	latency and throughput of each phase
	Stochastic Reward Nets(SRN) [135]	PBFT	HLF v1.0	throughput, utilization and mean queue length at each peer
Other Models	World State Prediction model [136]	PoW	Ethereum	transaction time cost
	Stochastic network model [137]	PoW	Ethereum	tx processing rate
	Random Graph model [138]	PoW	Bitcoin	block propagation delay and traffic overhead

2.3.1 Markov Chains for Modelling DLT Consensuses

In probability theory, Markov processes are a type of stochastic process with *Markov property*. Also called *memoryless property*, it refers to the fact that the future states

of the process depend *only* on the present state, but not on the previous ones. *Markov chain* is defined as a Markov process with discrete state space. It is a fundamental mathematical tool to evaluate the performance of distributed ledger systems [139]. In this subsection, we investigate how Markov chains are used to model two different consensus algorithms: *Raft* and *the tangle* for IOTA. The specific type of process used for this modelling is called discrete time Markov chain (DTMC).

DTMC for Modelling Raft: In a Raft [140] cluster, each node is at any given time in one of the three states: follower, candidate and leader. Normally, there is only one leader in a Raft cluster. We call it *network split* in the case of two or more leaders being elected simultaneously, which may dramatically impact the performance of the system. After the leader has been elected, it handles all requests from the client and sends them to followers for validation. Followers simply receive requests from and respond to leaders and candidates. Candidates are a mid-state transiting from follower to leader. The whole Raft consensus can be divided into several ever increasing timely manners called *terms* which have two processes: leader election and ledger replication. Each term starts with a leader election, in which all nodes start from follower state. Then, a node transits to candidate, candidate to leader or back to follower according to the rules depicted in Fig. 2.3 [140]. Once a leader is elected successfully, the ledger replication process starts with the leader sending heartbeat messages to all other nodes to establish its authority and prevent new elections. Once the leader receives responses of writing new transaction entry to the ledger from the majority of the followers, it notifies them and the client that the transaction is committed.

To explore the impact of network properties on the blockchain performance, Huang *et al.* [123] have built a simple Markov chain model for the process of a node transferring from follower state to candidate. They consequently present the network split probability as a function of the network size, the packet loss rate, and the election timeout period. Let us define the packet loss probability as a constant value p for a given network, the timeout period for each round of election as E_t uniformly initiated from a

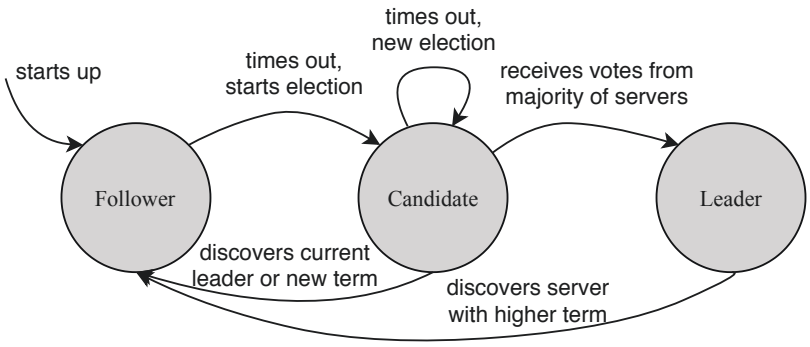


Figure 2.3: Node states transition illustration in Raft consensus.

range [a,b], and interval between two heartbeats as τ. Thus, the maximum number of heartbeats for an election to timeout is $K \in \{K_1, K_2, ..., K_r\}$, where $K_1 = \lfloor a/\tau \rfloor$ and $K_r = \lfloor b/\tau \rfloor$. Then, two discrete time stochastic processes at time n can be defined: ***g(n)*** as the stage status {1,2,...,r} of a given node, and ***b(n)*** as the remaining steps (i.e., number of heartbeats) for the election phase to timeout in a term.

Therefore, the transition process for an observed node from follower to candidate can be modelled as a two-dimensional stochastic process {***g(n),b(n)***}. It can be further transformed to an absorbing DTMC on the state space {(1,K_1),...,(1,0),..., (i,K_1),...,(i,0),...,(r,K_r),...,(r,0)}.

Using the mathematical derivations proven in [123], the network split probability before n-th step can be obtained.

DTMC for Modelling IOTA Tangle: IOTA tangle [1] is a DAG-based distributed ledger designed for the microtransactions in the IoT. Its consensus encourages all participants to contribute in maintaining the ledger through referencing (i.e., approving) two unapproved transactions called *tips* before issuing any new transaction. For the new coming transaction, IOTA tangle leverages the MCMC random walk algorithm to select two tips. All transactions directly or indirectly approved by this new transaction then add its weight to their cumulative weights, as shown in Fig. 2.4. For an approved transaction, its cumulative weight gradually increases to reach a predefined threshold. Finally, the corresponding transaction is considered confirmed

and permanently recorded in the ledger.

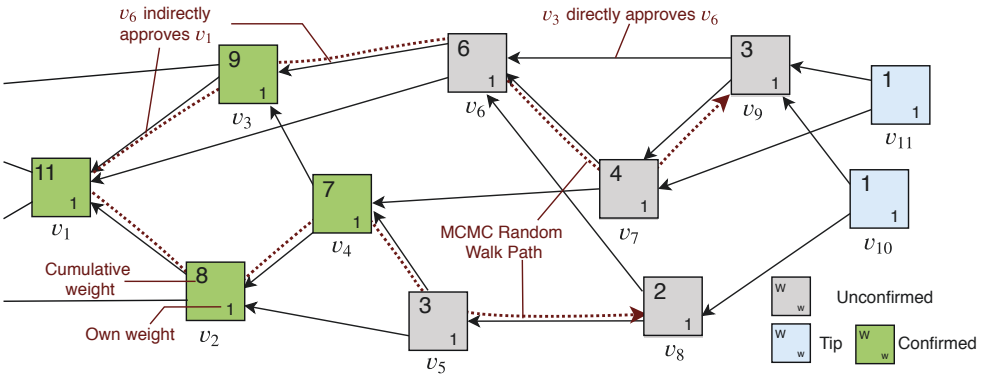


Figure 2.4: An example of the IOTA tangle.

To explore the impact of various transaction arrival rates on the cumulative weight and confirmation delay of an observed transaction, Cao *et al.* [124] built a Markov chain model to analyze the tangle consensus. They classified the network into four different regimes, according to the load situations: high load (HR), low load (LR), high-to-low load (H2LR) and low-to-high-load (L2HR). In each regime, the consensus process can be divided into two stages, namely the reveal stage and accumulating stage [1]. Since the first two steady regimes HR and LR have been analyzed in [1], the authors only focus on two unsteady regimes H2LR and L2HR.

The system can be modelled as a two-dimensional stochastic process $S(t) = W(t)$, $L(t)$ at an arbitrary time t, where $W(t)$ is the cumulative weight of a transaction observed at time t, and $L(t)$ is the total number of tips in the tangle at time t, $t = kh$, $k = 0,1,2,...,\infty$. Considering that $W(t+h)$ and $L(t+h)$ are only determined by the current states $W(t)$ and $L(t)$, but not related to the earlier status, the consensus process for a new observed transaction from issuance to confirmation can be formulated as a Markov process. Furthermore, this Markov process can be formalized as a DTMC on discrete transaction arrival time intervals. Here, one step transition of the observed transaction is defined as the arrival of an incoming transaction with randomly selecting two tips for reference from $L(t)$ tips. Based on this DTMC model, the expected cumulative weight and confirmation delay at a certain time in both H2LR and L2HR

can be obtained.

2.3.2 Queueing Theory for Modelling DLT Consensuses

Queueing theory was originally proposed by Agner Krarup Erlang in 1909, to describe the Copenhagen telephone exchange. It was later developed to solve different types of system problems that involve waiting, such as customers waiting for teller service in banks. In recent years, queueing theory has been widely used to model computer networks and systems [141], cloud computing centers [142], and blockchain systems. In a blockchain system, transactions issued by clients need to wait for servers (e.g., miner, validator or orderer) to provide service (e.g., mining, validating or ordering), and finally get confirmed.

Using queueing theory, different consensus processes of DLTs can be modelled as different types of queue systems, which are named according to the Kendall's notation [143]. Within a queue system, it is possible to quantitatively answer some system performance questions such as *what is the expected number of transactions in the system*, *what is the transaction throughput of the system* and *what is the average service time (i.e., transaction time)*. In this subsection, we focus on introducing the typical queueing models (e.g., M/M/1, M/G/1 and G/M/1 queues) used for addressing these performance questions of some mainstream consensus algorithms for blockchain.

Queueing Models for Proof-based Consensuses

Proof-based consensus is a type of consensus mechanism that requires the network participants to provide enough proof to compete for the chance of updating the ledger. Here, we review the queue systems for modelling some popular consensus mechanisms such as PoW and PoA.

Queueing Models for PoW: In PoW-based blockchain such as Bitcoin [39], the ledger is maintained and updated by the *mining* process. In the mining process, a bunch of nodes called *miners* compete for solving very difficult puzzle-like problems,

which consume a lot of computation power. Transactions issued by users are grouped into a container called a *block*, and the mining competition winner who first finds the algorithmic puzzle answer specialized for the block has the right to add the new block to the blockchain and accordingly gets incentives.

In 2017, Kawase and Kasahara [125] first built a modified $M/G^B/1$ queue with batch service to model the Bitcoin mining process, trying to deal with the transaction-confirmation time. In this model, transaction arrival was assumed to be a Poisson process and service time interval to be a general (or arbitrary) distribution. Arriving transactions are served in a batch manner with a maximum batch size b. In a typical $M/G^B/1$ queue system, an idle server starts service immediately if there are one or more customers awaiting service in the system [144]. But in this variant model, newly arriving transactions wait in the queue for getting served until the next block-generation time, even when the number of transactions is smaller than b. This is regarded as the service with multiple vacations. This is a very straightforward model description from the Bitcoin block generation and mining process based on Nakamoto's consensus, in which new transactions are grouped into a block to wait for being mined in the next block-generation time or even later on.

To analyze this queue system, the authors leveraged the joint distribution of the number of transactions in the system and the elapsed service time to derive the mean transaction-confirmation time. Then, by using the method of supplementary variables, a system of differential-difference equations was set up to formalize the problem. However, they have not successfully provided the unique solution of the differential-difference equations' system, leaving analysis of the blockchain queue system as an interesting open problem for future research.

To overcome the difficulties encountered in the original model [125], Li *et al.* introduced a new blockchain queueing model [126] by decomposing the mining process into two different exponential service stages: block-generation and blockchain-building processes. The sum of both stages' times is regarded as the *transaction-confirmation*

time, also called *service time*. In this model, all Bitcoin transactions are assumed to be arriving according to a Poisson process, namely the arrival interval times follow an exponential distribution with arrival rate λ. Service times in two stages of batch services are also simply assumed to be i.i.d. and exponentially distributed with rates μ_2 and μ_1, respectively. First, each transaction enters a queue waiting room and waits for services. Then, in the first service process, a group of transactions are mined into a block with rate μ_2 and, simultaneously, a nonce is appended to the block by the mining winner. The block has a limited transaction capacity of b, also called batch size in the model. In practice, the selection of transactions may not follow a first-come-first-serve (FCFS) discipline, meaning that some latter coming transactions in the queue may be preferentially first selected into the block. But in this model, all computations are based on the FCFS discipline for the reason of simplification. Finally, a generated block with all transactions wrapped in it is attached to the blockchain in a transaction rate of μ_1. The simple blockchain queueing model is illustrated in Fig. 2.5.

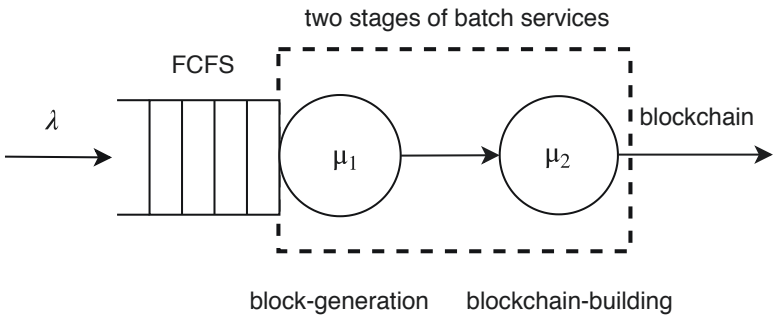


Figure 2.5: Blockchain queueing model with two batch service processes.

To analyze this queueing model, the authors defined two random variables $I(t)$ and $J(t)$ as the numbers of transactions in the block and in the queue at time t, respectively. Thus, the system can be modelled as a two-dimensional *continuous-time Markov chain (CTMC)* $X(t) = \{I(t), J(t)\}$ on the state space $\Omega = \{(i, j) : i = 0, 1, ..., b; j = 0, 1, 2...\}$. By analyzing the state transition diagram (see [126] for details), the only three possible transitions from an arbitrary state *(i, j)* are to state: ***(i, j+1)***, the same level; ***(0,***

j), i levels up; or ***(l, i+j-l)***, l ($1 \leq l \leq b$) levels down. With all these characteristics, the corresponding Markov transition matrix (or infinitesimal generator) Q is a lower Hessenberg matrix, which is constructed by different repetitive small matrix blocks. Therefore, $X(t)$ is a continuous-time Markov process of GI/M/1-type. This block-structured Markov chain (the other two examples are M/M/1-type and M/G/1-type) can be solved using the matrix-analytic (or matrix-geometric) approach.

However, this model has very strong assumptions on transaction arrival and service processes. It is too specific and not suitable for many practical conditions of blockchain systems. To generalize this model, in their more recent work [127], the authors changed the transaction arrivals from Poisson to Markov arrival process (MAP), the service times from exponential to phase-type (PH), and the service discipline from FCFS to service-in-random-order. Under the new assumptions, the blockchain queueing model description keeps the same. Note that this is also a structured GI/M/1-type Markov chain. For the solution, matrix-geometric approaches are adopted to analyze and find the stable condition. This is the same as the stationary condition of the previous model. The simple expressions for the average stationary number of transactions in the queue waiting room $E(N_1)$ and the average stationary number of transactions in the block $E(N_2)$ are obtained separately.

Because of the batch service and the Service-In-Random-Order discipline for choosing transactions from the queueing waiting room into a block, the Markov chain structure becomes more complicated. This makes the computation of transaction-confirmation time very difficult. To overcome the challenge, the authors borrowed a computational technique by means of both the PH distributions and the RG factorizations [127].

There have been other blockchain queueing models proposed for analyzing the performance of PoW consensus. Ricci *et al.* [128] proposed a framework combining machine learning with queueing theory to study Bitcoin transaction delays. They introduced a simple queueing model for characterization of the transaction confirmation that can

be consiered a variant of M/G/1 queue. Different from complicated mathematical derivations, the authors mainly leveraged the *operational laws* in queueing theory, such as *Little's Law* to solve the queue system. The most important result, namely average transaction delay experienced by a user, is given as $E(D) = \alpha E(B) + E(B_r)$, where α is the expected number of blocks that a user needs to wait until a transaction is confirmed, $E(B_r)$ denotes the residual time of the inter-block time, and $E(B)$ stands for the average time between block confirmations. This formalization is inspired by the standard M/G/1 queueing model, where the coefficient of the residual service time equals the system utilization. In this variant of the model, a block is always being mined, making the utilization 100% all the time.

Zhao *et al.* [129] established a type of non-exhaustive queueing model to study the *average transaction confirmation time* in a PoW-based blockchain system. For such system, any block has a size limitation, and the block cannot be confirmed during the mining process. Therefore, a non-exhaustive queue with a limited batch service and a possible zero-transaction service is naturally more suitable to capture the system features. In this queueing model, the mining process is treated as a *vacation*, and the block-verification process is regarded as a *service*. Transaction arrival is assumed to be a Poisson process with rate λ. The time duration V for a mining process and the time duration S for a block-verification process are both i.i.d. variables that follow a general distribution with distribution functions $V(t)$ and $S(t)$, respectively. *Laplace-Stieltjes transform (LST)* has been widely used in modelling both mining and block-verification processes to provide integral expressions for $E[V]$ and $E[S]$. Through a series of mathematical transformations and derivations, the authors eventually obtained the following expression for average transaction confirmation time: $E[C] = E[S] + E[V]$ (refer to [129] for details).

Queueing Models for PoA: To evaluate the performance of the mining process in Proof-of-X based blockchains, Geissler *et al.* [130] proposed a generic discrete-time $GI/GI^N/1$ queueing model. Their goal was to investigate key performance indicators,

such as the mean queue size and mean transaction waiting time, and to identify significant impact factors. To make this model general, the authors abstracted the blockchain network as a single server by neglecting the information propagation delays among network nodes. Then, the model was built around a fixed-point iteration of the queue size distribution by representing the system state with queue size Q_n.

In this system model, the transaction interarrival time A follows a general distribution $a(k)$ described as $A(k) = P(A < k) = \sum_{i=0}^{k} a(k), k \in [0, \infty)$. The service time T is also assumed to follow a general distribution. Every time a new transaction arrives, the size of queue $Q(k)$ increases by one, while every block generation decreases the queue size by confirming a batch of transactions from the queue. Thus, the queue size distribution can be defined recursively, with iteration based on an embedded Markov chain with embedding times right before a block generation event. Furthermore, the distribution of key performance indicator *transaction waiting time* can be defined by the recurrence time distribution of the block generation process $r_T(x)$ and the coefficient of weighted probability $c(k)$. The corresponding expressions are obtained through recursive solutions, Little's law of queueing theory and basic probability mathematical derivations, see [130] for details. In the evaluation part, the authors obtained a good match by comparing the model data and the experimental measurements, which showed the effectiveness and accuracy of the model. Unfortunately, this general model was only validated by using a specific Ethereum implementation based on the Proof-of-Authority (PoA) consensus. It discounts the versatility of this model, since the more popular PoX consensuses such as PoW and PoS have not been examined.

Queueing Models for Vote-based Consensuses

Vote-based consensus is a type of high performance algorithms relying upon voting to reach agreement on transaction processing among participant nodes in a distributed system. It is the most popular consensus mechanism used in permissioned blockchain. Three widely used representatives of the vote-based consensus implementations are

PBFT [145], BFT-SMaRt [146], and delegated Byzantine fault tolerance (dBFT).

Queueing Models for PBFT: The classic PBFT algorithm was firstly proposed in 1999 to solve the transmission errors and Byzantine faults in distributed systems [145]. It consists of five steps: request, pre-prepare, prepare, commit and reply. When the PBFT is adopted in constructing blockchain systems such as Hyperledger Fabric v0.6 [93], Zilliqa [147], and EOS [148], it has different implementations and/or combinations with other protocols. For example, EOS takes a hybrid consensus of combining PBFT with DPoS, to greatly reduce the required consensus time. Zilliqa uses an optimized version of classic PBFT binded with sharded PoW to achieve consensus in an efficient manner, yielding a high throughput for the blockchain system. HLF, as the most well-known permissioned enterprise-level distributed ledger platform, implements the PBFT consensus algorithm among the network peers (i.e., endorser, orderer and committer) mainly through three phases: endorsement, ordering and validation, as illustrated in Fig. 2.6.

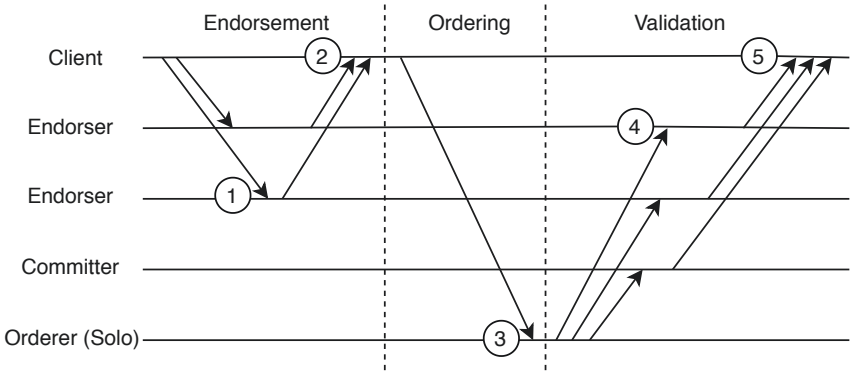


Figure 2.6: Hyperledger Fabric transaction workflow.

Phase 1. Endorsement (also called *proposal* or *execution*): ① The client generates transaction proposals and submits to endorsers for execution. ② The endorsers simulate the transactions by executing the operation previously written on the chaincode, and then return responses with signed endorsements to the client. The endorsements contain the values read or written called *read/write set* (or

rw-set) by the chaincode.

Phase 2. Ordering: The client sends the transaction together with the endorsements to the Solo orderer for ordering service. ③ The orderer collects transactions submitted from different clients, establishes a total order on them for each channel, packages multiple transactions into blocks and generates a hash-chained sequence of blocks. As for HLF v2.0, there are three implementations of ordering peers: Solo, Kafka, and Raft.

Phase 3. Validation (also called *validation and commit*): ④ The ordered blocks are delivered to committers through gossip protocol broadcasting. All peers are committers by default, including pure committers and committers with additional endorser responsibilities. Subsequently, the peers validate each transaction contained in the received blocks. If all validations are passed, the transaction's write set is applied to the peer's world state, and the client gets a notification about the successful execution of the transaction ⑤. Otherwise, any check fail will mark the transaction as invalid, and its effects are disregarded.

Geyer *et al.* [95] modelled the Solo ordering process of HLF as an $M/M^B/1$ queueing system. According to the previously described three phases, transactions with endorsements arrive at the orderer at different times and are queued. While the queued transactions reach a threshold number B (called batch size), the orderer immediately provides ordering service and packages them all at once into a block. If the transactions arrive according to a Poisson process with rate λ and the ordering service time is assumed to follow exponential distribution with rate μ and FCFS discipline, the service process can be described as an $M/M^B/1$ queueing system, as shown in Fig. 2.7.

To solve this model, the authors borrowed the results from a well-studied general bulk service queueing model $M/M_{a,b}/1$ [149]. They simply modified the batch size range to $a = b = B$. Then, the average time spent in the ordering phase $E(T)$ can be expressed by the given parameters, among which the batch size B is approved to

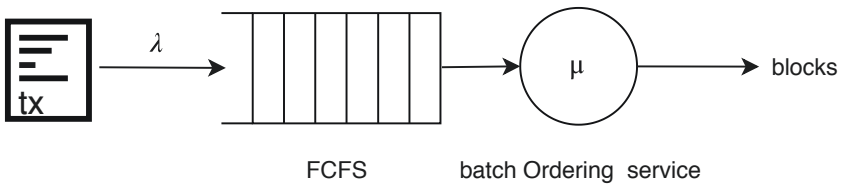


Figure 2.7: Hyperledger Fabric ordering service illustration and $M/M^B/1$ queueing model.

be significant to $E(T)$ from the numeric evaluations. This model well captures the characteristics of the ordering phase in Solo implementation. However, its shortcomings are obvious: 1) it is not suitable for Raft or Kafka implementations; and 2) it does not describe the whole transaction delay in the HLF system.

Alaslani *et al.* [131] focused on PBFT blockchain system end-to-end delay evaluation in IoT. To study the system delay, the authors built a model with two standard queues to capture the features of PBFT consensus from the system level. In this system, there are M IoT devices working as clients to send transaction requests, and K intermediate switches and R consensus replicas working together to process transactions. Since different IoT applications have different latency requirements to guarantee their service level agreement (SLA), network parameters need to be analyzed to meet the requirements. In the first part of the model, an M/G/1 queue is considered in which the maximum number of network hops K^* needs to be calculated under the application latency constrains. In the second part of the model, an M/M/1 queue is used to calculate R, the number of consensus replicas (i.e., blockchain consensus participants) needed to maintain the end-to-end requirements. Next, operational laws such as Little's law are used to analyze the network hops, and the number of consensus replicas, where three main phases (i.e., preprepare, prepare, and commit) of PBFT and its fault tolerance capability f out of $N = 3f + 1$ replicas are taken into consideration.

Fork-join Queue for Vote-based Consensus: In vote-based permissioned blockchain systems, transactions are broadcast to all authenticated voting peers

of the P2P overlay after being proposed. These voting peers, called miner nodes or validators, are selected and authorized to validate transactions, generate new blocks and record data to the local ledger if a transaction gets enough validation votes, e.g., k out of n. For example, in the PBFT, a block is accepted and recorded if $2f + 1$ out of $n = 3f + 1$ peers independently agree on the block of transactions, where f is the maximum number of Byzantine fault peers this system can tolerate.

The idea of leveraging an (n, k) fork-join queue to model vote-based blockchain is based on the fact that the service process of this queue system matches well with the above-mentioned transaction propagation and validation procedure. In an (n, k) fork–join queue, the incoming jobs are split/forked on arrival for simultaneous and independent service by numerous servers and joined before departure. While in a vote-based blockchain system, if we consider the confirmation of a transaction as a big job requiring enough validations from n nodes, this job can be split into n sub-tasks, associated with being broadcast to n nodes and being validated independently at the same time. Once any k out of n sub-tasks are finished, they are joined to finish the service and make the transaction confirmed and recorded to the local ledger. The remaining $n - k$ sub-tasks keep executing until being finished. This is called a non-purging (n, k) fork- queue. By contrast, a purging (n, k) fork-join queue removes all remaining sub-tasks of a job from both sub-queues and service nodes once it receives the job's kth answer.

In the literature, this model is highly prevalent for performance modelling (e.g., estimating the sojourn time of jobs in the queues) of parallel and distributed systems. Recently, it has been found effective for use in studying the delay performance of the synchronization process of the vote-based permissioned blockchain systems [132]. A typical non-purging (n, k) fork-join queueing model is illustrated in Fig. 2.8.

Even though few analytical results exist for fork–join queues, various approximation solutions are known. An example is the linear transformation approach [150], which can be used to approximately compute the sojourn time $t(n, k)$ of a general non-purging

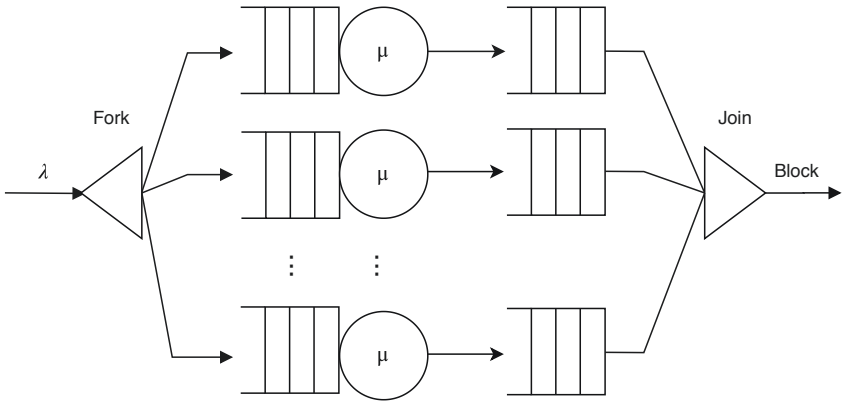


Figure 2.8: A typical fork-join queueing model. All blockchain voting nodes are homogeneous with the common service rate μ.

(n, k) fork-join queue for the vote-based blockchain system.

Fluid Queue for IOTA Tangle

In queueing theory, a fluid queue (also called fluid model) is a mathematical model used to describe the fluid level in a reservoir, for which the periods of filling and emptying are randomly determined. It can be viewed as a large tank connected to a series of pipes that pour fluid into the tank and a series of pumps that remove fluid from the tank. The capacity of this tank is typically assumed to be infinite. The fluid level $X(t)$ of this tank at time t is a random variable that can be calculated if the fluid arrival and leaving rates are given.

This model was successfully used to describe the dynamic behaviour of the IOTA tangle [133]. First, a fluid model was heuristically built based on some requisite stochastic models and the assumptions on the transaction arrival rate. Through solving the proposed delay differential equations system, the authors analyzed the stability of conflicts, which impacted the performance in return.

2.3.3 Stochastic Petri Nets for Modelling DLT Consensuses

Another type of commonly used analytical tool for BFT-based consensus performance modelling is *stochastic Petri net (SPN)*, especially its variants *generalized stochastic*

Petri net (GSPN) and *stochastic reward net (SRN)*. Petri nets (PNs) are a type of powerful mathematical modelling language used to model and simulate discrete-event distributed systems. They are graphs consisting of two types of nodes: places and transitions, which represent variables of system states represented by circles and actions made by the system represented by rectangles. When the *firing* times of all transitions are exponentially distributed (timed transitions), the model is called SPN. Built on SPN, a GSPN allows transitions to have zero firing times (immediate transitions) and *inhibitor arc* – an arc from a place to a transition that inhibits the firing of the transition when the input place is not empty. According to the literature, any GSPN model can be converted to an equivalent CTMC, and vice versa. At the net level, an SRN substantially improves the modelling power of the GSPN by adding guard functions, marking dependent arc multiplicities, general transition priorities, and reward rates.

HLF V1.0+ adopts a highly modular architecture design by decomposing the transaction process into three main stages as shown in Fig. 2.6. They can be also refined into five phases, namely HTTP, endorsement, ordering, validation & committing, and response [134]. HLF's modular design makes it possible to separately build a model for each phase and then cascade them to analyze the performance from the net/system level. There have been two studies on HLF performance analysis using GSPN [134] and SRN [135], respectively. Both follow these general steps: 1) clarify transaction process steps and the business logic behind them; 2) create the associated transition diagrams of Petri nets according to the corresponding rules under reasonable assumptions; 3) translate to Markov chains for analytical solutions or directly leverage mathematical tools for numerical simulation solutions. The second step is critical because it bridges the real system to an analytical model and paves the way to solutions for the performance indicators such as transaction throughput, latency, average queue length and utilization. Here, we focus on the Petri nets' transition diagrams of the ordering phase from the two reviewed studies, as shown in Fig. 2.9.

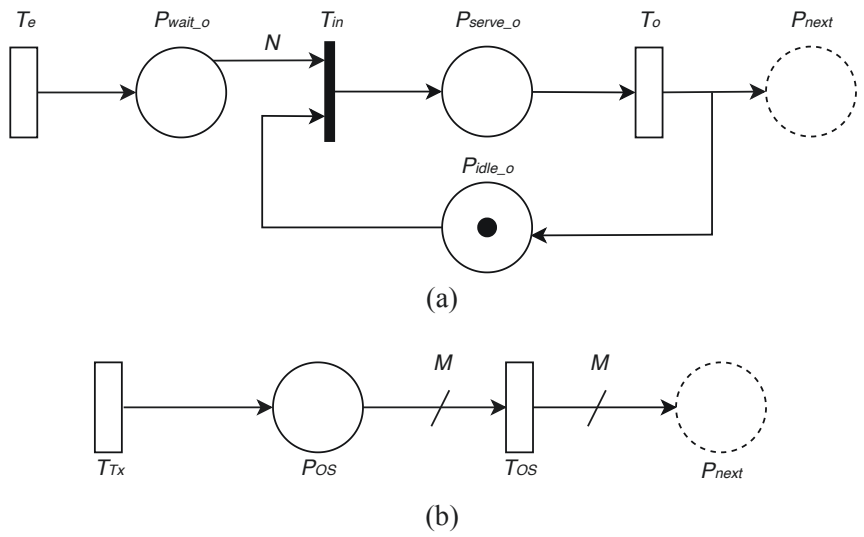


Figure 2.9: SPN models for ordering service in HLF V1.0+: (a) GPSN [134] (b) SRN [135].

In Fig. 2.9 (a), the ordering service starts with taking endorsed transactions as inputs under the assumptions of the exponentially distributed request arrival and constant size of each transaction. The symbols in the figure are interpreted as follows: T_e is a transition signifying the arrival of an endorsed transaction. P_{wait_o} is a place signifying the transaction is queuing, the number of token $\#(P_{wait\_o})$ denotes the queuing length. N is the batch processing size in number of transactions. P_{serve_o} is a place signifying transactions are being ordered. P_{idle_o} is a place signifying the server is idle now, the number of token $\#(P_{idle\_o})$ denotes the number of idle servers. T_{in} is an immediate transition whose enable predicate is $\#(P_{wait\_o})$ ¿ 0 & $\#(P_{idle\_o})$ ¿ 0, which means there are idle servers and queuing transactions. P_{next} is a place signifying the next processing phase.

Similarly, other phases can be modelled by following the same methodology. Consequently, the proposed analytic system model based on GSPN indicates that the HLF system is composed of multiple successive M/M/1 queue networks, and the system throughput is equal to the lowest throughput of all those phases. Using a tool embedded in Matlab named *pntool*, this system can be numerically solved to determine the latency and throughput.

The second part of Fig. 2.9 describes a simple SRN model for HLF ordering service in a network with one client, two endorsers and one peer running the validation logic. After the client receives a response from both endorsing peers, it sends the endorsed transaction to the ordering service (transition T_{Tx}), specified by a token deposited in place P_{OS}. When the number of pending transactions reaches block size (denoted by M) or block timeout for general, a number of transactions are ordered into a block (transition T_{OS}). The block is delivered to the committing peers (place T_{next}) for validations, such as VSCC validation and MVCC validation. Finally, all successfully validated transactions in the block are recorded into the local ledger. As for solving this SRN model, one can use the simulation approach called Stochastic Petri net Package (SPNP) [151] to numerically find answers for the following performance metrics.

- *throughput:* corresponds to the rate of each transition, using function *rate()* in SPNP to capture. E.g., the rate of transition T_{Ledger} signifies the block throughput of the system, which can be used to multiply by M to obtain transaction throughput.

- *utilization:* computed by the probability that the corresponding transition in SRN is enabled, using function *enabled()*, or computed using reward functions for transitions with function-dependent marking rate (such as TVSCC).

- *mean queue length:* obtained by the number of tokens in the corresponding phase, using function *mark()*. E.g., the mean number of tokens in place P_{OS} indicates the mean queue length at the ordering service.

2.3.4 Other Models in DLT Performance Modelling

Besides the stochastic models described earlier, there have been other analytical models proposed for analyzing blockchain performance. For example, a prediction model [136] derived from the core Ethereum's structure called *World State* was proposed to provide companies with a more accurate estimation of performance and required storage. By analyzing the modified Merkle Patricia tree (MPT), which is the implementation of the

World State in Ethereum, the expectation and the max tree height were derived as a function of the total number of transactions n. These results linked to the performance and storage, which were meaningful for decision making and early warnings. Another study [137] adopted *stochastic network models* to analyze the overall *block generation rate* for the PoW-based Ethereum. Through this model, the blockchain evolution and dynamics can be captured and used to analyze the impact of the block dissemination delay and hashing power of the member nodes on the *block generation rate*.

Random graph (also called Erdős-Rényi model) is a powerful mathematical tool first introduced by Erdős [152] and Rényi [153] to model and analyze complex networks. It has properties suitable for modelling the peer-to-peer overlay networks used by blockchain systems [138]. There are two main variants of the Erdős-Rényi model. One of them is $G_p(N)$, which is a graph constructed by randomly connecting nodes. Each edge is included in the graph with probability p independent from every other edge. Shahsavari *et al.* [138] presented a random graph using $G_p(N)$ to model the Bitcoin blockchain network, where N is the total number of nodes, and p refers to the independent probability that there exists a link between any two observed nodes in the peer-to-peer overlay network. Based on the well-established random graph analysis results, some key performance measures can be derived in terms of *block dissemination delay* and *traffic overhead*.

2.4 Findings and Suggestions for Future Research

In this section, we summarize the main findings from previous evaluation sections. First, we discuss the findings from the empirical and analytical evaluations. We then take a look at the performance bottlenecks identified from all reviewed solutions. Finally, we point out some open issues and provide suggestions for future research.

2.4.1 Findings for Empirical Analysis

Performance metrics and workloads: The evaluated performance metrics can be divided into two categories: macro (or overall) metrics and micro (or detailed) metrics. *Macro metrics* provide an overview of the system's performance for users from the application level, such as transaction throughput, latency, scalability, fault tolerance, transactions per CPU/memory second/disk IO/network data. The first two metrics (transaction throughput and latency) are evaluated most frequently, over all blockchains. *Micro metrics* depict the performance of different subprocesses of transactions or specific layers in the blockchain abstract model for developers, such as peer discovery rate, RPC response rate, transaction propagating rate, contract execution time, state updating time, consensus-cost time, encryption and hash function efficiency. Both macro and micro metrics are evaluated under well-designed workloads.

In blockchain performance benchmarking or monitoring frameworks, these workloads have been designed to evaluate the performance of different layers of blockchain. *Macro workloads*, such as YCSB, Smallbank, EtherId, Doubler and WavesPresale, are designed to evaluate the application layer in blockchain. *Micro workloads*, such as DoNothing, Analytics, IOHeavy and CPUHeavy, are designed to evaluate lower layers of blockchain, including execution, data and consensus layers [26].

In general, there are two popular ways to generate workloads for experiment-based performance evaluation. One is to construct a synthetic application with commonly used functions (e.g., CreateAccount, IssueMoney and TransferMoney), and leverage a client node to send requests of transactions (i.e., implemented functions) to a blockchain system [107]. The other is to leverage HTTP performance testing toolkit for generating requests, for example, using the *loadtest* library of Node.js to specify an HTTP request as a JSON-formated object, and constructing workloads for blockchains as separate JSON objects [106, 113].

Evaluated blockchains: HLF (v0.6 with PBFT and v1.0+ with BFT-SMaRt),

private Ethereum (Geth with PoW and Parity with PoA/PoW) and Ripple with XRP consensus are the most often comparatively evaluated blockchain platforms [26, 100, 107, 109]. Among them, HLF and Ripple can reach 1,000+ TPS within a small network and outperform the Ethereum platforms in terms of throughput and latency, under both macro and micro benchmark workloads. However, because of the underlying consensus algorithms they use, both HLF and Ripple fail to scale beyond a certain number of nodes in the network (e.g., 16 [26] for HLF v0.6). For HLF, it is well-known that BFT-based consensuses (e.g., PBFT and BFT-SMaRt) rely on a leader for processing transactions, which may act as a bottleneck and cause performance limitations. For Ripple, a limited and fixed number of validators receive and process numerous transaction requests, and finally fail to scale when the number of requests goes beyond the capability of the validators. This conclusion is shared by a number of early evaluation studies such as [26, 106, 107, 109]. Between the different versions of HLF, its new release v1.0+ has better performance than v0.6 [90] across all evaluated macro metrics such as execution time, latency, throughput and scalability. In addition, another blockchain proposed for IoT (i.e., Tendermint) outperforms HLF V0.6 and Ripple on both the throughput and the latency [113].

It is worth noting that we did not encounter any improvement solutions such as off-chain, side-chain, concurrent execution and sharding in the evaluated blockchain systems. In fact, many of the proposed solutions only exist at the conceptual stage at the time of writing this survey. Some of them provide a brief comparative evaluation and analysis under a specific use case for the purpose of proof-of-concept, but lack a systematic evaluation in a meaningful manner to demonstrate their effectiveness and efficiency.

Consensus finality: *Consensus finality* refers to the deterministic property of a blockchain where a block is considered confirmed once it is appended to the ledger. BFT-based blockchains are all with consensus finality, while those PoW-based are usually not. This property has a direct impact on the transaction latency. For

example, Bitcoin usually requires six successive confirmations as a secure finality that a transaction will not end up being pruned and removed from the blockchain, which makes the latency reach an unacceptable time of almost one hour. In contrast, HLF with BFT-based consensuses can finalize a transaction within seconds right after it is appended to the ledger. Therefore, BFT-based blockchains have an obvious advantage over PoW-based blockchain systems in terms of performance.

2.4.2 Findings for Analytical Modelling

Performance modelling strategies: Most models neglect information propagation delays in the network and simply collapse the whole network into a single node that provides service to process and confirm transactions. These models are usually queue systems that provide bulk services such as $M/M^B/1$ and $M/G^B/1$ queues. Only a small portion of models consider the system as separate disjoint nodes and take the network latency among network nodes into consideration. They aim to calculate system end-to-end output (e.g., delays) using queue networks or by cascading different queues such as M/G/1 and M/M/1 together to model the blockchain network.

An (n, k) fork-join queue combines both modelling strategies. It first regards the system as a single server when the system receives a job request. Then, it splits the job into several sub-tasks for independent and simultaneous processes on different network nodes. In the joint phase, process results are collected from different nodes to finish the original job (e.g., block validation).

2.4.3 Identified Performance Bottlenecks

From the perspective of users or managers, performance evaluation results can be used for decision making on blockchain system selection. Developers and system designers, on the other hand, may care more about the identified bottlenecks rather than the comparison results. They can analyze these bottlenecks and propose solutions for further performance optimization. All bottlenecks identified in the reviewed papers

are listed in Table 2.7.

Table 2.7: Identified performance bottlenecks for different blockchain systems

Blockchain	Bottlenecks Identified	Evaluation Approaches	Latest State
Ethereum v1.5.9	peer discovery, transactions propagation, consensus-cost	Monitoring [89]	Unresolved
Geth v1.4.18	consensus protocols	Benchmarking [26]	Unresolved
HLF v0.6.0	consensus protocols	Benchmarking [26]	Unresolved
Parity v1.6.0	transaction signing	Benchmarking [26]	Unresolved
HLF v1.0	endorsement policy verification, sequential policy validation of transactions in a block, and state validation and commit (with CouchDB)	Experimental analysis [92]	Resolved (HLF v1.1)
Byteball	data storage which is a relational database	Benchmarking [84]	Unresolved
HLF v1.0	no parallel transaction processing on the committing peer	Experimental analysis [84]	Unresolved
HLF	ordering service	Experimental analysis [95]	Unresolved
Private Ethereum	module responsible for reading and writing data	Experimental analysis [108]	Unresolved
Private Ethereum	consensus mechanism	Experimental analysis [109]	Unresolved
HLF	consensus mechanism	Experimental analysis [109]	Unresolved
HLF v1.0+	transmission from client to the ordering service and ledger write	Analytical modelling [135]	Resolved
HLF v1.2	committing phase if the number of transactions in a block is small and endorsement phase if it is large	Analytical modelling [134]	Unresolved

As we can see, most bottlenecks are still unresolved. This means that corresponding effective solutions to solve the performance problems have not yet been found. Another observation is that most bottlenecks are identified by empirical analysis, which can be attributed to two reasons. First, there are more empirical analyses conducted than performance modelling. Second, due to the involved mathematical expressions, analytical modelling is much more difficult than experimental solutions in exploring the impact of design parameters. In blockchain performance modelling, even one simple extra parameter can significantly increase the model complexity. Therefore, empirical analysis becomes more efficient and popular in bottleneck identification than its modelling counterpart.

2.4.4 Open Issues and Future Directions

As a fundamental component of blockchain research, performance evaluation plays an important role in boosting blockchain applications. Although numerous blockchain improvements have been proposed and implemented, only a small number of them have been well evaluated. The evaluation methods also need more analysis and explorations. Here, we identify some open issues and suggest potential directions for future research in this area.

- For empirical analysis, difficulties lie in comparative evaluation among different blockchain platforms, especially for those with very different consensus algorithms and data structures. The main reason is the lack of interface standards in running workloads. For example, when evaluating blockchain platforms for IoT such as HLF 1.0, Ripple and IOTA, it is difficult to design a common interface for uploading workload. Since smart contracts are not supported by Ripple or IOTA, one solution is to design an equivalent workload such as transferring a unity amount from account A to another account B [106]. However, this approach has limited extensibility, and requires to deploy a dedicated workload for each blockchain under evaluation. Thereby, there is a great potential for future research to develop more extensible tools for comparative evaluation of blockchain platforms.

- Many methods of experimental analysis rely on RPCs to communicate with blockchain consensus nodes and collect transaction statistic data (e.g., the total number of confirmed transactions of certain duration). Although the RPC API protocols (e.g., gRPC and JSON-RPC) claim to be efficient, they still induce extra overhead onto the consensus peers [89], which is counted as the peer consumption and in turn makes the evaluation results inaccurate. Therefore, a more light-weight and low-overhead data collection approach, such as log-based approach [89], deserves more attention in the future.

- RPC methods are widely used for data collection in empirical performance evaluation of blockchain systems. For micro metrics and micro workload design, it is challenging to decouple the impact from other layers. For example, two queries on transaction values are designed to evaluate the *data model* performance. For Ethereum, both queries can be easily implemented via invoking JSON-RPC APIs. However, for HLF, a chaincode (VersionKVStore) must be implemented as there are no APIs to query historical states in the system. Inevitably, this involves execution of a smart contract making the evaluation inaccurate by adding extra overhead. Therefore, for detailed evaluation of performance metrics in specific blockchain abstraction layers, it is an open issue to design a reasonable workload that alleviates the impact of other layers and improves accuracy.

- Besides the classic blockchain systems such as HLF and Ethereum, there is an urgent need for evaluating the performance of their proposed improvements. For example, *sharding* claims to be a promising solution and has been implemented in many blockchains. However, there is no evaluation work for comparing different shard-based blockchain systems. Different solutions, such as sharding v.s. DAG and off-chain v.s. side-chain also need to be comparatively evaluated. In addition, it would be beneficial to combine empirical and analytical approaches in blockchain performance evaluation in the future.

2.5 Conclusion

As blockchain has matured to receive more and more attention, its performance problems (e.g., low throughput and high latency) have became critical. To resolve these issues, there have been many improvements proposed, from system level optimization to new efficient consensus protocols. However, such blockchain modifications need to be evaluated in a meaningful manner to demonstrate their performance advantages. In this work, we present a systematic survey covering existing blockchain performance

evaluation approaches. From the high level perspective, they can be categorized into *empirical* and *analytical* evaluation methods.

The empirical analysis can be further divided into four groups: performance benchmarking, monitoring, experimental analysis and simulation. Three popular benchmark frameworks (i.e., Blockbench, DAGbench and Hyperledger Caliper) are introduced and comparatively analyzed. Performance monitoring is recognized as the best solution for performance evaluation of public blockchain.

Analytical modelling approaches are more powerful than empirical solutions especially for analyzing the consensus layer of blockchain system. There are three main types of modelling approaches compared in this survey: Markov chains, queueing models and stochastic Petri nets. This comparison can provide directions for selecting blockchain evaluation approach suitable for given purpose.

We also summarized the results of surveyed performance evaluation studies and identified the bottlenecks of major blockchain platforms. The survey concludes with identification of open issues and ascertainment of future research directions in this important area.

Chapter 3

Performance Analysis of the IOTA DAG-based Distributed Ledger

Distributed ledgers provide many advantages over centralized solutions in IoT projects including but not limited to improved security, transparency, and fault tolerance. In order to leverage distributed ledgers at scale, their well-known limitation, i.e., performance, should be adequately analyzed and addressed. DAG-based distributed ledgers have been proposed to tackle the performance and scalability issues by design. The first among them, IOTA, has shown promising signs in addressing above issues. However, due to the uncertainty and centralization of the deployed consensus, current IOTA implementation exposes some performance issues, making it less performant than the initial design. For example, the conflict resolution that solely relies on a random walk algorithm leads to a linearly increasing number of abandoned transactions and creates the need for a large number of reattachments. In this work, we first extend an existing simulator to support realistic IOTA simulations and investigate the impact of different design parameters on IOTA's performance. Then, we propose a layered model to help the users of IOTA determine the optimal waiting time to resend the previously submitted but not yet confirmed transaction. Our findings reveal the impact of the transaction arrival rate, tip selection algorithms (TSAs), weighted TSA randomness, and network delay on the throughput. Using the proposed layered model, we shed some light on the distribution of the confirmed transactions. The distribution

is leveraged to calculate the optimal time for resending an unconfirmed transaction to the distributed ledger. The performance analysis results can be used by both system designers and users to support their decision making.

3.1 Introduction

Distributed ledger technologies (DLTs), with features such as being decentralized, secure and trust-free, have obtained a lot of attention from both industry and academia to overcome a number of problems in IoT systems. They are essentially distributed systems, acting as a distributed database for storing and sharing data across all nodes in a network. Based on different usage contexts, various types of data, including transaction records (e.g., Bitcoin [154]), contracts [19], and even personal healthcare information [45], can be stored on a distributed ledger (DL) system. According to a recent survey [155], there are about 58 industries (e.g., law enforcement, ride-hailing and stock trading) that could be transformed by DLTs in the future. Clearly, DLT is potentially an effective solution to overcome the data management and security challenges in IT systems. However, various technical details, such as the consensus algorithms and the underlying data structure have a great impact on the performance of DLTs.

Among many different types of DLTs, directed acyclic graph (DAG) is considered to be the answer to the low latency, high throughput and scalability challenges in applications such as M2M micro-payment [20]. Within DAG, transactions can directly be attached to a graph without waiting to be wrapped into a block like in a standard blockchain system. Moreover, all newly added transactions can simultaneously run on different chains, making the performance much higher than a single chain. By contrast, traditional blockchain systems, such as Bitcoin [154], Ethereum [156] and Hyperledger [93], first need to put transactions into a container called *block*, and then linearly process the block; this will result in low performance. For example, in Bitcoin, it takes on average 10 minutes for a transaction to be confirmed. For the sake of

security of large transactions, it is usually recommended that the merchants wait for confirmation of at least six blocks, which implies one hour to complete a transaction. To avoid forking, the block generation rate is limited, thus dramatically limiting the transaction throughput, e.g., 7 transactions per second (TPS) in Bitcoin [154] and 20 to 30 TPS in Ethereum [156], which can hardly support the needs of IT systems in practice. Therefore, DAG-based DLs under well-designed consensus are theoretically more performant than traditional blockchains.

From the perspective of quality of service (QoS), service-level agreement (SLA) and blockchain as a service (BaaS) [157], the transaction throughput, average waiting time (transaction delay) and system scalability are extremely important for a DL system and user experience [96]. This is because these metrics reflect the transaction processing capacity, usability and extensibility of a DL system. In particular, transaction throughput, expressed as TPS at the network level, refers to the rate at which valid transactions are stored by the ledger in a defined time period [158]; transaction delay describes a network-wide view of the time taken for a transaction to be valid across the network since it has been issued [158]; and system scalability determines if the ledger can handle a growing amount of TPS as more resources (e.g., CPU, RAM, or nodes) are added to the existing network. As a counterpart to the traditional blockchain, the DAG-based IOTA claims to be scalable and to provide high throughput because of its innovative data structure and efficient consensus design. In our previous work [159], we designed an energy transactive system for smart communities using IOTA and explored the system scalability. Based on our experiments and analyses, we showed that the proposed system was scalable and effective for IoT use cases. In this work, we focus on the system performance, including throughput and transaction reattachment waiting time, of DAG-based DL in the same IoT scenario presented in [159].

Here, *confirmation* or *confirmed* means that the transaction gets enough proofs or validations to be trusted and included in the ledger. This is the target state for every normal transaction. *Validation* is an examination process including amount, signature

and time check, which is conducted by peers in a peer-to-peer network. We use the terms *throughput* and *CTPS*; *validation*, *approval* and *reference* interchangeably throughout this chapter.

From a system designer's perspective, it is vital to know the transaction processing capacity of the underlying DLT network. Also, from the users' point of view, it is critical to know about the optimal time that they need to wait before reattaching the transactions, if they have not been confirmed yet. If the waiting time is too short, the premature redundant transactions cause network congestion; if the waiting time is too long, the user experience declines as does the system throughput. Either way leads to a poor system efficiency. In this work, we strive to answer two vital research questions about the performance of a private IOTA network:

RQ1. From the system designer's perspective, which factors influence the throughput of an IOTA system? And how, quantitatively, do they impact the throughput?

RQ2. From the user's perspective, what is the optimal waiting time for confirmation before resubmitting the same original transaction?

To address the above questions, we perform the following steps:

1. Like other empirical analysis approaches [160] [161], we first study the system throughput by leveraging and extending the DAG-based DL simulator for simulating IOTA to identify significant factors such as transaction arrival rate (λ), different TSAs, weighted TSA randomness parameter (α), and network delay reflected by distance (D).

2. To find a pattern or relationship between the throughput and design parameters, we statistically analyze the performance data obtained from different configurations and parameter settings to identify potential influence factors for both

simulations and experiments. Here, experimental data are used to validate the simulation results and then collectively to answer RQ1.

3. We decompose the transaction confirmations into layers to explore the confirmation process in a fine-granular fashion. This way, we have a better understanding of transaction confirmation time with more details on how confirmations are distributed; provided that, we obtain a reasonable estimate for RQ2.

The key contributions of this work can be summarized as follows:

- It describes an extension of the DAGsim simulator [2], a simulation tool of DAG-based distributed ledger protocols, to support the currently running consensus on the public IOTA network.
- It provides abundant experimental evidence on how different design parameters influence the IOTA performance, and fills a gap in existing research on the optimal reattachment waiting time.
- It proposes a layered model to analyze the confirmed transactions' distribution in IOTA, providing a potential approach to investigate other DAG-based distributed ledgers, such as Byteball [24] and Nano [25].

The background information about distributed ledger technologies can be found in Section 1.2. Moreover, a detailed overview of existing work on DL system performance evaluation, including simulation and analytical models can be found in Chapter 2. The remainder of this chapter is organized as follows. Section 3.2 presents the background information on IOTA. In Section 3.3, we empirically analyze IOTA's performance through simulations to answer RQ1. In Section 3.4, we propose an analytical layered model to explore the distributions of confirmed transactions in IOTA, and leverage the proposed model to answer RQ2. The experimental results and main findings are also presented in these two sections. Finally, Section 3.5 concludes this work and states some potential future directions of research.

3.2 Background

In this section, we provide a brief introduction on the technical details of IOTA which is the focus of this study. For more information on distributed ledger technologies refer to Section 1.2.

IOTA is a DAG-based open-source value transfer platform designed for the Internet of Things. It is currently the most popular representative of DAG ledgers. This section briefly introduces the basic concepts and consensus mechanism of IOTA.

In the current IOTA network, nodes can be classified into two groups: full node and light node. A *full node* maintains an entire ledger; it receives and validates transactions by running an IOTA reference implementation (IRI) instance in the background. A *light node* sends transaction to a full node for services, acting as a client. There is another special participant called *Coordinator*, which is maintained by the IOTA Foundation and acts as a "finality device" to confirm transactions by periodically generating zero-value transactions, called *milestones*.

All transactions are linked together via a *reference* relationship to form a DAG (see Figure 3.1), called *Tangle* in IOTA. A transaction is the fundamental operation unit that can stand alone. A *tip* is a newly issued but not approved/validated transaction. Each transaction has its own weight and a cumulative weight. The *weight* reflects the computation resource that a sending node puts into this transaction. In order to issue a new transaction, a node must select two tips to validate. Once the validation is finished, this node will attach the issued transaction to the selected tip. This attachment is called a *reference*. At the same time, the newly issued transaction adds its own weight to all the predecessors' cumulative weight. For example, all the referenced transactions' (v_1 to v_9) cumulative weights in Figure 3.1 will add one after v_{10} is attached to the Tangle.

To find a good balance between punishing lazy behavior and not leaving too many tips behind, IOTA recommends a sampling approach named *weighted Markov*

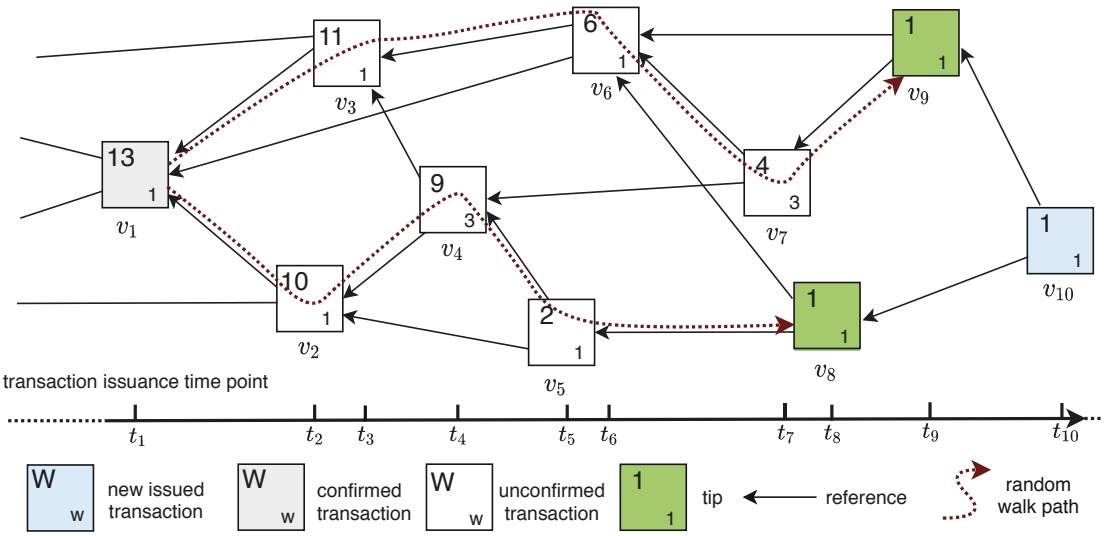


Figure 3.1: An example of DAG in IOTA. "w" stands for weight and "W" stands for cumulative weight.

chain Monte Carlo (MCMC) random walk or *weighted MCMC* to select two tips. For example, the selected tips in Figure 3.1 are v_9 and v_8, with the random walk paths $(v_1, v_3, v_6, v_7, v_9)$ and $(v_1, v_2, v_4, v_5, v_8)$, respectively. This approach leverages equation 3.1 [1] to calculate the probability of choosing the next particular approver in a random walk.

$$P_{\mathrm{xy}} = \frac{e^{\alpha H_{\mathrm{y}}}}{\sum_{z:z \rightsquigarrow x} e^{\alpha H_{\mathrm{z}}}} \tag{3.1}$$

where P_{xy} is the probability to walk from transaction x to y, H_{y} is the cumulative weight of y, and $z \rightsquigarrow x$ means z directly approves x. Therefore, in the random walk step, a transaction with higher cumulative weight has a much higher probability to be selected and approved. In other words, the probability of walking from x to y increases exponentially with the cumulative weight of y, multiplied by α. Here, α is a parameter regulating the strength of the bias in the weighted random walk. A higher α-value means that heavier tips are favoured when attaching new transactions to the DAG. If we set α to zero, the random walk is called *unweighted MCMC*. If we set α to a very high value, we get the super-weighted walk. We can also use a uniform random

tip selection (URTS) approach to select two from all available tips for comparison.

In IOTA, there are two consensus mechanisms: Coordinator and Coodicide [162]. The former is the currently running consensus, while the latter is still under development. With Coordinator, IOTA leverages the above discussed weighted (or biased) TSA to deal with conflicts and simply considers a transaction as confirmed if and only if it is referenced by a milestone. In this consensus, an issued transaction will continuously get approved/validated by peer nodes until it eventually gets confirmed. The more references a transaction gets, the more trustable and acceptable it becomes. As such, it has a higher probability to be referenced (indirectly) by the next coming milestone, and thus confirmed. With Coordicide, IOTA proactively resolves conflicts and reaches consensus through voting mechanism rather than the weighted MCMC TSA [162].

3.3 IOTA Performance Simulation

All research described in this contribution has been conducted within the context of a private IOTA network designed for smart communities. Two approaches, a simulation and an analytical layered model, are proposed to answer two different performance questions, i.e., system throughput and reattachment waiting time (RWT), respectively. In this section, we focus on IOTA performance simulation, while in the next section (Section 3.4), we discuss the layered model to explore the confirmation distributions in IOTA.

To conduct the simulation, we leverage and extend the DAGsim simulator [2], which is an asynchronous, continuous time, and multi-agent simulation framework for DAG-based distributed ledgers. In addition to the analysis of simulation results, we run experiments in a private IOTA network to validate the results.

3.3.1 Performance Metrics

Throughput. The throughput is defined as the confirmed transactions per second (CTPS), which represents the transaction processing power of the system. As for the definition of confirmation, it depends on the consensus used in the system, see Section 3.2 for details. Particularly, we choose COO as the consensus throughout this study to make it more practical, because this is the currently running consensus in IOTA.

Reattachment Waiting Time. Similar to a blockchain system, it may take seconds or minutes for a transaction to eventually be added to the IOTA ledger. In our system, latency is defined as the time between a transaction's arrival request at a full node and its confirmation. Based on this definition, each client wants a lower latency under the same security conditions. Sometimes a transaction may not get confirmed for a long time, causing high latency. It is also possible for a transaction to remain unconfirmed for a long time and be abandoned eventually. In these cases, the transaction should be *reattached* to a new position in the Tangle. *Reattachment* is the process of issuing the same original transaction to a new position in the Tangle, to increase the confirmation probability and decrease the latency. We define the time between two attachments as the *Reattachment Waiting Time (RWT)*. *Reattachment* requires performing PoW and tip selection again for determining the two new tips to be attached. So, too short RWT will not only waste power, but also cause network congestion due to the amount of redundant transactions. On the other hand, a long RWT will dramatically increase confirmation latency and decrease user satisfaction.

To explore the influence of some impact factors to the first IOTA performance metric, transaction throughput, an empirical study is conducted through simulations. These factors include transaction arrival rate λ, TSAs, network latency d and randomness parameter α of the weighted MCMC TSA. For other tested parameters such as network size, PoW difficulty and the number of Coordinators, refer to our previous work [159]

for details. Before presenting simulations, we first talk about the simulation tool.

3.3.2 IOTA Simulator Extension

COO consensus, based on the DAGsim simulator [2], has been developed by the authors to perform the simulations. The original DAGsim only supports consensus without COO, no milestones are generated in between regular transactions. We extended this simulator to support COO consensus. Then, we configured the extended DAGsim to generate and broadcast milestones to the network every 60 seconds, just like the current running IOTA Tangle. The generated milestones are acting exactly like regular transactions, but with the capability to confirm transactions. When we want to check whether a transaction is confirmed or not, we check if it is directly or indirectly referenced by a milestone. Our extended version of DAGsim is available publicly[1].

In the original DAGsim simulator, for each transaction, we only have access to the transactions that are directly referenced by this transaction in the simulation data. However, we also need indirect references when using COO consensus. To fetch this information, we propose a recursive solution named *Indirect References Extraction Algorithm (IREA)* shown in Algorithm 1.

It utilizes a recursive function to find the transactions directly referenced by the input transaction. According to the random walks, we know that there are always two (or at least one if the walks overlap) transactions directly referenced by any issued transaction. These two transactions are added to a list, and the recursive function is run again for each of them. This goes on until the genesis,i.e., the first block, is reached or a transaction that is already in the list is encountered. By running this algorithm, all transactions confirmed by a milestone can be found from the simulation data and, subsequently, the throughput (CTPS) can be calculated.

[1]https://github.com/pacslab/iota_simulation

Algorithm 1: Indirect References Extraction Algorithm

```
1  indirect_references ← empty
2  Function find_references(tx, direct_references):
3      APVD_1 ← The 1st transaction approved by tx
4      APVD_2 ← The 2nd transaction approved by tx
5      if tx is genesis then
6          return
7      else if tx is in indirect_references then
8          Append the new TXs to indirect_references
9          return
10     else
11         if APVD_1 is not in indirect_references then
12             Append APVD_1 to indirect_references
13         end
14         find_references(APVD_1, direct_references)
15         if APVD_2 exists then
16             if APVD_2 is not in indirect_references then
17                 Append APVD_2 to indirect_references
18             end
19             find_references(APVD_2, direct_references)
20         end
21     end
```

3.3.3 Simulation Setup and Results

To collect the simulation data, we run a group of 10 simulations with 6000 transactions, 20 agents, $d = 1, \alpha = 0.001$ and λ varying from 1 to 10 with $step = 1$, see Table 3.1. This is a base-line configuration which we use to explore the influence in comparison with other configurations. Then, we run 5 simulations by only changing λ varying from 10 to 30 with $step = 5$ and transactions from 3,000 to 9,000 with $step = 1,500$ to explore higher rates scenarios. In total, over 90,000 transactions are simulated. In all simulations, λ_{M} is set to be 1/60, i.e. one *Milestone* is issued to the Tangle every minute, because this is the setting in current running IOTA main net; for each simulation, *transactions* is set to 6,000, so that at least 10 *Milestones* (namely 10 replicas) are ensured for each λ configuration. The simulations are conducted on a DELL PC with Windows 10 OS, 8th Generation Intel Core$^{\mathrm{TM}}$ i7-8700 12-Core Processor and 16GB RAM.

Table 3.1: Default parameter configurations

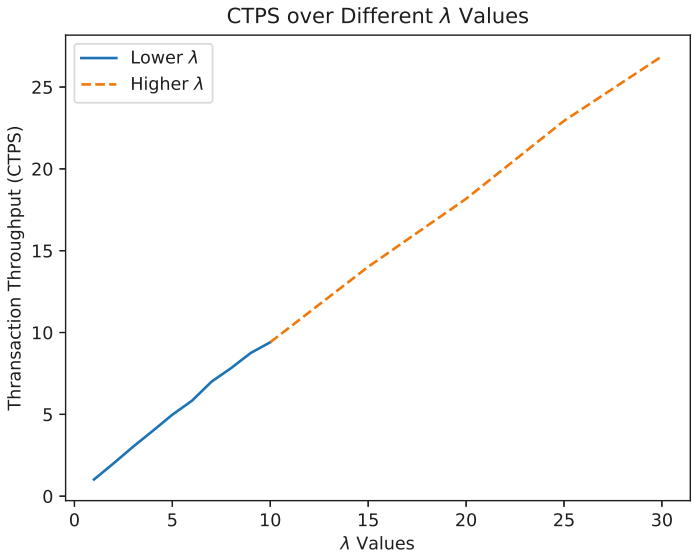
Parameter	Simulation	Configurable	Experiment	Configurable
λ[a]	1∼10	✓	1∼10	✓
α[b]	0.001	✓	0.001	✓
TSA[c]	MCMC	✓	MCMC	✗
d[d]	1	✓	≈1 ms	✗
agents	20	✓	20	✓
transactions	6000	✓	NA	✗
COOTick	60 s	✓	60 s	✓

[a] 1∼10 are explored in both; 15, 20, 25, and 30 are explored in simulation.
[b] 0.001, 0.01 and 0.1 are explored in both simulation and experiment.
[c] weighted MCMC, unweighted MCMC and URTS are explored in simulation.
[d] 1, 5 and 10 are explored in simulation.

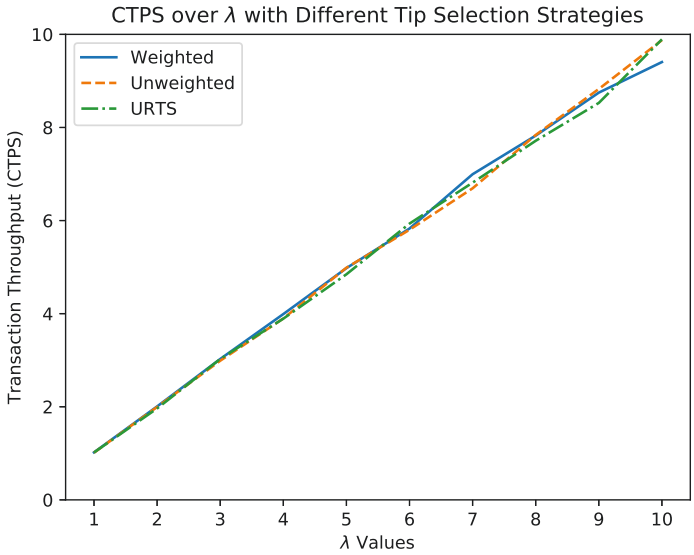
After simulations, the proposed IREA is used to extract the transaction confirmations data and conduct a statistical analysis on the data. The result provides an almost linear relationship between CTPS and λ, as shown in Figure 3.2a, in which all CTPS values are obtained by averaging over all confirmations of 10 replicas.

In order to examine the impact of different TSAs on CTPS, we conduct two more groups of simulations (10 simulations in each group) on the unweighted MCMC and URTS strategies, respectively. In the weighted MCMC random walk, to explore the influence of the randomness factor α on CTPS, we conduct 2 groups of simulations with α=0.01 and 0.1, respectively. In addition, different network delays indicated as distances are examined, i.e., d=1, 5 and 10, respectively. For each group of simulations, the value of λ is varying from 1 to 10 with $step = 1$ (refer to Figure 3.2b, Figure 3.3a and Figure 3.3b).

As can be seen in Figure 3.2b, there are almost no differences for CTPS under different TSAs, i.e., *weighted*, *unweighted* and $URTS$. This is because when α is set to the default value 0.001, there is sufficient randomness in tip selection random walk. Nevertheless, the α values have an obvious impact on CTPS in weighted random walk.



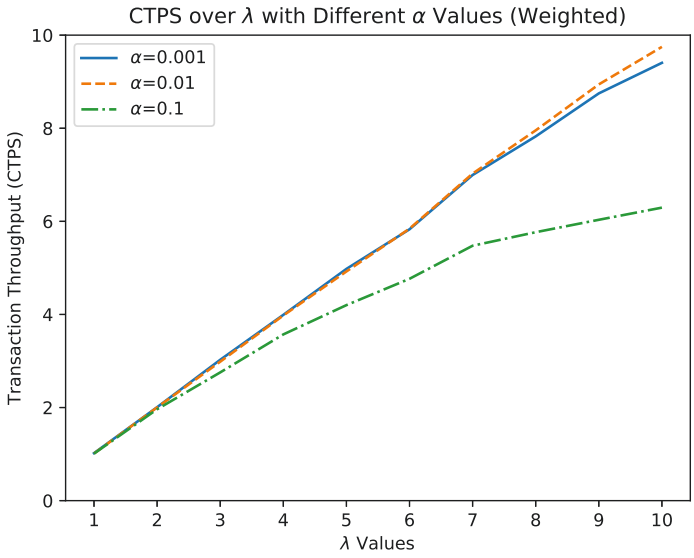
(a)



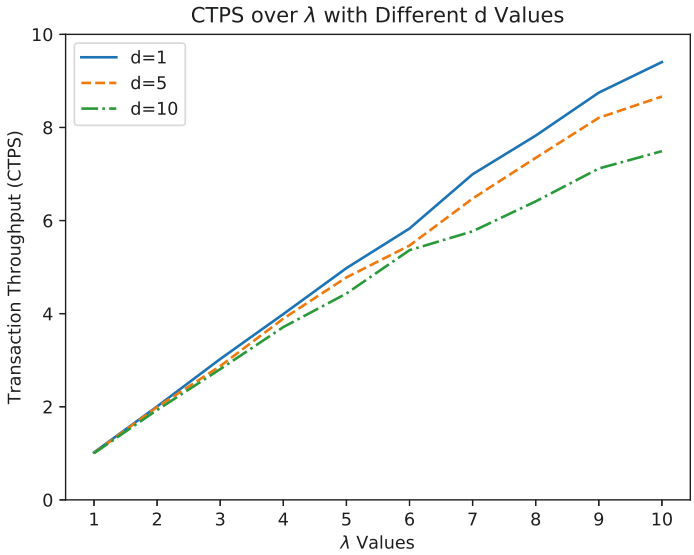
(b)

Figure 3.2: Simulation results: CTPS over λ under influence factors (a) λ only (b) TSAs.

As can be seen from the Eq. 3.1, larger α will increase the probability to select heavier tips so that most new coming transactions will always be attached to the heaviest path. It shows that as α increases from 0.001 to 0.01, there is no big difference on CTPS; but when α raises to 0.1, both CTPS and its increase rate decrease dramatically. In



(a)



(b)

Figure 3.3: Simulation results: CTPS over λ under different influence factors (a) randomness α (b) distance d.

more extreme case, CTPS even keeps flat when α is set to a big value, e.g., 10 [163], which implies a linked chain rather than a DAG in IOTA.

However, different distances which simulate the network delay of IOTA do not tell much difference in general as shown in Figure 3.3b, which means that network

delays have a limited influence on throughput under the examined situations. This can be explained as follows. On one hand, compared to the time spent on PoW and transaction validation, network delay only takes a very small portion of the whole transaction life from being sent to getting confirmed. On the other hand, all examined d values remain quite low, which further weakens their influence on the performance.

3.3.4 Parameter Analysis

To discuss how λ influences IOTA's performance, we model the IOTA system as a single-server network with first-come-first-served (FCFS) servicing policy. The service rate of this network refers to the maximum transaction processing capacity μ, which can be obtained from the load test in practice. In most simulation settings, the actual transaction throughput (denoted as X) keeps a near-linear growth against the transaction arrival rate λ, i.e., $X = k \cdot \lambda$. This trend continues until λ is larger than μ. As transactions are continuously issued and confirmed, the throughput should be the same as the arrival rate when $\lambda < \mu$ in this single-server network model. However, since there is a probability that an issued transaction will be eventually abandoned or orphaned under the random walk TSA, the actual transaction throughput becomes less than λ. In addition, more transactions get abandoned in high load (see Figure 3.2a). Therefore, λ dominates the growth of the transaction throughput when $\lambda < \mu$, while there is an obvious difference between λ and the actual transaction throughput due to the abandoned transactions. This difference is determined by the transaction attachment mechanism, namely TSA in the Tangle.

Different TSAs affect the growth of Tangle through impacting its properties such as the number of tips, transaction's cumulative weight and the time until the first approval [122]. The simplest TSA is the URTS, which selects a tip from the set of all available tips in a uniform random manner. Another strategy is MCMC (see Section 3.2), which is a sampling approach based on a random walk starting from an entry point in the Tangle towards a tip. If the random walk is biased to a more

weighted transaction in each step, we call it *weighted MCMC*; if there is no bias in random walks, we call it *unweighted MCMC*. Since the URTS and unweighted MCMC do not encourage to validate more weighted or honest transactions to build a main DAG, they leave vulnerabilities for attackers to build a parasite chain where double spending attacks can be launched (see [122] for details). Therefore, they are not secure against *parasite chain attacks* [122], and should be only treated as tools to study the Tangle rather than applied in production. To address this problem, IOTA employs the weighted MCMC (see Section 3.2) as its TSA to encourage new transactions verify honest tips and to achieve consensus. The bias level is controlled by a randomness parameter α. Theoretically, the greater the value of α, the greater the probability that the random walk will choose a more weighted tip in each step. Thus, the DAG is more secure by always expanding on the main chain. However, there are more abandoned honest transactions in this case, which will impact the transaction throughput.

We can observe that the Tangle performs almost the same on throughput under URTS and MCMC with $\alpha = 0, 0.01$ in Figure 3.2b, as well as the weighted MCMC with $\alpha = 0.001$ in Figure 3.3a. They all have sufficient randomness to ensure that every tip has a chance to be selected, leaving less abandoned transactions in the Tangle. But, the throughput drops significantly when $\alpha = 0.1$ in Figure 3.3a, where more honest transactions are abandoned because of the biased tip selection strategy. Therefore, it is an interesting research topic to find the optimal α, or propose more TSAs (e.g., hybrid TSA [164] and first order biased random walk [165]) to tackle the trade-off problem between security and efficiency of the Tangle. In addition, the time complexity of URTS is $\mathcal{O}(n)$, while both weighted and unweighted random walk TSAs have $\mathcal{O}(n^2)$ time complexity.

Even though the network delay has a very limited influence on throughput compared to on transaction latency, we can observe a drop on the throughput as the delay increases in Figure 3.3b. In the IOTA network, an IRI node relies on TCP/IP to broadcast transactions and synchronize local ledger to its neighbors. High network

delay will cause asynchronization problem among different local ledgers on the nodes. This asynchronization will further decrease the number of new tips from a delayed node's view. Since the weighted MCMC prefers new tips, more delayed transactions will get older and eventually abandoned as the network delay increases. Thus, the transaction throughput decreases.

3.3.5 Experimental Validation for Simulation

To validate the simulation , we conduct three groups of experiments on a medium-sized network, with the α configurations of 0.001, 0.01 and 0.1. For each α, the total λ values vary from 1 to 10 with $step = 1$, with other parameters remaining default as shown in Table 3.1. We employ IOTA Implementation Reference[2] (IRI 1.6.1) with Docker to deploy a private network of 20 nodes on the SAVI OpenStack cloud platform[3]. Each node is a virtual machine with the flavor of medium size, see Table 3.2 for configuration details. The open-source Compass[4] provided by the IOTA Foundation is used as the COO to generate milestones and confirm transactions in the Tangle. The COO is set to generate a milestone every 60 seconds, just as the simulations.

Table 3.2: Experimental environment

Network Nodes	Number	CPU	RAM	Disk
IRI Node	20	2 VCPU	4GB	40GB HD
Client Node	1	4-Core i5	8GB	256GB SSD

To better control the transaction arrival rate λ and avoid the impact of PoW to λ, we bring all PoW to a PC client with configurations shown in Table 3.2. We run all experiments with the transaction requests in a Poisson process, i.e., the transaction inter-arrival time follows an exponential distribution. The socket ZeroMQ[5] is employed

[2]https://hub.docker.com/r/iotaledger/iri
[3]https://www.savinetwork.ca
[4]https://github.com/iotaledger/compass
[5]https://docs.iota.org/docs/client-libraries/0.1/how-to-guides/python/listen-for-transactions

to listen and receive transaction data. To simplify the problem, we only send zero-value transactions.

In total, 151,104 confirmed out of around 169,701 generated transactions are collected from the three groups of experiments. More details on experimental and simulation results can be found in [166]. Then, we compare the experimental data with the previous simulation data on λ and α to examine the validity and accuracy of the simulation.

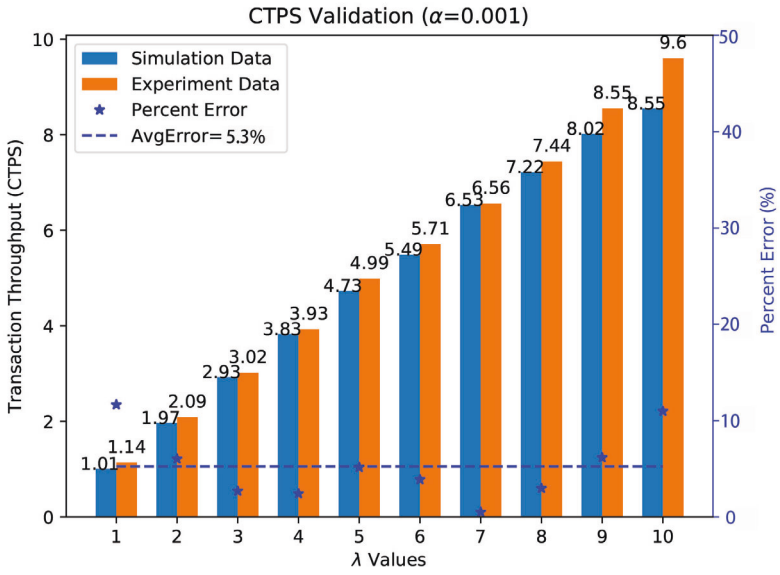


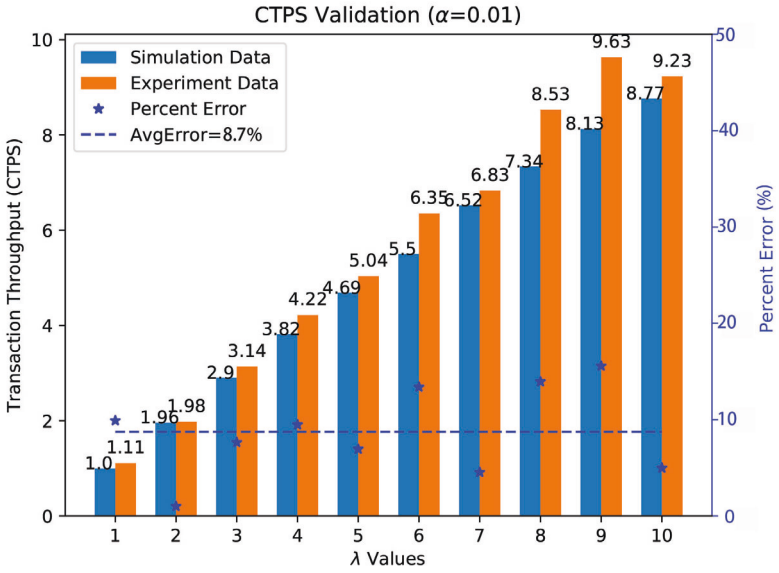
Figure 3.4: Experimental and simulation results comparison: λ only.

First, we compare the CTPS values of simulation and experiment under varying λ 1$\sim$10 by keeping other parameters as default (see Table 3.1) to calculate the percent error. As we can see in Figure 3.4, (1) both the simulation and experimental CTPS results increase almost linearly as λ values increase, implying a good scalability; (2) they match well to each other with an average percent error of 5.3%. Here, the percent error is calculated from the following equation,

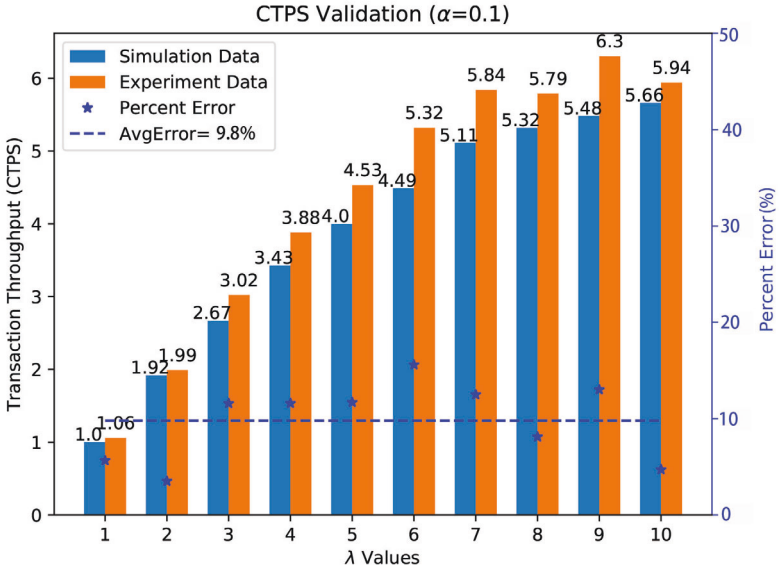
$$Percent\ Error = \frac{|CTPS_{\text{exp}} - CTPS_{\text{simu}}|}{CTPS_{\text{exp}}} * 100\% \tag{3.2}$$

Second, we compare the simulation and experimental results in terms of different

α values. As can be seen in Figure 3.5, simulation and experiment are well in tune for the two values of α. The average errors are 8.7% and 9.8%, respectively. Having the average errors below 10%, we assume that the simulation data can effectively and accurately capture the Tangle behaviour in real world.



(a)



(b)

Figure 3.5: Experimental and simulation results comparison under different α values: (a) α=0.01 (b) α=0.1.

3.4 Analytical Layered Model

The previous simulation explored some CTPS influential factors, identified the most important one and provided a quantitative relationship between CTPS and λ (i.e., linear relationship). However, it is difficult to describe more details such as how these confirmations are distributed in the Tangle, which keeps the second question about reattachment waiting time still unanswered. Therefore, we propose a layered model to explore the confirmation distributions in each single graph layer.

3.4.1 Model Description and Solution

In the IOTA Tangle, we define a layer as all the confirmed transactions with the same depth from a Milestone in a hierarchical architecture, as shown in Figure 3.6. In the case of two different transactions referencing the same transaction with different depths, we take the minimum layer index as the layer depth for this transaction. For example, in Figure 3.6, transaction 5 holds references from both 1 and 4, which are from different layers, $Layer_1$ and $Layer_2$. In this case, we assume that 5 is located in $Layer_2$ rather than $Layer_3$. With respect to this layering decomposition and using previously mentioned simulations data set with over 90,000 transactions, we extract the transaction confirmations number in each layer of the DAG, as shown in Figure 3.6.

After plotting the confirmed transactions over layer number for each λ, we observe a lot of bell-shaped curves, such as the blue dot "Data" curve shown in Figure 3.7, which points us to the nonlinear models fitting, e.g., Gaussian Model. Therefore, after taking the average confirmations of all milestones for each λ, we strive to fit our simulation data as a nonlinear model to characterize the relationship.

In total, for each λ we use 45 nonlinear models to fit our data in CurveExpert[6]. The results show that under all λ values except for λ=1, the Gaussian Model outperforms others and is always listed in top 3 models, as we can see from the example of λ=10 in Figure 3.7. In our fitting, we use the target data set under various λ values by

[6]https://www.curveexpert.net

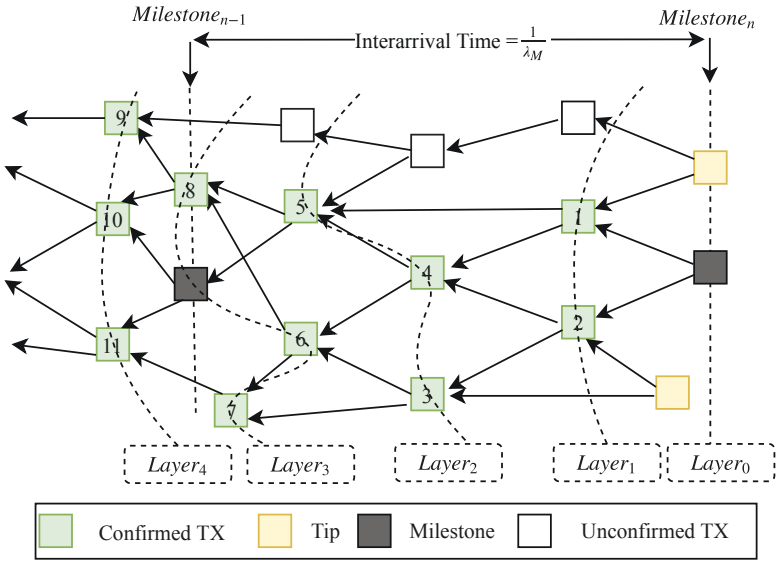


Figure 3.6: Layered model for transaction confirmations.

taking the mean layered transactions of milestone 5, 6, 7, 8 and 9, by getting rid of the potential warming up and cooling down phases. The fitting results of Gaussian Model are listed in Table 3.3.

Table 3.3: Gaussian model fitting results for different λ values

λ	1	2	3	4	5	6	7	8	9	10
Correlation Coefficient	0.934	0.988	0.987	0.982	0.992	0.997	0.984	0.996	0.990	0.991
Standard Error	0.864	0.909	1.274	2.271	1.911	1.476	3.469	2.242	3.733	4.146
a	7.598	16.561	22.439	32.087	40.232	49.378	51.804	66.691	71.021	81.009
b	7.944	8.360	8.976	9.172	9.726	9.390	10.169	9.745	10.298	9.722
c	4.246	3.517	3.742	3.656	3.296	3.298	3.859	3.296	3.536	3.283
AMUB*	13.600	15.000	15.600	16.800	16.400	16.200	17.600	17.000	17.400	16.600
CIUB+	16.265	15.252	16.310	16.337	16.187	15.854	17.733	16.205	17.229	16.156

*Actual Mean Upper Bound, +Confidence Interval Upper Bound

By checking the values of *Correlation Coefficient*, we carefully claim that the mean confirmed transactions located at different layers can be fitted as a Gaussian Model. So, we have the number of confirmed transactions to be a Gaussian function of layer x,

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{3.3}$$

Here, b has an almost linear increase trend as λ increases, while c almost remains the same from our simulation data in Table 3.3. This indicates that all examined Gaussian

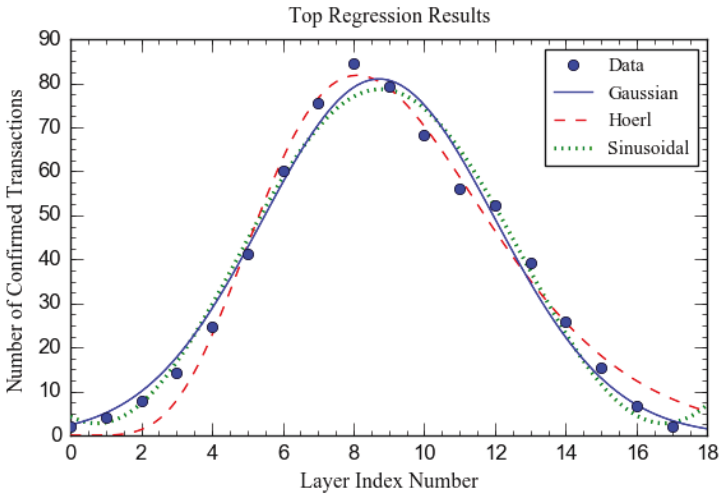


Figure 3.7: Simulation data and fitted models for λ=10.

Models have a very similar shape, and the next random confirmed transaction is expected to be located at a deeper layer in the Tangle with higher λ values. Theorem 1 summarizes our finding.

Theorem 1 *If a transaction remains unconfirmed more than $2/\lambda_M$ seconds after being submitted, the probability that it gets confirmed afterward is sufficiently low.*

Proof. Let layer index be the discrete time dimension, then the *Confidence Interval (CI)* of Gaussian models can be used to estimate the length of time to wait before reattaching transactions. First, let us look at the *Upper Bound* layers in our model. If we take a CI of 95%, the *critical value (Z-value)* for this CI is 1.96, where $(1 - 0.95)/2 = 0.025$. In our case, this means that there is a very small probability (2.5%) for a confirmation to happen after a specific *Upper Bound* layer. The layer of *CIUB* shown in Table 3.3 is calculated by the following estimation formula.

Given that

$$Z = \frac{X - b}{c} \tag{3.4}$$

we have

$$X_{\text{CIUB}} = Zc + b \tag{3.5}$$

Then, we translate the *Upper Bound* layer to the time dimension by analyzing the layered model. As we notice from Figure 3.7, the decreasing happens just after a specific layer. From Figure 3.6, we know that when the confirmation layers of $Milestone_{\text{n}}$ crosses the arrival time of $Milestone_{\text{n-1}}$, there will be a decrease in the number of transactions confirmed by $Milestone_{\text{n}}$ because of the overlap. For example, as shown in Figure 3.6, transaction 10 and 11 will not be counted as confirmations by $Milestone_{\text{n}}$ since they had already been confirmed by $Milestone_{\text{n-1}}$. Therefore, we empirically notice that the peak CTPS layer in confirmation Gaussian Model refers to the previous $Milestone$ arrival time, which is $1/\lambda_M$ seconds ago from the latest one. Thus, the optimal waiting time before reattaching for users is estimated to be around $\mathbf{2}/\lambda_M$ seconds.

3.4.2 Model Validation

To validate the deductive results of our layered model, we use the three groups of experimental data with different α values to conduct a statistical analysis. First, we find out all identical transactions by matching their hash values from the sent and confirmed records, respectively. Then, we leverage the $TimeStamp$ property to calculate the time difference from sending to getting confirmed for each transaction. Since some confirmed transactions recorded in a time period are generated from the previous time segment and cannot be matched within the corresponding sent records, the amount of matched transactions is usually less than all actually confirmed transactions. For example, there are 40,023 sent, 39856 confirmed and 39462 matched transactions when $\alpha = 0.001$.

In total, there are 110,726 confirmed transactions with confirmation waiting time (i.e., they are matched and have the sent-confirmed time difference). Within these transactions, we find that only 4,948 are confirmed after 2 minutes, see Table 3.4.

The detailed transaction confirmation distribution over time under different α values can be observed from Figure 3.8. As we know that the λ_M is set to be 1/60,

Table 3.4: Statistics of confirmations over 2 minutes ($\mathbf{2}/\lambda_M$ seconds)

α	Conf txs	Conf txs(>2 mins)	Percentage
0.001	39,462	2,235	5.7%
0.01	39,355	2,523	6.4%
0.1	31,909	190	0.6%

i.e., a milestone is issued every minute, so $\mathbf{2}/\lambda_M$ seconds are exactly 2 minutes in our experiments. Therefore, we have only 5.7%, 6.4% and 0.6% of the transactions confirmed after the time of $\mathbf{2}/\lambda_M$ for α=0.001, 0.01 and 0.1, respectively, which is relatively low and well matched with the prediction of our proposed layered model.

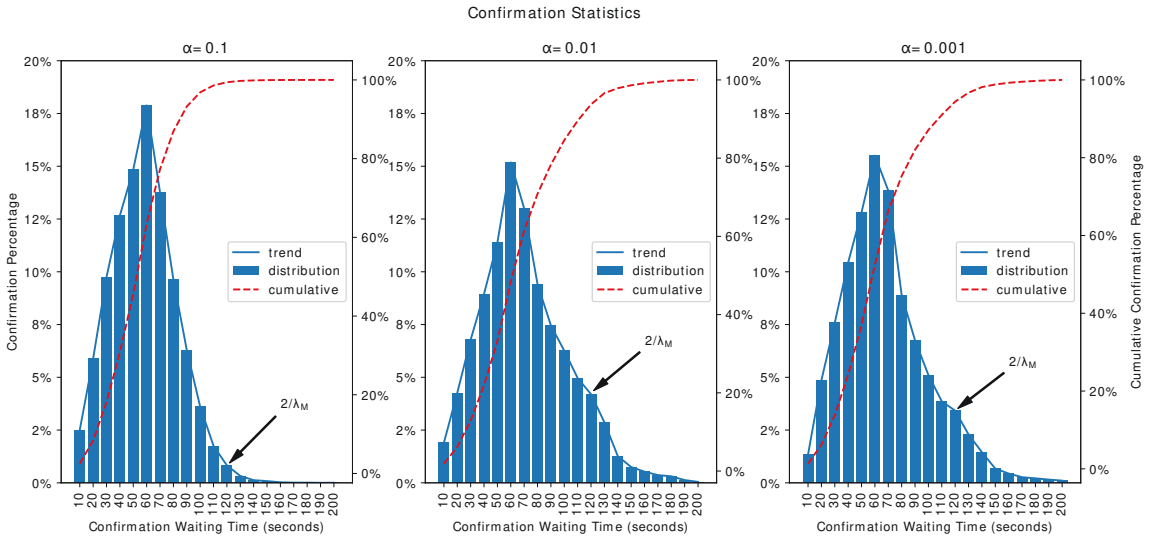


Figure 3.8: Experimental confirmation statistics in different time segments; the confirmation ratios are sufficiently low after the times of $2/\lambda_M$.

3.5 Conclusion

In this work, we have studied the performance of private IOTA network by experiments, simulations and a layered model. We leverage these approaches to answer two research questions on throughput and RWT, with high level of confidence. In particular, we extended the DAGsim simulator and used it to empirically analyze the influence of

transaction arrival rate λ, $TSAs$, randomness α in weighted random walk algorithm and network delay d on CTPS. Among all these impact factors, λ is the most important one, which has a near-linear relationship with the CTPS. Moreover, we leveraged the proposed analytical layered model to explore the confirmation distributions and found that the confirmations are normally distributed in DAG layers, which led to characterizing the Gaussian Model. Using this model, we estimated the RWT for a private IOTA network which was validated by our experimental results. In conclusion, our extended DAG ledger simulator and proposed layered model provide valuable insights into the performance of IOTA distributed ledger in private network scenarios.

Chapter 4

ChainFaaS: An Open Blockchain-based Serverless Platform

Due to the rapid increase in the total amount of data generated in the world, the need for more computational resources is also increasing dramatically. This trend results in huge data centers and massive server farms being built around the world, which have a negative impact on global carbon emissions. On the other hand, there are many underutilized personal computers around the world that can be used towards distributed computing. To better understand the capacity of personal computers, we have conducted a survey that aims to find their unused computational power. The results indicate that the typical CPU utilization of a personal computer is only 24.5% and, on average, a personal computer is only used 4.5 hours per day. This shows a significant computational potential that can be used towards distributed computing. In this work, we introduce ChainFaaS with the motivation to use the computational capacity of personal computers as well as to improve developers' experience of internet-based computing services by reducing their costs, enabling transparency, and providing reliability. ChainFaaS is an open, public, blockchain-based serverless platform that takes advantage of personal computers' computational capacity to run serverless tasks. If a substantial number of personal computers were connected to this platform, some tasks could be offloaded from data centers. As a result, the need for building new

data centers would be reduced with a positive impact on the environment. We have proposed the design of ChainFaaS, and then implemented and evaluated a prototype of this platform to show its feasibility.

4.1 Introduction

Due to the increasing need for more computational resources, many data centers with massive server farms have been built around the globe. Data centers around the world consumed about 416 terawatts of electricity during 2016, which was about 3% of the world's electricity consumption. This amount was also about 40% more than the consumption of the entire United Kingdom in the same year. This energy expenditure is expected to double every four years [167]. In another study, Jones [168] has found the contribution of the data centers in the global carbon emission to be about 0.3%, and the entire information and communications technology (ICT) ecosystem's contribution to be about 2%. It is hard to predict what the future holds, but in a worrying recent study conducted by Belkhir et al. [169], the information and communication industry's global carbon footprint is estimated to reach about 6-14% of the total worldwide footprint by 2040. This calls for new solutions to reduce the carbon emission of this industry.

On the other hand, hundreds of million units of personal computers are manufactured worldwide every year, each leaving their own share of emissions [170]. These computers are highly underutilized both in industries and in households. At any given time, they are either not being used at all or running on a small fraction of their capacity. Based on our survey, which is explained in Appendix A, personal computers run on an average CPU utilization of 24.5%. Moreover, the survey also shows that, on average, personal computers are only being used 4.5 hours per day. These numbers confirm the initial assumption that personal computers are highly underutilized.

The idea behind the proposed approach in this work, called ChainFaaS, is to use the untapped computational power of the current computers as a serverless platform.

In ChainFaaS, regular computer users can rent out their unused computational power by connecting it to a large network of resources. On the other hand, those who need computing resources can rent from this vast pool of compute at scale. If enough personal computers were connected to ChainFaaS, the need for building new data centers would decline. However, the goal in ChainFaaS is not to replace the data centers and servers but to only offload some of their tasks by reusing the idle cycles of personal computers.

Another motivation for creating ChainFaaS is to improve developers' experience of internet-based computing services. Currently, cloud giants, such as Amazon, Google, and Microsoft, are the leading players in serverless computing. Serverless computing platforms offered by these companies are highly centralized, and such companies control every single detail of the platforms. There is no way for the developers to verify the reported billing information. On the other hand, ChainFaaS offers a low-price, transparent, reliable and easy-to-use serverless platform which is not managed by one entity. Anyone can join the network to participate in the management of the platform as well as to observe the transactions. Moreover, since in ChainFaaS, functions run on personal computers, it can be a great platform for edge and fog computing. Developers who need to run IoT applications closer to end-users can use this platform instead of other computing services that run on servers.

For developers, one of the main incentives to use serverless computing platforms is to reduce their costs. In serverless, unlike in other cloud computing solutions, billing is based on the program execution time rather than on the provisioned capacity. In ChainFaaS, one of the motivations is to reduce these costs even more by running the tasks on excess computation power of personal computers.

ChainFaaS is designed with the motivation to provide a reliable and transparent serverless platform. Reliability is achieved through the dispersion of nodes in the network. Since the computing providers can be located anywhere in the world, and a large number of computers are available, ChainFaaS is reliable by design and sheer

scale. Moreover, transparency is achieved through the use of blockchain. Every single transaction on the network is recorded on the immutable ledger of the blockchain. Blockchain peers keep a record of all the changes and every user can easily access this information at any time and verify the transactions.

Rather than being managed by a central entity, the transparent public network of ChainFaaS uses blockchain. At first, this may seem to contrast with the initial motivation of ChainFaaS: to decrease carbon emissions of the ICT ecosystem. The main reason for this apparent conflict is that most well-known blockchains, based on proof-of-work consensus algorithms, require miners to complete computationally intensive tasks. For instance, the carbon emissions corresponding to the electricity consumption of Bitcoin is estimated to be about 22.0 to 22.9 $MtCO_2$, somewhere between the amounts produced by the nations of Jordan and Sri Lanka [171]. However, not all blockchains have such a negative impact on the environment. Many blockchain solutions have been introduced that do not require computationally intensive tasks and instead operate based on other consensus algorithms such as proof-of-stake, delegated proof-of-stake, practical Byzantine fault tolerance, and many others. The blockchain used in the prototype of ChainFaaS is Hyperledger Fabric, which works based on practical Byzantine fault tolerance consensus algorithm. As a result, the use of Hyperledger Fabric-based blockchain is in tune with the design goals of ChainFaaS.

To sum up, ChainFaaS offers an open blockchain-based serverless platform with the following features:

- It is public in the sense that anyone can be either a developer, provider or both.
- It is open and transparent to everyone.
- It is based on the excess computing power available on personal computers.
- It is affordable for developers, especially compared to similar centralized (in terms of management) cloud solutions.

- It is user-friendly and easy to use.
- It embodies edge and fog computing by nature as ChainFaaS can get very close to the end-user.

In this work, our objective is to design a public serverless platform with the above mentioned features, and to investigate the feasibility of the design. We first describe a high-level architecture of the platform and introduce the stakeholders. Then we assess the functional and non-functional properties of the platform and present a more detailed design. To examine the feasibility of the proposed platform, we follow a proof-of-concept approach by implementing a prototype of ChainFaaS. The implementation is based on a microservices architecture to better capture the needs of serverless computing. Finally, the performance of the prototype is evaluated in a series of experiments.

The rest of this article is organized as follows. Section 4.2 explains serverless computing and blockchain as the two main technologies used in ChainFaaS. It also outlines the related work done in this field. Section 4.3 presents the design details and architecture of ChainFaaS. Section 4.4 explains the implementation and deployment details of the prototype of ChainFaaS. Section 4.5 discusses the functional and non-functional evaluation of this prototype. In Section 4.6, the potential threats to the validity of the design and evaluation of ChainFaaS are discussed. Section 4.7 summarizes the conclusions of this work.

4.2 State of the Art

4.2.1 Public-Resource Computing

The idea of using the excess resources of personal computers in a distributed computing platform is not new. The first public-resource computing projects started in the mid-1990s. Great Internet Mersenne Prime Search (GIMPS) [172] and distributed.net [173] were two of the very first project in this area. Both projects started to work on solving

research questions on personal computers and they are still actively running. In 1999, SETI@home [174] was released, attracting millions of volunteers. These are just a few examples of many projects running on personal computers. However, there are only a few frameworks that enable the creation of such projects, and most of them only focus on research projects.

Public-resource computing (also known as global computing) refers to any distributed computing platform that uses the idle processing power of personal devices. Most known public-resource computing platforms are based on volunteer computing, a type of distributed computing in which volunteers contribute their idle computational power to perform computationally expensive research tasks [175]. Because of the heterogeneity of the computers in such a network, a task scheduler is needed to properly distribute the chunks of a research project on personal computers. This task distribution allows a peer-to-peer network to be formed that enables users to share their resources [176]. It is worth mentioning that volunteer computing is different from grid computing. In grid computing, the computing resources are managed and owned by organizations, whereas, in volunteer computing, the resources are highly unreliable and are managed by non-expert individuals.

One of the most well-known volunteer computing frameworks is the Berkeley Open Infrastructure for Network Computing (BOINC) [177], [178], which was first introduced in 2002. Currently, more than 700,000 devices participate in the BOINC network, clearly demonstrating the potential that personal computers hold. This framework consists of a central server and clients run on the volunteers' computers. Any project that wants to use BOINC has to host their own server and run the central BOINC server. This makes it extremely hard for developers to use this platform. Moreover, volunteers can choose the projects they would like to contribute to. As a result, the developers have to ensure that their projects gain enough visibility to attract volunteers. This makes it hard to predict if the project receives enough resources.

XtremWeb [179] is another popular volunteer computing framework, which is

the base of XtremWeb-HEP. As explained in the blockchain-based cloud computing subsection of Section 4.2.2, XtremWeb-HEP is used in the iExec project, which is a blockchain-based cloud computing. In XtremWeb, collaborators can host their own volunteer computing platform that can cooperate with the main Xtremweb server. This framework has three main components: the workers, the client, and the coordinator. The workers are volunteer computers that are responsible for executing the tasks. The client is the entity that submits the tasks to the network. Finally, the coordinator distributes the tasks, assigns workers to the tasks, and keeps track of them.

As another popular distributed computing framework, Cosm [180] provides a set of tools and libraries for distributed applications. This framework was introduced in 1995 and research projects such as Folding@Home [181, 182], and Genome@Home[183] used this platform. Since 1999, Folding@Home project has been running protein folding on personal computers to find how proteins work and to find cures for diseases. Genome@Home designed genes that match existing proteins from 2000 to 2004.

In the recent years, the development of cryptocurrencies has led to some interesting projects that enable payment to volunteers in public-resource computing platforms. Some research projects may choose to reward volunteers for participation in their project. FoldingCoin [184] and CureCoin [185] provide reward tokens for participants of the Folding@Home network. GridCoin [186] offers reward to volunteers in the BOINC platfrom.

Although the mentioned solutions are similar to ChainFaaS, there are some fundamental differences between them. Most of the current solutions are based on volunteer computing. In volunteer computing, the providers do not expect any income for the computations they execute. Moreover, the computing providers trust that the research tasks running on their computer would not tamper with their files and programs. This is mainly due to the fact that the research owners are known universities. On the other hand, in ChainFaaS, there is no trust in the developers, and the computing providers expect the income to be worth their time and effort. Also, in most volunteer

computing platforms, the developers need to host their own servers and promote their projects to gain popularity. This makes it hard for developers to adopt such solutions. Nevertheless, since ChainFaaS is a serverless computing platform, the developers only need to submit their functions to the network through the web application, and the network handles the rest. These characteristics make the design and implementation of serverless platforms, such as ChainFaaS, more challenging. Due to these fundamental differences, these platforms cannot be compared to ChainFaaS directly, especially in terms of performance.

The mentioned drawbacks of volunteer computing platforms have prevented their large-scale adoption. Despite the great potential that platforms such as BOINC have, they are still only being used by a small group of people. As mentioned earlier, the volunteers do not have any economic incentive to participate in the network and the developers have to spend much time modifying their program (i.e. high degree of customization) to be suitable to run on these platforms. Since these problems have been addressed in our design, ChainFaaS has a good potential to receive attention from both developers and providers.

4.2.2 Blockchain-based Cloud Computing

Recently, there have been a number of research activities on distributed cloud computing platforms that run on personal computers. These distributed systems have many management barriers that can be solved using blockchain. In this section, we discuss some of the well-known blockchain-based cloud computing platforms.

The iExec [187] platform is a distributed cloud computing infrastructure, which is based on XtremWeb-HEP [188] and Ethereum smart contracts. XtremWeb-HEP is an open-source desktop grid software that is developed by the same team. The Ethereum blockchain provides the support for distributed applications. For better performance, a new consensus algorithm called Proof-of-Contribution (PoC) is proposed for use in this platform. In iExec, there are three different types of providers: application providers,

computing providers, and data providers. Application providers are developers who want to use the platform to run their distributed applications. Similar to ChainFaaS, computing providers rent out their unused CPU cycles. Data providers can make their datasets available to others and charge them for each use. In this platform, the developers should divide their application into tasks and send them to the scheduler.

Golem project [189] aims to create a decentralized supercomputer. Golem is designed to use excess computational power on PCs to run heavy tasks such as deep learning or video rendering. However, currently, the only tasks that are accepted in this network are rendering tasks. In Golem, developers specify their price suggestions, and providers bid on the tasks. One of the limitations in the Golem network is the fact that computing providers are required to have public IPs, which prevents many from participating in the network.

Another project in this area is SONM [190], which is a decentralized fog computing platform. Similar to the previous two projects, in SONM, people in need of computational power are connected to those with excess computational capacity through a blockchain network. The main focus of this project is on IaaS, and the main applications are machine learning and video rendering.

CloudAgora [191, 192] is an academic project that proposes a blockchain-based platform providing access to storage and computing infrastructure. In this platform, providers bid on any incoming request in an auction-styled manner. The provider that offers the lowest price will be selected to execute the task. The current prototype of CloudAgora is implemented on Ethereum blockchain.

In their paper, Uriarte et al. [193] have surveyed blockchain-based cloud computing solutions and explained the projects in detail. They have also studied the challenges and standards in this field. In another recent research, Yang et al. [194] have studied the challenges and research issues of blockchain and edge computing integration. Westerlund et al. [195] surveyed the advantages and disadvantages of both centralized cloud computing platforms and distributed ledger technologies. They then identified

the principles that a distributed cloud platform should follow to use the advantages of both traditional clouds and DLTs. ChainFaaS's design is in accordance with these proposed principles.

All the projects mentioned above are similar to ChainFaaS in the sense that they use personal computers to run developers' tasks in a blockchain-based network. However, none of them are serverless platforms. The developers have to spend a lot of time modifying their applications for these platforms. In some platforms, developers even have to spend time selecting the provider for their job, which can be inconvenient. Moreover, in most of these platforms, it is hard for the providers to connect to the network. For instance, in Golem, they need to have a public IP, which may not be possible for most providers. In ChainFaaS, these problems are addressed, making the new platform easy to use for everyone. Unfortunately, none of the blockchain-based cloud computing solution providers have released performance analyses of their platform, and they cannot be directly compared to ChainFaaS.

4.3 System Design

4.3.1 High-Level Architecture

Fig. 4.1 shows a high-level architecture of ChainFaaS. This platform has three main parts. The blockchain network consists of all the blockchain peers who are responsible for keeping track of the transactions on the platform. The serverless controller manages the cloud portal and the job scheduling task. The computing resource providers cooperate to shape the execution network.

Blockchain Network

The blockchain network has two main tasks: keeping records of all transactions and managing payments. From now on, we will refer to the record-keeping ledger and payment management ledger as *monitoring ledger* and *monetary ledger*, respectively. Every single transaction is stored and kept by all blockchain peers on the monitoring

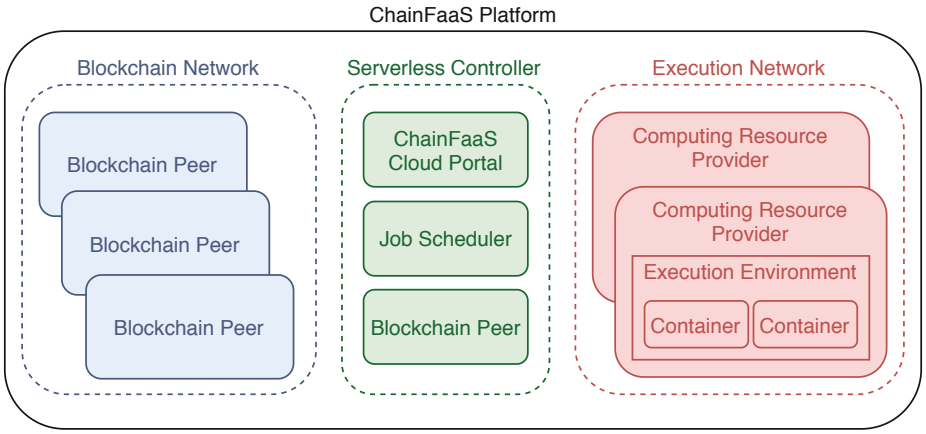


Figure 4.1: High-level architecture of ChainFaaS. The blockchain network stores and confirms all the transactions. The serverless controller is a blockchain peer itself and handles the cloud portal and scheduling the jobs. The execution network consists of all the computing resource providers.

ledger. Any user can easily access the information of a job and see how it has evolved over time. Information such as the owner of the job, the computing resource provider who has been responsible for the job, the time it has taken to run the job, and its cost, are accessible through the monitoring ledger. The transparency feature of ChainFaaS is achieved through the monitoring ledger. The monetary ledger stores the financial account information and account balances of the users. Every time a job is successfully executed by a provider (i.e., a personal computer owner), this monetary ledger automatically transfers the cost from the developer's account to the provider's account.

The blockchain network comprises many peers that can be owned by different proprietors. Each peer keeps records of all the transactions occurring in the network. To add a new transaction to the ledger, the peers should reach a consensus on whether to accept the transaction or not. The peers receive a commission for each transaction because of their contribution to the network. It is worth mentioning that a blockchain peer can also act as a computing resource provider in the execution network, which results in having two sources of income from ChainFaaS. The execution network is described in execution network subsection of Section 4.3.1.

Serverless controller

The serverless controller acts as the gateway and is responsible for providing the portal of ChainFaaS, publicly available here[1]. This portal is designed to help all users easily interact with the platform. Based on their enrolment in the system, users can submit new jobs, observe the status of their job, and monitor their contribution to the network. Moreover, the controller handles the job scheduling by finding a computing resource provider for each job. The job scheduling is based on a selection algorithm that may take into account many criteria to choose the provider, including the provider's availability and the computational power needed for the job. The serverless controller also acts as one of the blockchain peers and, just like any other blockchain peer, it receives a commission for the transaction.

Execution Network

The execution network consists of all computing resource providers. Anyone can easily connect their extra computational resources or their underutilized online computers to this network. This includes but is not limited to, personal computers, servers, and cloud resources. These providers get paid based on the time they spend on running the job to which they are assigned. Each job runs in an isolated execution environment on the provider's computer to ensure that it does not interfere with the provider's programs. This is also required to prevent different jobs from interfering with each other.

4.3.2 Stakeholders of ChainFaaS

Developer

A developer, or software owner, is an individual or a company that wants to submit a function to the serverless platform. In this role, the user wants to use an affordable function-as-a-service system to decrease their infrastructure costs and server

[1] https://chainfaas.com

management overhead.

Computing Provider

A provider is someone who wants to rent out their computer's idle CPU cycles and memory to earn money. The goal of this user is to serve as many jobs as possible to increase their income.

Blockchain Peer

Any computer with public IP address could act as a blockchain peer. The peers are responsible for running and storing the blockchain on their computers. Other blockchain users send requests to these peers to interact with the blockchain. As an incentive to participate in the network, these peers receive a portion of the transactions when they serve requests. In addition, the blockchain peers can also participate in ChainFaaS as computing providers to increase their income from the platform.

4.3.3 Functional Properties

A detailed architecture of ChainFaaS with a complete description of the process of serving a request is shown in Fig. 4.2. The developer is the owner of the function who can have clients sending requests to their function. These clients can be the developer themselves, a program owned by the developer, or anyone else. The developer can also choose to store the result of their function in a separate storage. This can be specified in the container they upload to the system. The result storage block in Fig. 4.2 represents this storage unit.

The scheduler in the serverless controller is responsible for scheduling the jobs, i.e., functions, and computing providers. It creates a new container that includes the function, a monitoring module (MM), and the controller's signature (Sig.). The module is designed to send back the run-time metrics of the job to the monitoring ledger. The signature is used to verify the container's creator to be the controller and not a malicious entity. When an appropriate provider is found for the job, the

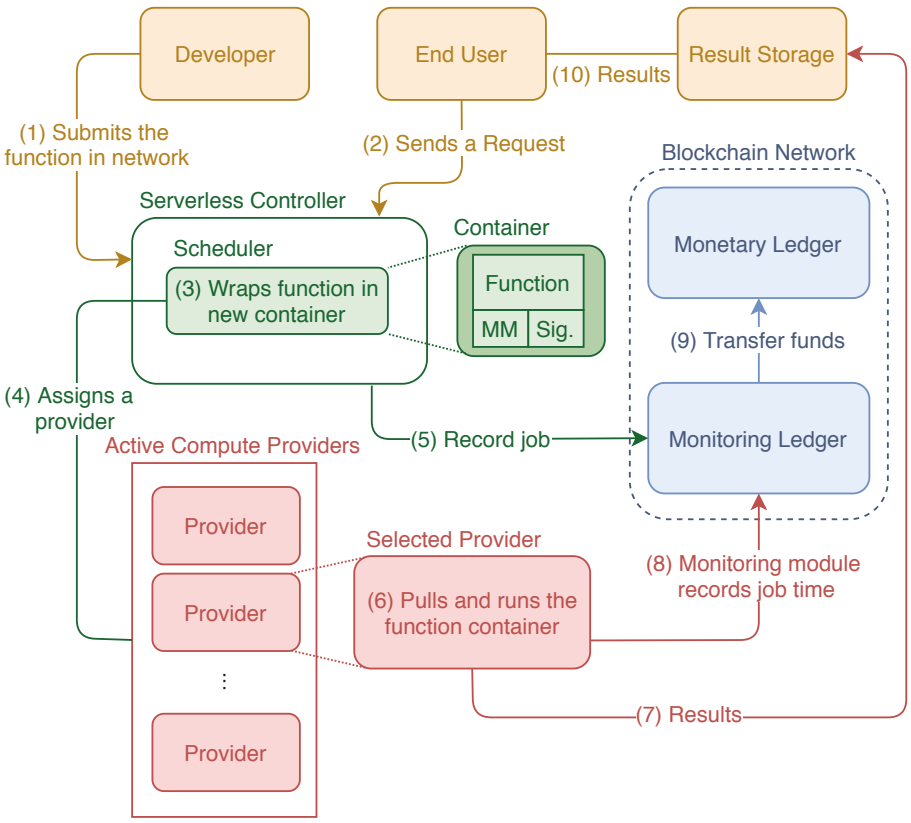


Figure 4.2: The detailed architecture of ChainFaaS and the way a request is served in the network (MM stands for monitoring module and Sig. stands for signature.)

controller asks the monitoring ledger to keep a record of the job. The job ID stored there should be the same as the one the monitoring module later uses to record the job service time. The assigned provider is the only one who can have the job's run time recorded on the ledger. A detailed explanation of each step in Fig. 4.2 follows:

1. For a function to be available in the system, the developer first needs to submit it to ChainFaaS. To do so, the developer sets the access link and characteristics of the function in the ChainFaaS cloud portal. From the moment the function is submitted, others can send requests for that function to the serverless controller.

2. As soon as the controller receives a request for a function, a job is created, and the process starts.

3. The scheduler then adds the monitoring module (MM) to the function and wraps

it in a new container with the controller's signature. The monitoring module is responsible for sending the job processing time to the monitoring chain in step 8. The signature is used in the provider to verify that the job has been sent from the controller.

4. The next step is for the scheduler to assign an appropriate provider to the job. This assignment is done based on a selection algorithm that takes into account the computational capability of the provider and its availability.

5. When a computing provider is found for the function, the controller sends a record-request to the monitoring ledger. In this request, the controller asks the blockchain network to add a new job to their records. The information added to the record includes the job ID, the developer of the job, and the provider assigned to the job. The detailed explanation of the information stored about each job in our implementation can be found in Section 4.4.3.

6. The selected provider then pulls the container from the registry and runs it.

7. In the next step, the provider sends back the results to the target storage.

8. The monitoring module then uses the job ID received from the controller to ask the monitoring ledger to keep a record of the job's run time metrics.

9. In the next step, the monitoring chain charges the developer's account by transferring the amount of bill to the provider's account on the monetary ledger.

10. Finally, the result storage, which is managed and owned by the developer, sends back the results to the end-user. The developer can have the end-user manually get the result from the results storage or have the storage share the results with end-user whenever available.

4.3.4 Non-Functional Properties

This section evaluates the most important non-functional properties of ChainFaaS: performance, availability, security, and usability.

Performance

One of the most important properties of a software system is performance. To understand the performance metrics of ChainFaaS, consider the timelines shown in the following two figures. Fig. 4.3 shows the timeline for submitting a function to ChainFaaS. From the developer's point of view, the time it takes for the function to become ready in the system after it has been submitted is a critical factor. It is called setup time $T_{stp} = t_{rd} - t_{sub}$.

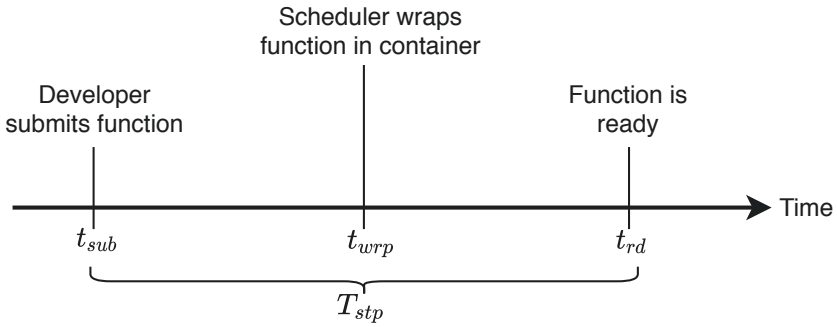


Figure 4.3: The timeline for submitting a function to ChainFaaS from the developer's point of view.

A new job is created in the system as soon as a request for a function is received by the controller, as shown by t_{req} in Fig. 4.4. The time it takes for the result storage to receive the result of the request is called response time $T_{rsp} = t_{stg} - t_{req}$. This time depends on both the request processing time of ChainFaaS and the completion time. The request processing time, T_{prc}, is defined as the time it takes for the provider to receive the job after the controller receives a request for the function. It corresponds to the time the serverless controller requires to schedule the job. The time it takes for the provider to pull and run the function is called completion time, T_{cmp}. This time depends heavily on the size of the developer's container, the network delay, and the

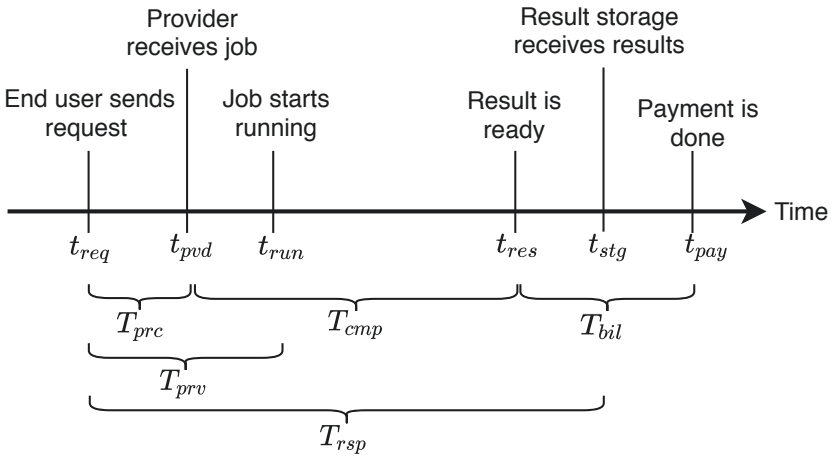


Figure 4.4: The lifetime of a request in ChainFaaS.

job itself. Since response time depends on the function, it can only be measured for a specific workload and not in general. Response times of sample workloads are shown in Section 4.5.

Another important performance metric in this system is the provisioning time, T_{prv}. It is the time it takes for the job to start when a request is received. The provisioning time consists of the request processing time and the container pull time. Finally, from the provider's point of view, the time it takes for them to receive the payment after the job is finished, T_{bil}, is of great importance.

The most important performance metrics of ChainFaaS are summarized below:

- *Setup time (T_{stp})* is the time it takes for the function to become available in the system after it is submitted.

- *Request processing time (T_{prc})* is the time it takes for the provider to receive the job after the serverless controller receives a request.

- *Provisioning time (T_{prv})* is the time it takes for the job to start running after the serverless controller receives a request.

- *Completion time (T_{cmp})* is the time it takes for the provider to pull and run the job's container.

- *End-to-end response time (T_{rsp})* is the time it takes for the result to be available in the result storage after the serverless controller receives a request.

- *Billing time (T_{bil})* is the time it takes for the provider to receive payments after they are finished running the job.

Availability

In a software system, availability is defined as the probability that the user receives a response for their request at a given time. There could be two main causes for jobs to be blocked in ChainFaaS: insufficient computing resources and job queue being full. When a new job is created in the network, there should be an available computing provider who has enough resources for the job. If the scheduler is unable to find a provider, the job request will be blocked.

In ChainFaaS, requests to functions will be queued until a provider is found; requests are served according to a queuing policy. When the queue is full, the serverless controller will block the new incoming requests. If the queue size is too big, some end-users may experience long response times. In such cases, the usual approach is to limit the queue size to prevent that from happening by immediate blocking.

Security

Since ChainFaaS is open to the public, security is of paramount concern. As a computing provider, the user needs to be assured that the code running on their computer is not going to harm their system or access their personal data. An open platform such as ChainFaaS should run completely isolated from the rest of the programs on the provider's computer. For this isolated environment, access to data and information should be restricted. The way ChainFaaS achieves this quality is explained in detail in Section 4.4.

Moreover, all stakeholders are storing important information, such as their account balance, on this platform. Everyone should trust the system to be secure. ChainFaaS

uses blockchain to ensure the security of shared information. Since blockchain is immutable, transparent, and secure, it can be used for this platform. No one, not even the controller, can change the information stored on the blockchain, and every single member can see all the transactions in the blockchain and verify their validity. All these characteristics make blockchain a great solution to open public platforms such as ChainFaaS.

Usability

ChainFaaS is designed to be used by anyone. Although it is a sophisticated platform based on the state of the art technology, the users are not necessarily computer professionals. Therefore, this system needs to be user-friendly. It should be designed in such a way that anyone, including users with no background in computer science, can easily become a computing provider or blockchain peer. To achieve this goal, ChainFaaS has a web portal along with easy-to-setup agents to be installed on computing nodes, as explained in Section 4.4.

4.3.5 Business Model

ChainFaaS has a few stakeholders that may benefit from this platform: the computing providers, the blockchain peers, and the serverless controller. When designing the system, their benefit should be considered an important factor. The computing providers get paid based on the computational power they contribute to the platform and the time they spend on running functions. Since the providers contribute their idle computing cycles, whatever they make is considered profit. Providers can easily connect their computers to ChainFaaS and rent out their excess cycle without any interference in their usual work. As the network extends in size, the providers' profit is likely to increase.

The blockchain peers get a portion of the transaction fees as an incentive for running the blockchain network. The reason blockchain peers are separate from the computing

providers is the necessity to have a public IP for blockchain peers. All the peers should be accessible by a public IP so that everyone can send requests to them. Anyone can own a blockchain peer in the network and help keep the blockchain network secure and transparent.

4.4 Implementation and Deployment

To demonstrate the feasibility of our design, we have implemented a prototype of ChainFaaS as a proof-of-concept. In this section, we present the details of the implementation and deployment of this prototype. Details of ChainFaaS implementation, including the complete code base, can be found on GitHub [2]. Fig. 4.5 shows the implementation and deployment view of ChainFaaS. The end-user represents the person who sends a request to a function. The end-users are FaaS users that can be the developers themselves, or their users who want to access the functions provided by the developer on ChainFaaS.

4.4.1 Serverless Controller

As explained earlier in Fig. 4.1, the controller has three main parts: cloud portal, job scheduler, and blockchain peer. The cloud portal is responsible for all interactions between the controller and the users. At the back-end of the web application, the job scheduler is also implemented. We have implemented a simple scheduler that randomly selects a provider from the available providers that can fit the request. The blockchain peer is just like other blockchain peers described in Section 4.4.3.

In our implementation of ChainFaaS, the serverless controller is running on an instance with 4 VCPUs, 15GB RAM, and 83GB disk with Ubuntu 18.04. The cloud portal is written in Python using the Django web framework [196]. The job scheduler is also implemented in the backend of the cloud portal. Gunicorn is used as the application server, and Nginx is used as a reverse proxy. Gunicorn is a Python Web

[2]https://github.com/pacslab/ChainFaaS

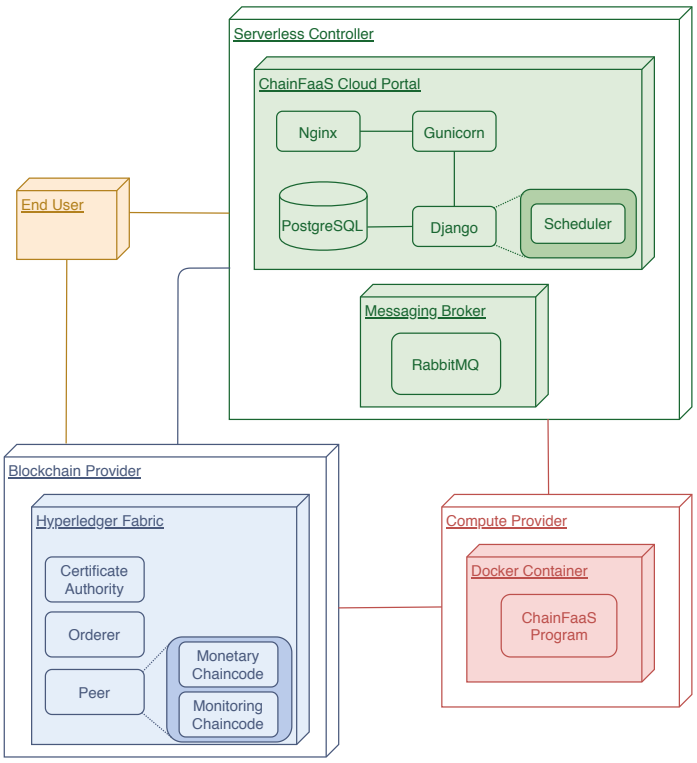


Figure 4.5: Deployment diagram of ChainFaaS showing the technologies and components used in different parts of the platform.

Server Gateway Interface (WSGI) HTTP server that communicates with the Python application. Gunicorn optimally creates as many instances as needed from the web application, distributes the requests between them, and restarts them if necessary. Nginx is used as a reverse proxy to Gunicorn to handle all incoming requests.

New users can use the ChainFaaS cloud portal to register. Each user should select what role they want to play in the platform: developer, provider, or blockchain peer. After registration is complete, the user can start interacting with the platform. As a developer, the user can submit the link to their Docker container in the Docker registry and set its characteristics.

When the scheduler matches a provider with a job, there should be a way for the serverless controller to inform the provider of the link to the function and all its information. Since providers are personal computers, they are not directly accessible by the controller. As a result, in ChainFaaS, a messaging queue has been implemented to

manage the interactions between the serverless controller and each provider. RabbitMQ has been chosen as the messaging broker for this system since it is a lightweight open-source broker that can be customized for different applications. During the registration step, the providers are also registered in the messaging broker to be able to access the appropriate queue. Each provider has its own queue that only the serverless controller and the provider can access. The serverless controller puts the jobs in the provider's queue, and the provider fetches it from the queue.

4.4.2 Compute Provider

On providers' computers, ChainFaaS runs a program that is written in the Python programming language. As long as the program is running on the provider's computer, it receives new jobs from the serverless controller, runs them, and waits for other jobs. Jobs run in isolated environments using Docker containers to keep them from interfering with the provider's computer and accessing their files. In ChainFaaS, having isolated environments for the jobs is particularly important since the provider may not necessarily trust the source of the job. Moreover, there may be more than one job running on the provider's computer at the same time. Using sandboxing solutions, we can ensure there is no conflict between the dependencies and resources of the jobs.

A popular solution for isolating the execution environment of software is using virtual machines. In this solution, a guest operating system runs on top of a host operating system and has virtual access to the system's underlying hardware. Another solution is using containers, which also provide an isolated environment for running a software service. Unlike virtual machines that virtualize the hardware stack, containers provide the developers with a logically isolated operating system by virtualizing the computer resources at the OS-level. As a result, compared to virtual machines, containers are far more lightweight, faster to start, and use far less memory.

In the current implementation of ChainFaaS, the security and privacy concerns of providers are addressed using containers as the sandboxing solution. The reason

for this choice is that the providers should be able to start using the platform with minimal effort and overhead. As discussed earlier, containers are fast, lightweight, and they provide the required isolated execution environment. Other open distributed computing platforms have different solutions to this security challenge. In BOINC [177, 178], the provider can choose to run the project applications under an unprivileged account on the operating system, which cannot access or modify data other than its own. This is called account-based sandboxing. This type of sandboxing is less scalable and secure compared to containers since it does not virtualize at the OS-level. This may not be a problem for volunteer computing platforms since there is some notion of trust between the researchers and the providers. However, in ChainFaaS, the providers do not necessarily trust the developers, and there could even exist a malicious entity in the network. In iExec [187], computing providers, which are called workers, can choose to run the program with a virtual machine, with a Docker container, or just running the script. Golem [189] uses a WebAssembly sandbox for this purpose. Both Golem and iExec are working on using Intel Software Guard Extensions (SGX) for isolation. Intel SGX offers hardware-level isolation, which is the most secure level. However, it is only supported on some modern Intel CPUs. Since this solution is hardware-specific and still experimental, it cannot be extended to all computers. Although beyond the scope of this work, the security and privacy of providers and developers in an open distributed computing platform are important topics for future research.

4.4.3 Blockchain Peer

The blockchain used in ChainFaaS is implemented using Hyperledger Fabric V1.4. When choosing the blockchain solution, one of the most important criteria was its energy consumption. As mentioned in Section 4.1, one of the initial motivations for developing this platform was to decrease the carbon emission of the ICT ecosystem by increasing the usage of personal computers. However, blockchains that use proof of work as their consensus mechanism, such as Bitcoin, require computationally

powerful computers to solve meaningless puzzles to get rewards. On the other hand, in Hyperledger Fabric, there is no need for solving such problems since the consensus algorithm is not based on proof of work. Moreover, Hyperledger Fabric has been designed explicitly for enterprises and takes into account their needs. In Hyperledger Fabric, everything is highly configurable and can be customized for different use cases. These features make it an excellent choice for ChainFaaS. The reason for selecting V1.4 is its stability and long term support. It is the first version with long term support, while V2.0 is still under development.

Hyperledger Fabric uses an execute-order-validate architecture for transactions. In this architecture, transactions are first executed and checked for correctness. Then, via a pluggable consensus algorithm, transactions are ordered. Finally, the transactions are validated against an application-specific endorsement policy and added to the ledger. Other blockchains that support smart contracts, such as Ethereum, Tendermint, and Quorom, use an order-execute architecture in which the transactions are validated and ordered first and only then executed by all peers. In order-execute based blockchains, smart contracts must be deterministic. As a result, these blockchains require smart contracts to be written in a domain-specific language to ensure that their operations adhere to this requirement. On the other hand, in Hyperledger Fabric smart contracts can be written in standard programming languages such as Go or Node.js. Also, since in order-execute blockchains all nodes execute the transactions, these blockchains face performance and scalability issues. Hyperledger Fabric's architecture enables applications to specify which peers and how many of them need to execute the transaction. As a result, only a subset of peers that are specified by the endorsement policy execute the transaction. This feature enables parallel execution of transactions which increases the performance and scalability of the network [197].

The Hyperledger Fabric's network is managed by a collection of organizations that come together to form a blockchain. As a permissioned blockchain, Hyperledger Fabric needs a membership component to overlook the participants in the network. A

trusted Membership Service Provider (MSP) is used for this purpose. Similar to other components in Hyperledger Fabric, the MSP is configurable by the network designer. In ChainFaaS, the default Fabric Certificate Authority (CA) is used as the MSP. Peers are important components of the blockchain network that are responsible for hosting the ledgers and smart contracts. They receive transactions, invoke corresponding smart contracts, and endorse the results. Each peer belongs to an organization and each organization can have many peers in the network. In Fabric, an ordering service is required which is responsible for collecting the endorsed transactions from the peers, ordering them and creating the blocks. The ordering service consists of one or more orderer nodes which reach consensus among each other based on a pluggable consensus algorithm. Currently, Hyperledger Fabric supports Raft, Kafka, and Solo consensus algorithms [197]. The implemented prototype of ChainFaaS uses one Solo orderer.

In the current implementation of ChainFaaS, all components of the Hyperledger Fabric network have been deployed on an instance with 8 VCPUs, 16GB RAM, and 160GB disk with Ubuntu 18.04. There is one Fabric Certificate Authority (CA), one Solo orderer, and two organizations, each with two peers. In the future version of ChainFaaS, there can be included more of these components, each running on different computers. Anyone with a public IP can run the blockchain peers. Each peer stores all the latest information about the ledgers and verifies the new transactions.

We are using chaincodes, which are Hyperledger Fabric's smart contracts, to implement the functionalities needed for ChainFaaS in the blockchain. There are two main chaincodes: monitoring and monetary. The monitoring chaincode is responsible for keeping track of every job that has been served in ChainFaaS. Fig. 4.6 shows the monitoring ledger with an example. Anything that is stored on Hyperledger Fabric blockchain is shown by a key-value pair. In the case of monitoring ledger, the key is the job ID, and the value is its developer, provider, function ID, time, cost, received, and payment-is-done information. The time shows the time the provider spends on running the function, received shows whether the end-user has received the result, and

payment-done shows whether the payment has successfully taken place. In Fig. 4.6, monitoring ledger keeps track of all changes that happen to each job. Consider JOB7 as an example. Right now, the job has finished, but the end-user has not yet received the result. As soon as the end-user receives it, a new transaction is created that consists of the change of JOB7's received value from False to True. The world state database is part of the Hyperledger Fabric structure and stores the latest version of every job. In other words, if anyone follows all the changes in the blocks from genesis to the last block, they will reach the value inside the world state. This implementation makes it easy to query the latest values very fast.

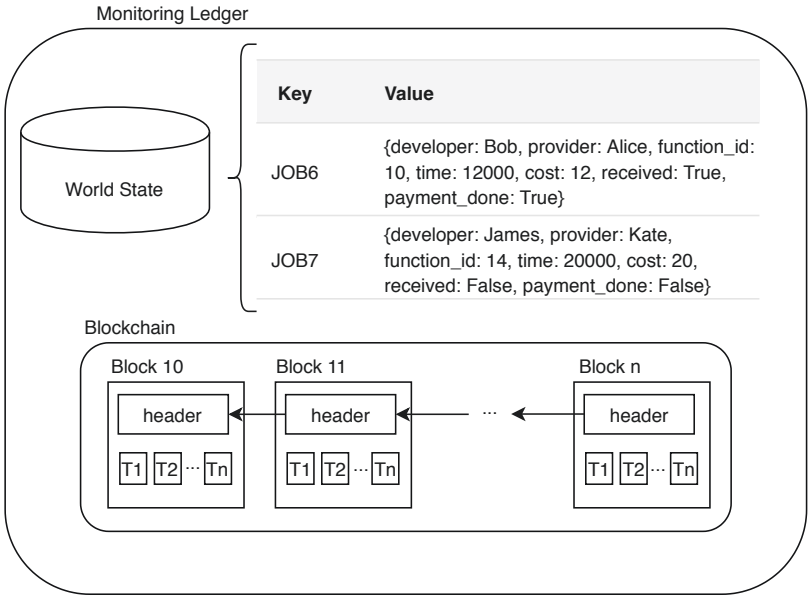


Figure 4.6: Details of the monitoring ledger. Blockchain stores changes to each job in terms of transactions. World state stores the latest version of the jobs.

The monetary chaincode is responsible for keeping track of user account balances. The key in this ledger is the username of the user, and the value is how much they own in ChainFaaS. Fig. 4.7 shows the details of the monetary ledger. Similar to the monitoring ledger, the blockchain keeps track of changes that happen in accounts, and the world state stores the latest version of account balances. To better understand the functionality of the two ledgers, consider the examples shown in Fig. 4.6 and Fig. 4.7.

Imagine that before JOB6 and JOB7, Bob, Alice, James, and Kate all had 600 units of money in their accounts. As soon as JOB6 is finished and received by the end-user, the cost (12 units) is deducted from Bob's account and added to Alice's account. Since the end-user has not yet received JOB7, the account balance of James and Kate has not changed.

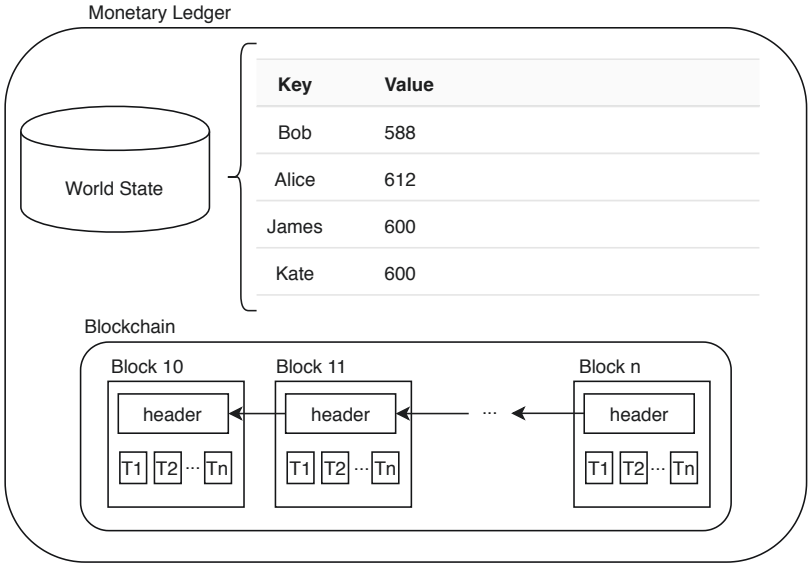


Figure 4.7: Details of the monetary ledger. Blockchain stores changes in each account in terms of transactions. World state stores the latest version of every account.

4.5 Experimental Evaluation

In this section, the functional and non-functional properties of the implemented prototype of ChainFaaS are evaluated. This prototype can be accessed through ChainFaaS's website[3].

4.5.1 Functional

We have discussed how each unit in ChainFaaS is implemented. In this part, we will show how all the units work together to achieve the functional properties of the platform, described in Section 4.3.3. The sequence diagram shown in Fig. 4.8 describes

[3]https://chainfaas.com

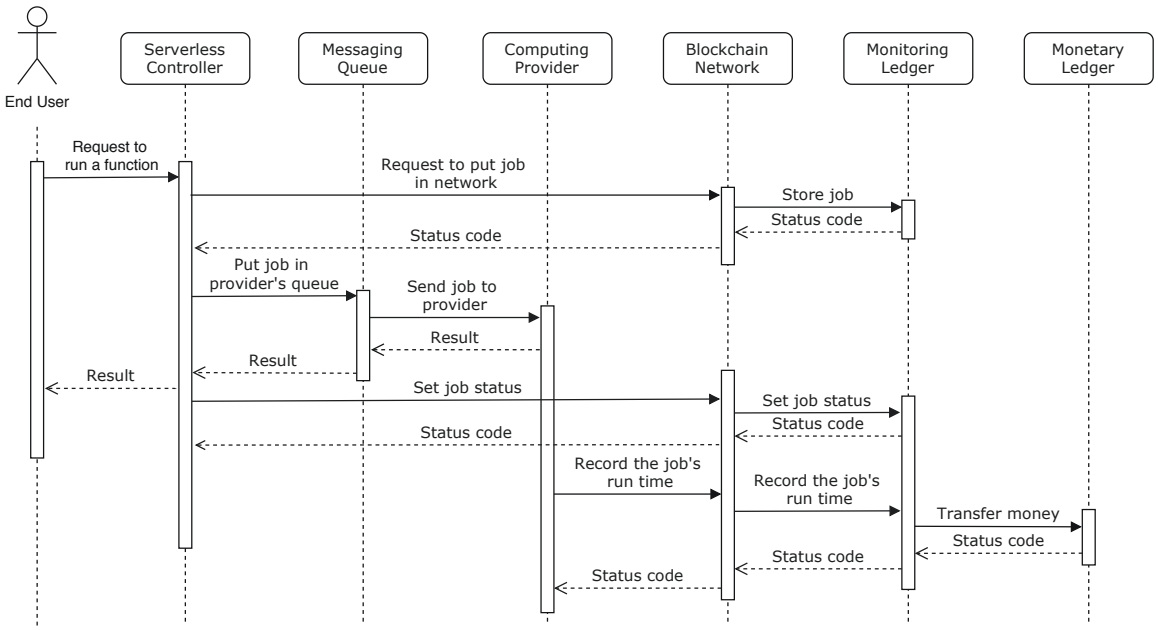


Figure 4.8: Sequence diagram showing the steps of processing a request in the prototype of ChainFaaS

the process in which the prototype of ChainFaaS serves a request from an end-user. In the current implementation, the result storage is the serverless controller.

When the serverless controller receives a request to run a function, the scheduler first finds an active provider capable of serving the request. When a provider is found, the controller asks the blockchain network to store the job on the monitoring ledger. In this step, the job is created in the ledger and information such as the developer, the provider who is supposed to execute the job, and the function ID is set. The controller then puts the job in the provider's messaging queue. The provider fetches the job from the queue, runs the Docker container, and sends back the results to the controller to be accessed by the end-user. As mentioned earlier, the serverless controller is set as the result storage for the prototype implementation. In the future versions of ChainFaaS, the result storage will be chosen by the developer. After receiving the result, the controller confirms that the result has been received by setting the status of the job in the blockchain network. In the meantime, the provider also sets the time it took to run the function. The provider is only able to set the run time if the job has already

been created by the controller in the blockchain network. Moreover, only the provider that has been assigned to the job can set the run time metrics. This ensures that others cannot tamper with the job's information in the blockchain network. When the monitoring chaincode receives both the confirmation from the controller and the run time from the provider, the service fee will be transferred to the provider's account from the developer's wallet.

4.5.2 Non-functional

In this section, we evaluate the non-functional properties of the prototype, most importantly, its performance. Our goal is to shed some light on the system response time for various use cases. This baseline evaluation will help to enhance the platform's performance in future versions.

To evaluate the system, we have selected a sample function to run on the prototype of ChainFaaS. We have created a Docker container that gets the node's information, such as its operating system, number of CPUs, and uptime. Two sets of experiments have been conducted. One focuses on virtual machines running on clouds as providers, and another focuses on personal computers as providers. From now on, we will refer to the first and second experiments as the cloud deployment and the personal computers deployment, respectively.

In the first experimental evaluation, the serverless controller, along with four computing power providers, run on virtual machines in the Compute Canada cloud. The blockchain network runs on a virtual machine on the SAVI testbed cloud. In the second set of experiments, the same serverless controller and blockchain network are used. However, the providers are two different personal computers. Table 4.1 summarizes the characteristics of the ChainFaaS components in the cloud deployment.

The workload generated for the cloud deployment is shown in Fig. 4.9a. Based on a recent research by Shahrad et al. [198], 81% of applications running on Microsoft Azure Functions are invoked, on average, once per minute or less. This information

Table 4.1: Experimental evaluation setup in cloud deployment.

Component	Size	CPU	RAM	Disk
Serverless Controller	1	4 vCPU	15 GB	20 GB
Computing Network	4	2 vCPU	7.5 GB	20 GB
Blockchain Network	1	8 vCPU	16 GB	160 GB

shows that most applications are invoked infrequently, and the generated workload selection, i.e. the number of requests sent to the function, is reasonable for serverless platforms. During a one-hour period, exponentially distributed requests are sent to the serverless controller to invoke the sample function, and the performance metrics of the platform are recorded.

For all experiments, we use a client running on an instance in the Compute Canada cloud with 8 vCPUs, 30 GB of memory, and 186 GB of disk, with less than 10 milliseconds latency to the controller server. A Python 3.7 script sends exponentially distributed requests to a function on ChainFaaS. The results, the request time, and the response time are stored in CSV files, and later processed to extract the response time. On the serverless controller, for each request, two times are stored: the time that the provider has received the job and the time the provider has spent to finish it. Finally, the blockchain server records the billing time for each job.

The most important metric for the end-user is the end-to-end response time, T_{rsp}, shown as response time in Fig. 4.9b. During this time, the scheduler finds a provider for the job, the provider pulls the Docker container, runs it, sends back the result to the end-user, and the blockchain network keeps track of every change in the job's status. As expected, T_{rsp} increases when the number of requests to the function is increased.

As can be seen in Fig. 4.9b, the response time (T_{rsp}) and provisioning time (T_{prv}) follow the same pattern as the processing time (T_{prc}). This behaviour is expected since T_{prc} is included in both other metrics. The completion time (T_{cmp}) contains the

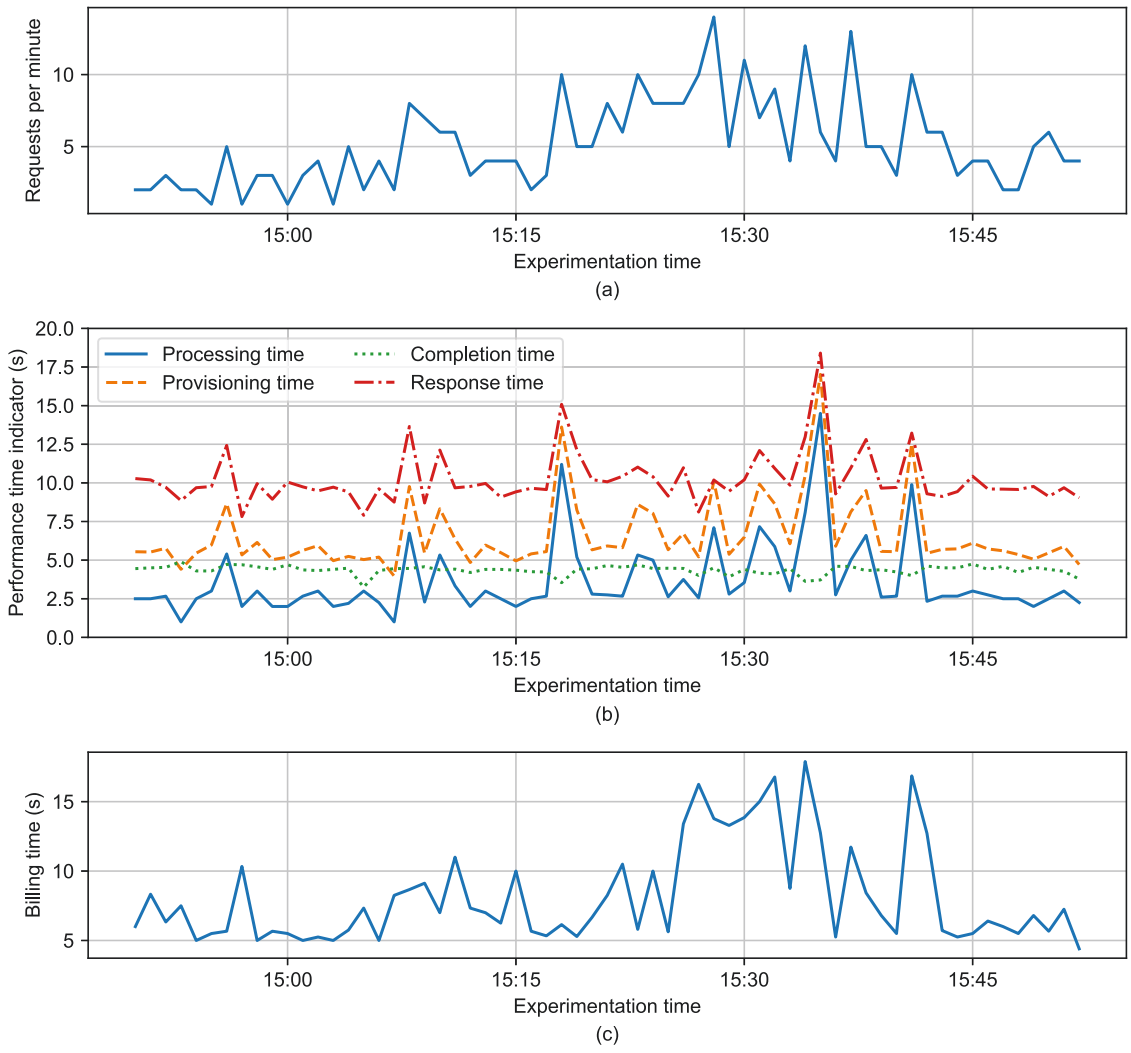


Figure 4.9: The trend of various performance metrics of ChainFaaS in the cloud deployment. (a) Workload generated to evaluate ChainFaaS over time. (b) Average processing time, completion time, provisioning time, and response time; averaged over a minute. (c) Average billing time in a given minute.

pull and run time of the Docker container. Since all four compute providers are in the same network with the same computational capacity, and they run the same function, completion time should remain nearly constant. Fig. 4.9b confirms this assumption.

From the provider's point of view, it is crucial to know the billing time (T_{bil}). Fig. 4.9c shows the average value of T_{bil} in a given minute in the cloud deployment. The billing time usually falls between 5-10 seconds and remains below 20 seconds.

Finally, for developers, it is crucial to know how fast they can deploy a new function.

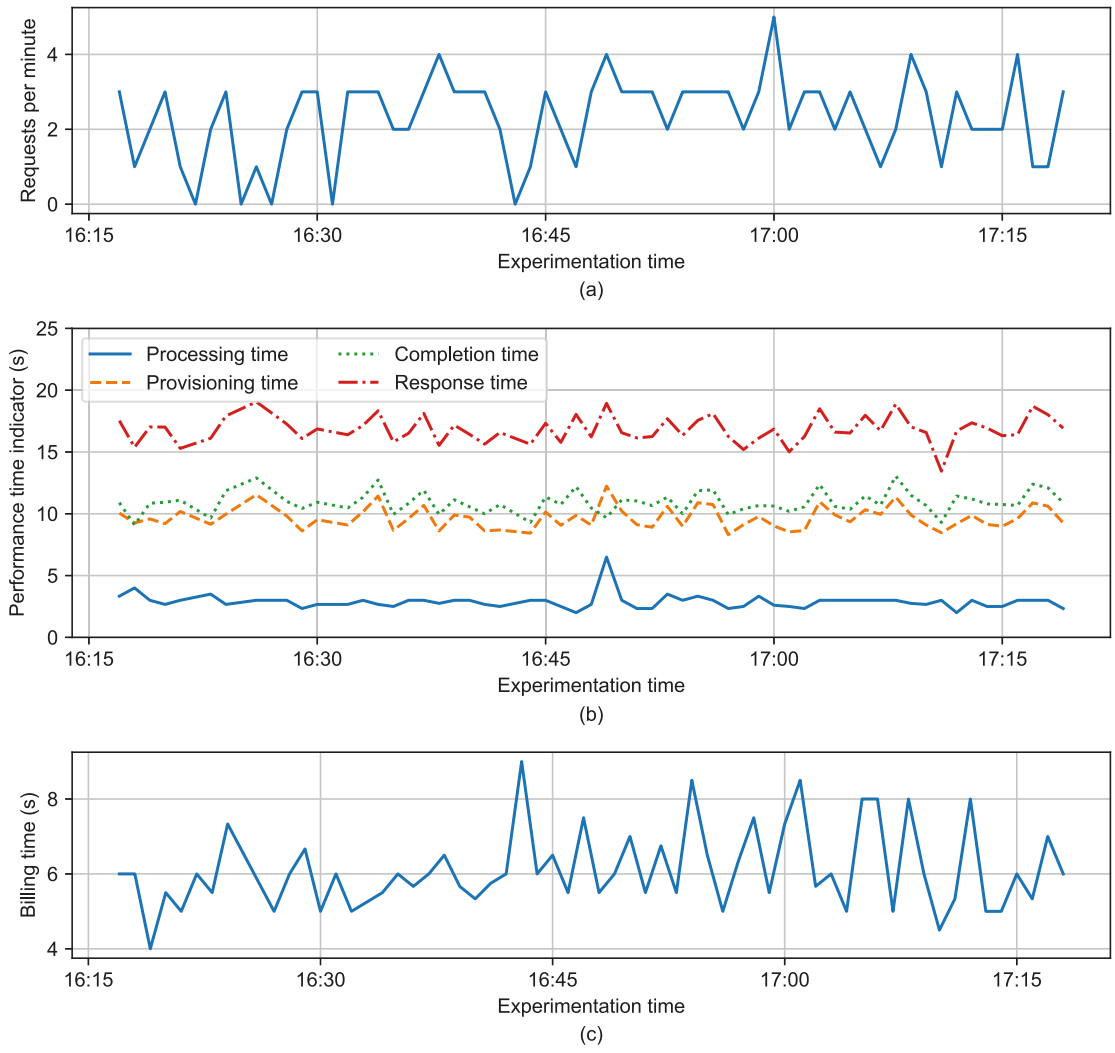


Figure 4.10: The trend of various performance metrics of ChainFaaS in the personal computers deployment. (a) Workload generated to evaluate ChainFaaS over time. (b) Average processing time, completion time, provisioning time, and response time; averaged over a minute. (c) Average billing time in a given minute.

Initially, a developer who wants to submit a function to ChainFaaS needs to register on the platform. During the registration process, an account is created for the user in the blockchain. This process takes about 5 seconds, which is a one-time-only wait. After that, the developer can create and submit a function in a matter of a few milliseconds by providing the controller with the Docker registry path.

In the second set of experiments, we focus on personal computers as providers. In these experiments, the serverless controller and the blockchain network are the same

as the cloud deployment, but the providers are personal computers instead of virtual machines. Table 4.2 shows the hardware specifications of the PCs used in the personal computers deployment.

Table 4.2: Hardware specifications of providers in personal computers deployment.

Component	CPU	RAM	Disk
Provider 1	Core i7-6700HQ	12GB	1TB
Provider 2	Core i7-6700HQ	16GB	1TB

The results of this experiment are shown in Fig. 4.10. It is interesting to see how the two experiments differ from each other. The completion time is almost doubled in the second set of experiments. This is mainly due to networking delays. In the cloud deployment, both the controller and provider were running on the same network (Compute Canada cloud), which makes communications much faster. Moreover, virtual machines running on cloud generally have more stable networks compared to personal computers. The provisioning time and response time are also influenced by the networking delay. The average response time in the cloud deployment is 10.2 seconds. This value is 16.8 seconds in the personal computers deployment.

Since the processing time is not influenced by the network delays, it is expected to be similar in both experiments when running on the same request rate. This expectation is shown to be true in Fig. 4.9b and Fig. 4.10b. The average processing time in personal computer deployment is 2.8 seconds. In the cloud deployment, during the time the request rate is less than five requests per minute, the processing time is around the same value (2.5 seconds). Similar to the cloud deployment, the billing time, which is shown in Fig. 4.10c, for the personal computer deployment, is still between 5-10 seconds.

4.6 Discussions and Threats to Validity

In this section, we discuss potential threats to the validity of the design implementation, and evaluation of ChainFaaS, as well as possible improvements of the platform. Although the implementation and evaluation of ChainFaaS confirm that the proposed platform is feasible, there is still much potential for its improvement.

From the computing providers' and the blockchain peers' point of view, their income from this platform should be worth their time and effort. They also need to consider the increased electricity consumption of their computers resulting from their participation in the network. Moreover, to motivate the developers to switch from public cloud providers to ChainFaaS, the platform should be reliable and cost-efficient. While the initial design of ChainFaaS indicates a sustainable income for the providers and the peers and a low-cost for the developers, a more in-depth cost analysis is needed to confirm this. It is necessary to take into account the cost efficiency of the platform for the developers as well as any possible costs for the providers and the blockchain peers.

Based on qualitative analysis, we have concluded that the use of ChainFaaS can have a positive impact on the environment by reusing the available computational capacity of personal computers. Although this statement appears reasonable, a quantitative analysis of power consumption is needed. This analysis should compare the consumption of public serverless platforms with the prototype of ChainFaaS using different scenarios and report the consumption values. However, this evaluation would be difficult to conduct since we do not have access to the underlying infrastructure of public serverless platforms.

In the current prototype of ChainFaaS, the scheduler randomly selects one of the available providers with enough CPU and RAM for the request. The scheduler can be improved to take into account other relevant factors. For instance, it can consider the reputation of the providers: those with a higher successful job completion score

may get a higher priority in scheduling compared to others. Moreover, instead of having the providers delete the Docker images that they have run, they could cache the images of recent requests they served. The scheduler can then take into account the images that each provider has stored when distributing new tasks. This way, the impact of cold starts will be minimized.

The compute provider's agent has been deployed as a Docker container to prevent the code from tampering with the provider's programs and files. However, there are still some concerns about the security of Docker containers [199]. An interesting research direction on open platforms could be to increase the security for providers while maintaining the performance and usability of the platform.

Any blockchain-based platform faces performance barriers, and ChainFaaS is not an exception. Hyperledger Fabric performs faster than most blockchains, especially those with a proof-of-work consensus algorithm. Nevertheless, as can be seen from the experiments, sometimes it can take up to 20 seconds to complete a task. The blockchain network could be optimized to enhance the overall performance of the platform.

Current implementation of ChainFaaS lacks a policy management component that defines different policies for the system such as fault tolerance, update and change, as well as workload aggregation policies. Since each provider node is an unreliable personal computer, the system's reliability per node is low. On the other hand, since a large number of providers participate in the network, the system achieves reliability by number. A fault tolerance policy is needed to specify what the system should do in case a provider node is unable to respond to a request. Also, an update policy is needed to specify the update and upgrade mechanisms of the system. This policy should include how the nodes are notified of the updates and what they should do if an update is received in the middle of a task. Finally, a workload aggregation policy would be helpful. With such policy, the network can accept large tasks and divide them into smaller subtasks that can run on individual personal computers. In the end,

using the aggregation policy, it could merge the results from different nodes.

In the current experiments, the number of providers in the network, and the number of blockchain peers are fixed. Changing the scale of the system may influence the performance. A possible future research direction is to identify the bottlenecks of the system by changing the parameters and conducting different experiments. Knowing the bottlenecks would help scale the system in the future versions.

4.7 Conclusion

Personal computers around the world are highly underutilized, and their computational power is being wasted every day. We have conducted a survey to gather more details and quantify this information. The results show that the typical CPU utilization of a personal computer is only 24.5% and, on average, a personal computer is only used 4.5 hours per day. Motivated by this, we have designed, implemented and evaluated an open blockchain-based serverless platform called ChainFaaS that uses the untapped computational power of personal computers. This platform can be used by developers to run tasks in a scalable environment with minimal infrastructure management overhead and a reasonable price. Any individual can rent out their extra computational power on ChainFaaS to make a profit. As proof of concept, we have implemented and evaluated a prototype of the proposed platform which is publicly available[4]. Also, the source code, documentations, and user guides are available on GitHub[5]. The prototype uses Hyperledger Fabric as the blockchain solution which enables decentralized and transparent management of the platform. Moreover, to ensure the security of computing providers, this platform runs jobs in isolated environments using containerization techniques. The evaluation process indicates the feasibility of the idea with satisfactory performance.

[4]https://chainfaas.com
[5]https://github.com/pacslab/ChainFaaS

Chapter 5

Towards Blockchain Interoperability Based on the Publish/Subscribe Architecture

Since the introduction of Bitcoin, many studies have worked on using the distributed ledger technology in different use cases and scenarios. This has resulted in many isolated and incompatible blockchain networks around the world. While the development of different blockchain networks shows great potential for DLTs, the isolated networks have led to data and asset silos, limiting the applications of this technology. Blockchain interoperability solutions are needed to enable distributed ledgers to reach their full potential. Such solutions allow blockchains to support asset and data transfer, resulting in the development of innovative applications. In this work, we propose a blockchain interoperability solution for permissioned blockchains based on the publish/subscribe architecture. We implemented a prototype of this platform to show the feasibility of our design. We evaluated our solution by implementing some example publisher and subscriber networks using Hyperledger Besu and two versions of Hyperledger Fabric. We then conducted a performance analysis on the whole network to determine its limits and bottlenecks. Finally, we discuss the extensibility and scalability of the platform in different scenarios.

5.1 Introduction

The distributed ledger technology (DLT) enables a set of independent untrusted nodes to establish an agreement on the state of a shared ledger. Blockchain, a type of DLT, is mostly known for its use cases in cryptocurrencies such as Bitcoin [3], Ethereum [4], XRP[200], etc. However, the technology can be used for many other applications and industries. Some examples are biomedical and health care [7], Internet of Things (IoT) [11, 104], and cloud computing [15, 201]. Due to the fact that each industry has its own unique sets of requirements, many isolated permissioned and permissionless blockchains have been introduced.

Currently, developers should decide which blockchain solution to use for their application, and they cannot use the capabilities of more than one blockchain. These isolated, incompatible networks have resulted in silos of data and assets, which cannot be used from other networks. Blockchain interoperability solutions are needed to enable asset and information transfer from one blockchain to another. However, interoperability for blockchains has some challenges that make it different from interoperability for other software networks. First, the solution should take into account the differences in the architecture of blockchain networks, and it should be technology agnostic. Although all blockchains have an immutable ledger that stores the history of assets, they usually reach a consensus using different algorithms. Moreover, the interoperability solutions should not require changes in the underlying blockchain networks, and it should be usable with minimal effort for existing blockchains.

In this work, we aim to tackle this problem by proposing a blockchain interoperability solution based on the publish/subscribe architecture for permissioned blockchains. In this platform, the goal is to provide a solution for blockchain networks to interoperate with minimal effort. We have implemented a broker blockchain that acts as a middleman in the interoperability process between the source network and the destination network. It is worth noting that since the broker is itself a blockchain network,

it is not a central authority and peers from the source and destination blockchains can also participate in the governance of this network. The broker blockchain keeps a copy of the information that needs to be shared in the form of a topic. A topic has a name, message, publisher, and a set of subscribers. The publisher is the source blockchain network that wants to share the information. The publisher is responsible for creating the topic on the broker and publishing to the corresponding topic whenever the information is changed. The subscribers are the destination networks that need some information from the source network. As soon as the subscriber network subscribes to a topic, the broker network notifies it whenever a change happens to a topic. This solution enables interoperability between blockchains with minimal effort. The source and destination network only need to deploy a connector smart contract on their blockchain networks, which manages the communications with the broker network.

The rest of this work is organized as follows. Section 5.2 summarizes state of the art in blockchain interoperability and blockchain-based publish/subscribe protocols. Section 5.3 presents the design of the proposed platform, as well as its components and message flow. Section 5.4 outlines the implementation and deployment details of each component in the platform. Section 5.5 discusses the performance evaluation of this platform. In Section 5.6, some potential threats to the validity of the design and future research directions are discussed. Finally, Section 5.7 summarizes and concludes this work.

5.2 State of the Art

5.2.1 Blockchain Interoperability

The research on blockchain interoperability has emerged in both the industry and academia. A recent survey conducted by Belchior et al. [202], classifies blockchain interoperability solutions into three categories: cryptocurrency-directed interoperability approaches, blockchain engines, and blockchain connectors. Cryptocurrency-directed

approaches are mostly industry solutions that provide interoperability across public blockchains. This category focuses on asset interoperability and is divided into sidechains, hash lock time contracts, notary schemes, and hybrid solutions. Sidechains offload transactions to a secondary chain, enhancing performance, as well as providing features that the main chain would not be able to provide. Sidechains also allow the representation of a token from the main chain at the secondary chain. Some sidechain solutions include the BTC Relay [203], Zendoo [204], and RSK [205]. Hash lock time contract solutions enable cross-chain atomic operations using smart contracts. Wanchain uses this scheme and provides loan services with cryptocurrencies [206]. Notary schemes are centralized or decentralized entities that mediate token exchange (e.g., cryptocurrency exchanges). Finally, hybrid solutions combine characteristics from previous approaches. The cryptocurrency-directed approaches only work for transferring different types of cryptocurrencies between blockchain network. As a result, these approaches cannot be used for permissioned blockchains with arbitrary assets and smart contracts, which are the focus of this work.

The second category is blockchain engines, which enable the creation of customized blockchains that can interoperate by providing reusable data, network, consensus, and contract layers. Platforms like Polkadot [207] and Cosmos [208] provide such infrastructure, with free interoperability among the instances they allow to create. These approaches are fundamentally different from what has been proposed in this work. Instead of enabling blockchain interoperability for currently running blockchains, blockchain engines propose blockchain networks that are interoperable by design. As a result, these solutions cannot be used for currently running permissioned blockchains.

The blockchain connector category is composed of interoperability solutions that are not cryptocurrency-directed or blockchain engines. They include blockchain agnostic protocols, blockchain of blockchains solutions, blockchain migrators, and trusted relays. Each of these subcategories is designed for a particular set of use cases. Blockchain agnostic protocols enable cross-blockchain communication between

arbitrary distributed ledger technologies, which typically include refactoring and making changes in the underlying blockchains. An example is the solution proposed by Abebe et al. [209], which enables interoperability between Hyperledger Fabric networks using Fabric chaincode and a protocol-buffer-based communication protocol. Blockchain of blockchains are approaches that allow users to build decentralized applications using multiple blockchains. Blockchain migrators enable the state of one blockchain to be migrated to another blockchain. Currently, only simple blockchain migration solutions have been proposed [210, 211]. Trusted relays allow the discovery of the target blockchains, often appearing in a permissioned blockchain environment where cross-blockchain transactions are routed by trusted escrow parties. An interesting trusted relay approach is Hyperledger Cactus [212], the most recent Hyperledger project aiming to connect a client to several blockchains, whereby transactions are endorsed by trusted validators. Cactus focuses on providing multiple use case scenarios via a trusted consortium.

The solution proposed in this work can be categorized as a trusted relay, as it contains a blockchain mediating communication across heterogeneous blockchains [202]. While trusted relays can provide a straightforward way of achieving interoperability, most of them are not trustless (e.g., contain a centralization point). Our solution is a decentralized trusted relay that implements a publish/subscribe system, anchored on the trust that underlying blockchains offer.

5.2.2 Blockchain-based Publish/Subscribe Protocol

The blockchain technology has been applied in the pub/sub paradigm in a few previous studies. However, those studies adopt blockchain to address the existing problems in other areas, such as IoT [213], supply chain [214], multi-tenant edge cloud [215], and digital trading [216].

Huang et al. [215] exploit blockchain technology to enhance the security of pub/sub communications in multi-tenant edge clouds. Mainly topic-based and broker-enabled

pub/sub streaming systems use centralized cloud servers to store sensitive metadata and access control lists, which can compromise the confidentiality, anonymity and integrity of tenants' data. Alternatively, critical data such as ACL and identity information, as well as the hash of raw messages, and operation logs, can be stored on the blockchain to guarantee data security and integrity. Smart contracts also implement access control mechanisms to authorize publishers and subscribers' requests.

Trinity [214] proposes a blockchain-based distributed publish/subscribe broker to solve the existing flows in centralized brokers in IoT and supply chain monitoring applications. Trinity has three main components: blockchain network, brokers, and clients. The blockchain network is responsible for consensus in the system and persistent storage. The broker handles the communications between the blockchain network and clients. The clients are the publishers and subscribers of the topics. The blockchain network is pluggable, and the authors have used Tendermint, Hyperledger Fabric, Ethereum, and IOTA. For the broker, they have used the Mosquitto MQTT broker.

Zhao et al. [217] have proposed Secure Pub-Sub (SPS), which provides fair payment based on the reputation for publishers and subscribers in cyber-physical systems. They use Bitcoin's network to enable payments between the entities, and they propose a reputation mechanism that helps calculate the price of data.

Lv et al. [213] presents a decentralized privacy preserving pub/sub model for IoT systems to solve centralized brokers' problems such as single point of failure, data tampering due to corrupter brokers, and heavy encryption algorithms. The presented model applies public-key encryption with equality test [218] and ElGamal [219] to protect participants' (both publishers and subscribers) privacy. A system prototype is implemented and evaluated against the feasibility of the proposed model.

Bu et al. [216] and Zupan et al. [220] have proposed blockchain-based pub/sub brokers to address the drawbacks of traditional pub/sub systems such as privacy and accountability. However, they have not explained their implementation and evaluation

in their studies.

As mentioned above, all the previous studies exploit blockchain to improve the centralized predicament in traditional pub/sub systems in distinct application domains. Our paper focuses on establishing effective and practical interoperability between multiple permissioned blockchains with different architecture and infrastructure. To the best of our knowledge, our paper is the first study that enhances blockchain interoperability utilizing the pub/sub communication model.

5.3 System Design

In this section, we first discuss the design principles that a blockchain interoperability solution should follow. We then propose our interoperability and explain its components and message flow.

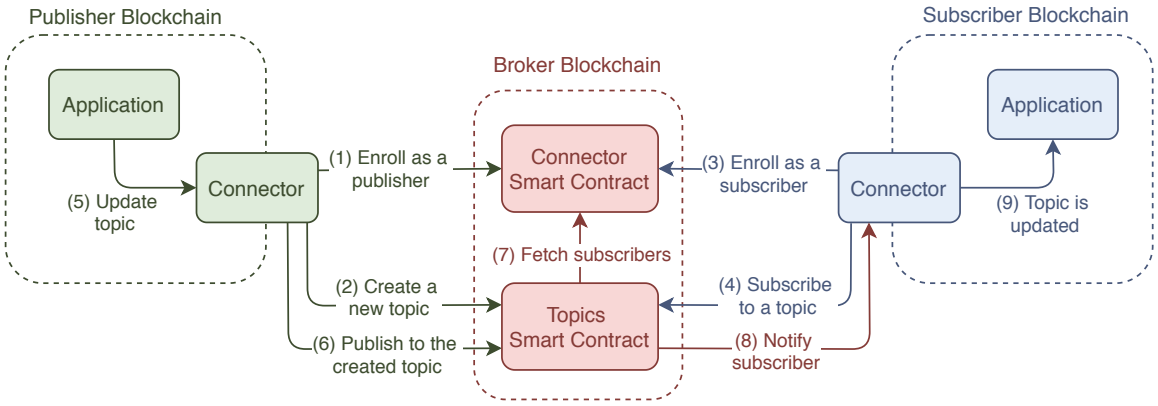


Figure 5.1: Architecture of the platform and the message flow.

5.3.1 Design Principles

Blockchain interoperability aims to enable applications to use the assets and information available on blockchains other than their main blockchain network. This allows for a greater range of applications. A blockchain interoperability solution should take into account the following design principles:

- The blockchain networks are independent, and they may have different architectures.
- The blockchain networks are in full control of their assets and information.
- The transfer protocol should be technology agnostic.
- The interoperability solution should not require significant changes in the source and destination networks.
- The blockchain networks should be able to incorporate the solution with minimal effort.

Following the mentioned principles, we present our solution, which allows interoperability based on a publish/subscribe architecture. Figure 5.1 shows the architecture and message flow of the platform.

5.3.2 Components

The proposed platform in this work aims to solve the interoperability problem in permissioned blockchains using the publish/subscribe pattern. When a blockchain wants to use the data from another blockchain, there needs to be a way to fetch and transfer this data between the networks securely. Moreover, if the data changes in the source network, the destination network should be notified of the change. Figure 5.1 shows the architecture of the platform and the message flow.

The *publisher blockchain* is the blockchain network that sends the data, also referred to as the source network. For a publisher to participate in this platform, it needs to run the appropriate connector smart contract on its blockchain and enroll as a publisher in the broker. The publisher can then create as many topics as they want and use the connector smart contract to publish the changes to the topic.

The *subscriber blockchain* is the blockchain network that received the data, also referred to as the destination network. Similar to the publisher, the subscriber also

needs to run the appropriate connector smart contract and enroll as a subscriber. The subscriber can then subscribe to any topic available on the broker blockchain. Every time the topic changes, the broker notifies the subscriber by invoking the connector smart contract. There can exist many subscribers for a topic.

The *broker blockchain* is the core component of the platform. It is a blockchain network that stores all the information about the topics and the blockchains that participate in the interoperability process. It has two different smart contracts, the *connector smart contract* and the *topics smart contract*. The connector smart contract stores the information about the participating blockchain networks and how the broker can interact with them. The topics smart contract is responsible for storing the information about the topics such as their publisher, subscribers, and the latest message.

5.3.3 Message Flow

The complete interoperation process in the platform is shown in Figure 5.1. For simplicity, only one publisher and one subscriber blockchains are shown in this figure. However, for each topic, we can have an unlimited number of subscribers, and in general, there is no limit on the number of publisher and subscriber blockchains. A detailed explanation of each step in Figure 5.1 follows:

1. For any blockchain network to interact with the broker blockchain, it must enroll in the connector smart contract. During this process, the information that the broker needs to be able to interact with the blockchain is registered in the connector smart contract. As a result, the publisher is required to enroll in the connector smart contract as a publisher. This step only needs to be done once when the publisher wants to create its first topic. It can then create topics or publish to existing ones without the need to be enrolled again.

2. A publisher that is registered in the connector smart contract can create a new

topic. In this step, the publisher needs to specify a name for the topic and the initial topic message. This step only needs to be done once for each topic.

3. Similar to the publisher blockchain, the subscriber blockchain should also enroll in the connector smart contract. This step only needs to be done once when the subscriber wants to subscribe to a topic for the first time.

4. To receive a notification when a topic has been changed, the subscriber should subscribe to the topic by sending a request to the topics smart contract. This results in the subscriber being added to the list of all subscribers for the topic. This step only needs to be done once for each topic.

5. Whenever needed, the application in the publisher blockchain can update the topic by sending a request to the connector smart contract.

6. The connector smart contract sends a publish request to the topics smart contract for the existing topic.

7. As soon as a publish request is received by the topics smart contract, the smart contract fetches the information about all the subscribers of the topic from the connector smart contract. This includes information on how the broker can interact with each of the subscribers.

8. The topics smart contract then uses the data fetched from the connector smart contract to notify all the subscribers of the change in the topic. This happens by sending an update request to the connector smart contract in each of the subscriber networks.

9. In each subscriber network, the connector smart contract receives the update for the topic and notifies the application.

5.4 Implementation and Deployment

An open-source prototype of the proposed platform has been implemented as a proof-of-concept to demonstrate the feasibility of the design. To show the interoperability capabilities of the platform, we implemented two example subscriber networks, as well as an example publisher network. The broker and the example publisher are implemented using Hyperledger Fabric v2 [221]. The two example subscribers are implemented using Hyperledger Fabric V1.4 [93, 222], and Hyperledger Besu [223]. In this section, the implementation and deployment details of the broker and the example networks are discussed. The codebase of the platform can be found on the Hyperledger Lab's GitHub[1] page.

5.4.1 Broker Blockchain

The broker blockchain acts as a messaging broker between other blockchains to enable interoperability. When choosing the blockchain solution to implement the broker network, we had to ensure that the solution fits well with the needs of this platform. First, since we are aiming to address interoperability in permissioned blockchains, the broker also needs to be permissioned. Moreover, many permissioned blockchains are enterprise-level, and they may have privacy and governance concerns. We need a broker blockchain that takes these needs into consideration. Finally, the smart contracts that need to be implemented on the broker blockchain are complicated, and the blockchain needs to support this kind of smart contract. Many blockchains only support smart contracts written in non-standard and domain-specific programming languages, making it hard to implement complicated smart contracts. Hyperledger Fabric has all the required features for the broker blockchain and more.

Hyperledger Fabric is an open-source permissioned blockchain that has been designed for enterprise use cases. Unlike the open permissionless blockchains that have scalability issues, Fabric enables high transaction throughput and low transaction confirmation

[1] https://github.com/hyperledger-labs/pubsub-interop

latency. The architecture of Hyperledger Fabric is highly modular and configurable, which enables customization for each specific use case. The consensus protocol, ordering service, membership service, database management system, and even the endorsement policy can be changed and modified to enable all kinds of applications. Most blockchain platforms that support smart contracts follow an order-execute architecture. In these blockchains, the transactions are first validated and ordered by the consensus protocol. Only then the transactions are propagated to all the peers in the network for sequential execution. This architecture requires the smart contracts to be deterministic to ensure that consensus is reached among the peers in the network. As a result, smart contracts need to be written in a non-standard and domain-specific programming language, limiting their functionalities. On the other hand, Hyperledger Fabric uses an execute-order-validate architecture that addresses the challenges of the order-execute architecture, and it also improves scalability and performance. In this architecture, the first step is to endorse the transaction by executing it and checking its correctness. Next, the transactions are ordered based on a configurable consensus protocol. At last, based on the application's endorsement policy, the transaction is validated and committed to the ledger. Since in this architecture, the transactions are executed before being ordered, any inconsistency can be detected in the execution step. Therefore a domain-specific language for the smart contract that ensures determinism is not required. This enables Hyperledger Fabric to support smart contracts in general-purpose programming languages such as Go, Node.js, and Java [221].

A Hyperledger Fabric network consists of different *organizations* that contribute resources to run the network. To manage the roles and identities in the permissioned ledger, Hyperledger Fabric uses *Certificate Authorities (CAs)* and *Membership Service Providers (MSPs)*. Usually, Each organization has its own CA and MSP. The CA is responsible for issuing identities by generating a key-pair that consists of a public and a private key. Using the public key, the MSP then assigns roles and defines permissions for the identity. In Hyperledger Fabric, an *ordering service* is used to

order the transactions and pack them into blocks. The ordering service consists of orderer nodes (or orderers for short) that reach a consensus among each other based on a pluggable consensus algorithm. Currently, Raft, Kafka, and Solo consensus algorithms are supported by Fabric. The design of Hyperledger Fabric relies on deterministic consensus, which prevents the creation of forks in the blockchain. Many other blockchains, such as Bitcoin and Ethereum, rely on probabilistic consensus algorithms, which can result in forks and many problems that come with it. As the nodes that host instances of the ledgers and the smart contracts, *peers* are a fundamental element of a blockchain network. Each peer belongs to an organization and is responsible for validating the transactions in a block, adding the new block to the ledger, and keeping a copy of the ledger. Some peers are also responsible for interacting with applications, executing the smart contracts, and endorsing transactions. These peers are called *endorsers*. Each smart contract has an *endorsement policy* that defines which peers should run the smart contract and endorse the transactions. Hyperledger Fabric allows a set of components, such as peer nodes, ordered nodes, and applications, to interact privately through *channels*. There can exist many channels in the network, and each peer can be a part of one or more channels. Each channel has its own ledger, and only those with access to the channel can interact with the ledger. Like any other blockchain that supports smart contracts, in Fabric, smart contracts define the executable logic for generating new transactions and making changes to the ledger. In Hyperledger Fabric, a chaincode is defined as a set of related smart contracts. Since each chaincode only consists of one smart contract in most use cases, the words chaincode and smart contract are often used interchangeably in the literature.

In the broker network, we leverage the capabilities of Hyperledger Fabric V2.2 to implement a messaging broker. The broker network has two peer organizations and an orderer organization, each with an independent certificate authority. Each of the peer organizations hosts one peer node, and the orderer uses Raft implementation. Two chaincodes have been implemented that run on one channel. In the current

prototype, all components of the broker network have been deployed on an instance with 8 VCPUs, 30GB RAM, and 288GB disk with Ubuntu 18.04.

We implement two chaincodes called the topics and the connector to support the features needed for the broker. The topics chaincode is responsible for keeping all the topics and their corresponding details. In Hyperledger Fabric, everything is stored as a key-value pair. In the topics smart contract, the key is a unique topic ID. The value is an object that includes the following properties: name, publisher, subscribers, and message. The name of a topic is a string value set by the publisher when creating the topic. Each topic also has one publisher, the blockchain network that has registered the topic on the broker. The publisher is the only blockchain that can make changes to the topic. The subscribers property stores a list of all the blockchains that have subscribed to the topic. It is worth mentioning that the publisher and the subscribers properties only accept objects stored on the connector blockchain. As a result, the publisher and subscriber blockchains should enroll in the connector chaincode before invoking the topics chaincode.

The connector chaincode is responsible for storing the connection details of other blockchain networks. Similar to the topics chaincode, the key in the key-value pair used in this chaincode is a unique ID for each blockchain. The value is an object that has the following properties: name, type, server IP, port, extra information. The name is a string value that can be selected when enrolling in the network. Type shows what kind of blockchain technology this network is using. Currently, support for Fabric and Besu has been implemented, and other blockchains will be added in future versions. The server IP and port are used by the broker blockchain to access the publisher or subscriber using an HTTP request. The extra information property stores network-specific details that may be needed when interacting with the blockchains. For instance, for a Hyperledger Besu network, this includes the private key, address, and the contract application binary interface (ABI) that the broker should use to send a request to the Besu network.

As mentioned earlier, each channel has its own separate ledger. A ledger stores facts about all the objects. A ledger consists of one or more world states and a blockchain. A world state is a database that stores the latest values of each object. Each chaincode has a separate world state, which is in a namespace that only the smart contracts inside that chaincode can access. The blockchain keeps the immutable history of all the objects in the ledger. In other words, each and every change that has happened to each object from its creation can be accessed from the blockchain. Transactions, which represent queries or updates to the world state, are bundled into blocks by the ordering service. Each block has a hash in its header, calculated based on the hash of the transactions in the block and the hash of the previous block. This mechanism results in a sequentially linked list of blocks, which enables immutability and increases the ledger's security. Figure 5.2 shows the ledger of the broker blockchain. One blockchain keeps records of all the objects, and there are two world states, one for the topics chaincode and another for the connector chaincode. Each peer in the network owns a copy of this complete ledger.

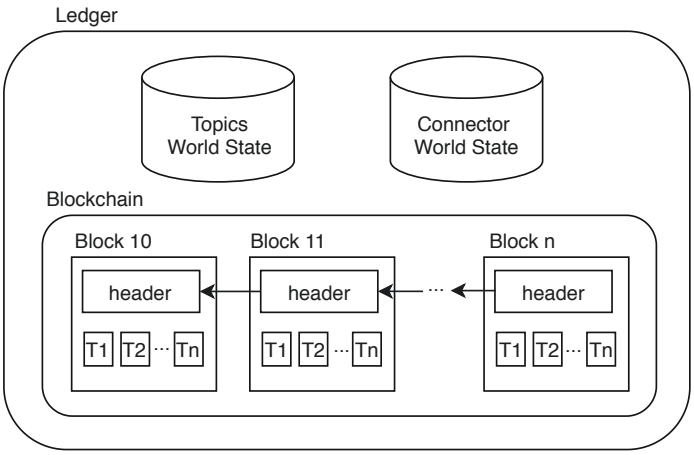


Figure 5.2: Broker blockchain's ledger which consists of one blockchain and two world states, one for each chaincode. The blockchain stores the history of the transactions. The world states store the latest version of each object.

To better understand how the topics and connector chaincodes work, we need to discuss their implemented functionalities. Figure 5.3 shows the UML class diagram

of the implemented chaincodes. The Hyperledger Fabric contract API provides an interface for developing smart contracts and applications. Each developed chaincode should extend the contract class from this API and then implement the required logic. In each smart contract, the *InitLedger* function is used to create a set of initial assets on the ledger when the chaincode is deployed. In the topics chaincode, the *CreateTopic* function can be used to create a new asset of type topic. The *QueryTopic* and the *QueryAllTopics* functions can be used to query one specific topic and all the existing topics, respectively. The connector chaincode implements the same initialize, create, and query functionalities but for assets of type blockchain.

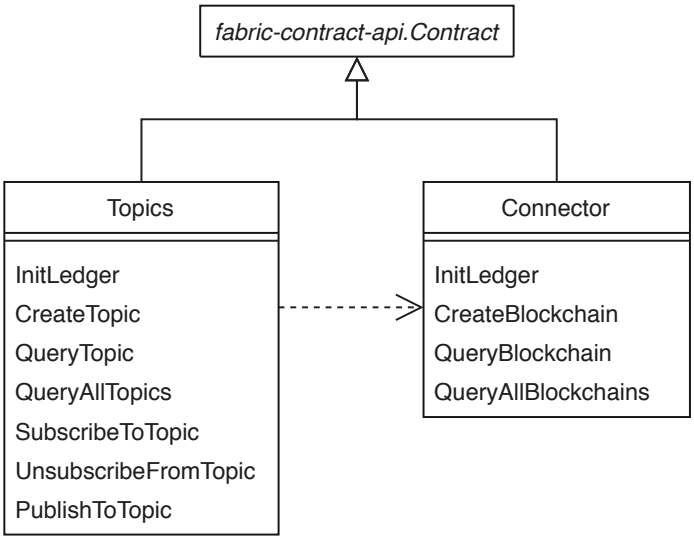


Figure 5.3: UML class diagram of the implemented chaincodes.

Other than the mentioned functions, the topics blockchain also implements *SubscribeToTopic*, *UnsubscribeFromTopic*, and *PublishToTopic* functionalities. When a destination blockchain wants to get notified of a topic's change, it subscribes to that topic by invoking the *SubscribeToTopic* function. In this case, the broker blockchain retrieves the latest version of the topic from the world state, adds the new destination blockchain to the list of subscribers for that topic, and updates the ledger. The subscriber can also unsubscribe from the topic at any time by invoking the *UnsubscribeFromTopic* function. Finally, the *PublishToTopic* function is used by the

source blockchain network when they want to update the topic's message. An invoke request to this function triggers update requests to all the subscribers of the topic. Algorithm 2 shows the detailed implementation of the *PublishToTopic* method. First, the broker retrieves the latest version of the topic from the topics world state. In the case that no topic was found, it immediately throws an error. Next, the topic's message is updated with the new message value and the topic's state is put to the world state. The next step is for the broker to notify all the subscribers. For each of the subscribers of the topic, the blockchain object is queried from the connector contract. This inter-chaincode communication is also shown in Figure 5.3. Then given the type of subscriber blockchain, the steps to invoke the remote network are followed.

Algorithm 2: PublishToTopic Method

```
   Input: topicID, newMessage
   Result: Subscribers are notified of the new message
 1 topicState ← getState(topicID)
 2 if !topicState then
 3 |   throw error
 4 end
 5 topicState.message ← newMessage
 6 putState(topicID, topicState)
 7 for subID in topicState.subscribers do
 8 |   subState ← query subID from connector contract
 9 |   if subState.type = Fabric then
10 |   |   follow steps to invoke a remote Fabric network
11 |   else if subState.type = Besu then
12 |   |   follow steps to invoke a remote Besu network
13 |   end
14 end
```

5.4.2 Subscriber Blockchains

The subscriber, or destination blockchain, is the blockchain that requires information from another blockchain to run a task. For the subscriber to be able to participate in the platform, it needs to have the appropriate connector smart contract deployed on it. We have implemented subscriber connector contracts for Hyperledger Fabric v1.4

and Hyperledger Besu. However, the connector is a simple smart contract that can also be developed by the owners of the subscriber blockchain. This smart contract needs to keep track of the topics that the subscriber has subscribed to and store their latest version for other smart contracts to access at any time. Two example subscriber networks have been implemented to demonstrate the interoperability capabilities of the platform.

The first example subscriber is implemented using Hyperledger Fabric V1.4. The two versions of Fabric used in this work are similar in terms of architecture. However, version 2 offers improved performance, usability, and security when compared to version 1.4. The Fabric example subscriber has been implemented on an instance with 2 VCPUs, 7.5GB RAM, and 36GB disk with Ubuntu 18.04. There are two organizations, each hosting two peers. One Solo orderer and one Fabric certificate authority are used by the whole network.

The second example subscriber is implemented using Hyperledger Besu, an open-source Ethereum client that supports private and permissioned blockchains. Besu can be used to create networks that work based on a proof of work (PoW) or a proof of authority (PoA) consensus algorithm. In this work, we implemented a PoW network using Besu, which can be thought of as a private Ethereum network. We then implemented a connector smart contract in Solidity to keep a record of the subscribed topics. The Besu network runs on an instance with 2 VCPUs, 7.5GB RAM, and 36GB disk with Ubuntu 18.04.

5.4.3 Publisher Blockchains

The publisher, or the source blockchain, is the blockchain network that needs to send information to other blockchains. Similar to what we have in the subscriber blockchain, a connector smart contract is also required for the publishers. However, the connector is slightly different in the publisher. The publisher connector should not only keep track of the topics, but it should also connect to the broker blockchain to publish the

topics. We implemented an example publisher network using Hyperledger Fabric V2.2 with the same configurations as the broker network on an instance with 2 VCPUs, 7.5GB RAM, and 36GB disk with Ubuntu 18.04.

5.5 Evaluation

We have discussed the design of our interoperability solution as well as the implementation details of the prototype. In this section, we focus on evaluating the performance of the implemented prototype of the broker blockchain. The goal is to see how the throughput and latency of the system changes in different scenarios. We have conducted two series of experiments to achieve this goal. The first set of experiments aims to show the performance metrics of different functionalities in the broker blockchain. The second set of experiments is focused on the publish function, which is the most important and time-consuming part of broker blockchain.

We have used Hyperledger Caliper [224] to run the experiments. Caliper is an open-source blockchain performance benchmark tool that allows performance measurement for different blockchains such as Hyperledger Fabric, Ethereum, Hyperledger Besu, etc. In Hyperledger Caliper, the workloads or benchmarks are responsible for generating the content of each transaction that is sent to the blockchain network. Given the network and benchmark configurations, Caliper uses a set of independent workers to send scheduled requests to the blockchain network and monitor the response. When the tests are finished, Caliper generates a performance report which consists of the average throughput and minimum, maximum, and average latency throughout the test. The throughput shows the number of transactions that were processed in the system in a given time. The latency shows the amount of time it takes for a transaction to be finished and added to the ledger.

We have set up Hyperledger Caliper on a separate machine to ensure that its process does not affect the performance of the broker network. The machine has 8 VCPUs, 30GB RAM, and 288GB disk with Ubuntu 18.04. We use five workers, a fixed rate

controller, and a test duration of 60 seconds for each benchmark round.

The first set of experiments focuses on the performance evaluation of broker blockchain. In these experiments, we conduct a series of tests using Hyperledger Caliper for each functionality that broker blockchain offers. Figure 5.3 summarizes all these functionalities. Each type of transaction goes through a specific set of steps in Hyperledger Fabric, which highly influences the response time for that transaction. For instance, an invoke transaction goes through endorse, order and commit steps. On the other hand, a query transaction is not transferred to the orderer, and the response is immediately sent back by the peer. The create actions in the connector and topics smart contract are invoke actions that have very similar implementations. The same goes for the query actions in the two smart contracts. As a result, it would be repetitive to run performance evaluation experiments for both smart contracts. Therefore, we run the experiments on the topics smart contract.

The topics smart contract has five important functionalities: create a topic, query a topic, publish to a topic, subscribe to a topic, and unsubscribe from a topic. For each of these actions, we run a set of experiments by changing the transaction send rate in the Hyperledger Caliper benchmark. The goal is to see how the system's throughput and average latency changes when the send rate is changed. Figure 5.4 shows the details of these experiments. It can be seen that the send rate follows the same pattern for all the actions except for *PublishToTopic*. The reason for this difference is that the *PublishToTopic* action takes more time and needs more resources to run compared to other actions. Consequently, the hardware limits of the broker blockchain are reached when the network receives more than roughly 100 publish transactions in each second. We discuss the behaviour of the network with different *PublishToTopic* requests in the second set of experiments shown in Figure 5.5. As a result of this limitation, we lowered the send rate for the *PublishToTopic* action in our experiments.

It can be seen in Figure 5.4 that the *SubscribeToTopic*, *UnsubscribeFromTopic*, and *CreateTopic* have similar behaviours under the same send rate. These three actions are

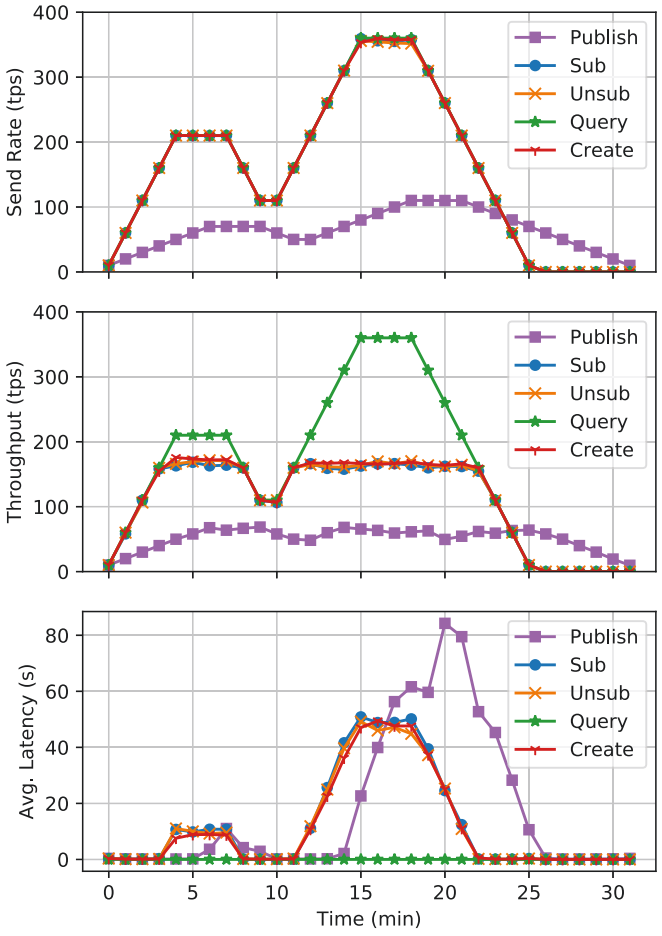


Figure 5.4: The trend of system throughput and average latency for various functionalities throughout time with the change of request send rate. The words publish, sub, unsub, query, and create in the plots stand for PublishToTopic, SubscribeToTopic, UnsubscribeFromTopic, QueryTopic, and CreateTopic functions, respectively.

of type invoke. As mentioned earlier, an invoke transaction goes through all the steps of execution, endorsement, ordering, and committing. In other words, since an invoke transaction proposes a change in the blockchain, it needs to go through the consensus algorithm, which can be time-consuming. Since the three actions are of the same type, and none need heavy computations in execution, the system throughput and latency for all of them are similar. As can be seen in the experimentation results, when the send rate is lower than a threshold (around 160 TPS in this case), the throughput is the same as the send rate, and the average latency is only a few milliseconds. This shows that with send rates below the threshold, all transactions are processed immediately. When the number of create, subscribe, or unsubscribe transactions sent in each second is more than the threshold, the processing limit of the broker network is reached. The throughput is limited to the broker's maximum capacity (around 160 TPS), and the transactions are queued before being processed, which results in an increase in the latency. Figure 5.4 shows that when the send rate for the create, subscribe, or unsubscribe transactions is around 210 TPS, the average latency increases to about 11 seconds. The latency keeps increasing with higher send rates and reaches approximately 50 seconds with a send rate of 360 TPS.

The *QueryTopic* action is very different from the previous ones. Since a query transaction does not go through the consensus protocol, its process is much faster. The send rate pattern used for query is similar to that of create, subscribe, and unsubscribe. However, the throughput and average latency act very differently. The throughput follows the same pattern as the send rate, and the average latency is around a few milliseconds throughout the whole experiment. These results show that this experiment does not reach the process limit for *QueryTopic*.

Finally, the *PublishToTopic* action is very different from the previous actions. It is similar to create, subscribe, and unsubscribe because they are all invoke transactions. However, the publish action requires heavier computations. The implementation details of the *PublishToTopic* method can be found in Algorithm 2. As mentioned

earlier, since the publish action needs more time and computational resources, we use a different send rate pattern for it. If we were to use the same send rate, the hardware limits of the broker blockchain would be reached, resulting in the experiments being halted. We discuss this in more detail in the second set of experiments shown in Figure 5.5. To ensure that the performance of the remote source and destination networks do not influence the performance evaluation of the broker network, we only send dummy requests to the subscriber networks during the experiments. It can be observed from Figure 5.4 that the publish action reaches the processing limit of the broker network much faster than the other invoke transactions. With send rates of about 70 TPS and more, the throughput is limited to 65 TPS. The average latency for the publish action has more fluctuations compared to other invoke actions. The main reason for this fluctuation is that in the publish method, depending on the number of subscribers that the topic has, the processing time can vary. In this experiment, the average latency gets as high as 80 seconds, with the send rate of 110 TPS.

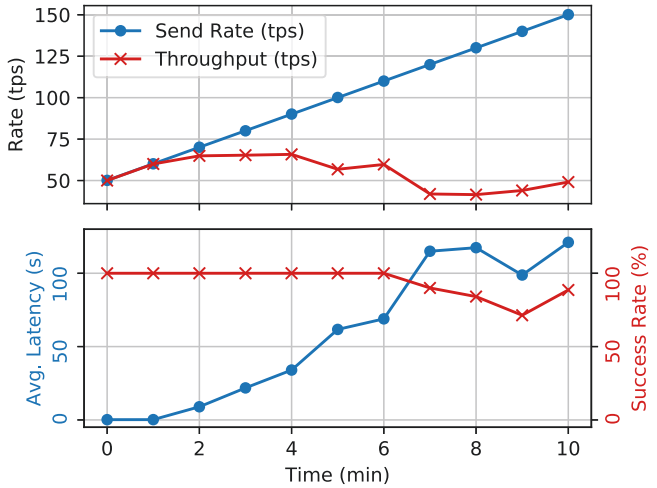


Figure 5.5: The trend of system throughput, average latency, and request success rate throughout time with the change of send rate.

Given the limits of the *PublishToTopic* action, we decided to run some extra experiments on this type of transaction. This experiment aims to find the limits of the

broker network and discover what happens when the limit is reached. In the previous experiment, we discovered that the processing limit for the publish transactions is reached at the sent rate of around 70 TPS. We also observed that the latency increases, and the throughput is limited for send rates above 70 TPS and below 110 TPS. However, we would like to know what happens if the send rate is more than 110 TPS. In this experiment, we linearly increase the send rate from 50 to 150 TPS and observe the throughput, latency, and transaction success rate. Figure 5.5 shows the results of this experiment. Similar to the previous experiment, we see that the throughput is limited, and the latency is increased when the send rate reaches 70 TPS. Nevertheless, the interesting change happens at the 120 TPS send rate. At this point, a significant drop and a significant rise is observed in the throughput and latency, respectively. Moreover, the transaction success rate is not 100% anymore. From this point on, a portion of the transactions fail since the broker network has reached its hardware limits.

5.6 Discussions

To enable blockchain interoperability, we have proposed the use of a broker blockchain as a middleman. The broker blockchain acts as a decentralized trusted relay between the source and destination network. Using a relay enables the interoperating networks to transfer data with minimal effort. The task of verifying the data and handling the communications between different blockchain networks can be delegated to the relay. As a result, there is no need for source and destination networks to make fundamental changes to their underlying networks. The use of a middleman may seem contradictory to the nature of blockchain at first. However, in our solution, the relay network is also a blockchain network that runs on smart contracts. Therefore, the broker blockchain allows the interoperation to be seamless, transparent, and secure.

In our evaluations, we identified the PublishToTopic functionality as the bottleneck of the platform. A possible future research direction can be to improve the performance

of this function. Different solutions can be implemented and compared for performance. One idea is to add a time component that the subscribers can set to specify the time interval that they want to receive updates for a topic. For instance, a subscriber may choose to receive notifications for a topic no sooner than every 10 minutes. This way, if the topic changes during this time, the subscriber gets notified of all the changes at once after the time interval.

In the current prototype of our platform, everyone can subscribe to a topic, and everything is transparent to all the participants in the network. An access control layer can be added to the platform as an extra feature. In some use cases, the source and destination blockchains may need to interact with each other privately and confidentially. The channels in Hyperledger Fabric can be used to enable such communications. Moreover, the publisher network may choose to only make a topic available to a subset of subscribers. An access control module can be used to manage the blockchains that can access each topic.

5.7 Conclusion

With blockchain technology gaining popularity in academia and industry every day, many blockchain networks are being introduced worldwide. These networks are highly isolated and incompatible with each other, resulting in silos of data. Blockchain interoperability solutions can revolutionize this technology by enabling data and asset transfers between heterogeneous blockchains. In this work, we propose a blockchain interoperability solution based on the publish/subscribe architecture. Our solution consists of a broker blockchain that keeps a record of the data being transferred between blockchain networks. The blockchains that want to participate in the interoperability can connect to the broker network as publishers or subscribers, depending on their role. A prototype of the broker blockchain has been implemented using Hyperledger Fabric. Moreover, an example publisher and two example subscribers have been implemented using Hyperledger Besu and two versions of Hyperledger Fabric to show that the design

works for heterogeneous blockchains. The network's performance has been analyzed using a benchmark tool to identify the limits and bottleneck of the platform. The implementation and evaluations indicate the feasibility of the idea with satisfactory performance, and the bottleneck is identified to be the process of publishing a new message to a topic. Finally, a discussion on the extensibility, scalability, and possible improvements of the system is presented.

Chapter 6

Conclusion and Future Work

In this work, we first studied and analyzed the performance of different distributed ledgers to identify their potential for cloud computing applications. We then use the results to design an open blockchain-based serverless computing platform. Finally, we propose a blockchain interoperability solution to enable our serverless platform to work with other blockchain networks and offer payments in other cryptocurrencies. The following summarizes the findings and results of each of our studies as well as possible future research directions.

In Chapter 2, we conducted a systematic survey covering the performance evaluation approaches for distributed ledger technologies that exist in the literature. We categorized these solutions into empirical and analytical evaluation methods. The empirical methods can further be grouped into four categories: performance benchmarking, monitoring, experimental analysis and simulation. We identified performance monitoring as the best solution for evaluating public blockchains. Compared to the empirical solutions, analytical modelling approaches were found to be more powerful, especially for analyzing the consensus layer of blockchain systems. We compared three main types of modelling approached and discussed their advantages and disadvantages for each blockchain network. These approaches are Markov chains, queueing models and stochastic Petri nets. Lastly, we summarize the bottlenecks of major blockchain platforms and proposed the open issues and possible future research directions in this

area.

In Chapter 3, we studied the performance of IOTA, a well-known DAG-based distributed ledger. In this work, we analyzed different performance metrics of the IOTA network using simulation, experiments, and an analytical layered model. We extended a DAG simulation tool to support the currently running consensus algorithm on the public IOTA network. We then used experiments and simulations to investigate the impact of arrival rate, tip selection algorithm, network delay, and randomness parameter of the random walk algorithm on the number of confirmed transactions per second in the network. Finally, we proposed an analytical model to estimate the transaction reattachment waiting time from the user's perspective.

As for the future work, on one hand we would like to study how other factors such as encryption algorithms and ledger databases influence the throughput. It would be also interesting to evaluate IOTA's performance under consensus without COO in further study. On the other hand, our extended simulator did not resolve the efficiency problem in large-scale simulations, which would be another direction of our future research.

In Chapter 4, we proposed an open blockchain-based serverless computing platform called ChainFaaS that leverages the untapped computational power of personal computers. Developers can use this platform to run tasks in a scalable environment with minimal infrastructure management overhead and a reasonable price. Any individual can rent out the extra computational power of their personal computer on ChainFaaS to make a profit. As proof of concept, we implemented and evaluated a prototype of the proposed platform, which is publicly available[1]. Also, the source code, documentation, and user guides are available on GitHub[2]. The prototype uses Hyperledger Fabric as the blockchain solution, which enables decentralized and transparent management of the platform. Moreover, to ensure the security of computing providers, this platform

[1]https://chainfaas.com
[2]https://github.com/pacslab/ChainFaaS

runs jobs in isolated environments using containerization techniques. The evaluation process indicates the feasibility of the idea with satisfactory performance.

From the computing providers' and the blockchain peers' point of view, their income from this platform should be worth their time and effort. They also need to consider the increased electricity consumption of their computers resulting from their participation in the network. Moreover, to motivate the developers to switch from public cloud providers to ChainFaaS, the platform should be reliable and cost-efficient. While the initial design of ChainFaaS indicates a sustainable income for the providers and the peers and a low-cost for the developers, a more in-depth cost analysis in future research can be valuable. It is necessary to take into account the cost efficiency of the platform for the developers as well as any possible costs for the providers and the blockchain peers.

In the current prototype of ChainFaaS, the scheduler randomly selects one of the available providers with enough CPU and RAM for the request. The scheduler can be improved to take into account other relevant factors. For instance, it can consider the reputation of the providers: those with a higher successful job completion score may get a higher priority in scheduling compared to others. Moreover, instead of having the providers delete the Docker images that they have run, they could cache the images of recent requests they served. The scheduler can then take into account the images that each provider has stored when distributing new tasks. This way, the impact of cold starts will be minimized.

The compute provider's agent has been deployed as a Docker container to prevent the code from tampering with the provider's programs and files. However, there are still some concerns about the security of Docker containers [199]. An interesting research direction on open platforms could be to increase the security for providers while maintaining the performance and usability of the platform.

Any blockchain-based platform faces performance barriers, and ChainFaaS is not an exception. Hyperledger Fabric performs faster than most blockchains, especially

those with a proof-of-work consensus algorithm. Nevertheless, as can be seen from the experiments, sometimes it can take up to 20 seconds to complete a task. The blockchain network could be optimized to enhance the overall performance of the platform.

Current implementation of ChainFaaS lacks a policy management component that defines different policies for the system such as fault tolerance, update and change, as well as workload aggregation policies. Since each provider node is an unreliable personal computer, the system's reliability per node is low. On the other hand, since a large number of providers participate in the network, the system achieves reliability by number. A fault tolerance policy is needed to specify what the system should do in case a provider node is unable to respond to a request. Also, an update policy is needed to specify the update and upgrade mechanisms of the system. This policy should include how the nodes are notified of the updates and what they should do if an update is received in the middle of a task. Finally, a workload aggregation policy would be helpful. With such policy, the network can accept large tasks and divide them into smaller subtasks that can run on individual personal computers. In the end, using the aggregation policy, it could merge the results from different nodes.

Finally, in Chapter 5, we proposed an interoperability solution for permissioned blockchains based on the publish/subscribe architecture to enable different blockchain networks to work together. In this platform, a broker blockchain acts as a messaging broker and uses smart contracts to track the topics and messages in the network. When a publisher blockchain publishes a new message to a topic, all the subscriber blockchains who have subscribed to the topic will be notified of the change. This method enables seamless interoperability as there is no need for the subscribers to keep checking for the changes on the publisher blockchain. We implemented a prototype of the broker blockchain and three publisher and subscriber blockchains to assess our design. The codes base and documentation can be found on Github[3]. We concluded

[3]https://github.com/hyperledger-labs/pubsub-interop

this work by evaluating the implemented prototype using performance benchmarking tools. In our evaluations, we identified the PublishToTopic functionality as the bottleneck of the platform. A possible future research direction can be to improve the performance of this function. Different solutions can be implemented and compared for performance. One idea is to add a time component which the subscribers can set to specify the time interval that they want to receive updates for a topic. For instance, a subscriber may choose to receive notifications for a topic no sooner than every 10 minutes. This way, if the topic changes during this time, the subscriber gets notified of all the changes at once after the time interval. Moreover, in the current prototype of our platform, everyone can subscribe to a topic and everything is transparent to all the participants in the network. An access control can be added to the platform as an extra feature. In some use cases, the source and destination blockchains may need to interact with each other privately and confidentially. The channels in Hyperledger Fabric can be used to enable such communications. Moreover, the publisher network may choose to only make a topic available to a subset of subscribers. An access control can be used to manage the blockchains that can access each topic.

Bibliography

[1] S. Popov, "The tangle," *cit. on*, p. 131, 2016.

[2] M. Zander, T. Waite, and D. Harz, "Dagsim : Simulation of dag-based distributed ledger protocols," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 46, no. 3, pp. 118–121, 2018, ISSN: 01635999. DOI: 10.1145/3308897.3308951.

[3] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, Last accessed 2020-07-17, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[4] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[5] D. Schwartz, N. Youngs, A. Britto, *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, no. 8, 2014.

[6] CB Insights, *Banking Is Only The Beginning: 42 Big Industries Blockchain Could Transform*, Last accessed 2020-02-24, 2019. [Online]. Available: https://www.cbinsights.com/research/industries-disrupted-blockchain/.

[7] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado, "Blockchain distributed ledger technologies for biomedical and health care applications," *Journal of the American Medical Informatics Association*, vol. 24, no. 6, pp. 1211–1220, 2017.

[8] J. Zhang, N. Xue, and X. Huang, "A secure system for pervasive social network-based healthcare," *Ieee Access*, vol. 4, pp. 9239–9250, 2016.

[9] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "Medshare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE Access*, vol. 5, pp. 14 757–14 767, 2017.

[10] M. F. Zia, M. Benbouzid, E. Elbouchikhi, S. M. Muyeen, K. Techato, and J. M. Guerrero, "Microgrid transactive energy: Review, architectures, distributed ledger technologies, and market analysis," *IEEE Access*, vol. 8, pp. 19 410–19 432, 2020.

[11] T. M. Fernández-Caramés and P. Fraga-Lamas, "A review on the use of blockchain for the internet of things," *IEEE Access*, vol. 6, pp. 32 979–33 001, 2018.

[12] P. K. Sharma, M.-Y. Chen, and J. H. Park, "A software defined fog node based distributed blockchain cloud architecture for iot," *Ieee Access*, vol. 6, pp. 115–124, 2017.

[13] K. Korpela, J. Hallikas, and T. Dahlberg, "Digital supply chain transformation toward blockchain integration," in *proceedings of the 50th Hawaii international conference on system sciences*, 2017.

[14] K. Toyoda, P. T. Mathiopoulos, I. Sasase, and T. Ohtsuki, "A novel blockchain-based product ownership management system (poms) for anti-counterfeits in the post supply chain," *IEEE Access*, vol. 5, pp. 17 465–17 477, 2017.

[15] J. H. Park and J. H. Park, "Blockchain security in cloud computing: Use cases, challenges, and solutions," *Symmetry*, vol. 9, no. 8, p. 164, 2017.

[16] K. Wüst and A. Gervais, "Do you need a blockchain?" In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, IEEE, 2018, pp. 45–54.

[17] F. M. Benčić and I. P. Žarko, "Distributed ledger technology: Blockchain compared to directed acyclic graph," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2018, pp. 1569–1570.

[18] N. Szabo, *Smart contracts*, Last accessed 2020-02-24, 1994. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

[19] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *Ieee Access*, vol. 4, pp. 2292–2303, 2016.

[20] K. Yeow, A. Gani, R. W. Ahmad, J. J. Rodrigues, and K. Ko, "Decentralized consensus for edge-centric internet of things: A review, taxonomy, and research issues," *IEEE Access*, vol. 6, pp. 1513–1524, 2017.

[21] H. Gupta and D. Janakiram, "Cdag: A serialized blockdag for permissioned blockchain," *arXiv preprint arXiv:1910.08547*, pp. 1–13, 2019.

[22] Y. Sompolinsky and A. Zohar, "Phantom: A scalable blockdag protocol.," *IACR Cryptology ePrint Archive*, vol. 2018, p. 104, 2018.

[23] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol.," *IACR Cryptology ePrint Archive*, vol. 2016, p. 1159, 2016.

[24] A. Churyumov, "Byteball: A decentralized system for storage and transfer of value," *URL https://byteball. org/Byteball. pdf*, 2016.

[25] C. LeMahieu, "Nano: A feeless distributed cryptocurrency network," *Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018)*, 2018.

[26] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1085–1100.

[27] V. Acharya, A. E. Yerrapati, and N. Prakash, *Oracle Blockchain Quick Start Guide: A practical approach to implementing blockchain in your enterprise*. Packt Publishing Ltd, 2019.

[28] G.-T. Nguyen and K. Kim, "A survey about consensus algorithms used in blockchain.," *Journal of Information processing systems*, vol. 14, no. 1, 2018.

[29] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, 2017, pp. 2567–2572.

[30] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "Sok: Consensus in the age of blockchains," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 183–198.

[31] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, "A survey on consensus mechanisms and mining strategy management in blockchain networks," *IEEE Access*, vol. 7, pp. 22 328–22 370, 2019.

[32] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[33] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D Patterson, A. Rabkin, I. Stoica, *et al.*, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, no. 13, p. 2009, 2009.

[34] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, *et al.*, "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*, Springer, 2017, pp. 1–20.

[35] Amazon Web Services Inc., *Aws lambda*, Last accessed 2020-01-23, 2019. [Online]. Available: https://aws.amazon.com/lambda/.

[36] Microsoft, *Azure functions*, Last accessed 2020-02-21, 2020. [Online]. Available: https://azure.microsoft.com/en-us/services/functions/.

[37] Google, *Google cloud functions*, Last accessed 2020-02-21. [Online]. Available: https://cloud.google.com/functions.

[38] IBM, *Ibm cloud functions*, Last accessed 2020-02-23. [Online]. Available: https://cloud.ibm.com/functions/.

[39] S. Nakamoto and A Bitcoin, "A peer-to-peer electronic cash system," *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, 2008.

[40] Q. K. Nguyen, "Blockchain-a financial technology for future sustainable development," in *2016 3rd International conference on green technology and sustainable development (GTSD)*, IEEE, 2016, pp. 51–54.

[41] L. Cocco, A. Pinna, and M. Marchesi, "Banking on blockchain: Costs savings thanks to the blockchain technology," *Future internet*, vol. 9, no. 3, p. 25, 2017.

[42] M. Hölbl, M. Kompara, A. Kamišalić, and L. Nemec Zlatolas, "A systematic review of the use of blockchain in healthcare," *Symmetry*, vol. 10, p. 470, 2018.

[43] C. C. Agbo, Q. H. Mahmoud, and J. M. Eklund, "Blockchain technology in healthcare: A systematic review," in *Healthcare*, Multidisciplinary Digital Publishing Institute, vol. 7, 2019, p. 56.

[44] L. A. Linn and M. B. Koo, "Blockchain for health data and its potential use in health it and health care related research," in *ONC/NIST Use of Blockchain for Healthcare and Research Workshop. Gaithersburg, Maryland, United States: ONC/NIST*, 2016, pp. 1–10.

[45] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *2016 IEEE 18th international conference on e-health networking, applications and services (Healthcom)*, IEEE, 2016, pp. 1–3.

[46] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, "Consortium blockchain for secure energy trading in industrial internet of things," *IEEE transactions on industrial informatics*, vol. 14, no. 8, pp. 3690–3700, 2017.

[47] E. Mengelkamp, B. Notheisen, C. Beer, D. Dauer, and C. Weinhardt, "A blockchain-based smart grid: Towards sustainable local energy markets," *Computer Science-Research and Development*, vol. 33, no. 1-2, pp. 207–214, 2018.

[48] N. Z. Aitzhan and D. Svetinovic, "Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 5, pp. 840–852, 2016.

[49] G. Perboli, S. Musso, and M. Rosano, "Blockchain in logistics and supply chain: A lean approach for designing real-world use cases," *IEEE Access*, vol. 6, pp. 62 018–62 028, 2018.

[50] E. Tijan, S. Aksentijević, K. Ivanić, and M. Jardas, "Blockchain technology implementation in logistics," *Sustainability*, vol. 11, no. 4, p. 1185, 2019.

[51] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, *et al.*, "On scaling decentralized blockchains," in *International conference on financial cryptography and data security*, Springer, 2016, pp. 106–125.

[52] J. Reed, *Litecoin: An Introduction to Litecoin Cryptocurrency and Litecoin Mining*. CreateSpace Independent Publishing Platform, 2017.

[53] J. Teutsch and C. Reitwießner, *Truebit: A scalable verification solution for blockchains*, 2018.

[54] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1353–1370.

[55] J. Poon and T. Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

[56] R. Network-Fast, *Cheap, scalable token transfers for ethereum*, 2018.

[57] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling blockchain innovations with pegged sidechains," *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, vol. 72, 2014.

[58] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, pp. 1–47, 2017.

[59] P. Gazi, A. Kiayias, and D. Zindros, "Proof-of-stake sidechains.," *IACR Cryptology ePrint Archive*, vol. 2018, p. 1239, 2018.

[60] A. Kiayias and D. Zindros, "Proof-of-work sidechains," in *International Conference on Financial Cryptography and Data Security*, Springer, 2019, pp. 21–34.

[61] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen, "Adding concurrency to smart contracts," *Distributed Computing*, pp. 1–17, 2019.

[62] L. Yu, W.-T. Tsai, G. Li, Y. Yao, C. Hu, and E. Deng, "Smart-contract execution with concurrent block building," in *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, IEEE, 2017, pp. 160–167.

[63] P. S. Anjana, S. Kumari, S. Peri, S. Rathor, and A. Somani, "An efficient framework for optimistic concurrent execution of smart contracts," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, IEEE, 2019, pp. 83–92.

[64] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 17–30.

[65] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 583–598.

[66] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 931–948.

[67] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 2019, pp. 95–112.

[68] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 123–140.

[69] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," in *International Conference on Financial Cryptography and Data Security*, Springer, 2015, pp. 528–547.

[70] C. Li, P. Li, D. Zhou, W. Xu, F. Long, and A. Yao, "Scaling nakamoto consensus to thousands of transactions per second," *arXiv preprint arXiv:1805.03870*, 2018.

[71] S. D. Lerner, "Dagcoin: A cryptocurrency without blocks," *White paper*, 2015.

[72] S. Kim, Y. Kwon, and S. Cho, "A survey of scalability solutions on blockchain," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, IEEE, 2018, pp. 1204–1207.

[73] S. Rouhani and R. Deters, "Security, performance, and applications of smart contracts: A systematic survey," *IEEE Access*, vol. 7, pp. 50 759–50 779, 2019.

[74] X. Zheng, Y. Zhu, and X. Si, "A survey on challenges and progresses in blockchain technologies: A performance and security perspective," *Applied Sciences*, vol. 9, no. 22, p. 4731, 2019.

[75] R. Wang, K. Ye, and C.-Z. Xu, "Performance benchmarking and optimization for blockchain systems: A survey," in *International Conference on Blockchain*, Springer, 2019, pp. 171–185.

[76] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16 440–16 455, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2967218.

[77] G. Yu, X. Wang, K. Yu, W. Ni, J. A. Zhang, and R. P. Liu, "Survey: Sharding in blockchains," *IEEE Access*, vol. 8, pp. 14 155–14 181, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2965147.

[78] S. T. Leutenegger and D. Dias, "A modeling study of the tpc-c benchmark," *ACM Sigmod Record*, vol. 22, no. 2, pp. 22–31, 1993.

[79] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.

[80] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux, "Oltp-bench: An extensible testbed for benchmarking relational databases," *Proceedings of the VLDB Endowment*, vol. 7, no. 4, pp. 277–288, 2013.

[81] V. Abramova and J. Bernardino, "Nosql databases: Mongodb vs cassandra," in *Proceedings of the international C* conference on computer science and software engineering*, 2013, pp. 14–22.

[82] Y. Abubakar, T. S. Adeyi, and I. G. Auta, "Performance evaluation of nosql systems using ycsb in a resource austere environment," *Performance Evaluation*, vol. 7, no. 8, pp. 23–27, 2014.

[83] H. Matallah, G. Belalem, and K. Bouamrane, "Experimental comparative study of nosql databases: Hbase versus mongodb by ycsb," *International Journal of Computer Systems Science and Engineering (IJCSSE)*, vol. 32, no. 4, pp. 307–317, 2017.

[84] Z. Dong, E. Zheng, Y. Choon, and A. Y. Zomaya, "Dagbench: A performance evaluation framework for dag distributed ledgers," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, IEEE, 2019, pp. 264–271.

[85] Performance and S. W. Group, "Hyperledger blockchain performance metrics(white paper v1.01)," HyperLedger Found., Tech. Rep., 2018.

[86] A. Aldweesh, M. Alharby, M. Mehrnezhad, and A. Van Moorsel, "Opbench: A cpu performance benchmark for ethereum smart contract operation code," in *2019 IEEE International Conference on Blockchain (Blockchain)*, IEEE, 2019, pp. 274–281.

[87] A. Aldweesh, M. Alharby, E. Solaiman, and A. van Moorsel, "Performance benchmarking of smart contracts to assess miner incentives in ethereum," in *2018 14th European Dependable Computing Conference (EDCC)*, IEEE, 2018, pp. 144–149.

[88] M. T. Oliveira, G. R. Carrara, N. C. Fernandes, C. V. Albuquerque, R. C. Carrano, D. S. Medeiros, and D. M. Mattos, "Towards a performance evaluation of private blockchain frameworks using a realistic workload," in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, IEEE, 2019, pp. 180–187.

[89] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu, "A detailed and real-time performance monitoring framework for blockchain systems," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, IEEE, 2018, pp. 134–143.

[90] Q. Nasir, I. A. Qasse, M. Abu Talib, and A. B. Nassif, "Performance analysis of hyperledger fabric platforms," *Security and Communication Networks*, vol. 2018, 2018.

[91] A. Baliga, N. Solanki, S. Verekar, A. Pednekar, P. Kamat, and S. Chatterjee, "Performance characterization of hyperledger fabric," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, IEEE, 2018, pp. 65–74.

[92] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE, 2018, pp. 264–276.

[93] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–15.

[94] T. S. L. Nguyen, G. Jourjon, M. Potop-Butucaru, and K. L. Thai, "Impact of network delays on hyperledger fabric," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2019, pp. 222–227.

[95] F. Geyer, H. Kinkelin, H. Leppelsack, S. Liebald, D. Scholz, G. Carle, and D. Schupke, "Performance perspective on private distributed ledger technologies for industrial networks," in *2019 International Conference on Networked Systems (NetSys)*, IEEE, 2019, pp. 1–8.

[96] M. Kuzlu, M. Pipattanasomporn, L. Gurses, and S. Rahman, "Performance analysis of a hyperledger fabric blockchain framework: Throughput, latency and scalability," in *2019 IEEE International Conference on Blockchain (Blockchain)*, IEEE, 2019, pp. 536–540.

[97] S. Wang, "Performance evaluation of hyperledger fabric with malicious behavior," in *International Conference on Blockchain*, Springer, 2019, pp. 211–219.

[98] Z. Shi, H. Zhou, Y. Hu, S. Jayachander, C. de Laat, and Z. Zhao, "Operating permissioned blockchain in clouds: A performance study of hyperledger sawtooth," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, IEEE, 2019, pp. 50–57.

[99] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.

[100] S. Rouhani and R. Deters, "Performance analysis of ethereum transactions in private blockchain," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, IEEE, 2017, pp. 70–74.

[101] R. Yasaweerasinghelage, M. Staples, and I. Weber, "Predicting latency of blockchain-based systems using architectural modelling and simulation," in *2017 IEEE International Conference on Software Architecture (ICSA)*, IEEE, 2017, pp. 253–256.

[102] C. Rathfelder and B. Klatt, "Palladio workbench: A quality-prediction tool for component-based architectures," in *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, IEEE, 2011, pp. 347–350.

[103] M. Bez, G. Fornari, and T. Vardanega, "The scalability challenge of ethereum: An initial quantitative analysis," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, 2019, pp. 167–176.

[104] C. Fan, H. Khazaei, Y. Chen, and P. Musilek, "Towards a scalable dag-based distributed ledger for smart communities," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, IEEE, 2019, pp. 177–182.

[105] F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telematics and Informatics*, vol. 36, pp. 55–81, 2019.

[106] R. Han, V. Gramoli, and X. Xu, "Evaluating blockchains for iot," in *2018 9Th IFIP international conference on new technologies, mobility and security (NTMS)*, IEEE, 2018, pp. 1–5.

[107] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, "Performance analysis of private blockchain platforms in varying workloads," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, IEEE, 2017, pp. 1–6.

[108] S. Chen, J. Zhang, R. Shi, J. Yan, and Q. Ke, "A comparative testing on performance of blockchain and relational database: Foundation for applying smart technology into current business systems," in *International Conference on Distributed, Ambient, and Pervasive Interactions*, Springer, 2018, pp. 21–34.

[109] Y. Hao, Y. Li, X. Dong, L. Fang, and P. Chen, "Performance analysis of consensus algorithm in private blockchain," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 280–285.

[110] H. M. A. Aljassas and S. Sasi, "Performance evaluation of proof-of-work and collatz conjecture consensus algorithms," in *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, IEEE, 2019, pp. 1–6.

[111] R. Deloin, "Proof of collatz conjecture," *Asian Research Journal of Mathematics*, pp. 1–18, 2019.

[112] S. Benahmed, I. Pidikseev, R. Hussain, J. Lee, S. A. Kazmi, A. Oracevic, and F. Hussain, "A comparative analysis of distributed ledger technologies for smart contract development," in *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, IEEE, 2019, pp. 1–6.

[113] R. Han, G. Shapiro, V. Gramoli, and X. Xu, "On the performance of distributed ledgers for internet of things," *Internet of Things*, p. 100 087, 2019.

[114] S. Park, S. Oh, and H. Kim, "Performance analysis of dag-based cryptocurrency," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, IEEE, 2019, pp. 1–6.

[115] S. Chandel, W. Cao, Z. Sun, J. Yang, B. Zhang, and T.-Y. Ni, "A multi-dimensional adversary analysis of rsa and ecc in blockchain encryption," in *Future of Information and Communication Conference*, Springer, 2019, pp. 988–1003.

[116] J. Ferreira, M. Antunes, M. Zhygulskyy, and L. Frazão, "Performance of hash functions in blockchain applied to iot devices," in *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, IEEE, 2019, pp. 1–7.

[117] M. Alharby and A. van Moorsel, "Blocksim: A simulation framework for blockchain systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 135–138, 2019.

[118] S. Pandey, G. Ojha, and B. Shrestha, "Blocksim: A practical simulation tool for optimal network design, stability and planning.," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2019, pp. 133–137.

[119] C. Faria and M. Correia, "Blocksim: Blockchain simulator," in *2019 IEEE International Conference on Blockchain (Blockchain)*, IEEE, 2019, pp. 439–446.

[120] M. Zander, T. Waite, and D. Harz, "Dagsim: Simulation of dag-based distributed ledger protocols," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 118–121, 2019.

[121] M. Bottone, F. Raimondi, and G. Primiero, "Multi-agent based simulations of block-free distributed ledgers," in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, IEEE, 2018, pp. 585–590.

[122] B. Kusmierz, W. Sanders, A. Penzkofer, A. Capossele, and A. Gal, "Properties of the tangle for uniform random and random walk tip selection," in *2019 IEEE International Conference on Blockchain (Blockchain)*, IEEE, 2019, pp. 228–236.

[123] D. Huang, X. Ma, and S. Zhang, "Performance analysis of the raft consensus algorithm for private blockchains," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[124] B. Cao, S. Huang, D. Feng, L. Zhang, S. Zhang, and M. Peng, "Impact of network load on direct acyclic graph based blockchain for internet of things," in *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, IEEE, 2019, pp. 215–218.

[125] Y. Kawase and S. Kasahara, "Transaction-confirmation time for bitcoin: A queueing analytical approach to blockchain mechanism," in *International Conference on Queueing Theory and Network Applications*, Springer, 2017, pp. 75–88.

[126] Q.-L. Li, J.-Y. Ma, and Y.-X. Chang, "Blockchain queue theory," in *International Conference on Computational Social Networks*, Springer, 2018, pp. 25–40.

[127] Q.-L. Li, J.-Y. Ma, Y.-X. Chang, F.-Q. Ma, and H.-B. Yu, "Markov processes in blockchain systems," *Computational Social Networks*, vol. 6, no. 1, pp. 1–28, 2019.

[128] S. Ricci, E. Ferreira, D. S. Menasche, A. Ziviani, J. E. Souza, and A. B. Vieira, "Learning blockchain delays: A queueing theory approach," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 122–125, 2019.

[129] W. Zhao, S. Jin, and W. Yue, "Analysis of the average confirmation time of transactions in a blockchain system," in *International Conference on Queueing Theory and Network Applications*, Springer, 2019, pp. 379–388.

[130] S. Geissler, T. Prantl, S. Lange, F. Wamser, and T. Hossfeld, "Discrete-time analysis of the blockchain distributed ledger technology," in *2019 31st International Teletraffic Congress (ITC 31)*, IEEE, 2019, pp. 130–137.

[131] M. Alaslani, F. Nawab, and B. Shihada, "Blockchain in iot systems: End-to-end delay evaluation," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8332–8344, 2019.

[132] U. R. Krieger, M. H. Ziegler, and H. L. Cech, "Performance modeling of the consensus mechanism in a permissioned blockchain," in *International Conference on Computer Networks*, Springer, 2019, pp. 3–17.

[133] P. Ferraro, C King, and R. Shorten, "Distributed ledger technology for smart cities, the sharing economy, and social compliance," *IEEE Access*, vol. 6, pp. 62 728–62 746, 2018.

[134] P. Yuan, K. Zheng, X. Xiong, K. Zhang, and L. Lei, "Performance modeling and analysis of a hyperledger-based system using gspn," *Computer Communications*, 2020.

[135] H. Sukhwani, N. Wang, K. S. Trivedi, and A. Rindos, "Performance modeling of hyperledger fabric (permissioned blockchain network)," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, IEEE, 2018, pp. 1–8.

[136] H. Zhang, C. Jin, and H. Cui, "A method to predict the performance and storage of executing contract for ethereum consortium-blockchain," in *International Conference on Blockchain*, Springer, 2018, pp. 63–74.

[137] N. Papadis, S. Borst, A. Walid, M. Grissa, and L. Tassiulas, "Stochastic models and wide-area network measurements for blockchain design and analysis," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, 2018, pp. 2546–2554.

[138] Y. Shahsavari, K. Zhang, and C. Talhi, "Performance modeling and analysis of the bitcoin inventory protocol," in *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, IEEE, 2019, pp. 79–88.

[139] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[140] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 305–319.

[141] T. G. Robertazzi, *Computer networks and systems: queueing theory and performance evaluation*. Springer Science & Business Media, 2012.

[142] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *IEEE Transactions on parallel and distributed systems*, vol. 23, no. 5, pp. 936–943, 2011.

[143] E. Gelenbe, G. Pujolle, E. Gelenbe, and G. Pujolle, *Introduction to queueing networks*. Wiley New York, 1998, vol. 2.

[144] M. Chaudhry and J. Templeton, "The queuing system m/gb/l and its ramifications," *European Journal of Operational Research*, vol. 6, no. 1, pp. 56–60, 1981.

[145] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[146] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with bft-smart," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE, 2014, pp. 355–362.

[147] Z. Team *et al.*, "The zilliqa technical whitepaper," *Retrieved September*, vol. 16, p. 2019, 2017.

[148] E. IO, "Eos. io technical white paper," *EOS. IO (accessed 18 December 2017) https://github. com/EOSIO/Documentation*, 2017.

[149] J Medhi, "Waiting time distribution in a poisson queue with a general bulk service rule," *Management Science*, vol. 21, no. 7, pp. 777–782, 1975.

[150] H. Wang, J. Li, Z. Shen, and Y. Zhou, "Approximations and bounds for (n, k) fork-join queues: A linear transformation approach," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, IEEE, 2018, pp. 422–431.

[151] G. Ciardo, J. K. Muppala, K. S. Trivedi, *et al.*, "Spnp: Stochastic petri net package.," in *PNPM*, vol. 89, 1989, pp. 142–151.

[152] P. Erdos, "On random graphs," *Publicationes mathematicae*, vol. 6, pp. 290–297, 1959.

[153] B. Bollobás and B. Béla, *Random graphs*, 73. Cambridge university press, 2001.

[154] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *White Paper*, pp. 1–9, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[155] CBinsights, "Banking Is Only The Beginning: 58 Big Industries Blockchain Could Transform," CBinsights, Tech. Rep., 2018. [Online]. Available: https://www.cbinsights.com/research/industries-disrupted-blockchain/.

[156] V. Buterin, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, p. 37, 2014. [Online]. Available: https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf.

[157] M. Samaniego and R. Deters, "Blockchain as a service for iot," in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, New York, NY, USA: IEEE, 2016, pp. 433–436.

[158] S. W. Group *et al.*, "Hyperledger blockchain performance metrics," *Hyperledger. org*, pp. 1–17, 2018.

[159] C. Fan, H. Khazaei, Y. Chen, and P. Musilek, "Towards a scalable dag-based distributed ledger for smart communities," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, Limerick, Ireland, Ireland: IEEE, 2019, pp. 177–182. DOI: 10.1109/WF-IoT.2019.8767342.

[160] A. Ilyushkin, A. Ali-Eldin, N. Herbst, A. Bauer, A. V. Papadopoulos, D. Epema, and A. Iosup, "An experimental performance evaluation of autoscalers for complex workflows," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, no. 2, 8:1–8:32, Apr. 2018, ISSN: 2376-3639. DOI: 10.1145/3164537. [Online]. Available: http://doi.acm.org/10.1145/3164537.

[161] C. Wang, Q. Liang, and B. Urgaonkar, "An empirical analysis of amazon ec2 spot instance features affecting cost-effective resource procurement," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, no. 2, 6:1–6:24, Mar. 2018, ISSN: 2376-3639. DOI: 10.1145/3164538. [Online]. Available: http://doi.acm.org/10.1145/3164538.

[162] S. Popov, H. Moog, D. Camargo, A. Capossele, V. Dimitrov, A. Gal, A. Greve, B. Kusmierz, S. Mueller, A. Penzkofer, *et al.*, "The coordicide," *White Paper*, pp. 1–55, 2020. [Online]. Available: https://files.iota.org/papers/20200120_Coordicide_WP.pdf.

[163] B. Kusmierz and A. Gal, "Probability of being left behind and probability of becoming permanent tip in the tangle v0.2," *White Paper*, pp. 1–15, 2018.

[164] P. Ferraro, C. King, and R. Shorten, "Iota-based directed acyclic graphs without orphans," *arXiv preprint arXiv:1901.07302*, pp. 1–11, 2018.

[165] A Cullen, P Ferraro, C King, and R Shorten, "On the resilience of dag-based distributed ledgers in iot applications," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7112–7122, 2020.

[166] C. Fan, "Performance analysis and design of an iot-friendly dag-based distributed ledger system," Master of Science Thesis, University of Alberta, 2019.

[167] R. Danilak, "Why energy is a big and rapidly growing problem for data centers," *Forbes*, vol. 15, pp. 12–17, 2017.

[168] N. Jones, "How to stop data centres from gobbling up the world's electricity," *Nature*, vol. 561, no. 7722, pp. 163–167, 2018.

[169] L. Belkhir and A. Elmeligi, "Assessing ict global emissions footprint: Trends to 2040 & recommendations," *Journal of Cleaner Production*, vol. 177, pp. 448–463, 2018.

[170] International Data Corporation (IDC), *Personal Computing Device Shipments Forecast to Continue Their Slow Decline with a Five-Year Compound Annual Growth Rate of -1.2%, According to IDC*, Last accessed 2020-05-03, 2019. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS44908319.

[171] C. Stoll, L. Klaaßen, and U. Gallersdörfer, "The carbon footprint of bitcoin," *Joule*, vol. 3, no. 7, pp. 1647–1661, 2019.

[172] G. Woltman and S. Kurowski, *The great internet mersenne prime search*, Last accessed 2020-05-12, 1996. [Online]. Available: http://www.mersenne.org.

[173] E. Ady *et al.*, *Distributed.net*, Last accessed 2020-05-12, 1997. [Online]. Available: https://www.distributed.net/.

[174] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@ home: An experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[175] K. Watanabe, M. Fukushi, and M. Kameyama, "Adaptive group-based job scheduling for high performance and reliable volunteer computing," *Journal of Information Processing*, vol. 19, pp. 39–51, 2011.

[176] P. Dharmapala, L. Koneshvaran, D. Sivasooriyathevan, I. Ismail, and D. Kasthurirathna, "Peer-to-peer distributed computing framework," in *2017 6th National Conference on Technology and Management (NCTM)*, IEEE, 2017, pp. 126–131.

[177] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Fifth IEEE/ACM international workshop on grid computing*, IEEE, 2004, pp. 4–10.

[178] D. P. Anderson, "Boinc: A platform for volunteer computing," *Journal of Grid Computing*, pp. 1–24, 2019.

[179] G. Fedak, C. Germain, V. Neri, and F. Cappello, "Xtremweb: A generic global computing system," in *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*, IEEE, 2001, pp. 582–587.

[180] Mithral Inc, *The mithral (cosm)*, Last accessed 2020-06-23, 2020. [Online]. Available: http://www.mithral.com.

[181] The Folding@home Consortium (FAHC), *Folding@home*, Last accessed 2020-06-23, 2020. [Online]. Available: https://foldingathome.org/.

[182] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@ home: Lessons from eight years of volunteer distributed computing," in *2009 IEEE International Symposium on Parallel & Distributed Processing*, IEEE, 2009, pp. 1–8.

[183] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande, "Folding@ home and genome@ home: Using distributed computing to tackle previously intractable problems in computational biology," *arXiv preprint arXiv:0901.0866*, 2009.

[184] FoldingCoin, Inc., *Foldingcoin whitepaper*, Last accessed 2020-06-23, 2018. [Online]. Available: https://foldingcoin.net/index.php/whitepaper.

[185] CureCoin Team, *Curecoin whitepaper*, Last accessed 2020-06-23, 2019. [Online]. Available: https://curecoin.net/white-paper/.

[186] GridCoin, *Gridcoin whitepaper*, Last accessed 2020-06-23, 2018. [Online]. Available: https://gridcoin.us/assets/img/whitepaper.pdf.

[187] *Iexec: Blockchain-based decentralized cloud computing v3.0*, Last accessed 2020-03-24, 2018. [Online]. Available: https://iex.ec/wp-content/uploads/pdf/iExec-WPv3.0-English.pdf.

[188] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 153–167.

[189] *The golem project: Crowdfunding whitepaper*, Last accessed 2020-03-24, 2016. [Online]. Available: https://golem.network/crowdfunding/Golemwhitepaper.pdf.

[190] Sonm Pte. Ltd., *Sonm: Decentralized fog computing platform*, Last accessed 2020-03-24, 2020. [Online]. Available: https://sonm.com/.

[191] K. Doka, T. Bakogiannis, I. Mytilinis, and G. Goumas, "Cloudagora: Democratizing the cloud," in *International Conference on Blockchain*, Springer, 2019, pp. 142–156.

[192] T. Bakogiannis, I. Mytilinis, K. Doka, and G. Goumas, "Leveraging blockchain technology to break the cloud computing market monopoly," *Computers*, vol. 9, no. 1, p. 9, 2020.

[193] R. B. Uriarte and R. De Nicola, "Blockchain-based decentralized cloud/fog solutions: Challenges, opportunities, and standards," *IEEE Communications Standards Magazine*, vol. 2, no. 3, pp. 22–28, 2018.

[194] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, "Integrated blockchain and edge computing systems: A survey, some research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1508–1532, 2019.

[195] M. Westerlund and N. Kratzke, "Towards distributed clouds: A review about the evolution of centralized cloud computing, distributed ledger technologies, and a foresight on unifying opportunities and security implications," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2018, pp. 655–663.

[196] Django Software Foundation, *Django*, Last accessed 2020-02-13, 2020. [Online]. Available: https://www.djangoproject.com/.

[197] Hyperledger, *Hyperledger fabric documentation*, Last accessed 2020-02-24, 2019. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html.

[198] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," *arXiv preprint arXiv:2003.03423*, 2020.

[199] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016.

[200] B. Chase and E. MacBrough, "Analysis of the xrp ledger consensus protocol," *arXiv preprint arXiv:1802.07242*, 2018.

[201] S. Ghaemi, H. Khazaei, and P. Musilek, "Chainfaas: An open blockchain-based serverless platform," *IEEE Access*, vol. 8, pp. 131 760–131 778, 2020.

[202] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *arXiv preprint arXiv:2005.14282*, 2020.

[203] Ethereum Foundation and Consensys, *BTC Relay*, Last accessed 2020-11-6, 2015. [Online]. Available: http://btcrelay.org/.

[204] A. Garoffolo, D. Kaidalov, and R. Oliynykov, "Zendoo: A zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains," *arXiv preprint arXiv:2002.01847*, 2020.

[205] S. Lerner, *RSK Whitepaper*, Last accessed 2020-11-6, 2015. [Online]. Available: https://docs.rsk.co/RSK_White_Paper-Overview.pdf.

[206] J. Lu, B. Yang, Z. Liang, Y. Zhang, S. Demmon, E. Swartz, and L. Lu, *Wanchain: Building super financial markets for the new digital economy.* Last accessed 2020-11-6, 2017. [Online]. Available: https://wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf.

[207] G. Wood, *Polkadot: Vision for a heterogeneous multi-chain framework.* Last accessed 2020-11-6, 2016. [Online]. Available: https://github.com/w3f/polkadotwhite-paper/raw/master/PolkaDotPaper.pdf.

[208] J. Kwon and E. Buchman, *Cosmos whitepaper*, Last accessed 2020-11-6, 2019. [Online]. Available: https://cosmos.network/cosmos-whitepaper.pdf.

[209] E. Abebe, D. Behl, C. Govindarajan, Y. Hu, D. Karunamoorthy, P. Novotny, V. Pandit, V. Ramakrishna, and C. Vecchiola, "Enabling enterprise blockchain interoperability with trusted data transfer (industry track)," in *Proceedings of the 20th International Middleware Conference Industrial Track*, 2019, pp. 29–35.

[210] E. Scheid, B. Rodrigues, and B. Stiller, "Toward a policy-based blockchain agnostic framework," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, IEEE, 2019, pp. 609–613.

[211] P. Frauenthaler, M. Borkowski, and S. Schulte, "A framework for blockchain interoperability and runtime selection," *arXiv preprint arXiv:1905.07014*, 2019.

[212] H. Montgomery, H. Borne-Pons, J. Hamilton, M. Bowman, P. Somogyvari, S. Fujimoto, T. Takeuchi, T. Kuhrt, and R. Belchior, *Hyperledger Cactus Whitepaper*, Last accessed 2020-09-28, 2020. [Online]. Available: https://github.com/hyperledger/cactus/blob/master/whitepaper/whitepaper.md.

[213] P. Lv, L. Wang, H. Zhu, W. Deng, and L. Gu, "An iot-oriented privacy-preserving publish/subscribe model over blockchains," *IEEE Access*, vol. 7, pp. 41 309–41 314, 2019.

[214] G. S. Ramachandran, K.-L. Wright, L. Zheng, P. Navaney, M. Naveed, B. Krishnamachari, and J. Dhaliwal, "Trinity: A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2019, pp. 227–235.

[215] B. Huang, R. Zhang, Z. Lu, Y. Zhang, J. Wu, L. Zhan, and P. C. Hung, "Bps: A reliable and efficient pub/sub communication model with blockchain-enhanced paradigm in multi-tenant edge cloud," *Journal of Parallel and Distributed Computing*, 2020.

[216] G. Bu, T. S. L. Nguyen, M. P. Butucaru, and K. L. Thai, "Hyperpubsub: Blockchain based publish/subscribe," in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, IEEE, 2019, pp. 366–3662.

[217] Y. Zhao, Y. Li, Q. Mu, B. Yang, and Y. Yu, "Secure pub-sub: Blockchain-based fair payment with reputation for reliable cyber physical systems," *IEEE Access*, vol. 6, pp. 12 295–12 303, 2018.

[218] G. Yang, C. H. Tan, Q. Huang, and D. S. Wong, "Probabilistic public key encryption with equality test," in *Cryptographers' Track at the RSA Conference*, Springer, 2010, pp. 119–131.

[219] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[220] N. Zupan, K. Zhang, and H.-A. Jacobsen, "Hyperpubsub: A decentralized, permissioned, publish/subscribe service using blockchains," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos*, 2017, pp. 15–16.

[221] Hyperledger, *Hyperledger fabric v2.2 documentation*, Last accessed 2020-10-22, 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/.

[222] Hyperledger, *Hyperledger fabric v1.4 documentation*, Last accessed 2020-10-22, 2019. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/.

[223] Hyperledger, *Hyperledger besu documentation*, Last accessed 2020-10-22, 2020. [Online]. Available: https://besu.hyperledger.org.

[224] T. L. Foundation, *Hyperledger Caliper*, Last accessed 2020-11-5, 2020. [Online]. Available: https://www.hyperledger.org/use/caliper.

[225] P. Domingues, P. Marques, and L. Silva, "Resource usage of windows computer laboratories," in *2005 International Conference on Parallel Processing Workshops (ICPPW'05)*, IEEE, 2005, pp. 469–476.

Appendix A: Personal Computer Survey

The ChainFaaS platform is based on the assumption that personal computers are highly underutilized. However, there are no recent studies to verify this premise. In 2005, Domingues et al. [225] studied the resource usage of Windows 10 machines from classroom laboratories. The results show an average CPU idleness of 97.9%. Although this study shows much wasted computational capacity, it focuses only on laboratory computers, and it was conducted about 15 years ago. Therefore, to provide stronger support of the idleness assumption, we conducted a survey and asked participants to report their computer's CPU utilization and unused memory when running their regular programs.

In this survey, we first gathered some demographic information about the participant, such as their age, gender, and level of education. The rest of the survey questions were aimed to find the amount of computational power that is not being used. The participants first state the number of personal computers that they own. Then they answer the following questions for each of their computers:

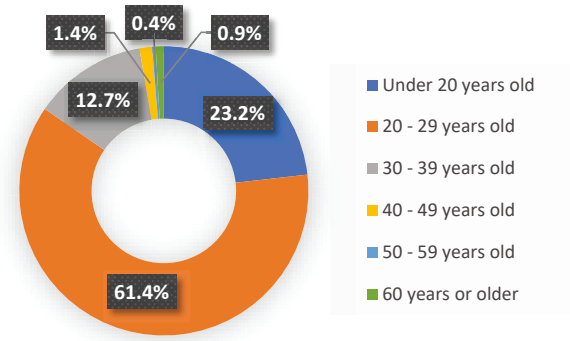
- On average, how many hours per day do you work with your personal computer?
- What is the primary operating system that you use on your computer?
- What is your computer's average CPU utilization when running regular programs?
- What is the amount of unused memory, in gigabytes, on your computer when running regular programs?

About 700 people participated in this survey, mostly university students of computer science or computer engineering. Fig. A.1 shows the demographic information of the participants. Most of them are 20 - 29 years old, male, with a high school diploma. In total, we gathered information about the utilization of about 1150 computers. The gathered data in this survey is available publicly on GitHub [1].

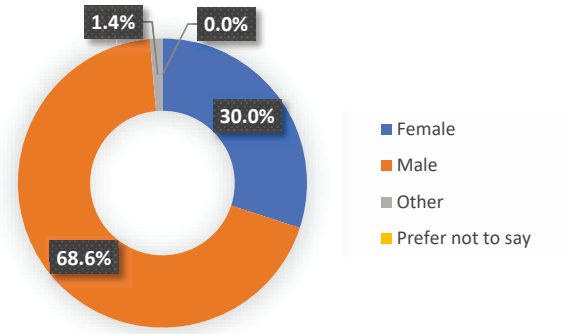
To get a better understanding of the gathered data, the average usage time per day, CPU utilization, and unused memory of the participants' main computer for the education groups and age groups are shown in Fig. A.2 and Fig. A.3, respectively. Since there were less than 10 participants with an education level of less than a high school diploma, it was not possible to calculate a reliable average value for this group. The situation is similar for the age groups of 50-59 years old, and 60 years and older.

It is also interesting to see the popularity of different operating systems among the participants. About 73.8% of the computers are running on Windows, which shows the popularity of this operating system. 16.6% run on macOS, 8.4% on Linux, and the remaining 1.2% run on other operating systems.

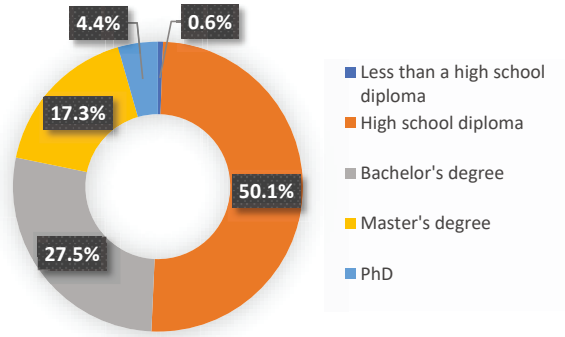
[1] https://github.com/pacslab/PC-Survey



(a)



(b)



(c)

Figure A.1: Demographic information of the participants in the personal computer survey. (a) Age (b) Gender (c) Highest level of education

Table A.1: Average usage time, CPU utilization, and unused memory for the main computers and all the computers in the survey.

	Time (Hours)	CPU (%)	Memory (GBs)
Main Computers	6.34	25.72	8.6
All Computers	4.53	24.54	7.91

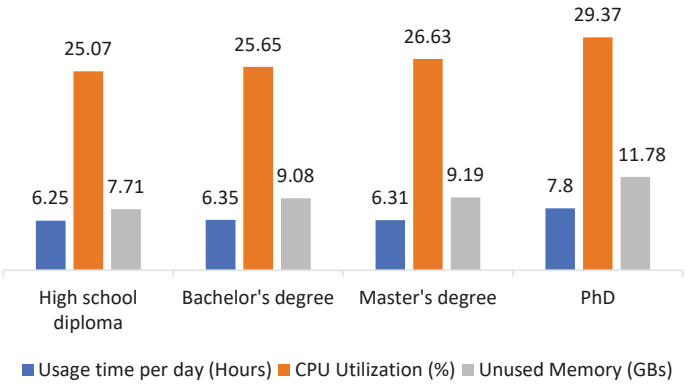


Figure A.2: Average main computer's usage time per day, CPU utilization, and unused memory for each of the education groups.

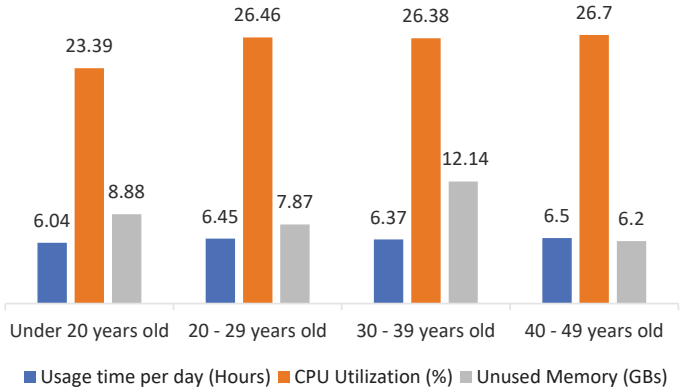


Figure A.3: Average main computer's usage time per day, CPU utilization, and unused memory for each of the age groups.

Finally, Table A.1 shows the average usage time per day, CPU utilization, and unused memory for the main computers and all the computers in the survey. It can be seen that the average CPU utilization for all the personal computers in this survey is 24.54%. Since most participants are computer engineering and computer science students, the average CPU utilization for a more general audience is expected to be even less than this number. This result confirms our initial assumption that personal computers are highly underutilized.