Blockchain based remote voting system: A performance perspective

by

Sushil Paneru
B.Tech., National Institute of Technology, Warangal, India, 2017

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Blockchain based remote voting system: A performance perspective

by

Sushil Paneru
B.Tech., National Institute of Technology, Warangal, India, 2017

Supervisory Committee

Dr. Sean Chester, Supervisor
(Department of Computer Science)

Dr. Yvonne Coady, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Sean Chester, Supervisor
(Department of Computer Science)

Dr. Yvonne Coady, Departmental Member
(Department of Computer Science)

ABSTRACT

Although cryptography based remote voting protocols have been researched since 1981, most of the previous protocols [9], [5], [13] assume the existence of public bulletins or, in other words, a publicly readable, tamper-proof, append-only log. As blockchain or distributed ledger technology (DLT) offers properties like irreversibility, transparency and decentralization, it is suitable for realization of public bulletin board for the voting system. We see a gap in the research of blockchain based voting systems because there either exists work on just the protocol aspect of the voting system or the performance aspect of the blockchain. As blockchain is a general purpose tool, we believe that there lies opportunities for micro-optimizations that could specifically benefit the voting system. This ushered us to focus our effort on the performance aspect of integration of voting protocol with blockchain. Hence, in this thesis, we first introduce a homomorphic encryption based voting protocol that uses blockchain, Hyperledger Fabric (HLF), as bulletin board. The protocol is designed such that it leverages the transaction processing characteristics of underlying DLT. We then created an experiment where we designed a smart contract, set up a blockchain network and exposed the system to 40k concurrent voting transactions to profile the code of HLF. From the profile data, it was found that execution of cryptographic operations constitutes most of the transaction processing time. This led us to benchmark cryptographic libraries for SHA256 and digital signature algorithm and integrate the faster library into HLF for better performance. We also found that the transaction manager of HLF does not need read-write locks to ensure transaction isolation in special scenarios, which alleviates the performance drop due

to lock contention. Altogether we were able to improve the throughput and latency of the baseline system by more than 30%. Lastly, we make a comparison between public and permissioned DLT based remote voting system and discuss the suitability of permissioned blockchain for the application of voting systems.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

my family for encouraging me to pursue graduate studies and supporting me through thick and thin.

Dr. Sean Chester, for providing this opportunity to initiate research and having faith in me.

Dr. Yvonne Coady, for sharing her knowledge to improve this thesis.

Bhumika Bhatta, for supporting and encouraging me in difficult times.

You are what you believe in. You become that which you believe you can become.
- Bhagavad Gita

DEDICATION

To my late mother who dedicated her whole life for my family.

Chapter 1

Introduction

Remote voting is a system where a voter casts a vote without being physically present at the ballot station. The votes can be cast either via mail or internet. With globalization, people are spread more than ever and such a dispersed group of people might become barred from voting if they are not present in their home state. Also, at the time of a pandemic due to the fear of spreading the contagion, it is quite difficult to make ballot station based voting functional. As per [7] in ballot station based voting system, the need to travel to polling station imposes geolocation constraints on the voters which causes lower voter turnout. Because vote-by-mail reduces the geolocation dependency, it explains the higher voter turnout in 2020 US election. For example, during the 2020 US elections, a record number of the ballots, 65%, were cast by mail and voter turnout was the highest in the last 120 years [28]. Even though geolocation dependency is removed, one of the problems with voting by mail is that it is slow. In states where pre-counting before election date was not allowed, like Wisconsin, Pennsylvania, etc, it was estimated it would take 5 - 10 days for vote counting to complete. Not only was the process slow, there were a series of allegations regarding fairness of the election being made by the opposition as there is no transparency to the system. With increased accessibility of the internet to the people, one of the main questions that arises in our society is that when banking, ticket booking are online, why not election?

There already exist internet-based remote voting systems in countries like Estonia, Switzerland, Canada [18] and even though such remote voting allows a voter to cast a ballot via the internet, the authority over election is centralized and is not transparent. Without decentralization and transparency, it is difficult to do an end-to-end

verification of the voting process. Because of this, it is not very different from a ballot station based voting system. Threshold signature scheme and threshold encryption scheme with zero knowledge proofs are used in previous work [9], [5], [13] to decentralize the authority and make the process verifiable but these work make assumptions on the existence of tamper proof append only public bulletins. These public bulletins can be implemented using the traditional centralized approach with a typical relational database management system, but it does not ensure tolerance against Byzantine faults, a class of faults presenting different symptoms to different observers [15]. For example, a dishonest political influencer's conspiracy or faulty hardware can compromise the correctness of the system and make the election untrustable. Blockchain as an underlying data store can come to rescue to track the aforementioned faults. It can be said that blockchain offers properties like irreversibility, traceability and decentralization, which allows to ensure transparency of state change in the underlying data. Despite blockchain providing aforementioned properties, the performance of the system can be quite slow [20] as it is not a mature technology. The throughput of public blockchain like Bitcoin and Ethereum is around 6 - 7 transactions per second [11] and if we were to use such a system to conduct an election, it could take tens of days just to ingest the votes. This makes a strong case that blockchain based voting system should prioritize improving the throughput and latency of the system. Even though there exist previous work [38], [19] which solely focus on general performance optimization of the blockchain, we believe there lies a room for micro-optimization of HLF that can specifically benefit the voting system; one of such specific optimization for the voting application is explained in the section 5.4. Hence, in this thesis, we primarily focus on performance aspect of integration of voting protocol with underlying blockchain.

For studying the performance aspect of the voting system, we required a voting protocol as our first step. So, we first introduced a voting protocol quite similar to [5]; the major difference being that for regional security layered encryption is not used and ballots are stored in blockchain. We then chose Hyperledger fabric as our choice of blockchain, the primary reasons being that it is a permissioned blockchain which is supposed to be faster than public blockchain and very few voting systems have been implemented on top of permissioned blockchain. As our next step, we explored ways to improve the performance of the baseline system we created and in order to so, we profiled the code of HLF to understand about its subsystems. As part

of our task for code profiling, we created an experiment where we designed a smart contract for the voting protocol, set up blockchain network and exposed the system to 40k concurrent voting transactions using an open source blockchain benchmarking tool known as Hyperledger Caliper. We found benchmarking a blockchain system in a single machine is resource intensive which led us to modify the internal working of Hyperledger Caliper to simplify the task of profiling HLF —a new feature was added that could skip either endorsement or commit phase of HLF. We then used the profile data to break down the execution time of different operations, which was used to come up with optimizations like expediting cryptographic operations and improving concurrency control mechanism for improving the latency and throughput of the voting system. It was found that performance metrics are dependent on execution time of cryptographic operations which led us to benchmark cryptographic libraries for SHA256 and digital signature algorithm and integrate the faster library into HLF for better performance. We also found that the transaction manager of HLF does not need read-write locks to ensure transaction isolation in scenarios where a transaction reads or writes only a single key; this insight was used to design single keyed transactions which alleviated lock contention between endorser and commit thread, and improved of the performance of the system; HLF transaction workflow is further explained in section 3.2.1. Altogether we were able to improve the throughput and latency of the system by more than 30%. Lastly, we also make a comparison between public and permissioned DLT based remote voting system on the basis of cost, speed and security, and discuss the suitability of permissioned blockchain for the application of voting systems. Hence, in this thesis, we show that not only electronic remote voting systems can be made transparent and verifiable by the use of blockchain, we also analyze its performance and show how the performance of the underlying blockchain can be improved and make the voting process more feasible.

1.1 Motivation

After the Bitcoin whitepaper was published in 2008 [31], blockchain has been regarded as a technology that can decentralize different systems of society and bring more transparency into the system. Non-repudiation, high availability, and irreversibility are some of the properties of blockchain due to which it has been widely adopted to solve a variety of problems in different domains [23], [39] like healthcare, supply chain management, governance, etc. For example, government offices

of China use blockchain to store the records related to public work to gain trust of public and reduce the friction between different departments [23]. The long studied voting problem can also benefit from blockchain; researchers have proposed numerous protocol design and system design based on blockchain to implement a secure voting system. But the problem with blockchain based voting system is that they are slow and not feasible [17], [22], [27]. We have observed that either there exist work on just the protocol aspect of the voting system or the performance aspect of the blockchain, but lacks the study of performance aspect of voting protocol coupled with blockchain. In the context of performance optimization, we believe that when an application is built using general purpose tool like blockchain, there always lies opportunities for micro-optimization for improving the performance of the system. For example, in a general purpose database, characteristics of internal data structures or algorithms can always be tweaked for the use case of the application being designed. Hence, we believe that blockchain being a general purpose tool, there lies an interesting area of research for performance analysis with respect to a particular use case i.e., in this thesis, voting application. Such improvement in the performance of a slow system is important because it makes the implementation of voting system on top of blockchain more feasible, which in turn can make the election process more transparent and trustable. Thus, if we are to ensure that the system can handle the volume and velocity of incoming votes of the election day, it is important to study the performance aspect of the system for better feasibility. This led us to analyze the performance of the underlying blockchain and address the bottlenecks specific for the application of voting system; the solutions to the bottlenecks can also be applicable for other use cases like supply chain management, healthcare management system, etc. We show how replacing the existing cryptographic library with faster library can result in 20% improvement in throughput. We also show how modifying concurrency control mechanism can lead to improvement of throughput and latency of the system by more than 10% without affecting the correctness of the system.

1.2 Contribution

- As part of performance analysis of the voting system, we needed a voting protocol. So, we implemented blockchain based voting protocol which is similar to [5]; the major difference being that for regional security layered encryption is not used and ballots are stored in blockchain. This protocol with blockchain

makes the system verifiable, transparent and confidential.

- As per our observation, performance analysis of blockchain based voting system has not been done for a voting system. We profiled the code of HLF for performance analysis to gain more insights about the system's characteristics. In order to do so, we created an experiment where we designed a smart contract for the voting protocol, set up blockchain network and exposed the system to 40k concurrent voting transactions. The profile data was used to study the effects of crypto operations, golang runtime, IO, etc.

- As performance benchmarking of a blockchain system is resource intensive, we added a new feature to an open source benchmarking tool, Hyperledger Caliper, to skip endorsement or commit phase of HLF as per requirement. This was done to simplify the process of benchmarking HLF in a single machine and make the process less resource hungry.

- From profile data, we found that cryptographic operations dominate the execution time of transaction processing. This led us to search for faster implementation of SHA256 and digital signature algorithm and benchmark them. We then integrated the faster library into HLF for better performance. Our approach of speeding the crypto operations is different from [38] where the authors concentrated on parallelizing tasks involving crypto operations.

- From profile data and analysis of logs, we found that there's lock contention between the endorser and commit thread. This led us to make an architectural change to concurrency control mechanism i.e. remove the read-write lock and do safety analysis for improving the performance of the system without affecting the correctness of the system.

- As there is option to choose private or public blockchain for designing a voting system, we made a comparison of their efficacy for the purpose of voting application on the basis for cost, speed and security.

1.3 Agenda

Chapter 1 includes a brief introduction about the relevance of decentralized remote voting and the motivation behind taking this thesis and the contributions.

Chapter 2 provides the literature review for the voting systems. It provides brief details voting protocols proposed before and after introduction of blockchain. It also provides pros and cons of relevant work.

Chapter 3 explains about the preliminary concepts and technologies required to understand the work done as part of this thesis.

Chapter 4 demonstrates about the voting protocol we have used. This chapter explains how different entities interact with each other and the assumptions we have made for the system.

Chapter 5 explains the result of performance analysis and our efforts to improve the performance of the system

Chapter 6 includes the discussion about the experimental setup and the result we observed for the experiments. This chapter contains graphs that show the improvement of throughput and latency caused by the optimization we made.

Chapter 7 presents discussion about the benefits of permissioned blockchain for implementing voting system and cost benefits introduced as part of the optimizations we did.

Chapter 8 concludes the findings made as part of this thesis and also contains a discussion about possible future work to extend the scope of this thesis.

Chapter 2

Related Work

In this chapter we discuss the past relevant work on the remote voting system. As there has been work on voting protocol since 1981, we first talk about papers that explore different cryptographic primitives to come up with a voting system that has properties like confidentiality, privacy, uncoercibility, verifiability. These works are used as inspiration to design a voting protocol which we use later for the performance analysis of the voting system. We then introduce papers related to blockchain based voting systems; these were helpful in understanding their limitations and designing a scope for this thesis. As our center of attention is the performance of blockchain based voting system, we also introduce papers that attempt to improve the performance of blockchain framework we are concerned about i.e. Hyperledger fabric. These papers were quite resourceful in understanding different kinds of optimizations that have been applied to HLF and also determining what more can be done to improve its performance.

A cryptography based voting protocol was first introduced in 1981 by Chaum [8] which made the use of mix-nets and digital signature to design a election system. This work sprung the research on design of voting protocol using different cryptography primitives. One of the most notable works that uses the mixnets is [9] which ensures public verifiability and receipt-freeness. Every operation from voter and registrar is publicly verifiable and makes assumption of append-only irreversible append only log. The other notable and implemented work is Helios [1] which also uses mixnets and Elgamal encryption scheme. Helios seems to have public confidence because of the fact that it has been used to conduct elections at a university [2] and at IACR. Shoenmakers [35] used publicly verifiable secret sharing schemes (PVSS) and homomorphic

property of Elgamal encryption scheme to show how a simple voting protocol can be designed. Baudron [5] showed how zero knowledge proofs, homomorphic property of Paillier encryption and vote encoding can be used to implement a multi candidate encryption. It also shows the protocol can be made distributed to handle national scale elections. All the aforementioned protocols make assumptions that there exist append-only, irreversible public bulletins which can actually be implemented using blockchain. Apart of blockchain as a solution, Volkamer et al [40] addressed a serious security issue in the client side. He proposed a simple voting schema and analyzed the security client side of the remote voting and claimed that one of the major hurdles to remote voting is unsafe client application or the hardware the client is using. Volkamer suggested the use of a trusted computing base [41] and secure operating base to implement the cryptographic operations for the client.

After the original Bitcoin whitepaper [31] in 2008, a lot of problems have been tried to solve via blockchain and one of them is a remote election system. Most of the proposed work follow a general pattern and that is use encryption to conceal a vote using homomorphic encryption scheme, store it in a blockchain and use the homomorphic property of the encryption scheme to tally votes. McCorrey [27] proposed and implemented the first remote voting system using Ethereum as a blockchain platform and Elgamal encryption scheme. They designed smart contracts, a programmable unit of blockchain, to simulate an election authority and performed data sharing via writing data to the ledger which makes it slow and not suitable for large scale elections. Hjálmarsson et al [22] also designed their solution on top Ethereum but does not provide voter verifiability. [6], [12] are some of the voting system designed on top of bitcoin but due to high latency and low throughout of Bitcoin and Ethereum, these systems are also not suitable for large scale elections. Liu [26] used blind signatures to design an end-to-end verifiable voting protocol irrespective of blockchain platform used but provides no implementation detail or performance analysis. [34] treats secure voting as a subset of secure multi-party computation and uses secret sharing to come up with a fraud detectable voting protocol. This work works for only 2 candidate election system and also provides no detail about implementation and its performance. [17] uses threshold blind signature to design a protocol which is implemented on top of a permissioned blockchain platform, hyperledger fabric. Even though it provides a limited level of performance analysis of the underlying blockchain, we believe underlying DLT, HLF, can be analyzed further to improve the

overall performance of the system.

The blockchain platform whose performance analysis we will be doing is Hyperledger fabric, a permissioned distributed ledger. Thakker et al [38] and Gorenflo [19] have notably provided general optimization that can speed up the system performance. Thakker et al [38] showed general behaviour of the system like how block size, endorsement policy, type of key value store the throughput of the system. They also pointed out that sequential operation of validation system chaincode (VSCC), the major verification phase, is the major bottleneck. They introduced parallelization and caching to VSCC workflow to gain more than 10× speedup. We believe the performance of VSCC can be further improved by examining the performance of crypto libraries that are used in HLF. Gorenflo [19] introduced aggressive optimizations and claimed more than 5× speed-up but many of the suggested architectural changes were not quite practical. For example, they suggested storing everything in memory to prevent disk writes but most of the applications require persistence of the data. They successfully increased the CPU utilization of the ordering phase by parallelizing transaction processing workflow. They pointed out that marshalling and unmarshalling operations are expensive and used caching mechanisms to avoid repetition of the marshalling and unmarshalling operations. There exist works [36], [29] on improving the transaction isolation mechanism used in Hyperledger Fabric. [36] showed how the use of MVCC can increase the failed the transactions and proposed transaction ordering mechanism for conflicting transactions which resulted in the reduction of number of failed transactions. [29] showed that lockless transaction isolation mechanism results to performance improvement. Our approach for removing read-write lock to improve concurrency of transaction is quite similar to their work, and we discuss the similarities and differences in Chapter 7.

Either there exists work on protocol design without giving much emphasis on performance of underlying blockchain or there exists work on just performance optimization on blockchain, hence, we first propose a protocol similar to [5] and then perform optimization as per the needs of the protocol. As part of our work on performance analysis of Hyperledger fabric (HLF), we show that performance of validation phase can be further improved by using faster crypto libraries. We then show concurrency control mechanisms can be tuned as per the nature of transaction to improve the performance of the system.

Chapter 3

Background

The purpose of this chapter is to describe the techniques and terminologies required to understand the future chapters that dives into internals of HLF and voting protocols. We discuss blockchain in general in section 3.1 and delve into internals of Hyperledger Fabric in section 3.2. In section 3.3 discussion on Hyperledger Caliper, is done to understand the benchmarking we did as part of performance analysis. In the last two sections 3.4, 3.5, we briefly discuss cryptographic primitives which are required to understand the design of the voting protocol.

3.1 Blockchain

According to [3], *Blockchain*, also called *distributed ledger*, is defined as immutable ledger for recording transactions, maintained within a distributed network of mutually untrusting peers, where peers execute a *consensus protocol* to validate transactions, group them into blocks, and build a hash chain over the blocks. In simple terms, it is a way to replicate computation and data amongst multiple nodes to ensure immutability of data by achieving consensus amongst the majority of the nodes (51%) in the network.

Each node in the network maintains a copy of the log which is stored as a chain of blocks. Each block stores some number of transactions which are cryptographically signed and the hash of each block is stored for integrity checks. In figure 3.1, it can be seen that nodes in the network are connected to each other and each of the nodes maintain the same copy of verified data. The transactions are stored as blocks to utilize the maximum available bandwidth of the system. The ordering of the

transactions within a block is determined via a consensus protocol and the order of transactions is important as it can define the state of the data. At any particular point of time when all transactions are applied to the initial state (genesis), we get a current state of the facts stored in the blockchain which is also known as the world state. Transactions are run against world state which is stored in a fast key value store for better performance. It can be said that the correctness of the current state of data which is stored in a fast key value store is backed up by a log of transactions stored as a chain of blocks in flat files. The clients interact with the blockchain by invoking smart contracts. Smart contracts contain the business logic that a transaction wishes to run, and it determines how the state of the blockchain should be transformed.

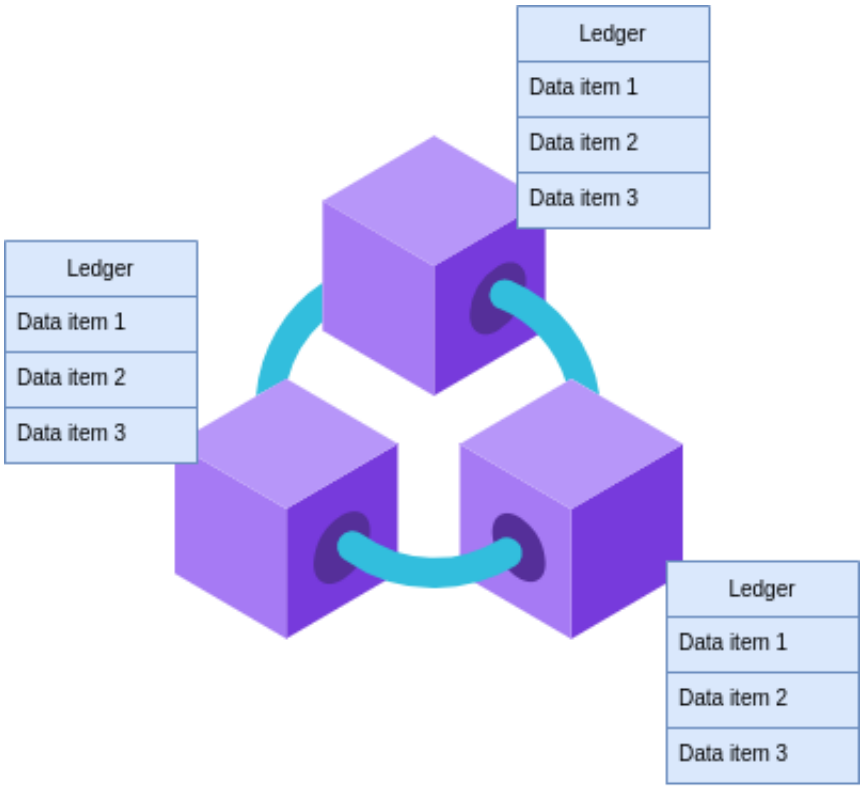


Figure 3.1: High level architecture of blockchain; blue cubes represent nodes in a blockchain network and the same ledger is replicated amongst all the nodes.

Blockchain was first used to create cryptocurrencies like Bitcoin and Ethereum [37] to decentralize the finance industry. The fact that blockchain is distributed, immutable and fault tolerant has made it quite popular in the field where discrepancy arises amongst the different organizations such as supply chain management, finance, healthcare sector, e-governance [30]. Some of the popular use cases of blockchain are

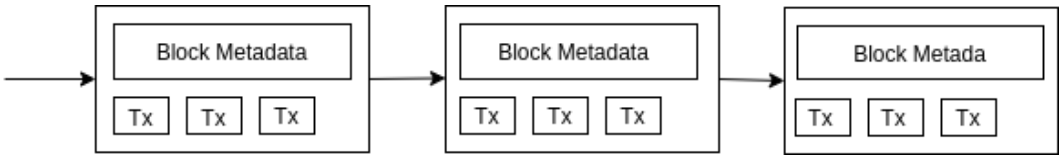


Figure 3.2: Chaining of blocks of transactions

music royalty tracking, cross border payment, real estate data store platform [30]. In all these use cases, the participating organizations cannot fully trust each other, and in such scenarios blockchain is used as a tool to bridge the gap of trust. This is because the data is shared between all the entities and any change to it has to go through a consensus mechanism. This allows the network to detect the misbehaving organizations, and they can be held accountable for their actions.

In this paragraph we demonstrate a scenario where there might arise distrust due to opaqueness and mutability of the data. Government offices are tasked to administer different kinds of government work and maintain data related to government projects, property ownership, etc. In the current approach, the data is stored in a centralized database which is opaque to the public. There might arise a scenario where a wrongly motivated entity, person or an organization, influences the officials of the government office to change any data of their interest as per their advantage without letting the public know about it. In the long run, this can be detrimental to an individual or the whole community, and public distrust in public offices may arise. If the data is stored in blockchain in which civil organizations are part of it, such illegal mutation of data is not possible due to chain of hashes, and also it is possible to detect such miscreant activities by looking at the log of consensus protocol. In this way, the usage of blockchain can fix the issue of distrust.

There are primarily two types of blockchain and they are *permissionless or public blockchain and* permissioned or private blockchain. They are explained below in detail.

3.1.1 Public or permissionless blockchain

As per Swan [24], *permissionless or* public blockchains are shared by all network participants with equal importance, updated by miners, monitored by everyone and controlled by no one. Bitcoin and Ethereum are examples of public blockchain as

anyone can join the network and become a miner. Permissionless blockchain provides the strongest level of decentralization and is open to everyone as a user does not need any requirement to join the network. As there is no requirement of identification of the user, computationally expensive processes like proof of work [31] are used to achieve consensus. The process is so expensive that as per Baraniuk [4], in one year the power consumption of Bitcoin mining, which uses proof of work mechanism, is equal to the total power consumption of Switzerland. The block formation time is also quite long due to which blockchains like Bitcoin and Ethereum have throughput close to 6–10 transactions per second and the confirmation time for transaction completion is more than 10 minutes. Such lower transaction per second is the reason why public blockchain is not suitable for high throughput application. This is the primary factor that ushered us not to choose a public blockchain for designing a voting system. Hence, it can be said that permissionless blockchain provides a higher level of decentralization at the expense of throughput and latency.

3.1.2 Permissioned or Private blockchain

In *permissioned* blockchains, membership is required to join the network. The nodes in the network have different roles like membership nodes, transaction ordering nodes, comitter nodes, etc. Some of the examples of permissioned blockchain is Ripple, Hyperledger fabric, Hyperledger sawtooth. Permissioned blockchains came into rise to meet the requirements of the high throughput applications in the industries like supply chain management, healthcare, etc. where only a few entities are required to share the data and involve in the validation process. Private blockchain are expected to be inherently faster than public blockchain because the replication of computation and data is not done on all the nodes in the network. As the nodes in the network have identities, Byzantine-fault tolerant consensus algorithms like Raft [32] can be used instead of expensive Proof-of-work. Permissioned blockchain provided performance at the expense of partial decentralization. With proper implementation of service access, role management permissioned blockchain can provide all the features that a permissionless blockchain can provide.

3.2 Hyperledger Fabric

Hyperledger fabric (HLF) [1] [3] is a modular tool to implement scalable permissioned blockchains. It is an open source project maintained by the Linux foundation. It is quite popular in industry, with production uptake by large corporations like Walmart and Deloitte. The main reasons for the popularity of HLF is that it is modular and extensible. Consensus protocol, transaction validation mechanism can be changed as per the requirement of the system. Unlike other blockchain platforms where smart contracts are written in a domain specific language, HLF provides options to write smart contracts in general purpose languages like Java, Golang and Javascript. As writing smart contract logic can become quite complicated, the ability to implement them in such popular languages has boosted HLF's adoption rate.

The main objective of all the blockchain systems is to implement a state machine replication via active replication i.e. request for the change of replicated data is processed by all the nodes. First, the transactions are ordered via a consensus protocol, and then they are executed sequentially by all the nodes in the network. This is called order-execute architecture which is used in almost all the blockchain, be it permissioned or permissionless systems. The novelty of HLF is that it uses a hybrid model of replication which gave rise to execute-order-validate architecture. It is claimed to be hybrid as it uses both passive and active replication. The transactions are executed in selected nodes, which is passive replication, and then they are committed in all the nodes at once which is active replication. Hence, in HLF, a transaction has to go through three phases: endorsement phase, ordering phase and commit/validation phase, which are further explained in section 3.2.1. As explained in [3], the advantages of the execute-order-validate over traditional order-execute model are:

- **Better Concurrency**: As a transaction can be in one of the three phases and as each phase can be executed on 3 different machines, the capacity to handle concurrent transactions increases. Not only the concurrency, the ability of the system to handle long running transactions also improves. If a transaction takes a long time to run, it will not block other transactions from running in the execute-order-validate model whereas in the order-validate model, the whole system might get blocked. In HLF, a non terminating transaction can be aborted after a timeout and as transactions are first run concurrently in

[1] https://github.com/hyperledger/fabric

an isolated manner, the behaviour of one transaction won't affect the others. This does not happen in the order-execute model as transactions are executed after they are ordered which is why special provisions need to be made. For example, Ethereum uses the concept of gas which is basically a measurement of instruction execution. If the number of instructions of a transaction exceeds a threshold, the transaction is aborted.

- **Modularity**: As execution, ordering and validation are separate, it is easy to make the system modular. There's no such one-for-all algorithm that fits in all kinds of scenarios. For example, if the trust model of a system is different, BFT consensus algorithms are not required at all. Similarly, if the system requires custom logic for transaction validation, system designers can easily implement them in modules and plug it into the system. Hence, as execute-order-validate architecture is more modular than order-execute architecture, it allows designers the flexibility to change the behaviour of the system as requirements change.

- **Better non-determinism control**: In case of execute-order-validate architecture, if a transaction is non-deterministic, it is rejected during the validation phase as the result of execution in different nodes will have different results. In order-execute architecture, non-determinism can cause forks of the chain if they are not detected. Hence, the system has to enforce a developer to write smart contracts in domain specific language which is not deterministic. General purpose languages cannot be used as the system designer has to trust the smart contract writer to not bring any kind of non-determinism.

3.2.1 Transaction Flow in HLF

Figure 3.3 gives the high level view of the transaction flow. The numbered workflow is explained in the sections below.

- **Endorsement Phase**: In execute-order-validate architecture, the execution of a transaction takes place in the endorsement phase. As per the endorsement policy, a client sends input to a set of peers; this is denoted by (1) in figure 3.3. Each peer executes the corresponding chaincode, another term for smart contract. The execution takes place against the state maintained by the peer in a statedb which then produces readset and writeset. Readset and writeset are used as part of transaction validation; readset is basically a set of keys which

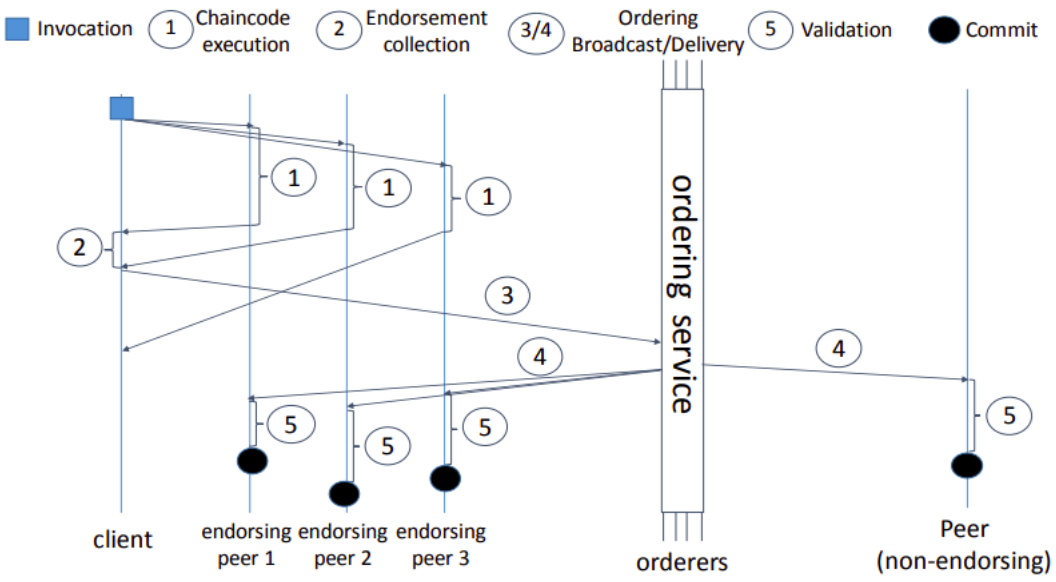


Figure 3.3: Interaction diagram for transaction workflow in HLF.

needs to be read and writeset is the set of keys that gets updated in the world state. If the transaction does not violate any condition then the result of the transaction signed by the peer, and the transaction is also said to be endorsed by the peer. It is important to note that executing the transaction first simplifies the design of HLF because non-deterministic code can be easily caught.

- **Ordering Phase**: After the transaction proposal is processed by the peers i.e. endorsed, client collects this endorsed response of transaction proposals from all the peers; this is denoted by (2) in figure 3.3. The endorsed transaction is sent to the ordering service, denoted by (3) in the above figure. The main purpose of the ordering service is to fix the order of the transaction, create a block of transactions and broadcast the block to all the peers in the network. The ordering service is not concerned about the semantics of the transaction. The integrity of the whole blockchain network is dependent on how ordering service reaches consensus. There are a number of ways in which consensus of the transaction order can be reached. They are termed as single node based ordering, Apache kafka based ordering and raft based ordering. Single orderer is used for testing purposes as it offers no decentralization and kafka based ordering is deprecated as it also does not offer true decentralization. The recommended

approach for consensus is using the raft algorithm which is crash fault tolerant. As ordering nodes from each organization participates as part of the raft algorithm, the ordering phase is said to be decentralized. As raft is not BFT, the network cannot tolerate Byzantine faults. It is important for the ordering service to be Byzantine fault tolerant as the sole purpose of a blockchain is to provide a functional network when the participating organizations cannot fully trust each other. Hence, the HLF community is working towards providing a native implementation of the BFT consensus algorithm.

- **Validation Phase**: A peer receives a block either directly from the orderer or another peer via gossip broadcast; denoted by (4) in the figure 3.3. Peer performs a series of validation for each transaction in the block; denoted by (5) in the figure 3.3. First, the endorsement policy of each transaction is checked against the signature present in the transaction proposal. This validation is done by a configurable component called validation system chaincode (VSCC) which can be replaced by the system designer as per the requirement. After this, the readset and writeset are validated as part of multiversion concurrency check (MVCC). As part of MVCC, overlap between the readset and writeset is checked amongst the transaction from a block. If they overlap one of the transactions is aborted whereas the rest of the transactions are said to be validated. MVCC validation is explained in detail in section 7.1.2. After the validation of the transaction is complete, the state database is updated with the writeset of all transactions and the block is written to a file that contains all the other blocks.

3.3 Hyperledger Caliper

Hyperledger caliper [2] is an open source project that is used to measure the performance metrics like throughput and latency of a blockchain system. It is very modular in design as it can be extended to support any blockchain and as of now it supports HLF, Sawtooth, Ethereum, etc. It is also configurable; for example, the settings allow configuring the deployment mode i.e. distributed or single node deployment, number of transactions to be sent and the rate at which transactions are to be sent. As blockchains are mostly used as transactional system, their behaviour is mostly influenced by the number of transaction they can process in a certain amount of time

[2] https://hyperledger.github.io/caliper/

which is why Caliper offers different kinds of rate controllers like fixed rate controller, linear rate controller, feedback based, etc to study the behaviour of the system at different rate of incoming transaction.

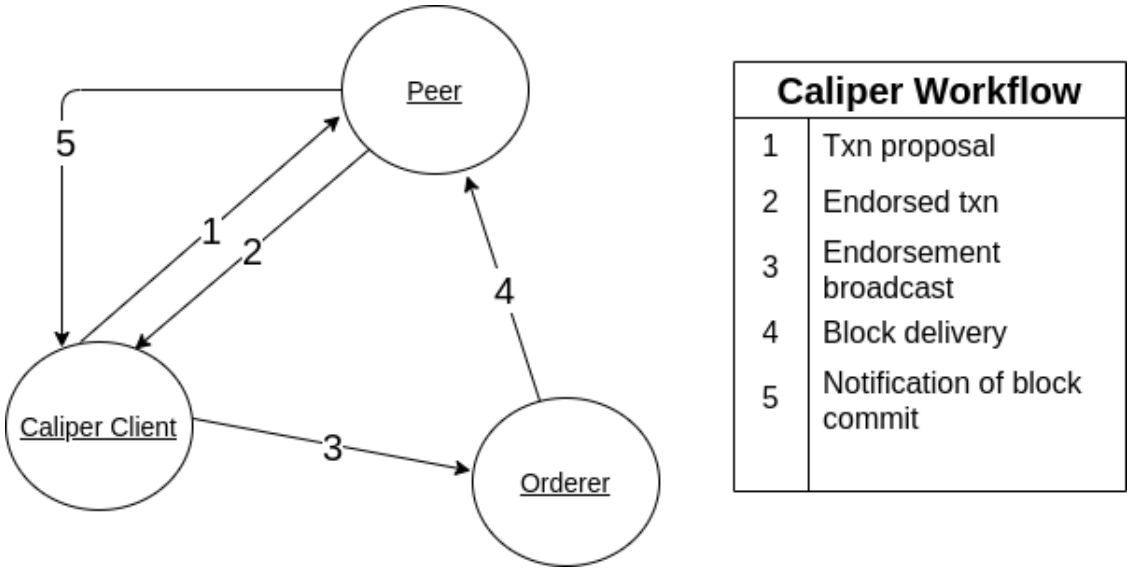


Figure 3.4: Hyperledger Caliper workflow for HLF. Circle represents nodes in the network whereas arrows with numbers represents sequence of interactions

Figure 3.4 describes the workflow of the hyperledger caliper. As per the requirement for scale of transaction to be sent per second, caliper creates multiple clients. These caliper clients send transaction proposals, start a timer and also register for an event where they get notified when a new block is created. The clients keep track of the timer to calculate the latency and throughput of the system. After the transactions are executed, they are received by the client and sent to the ordering service which then creates a block and sends it to the peer. When the peer commits the block, a notification is sent to the caliper client which then updates the timer and calculates the latency and throughput.

3.4 Paillier Cryptosystem and Homomorphic Property

Paillier cryptosystem is a probabilistic public key cryptography algorithm. The security of this encryption scheme is dependent on hardness for solving the decisional composite residuosity problem. Like any encryption scheme, it has 3 algorithms, and

they are described below -

- **Key Generation**: Let N be a modulus $N = pq$, which is a product of prime numbers p, q such that $gcd(pq, (p-1)(q-1)) = 1$. Let g be an integer of order a multiple of N modulo N^2 and let $\lambda(N) = LCM((p-1)(q-1))$, where LCM means lowest common modulo. The public key is $P_k = (N, g)$ and the secret key is $S_k = \lambda(N)$.

- **Encryption**: In order to encrypt a message m, a random number r is chosen and ciphertext is computed as $c = E(P_k, m) = g^m * r^N \mod N^2$

- **Decryption**: If c is the ciphertext then it is decrypted as $m = L(c^\lambda \mod N^2) * \mu \mod N$ where $\mu = (L(g^\lambda \mod N^2)^{-1})$

Paillier cryptosystem is also best known for its additive homomorphic property. If $E(k, x)$ is encryption of x under key K then additive homomorphic property is when $E(k, x) * E(k, y) = E(k, x+y)$. Such a relation can be seen with the encryption function as described above.

3.5 Zero Knowledge Proof

Zero knowledge proof is a method which can be used by a party (P) known as prover to prove that it knows a value x to a party (V) also known as verifier without giving out any information about x. The pair (P, V) can be modeled as turing machine where P works in probabilistic polynomial time and V works in deterministic polynomial time. A zero knowledge proof is expected to satisfy 3 properties, and they are —(i) *completeness*: if the statement that is to be proved is true, a honest prover should be able to convince an honest verifier, (ii) *soundness*: a dishonest prover can prove a false statement is true to an honest verifier only with very small probability (iii) *zero knowledge*: no verifier can gain any information other than the truth of the statement.

In this paragraph, we will give an example of a simple example of zero knowledge proof. Assume there exists 2 parties, prover P and verifier V. For a given large prime number p, a generator g and a value y, P wants to prove V that it knows a value x such that $y = g^x \pmod p$. In order to do so, P can generate a random number r and produce C such that $C = g^r \pmod p$. Along with this, P produces C' such that

$C' = g^{x+r} \pmod{p}$ and presents r, C and C' to V. With all these values, V can verify the validity of the original statement by checking if y equals to C'/C. This way P can prove V that it knows x without leaking any information about x to V.

Chapter 4

The Voting Protocol Design

In this chapter, we discuss the voting protocol we used to realize the voting system. We explain about the assumptions we have made and the details about the interaction between voters, election authority and other participants of the system.

4.1 Voting Scheme

Over the years, many voting protocols have been proposed [5], [9], [13] to design a voting system that is secure. Unlike the aforementioned work where an assumption is made regarding the existence of public bulletins that is irreversible and transparent, we use blockchain to actually realize a public bulletin, and the design of protocol is in synchrony with concurrency control mechanism of the underlying blockchain. Our protocol is quite similar to [5] as we use homomorphic encryption scheme with zero knowledge proof to ensure correctness and confidentiality of the votes. Even though intricate details of the existing voting systems may vary, they all try to achieve similar properties. As per [16], some of the properties that different voting system try to accomplish are:

- **Verifiability** - Voters should be able to verify how they voted or how tallying of the votes took place. This enables to make the voting process to be more trustable. In the current ballot station based voting and voting by mail system, a voter cannot verify how the ballots were recorded and how the tallying of votes took place. This makes a voter to assume that the election authority is honest, but it is not viable assumption to make.

- **Privacy** - Identifying the content of the ballot should not be possible by anyone other than the voter himself/herself. In the current system of voting, there's a chance that a tallier agent at the polling station can look at the content of the ballot and find out whom a voter cast his or her vote to. This can impose a security risk to the voter.

- **Authentication** - Only registered voters should be able to cast ballot. This helps to ensure that only eligible voters can vote. Authentication exist in the current system of voting as only voter with eligible voter card can cast a ballot.

- **Double voting prevention** - A voter should be able to vote only once. This property is also fulfilled by ballot station based existing voting system as a voter is allowed to cast a vote from a particular ballot station only, and casting double vote at the same ballot station with same voter identification is detectable.

- **Robustness** - If there is coalition of minority of fraud agents inside the system, it should be tolerated and later detected by the system. This is important because it makes the system tolerant to unexpected faults. In the current system, sometimes fraud inside the election system is not detectable which can violate the correctness of voting process.

- **Uncoercibility** - According to [16], no voter should be able to convince any other participant of the value of its vote. It prevents buying and selling of votes. As part of the protocol design, we do concentrate our efforts to achieve uncoercibility.

The process of designing voting protocols is quite complicated because the requirements conflict with each other. For example, verifiability and privacy are conflicting because we want every process to be verifiable and at the same time we also want voters to have privacy. Hence, a voting protocol cannot have strong guarantee of all properties so we need to trade off amongst what levels of properties we want. It is important to note that blockchan is important as part of protocol for 2 reasons: (i) integrity of data is ensured as hash of new block is dependent of the content of older blocks; this ensures data cannot be tampered unless majority of the nodes are wrongly motivated. (ii) the logic for validation of votes can be checked by the participating nodes in the blockchain network; this ensures malicious activity in the network can be prevented and tracked.

For the purpose of protocol design, we consider a voting system where a voter can cast a ballot for 1 out of p candidates. The entities involved in the voting system are election authority, validators and voters. Validators can be political parties, local civil organizations and the election authority itself. Validator entities are the part of the blockchain network and as long as the majority of validators do not have common interest and do not conspire, any tampering of the public bulletin will be detectable. One important thing to note is that there are lots of factor involved in system design. This requires us to make some assumptions to simplify the task of protocol design. The assumptions we make are related to safety of hardware, nature of communication channel and adoption of cryptographic system which are unrelated to the working of blockchain, and this allows us to continue our exploration of blockchain layer of the voting system. The assumptions we have made are described below:

1. The voting client software or the hardware should not be corrupt or else malware infected system software like operating system, device driver, etc can change the behavior of the voting client software. The infected client software may leak ballot data without voter knowing about it. Trusted computed base [41] are quite popular option to achieve computation in an isolated environment which is suitable for use cases of client voting application.

2. There exists secure channel between voter and election authority. This ensures an adversary cannot observe the communication between a voter and the election authority. This can be accomplished communicating using HTTPS or TLS over TCP.

3. Adversary can't fake the credentials required for voter registration. This implies the private key of the election authority is kept safe or else this can enable an adversary to register as many voters as they want. If it is a private key of threshold cryptosystem, it is relatively easier to keep the key safe as the key is split amongst multiple parties. Hardware security modules like Yubikey [25] is a popular way to keep private key safe.

4. Fewer than half of the validator nodes are corrupt. Without this constraint, the data stored in blockchain is not expected to be untampered. If the validators are chosen such that they do not have common interest, it is quite reasonable to make this assumption.

5. Every voter has a pair of private, public cryptographic key and a certificate validated by election authority. This implies every voter can be identified by a pair of cryptographic keys. As digital identity are getting popular, this assumption is also reasonable.

4.2 Protocol Design

4.2.1 Setup

Election authority sets up a Public Key Infrastructure (PKI) that is used to associate entities involved in election with their public key and certificate signed by certificate authority. Every voter and election authority is identified by (Pk, $Cert$) where Pk is the public key of the entity whereas $Cert$ is the public certificate signed by a trusted authority like government entities. Election authority sets up the permissioned blockchain network where the nodes can be participating political parties, local civil organizations, etc. All the interactions between the entities are validated using these public key and public key certificate.

4.2.2 Registration

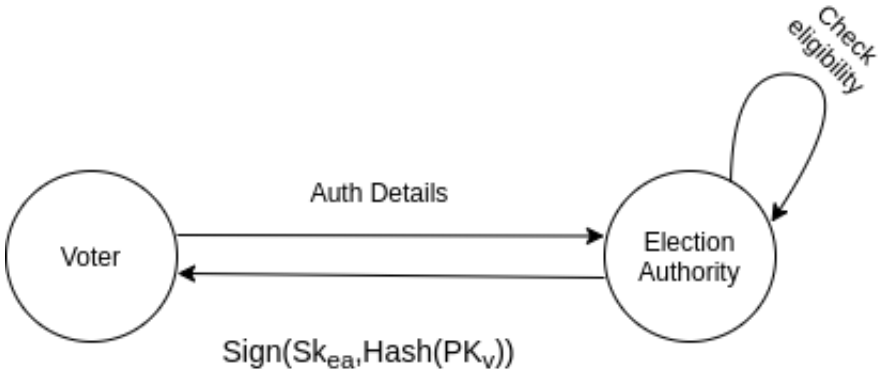


Figure 4.1: Registration of a voter

Unlike [5], we discuss the registration process in detail. Like a station based voting system, the registration process requires the voter to present authentication details to the election authority server. For example, in Canada, the Canada Revenue Agency (CRA) account details could be authentication credential or from public key

cryptography point of view, if I(PK, SK, Cert) is the identity of a voter, a credential can be (a, Sign(SK, a), Cert). If a voter provides authentication details that is legitimate, election authority registers the voter and provides a secret signed token t = Sign(Sk_{ea}, Hash(Pk_v)) where Sk_{ea} is the private key of election authority and Pk_v is the public key of the voter. This token is supposed to be kept secret and as part of authentication process, it is to be presented when a vote is to be cast. Having a single entity for the registration can lead to a single point of failure of trust. The aforementioned registration process can be extended in a distributed setting using threshold signature schemes.

4.2.3 Ballot Cast

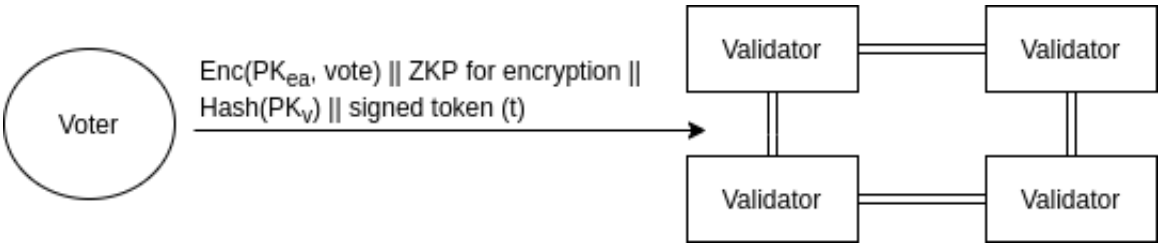


Figure 4.2: Ballot cast

A ballot is a set of signed token from the election authority (t), encrypted vote, zero knowledge proof for validity of encryption. Along with the ballot, a digital signature on the content of the ballot is required to verify the authenticity of the voter. The secret token (t) is used to authenticate if the voter is eligible to cast a ballot or not; the validator node (peer) in HLF has to validate if Verify(Pk_{ea}, t) == Hash(PK_v). The zero knowledge proof of encryption is same as used in [5]. The zero knowledge proof (ZKP) of encryption is necessary to assure the ciphertext contains a valid value for the vote without leaking any information about it. With an invalid value for the vote, the tallying process might result in incorrect results. For example, if the vote value is supposed to be one of {1, 4, 8} but the voter used the value 19 then the result obtained from tally process, which is basically summation of encrypted votes, would be incorrect. Unlike [5], the logic for validation of ballots is written as part of the chaincode which is installed in the validator node or in other words peer entity of the blockchain network. If all validations pass, the ballot is written into the blockchain as a key value pair where key is the Hash(PK_v) and the value is the

encrypted ballot.

Algorithm 1: Chaincode Logic

```
Function ValidateBallot(ballot):
    if validSignature(ballot) and withinTimeLimit(ballot.timestamp) and
     validZKP(ballot.proof) then
        acceptBallot(ballot) ;
    else
        rejectBallot();
    end
```

4.2.4 Ballot Tally

Once all votes are written to the blockchain, anyone having access to the network can read the votes and start tallying. We consider using the Paillier encryption scheme and because of its additive homomorphic property, the only thing a tallier needs to do is calculate the product of all encrypted votes.

E(Pk, R) = E(Pk, V1) * E(Pk, V2) * ..... * E(Pk, Vn) = E(Pk, V1 + V2 + ... + Vn)

The votes are encoded with log(Nv) * p bits where p is number of candidates and Nv is total number of voters. For example, votes are encoded such that for 4 candidates and 4 voters, the vote value for 1st candidate would be 1 (000,000,000,001), vote value for 2nd candidate would be 8 (000,000,001,000), vote for 3rd, candidate would be 64 (000,001,000,000) and vote for 4th candidate would be 512 (001,000,00,000). Hence, summation of votes should result in a number such that information on the number of votes of ith candidate is represented by ith block of log(Nv) bits from the left.

Algorithm 2: Tally Phase

```
Function TallyVotes(votes):
    finalTally = 1
    foreach vote in votes do
        finalTally = finalTally * vote
    end
    return finalTally
Function ExtractCandidateVote(finalTally):
    resultTally = [0 for i in range(candidates)]
    foreach i ∈ range(0, vBits * (candidates + 1)) do
        if (1<<i)&finalTally ≠ 0 then
            candidate = int(i/vBits)
        end
        resultTally[candidate]| = (1<<(i%vBits))
    end
    return resultTally
```

Once the final tallied value is calculated, any entity can request the election authority to decrypt it. The election authority will respond only if the presented value is the correct tallied value. The response for the decryption will contain the zero knowledge proof for the decryption [13] which the requester can use to verify if the election authority actually decrypted the value which he or she sent. Even though blockchain based voting protocols already exist, the novelty in our protocol is that while writing votes to the blockchain, we use single key transaction so that we can expose optimization from the underlying blockchain platform we used. This optimization is explained in the section 5.4. Even though we consider a ballot casting to be 1 out of p candidates, the protocol can be extended to accomodate ranked voting system too. Ballot structure can be modified such that a voter can specify p numbers to specify the ordering of the candidates. Also, we would require p proof of encryption; each for a candidate.

Chapter 5

Methodology

In this chapter we discuss the methodologies we applied to improve the capacity of voting system to ingest high volume and velocity of votes. We first explain about the performance analysis of Hyperledger Fabric which was done to understand its internals. Based on the insights observed from performance analysis, we then explain about the optimizations we proposed to improve the performance of the system.

5.1 Performance Analysis

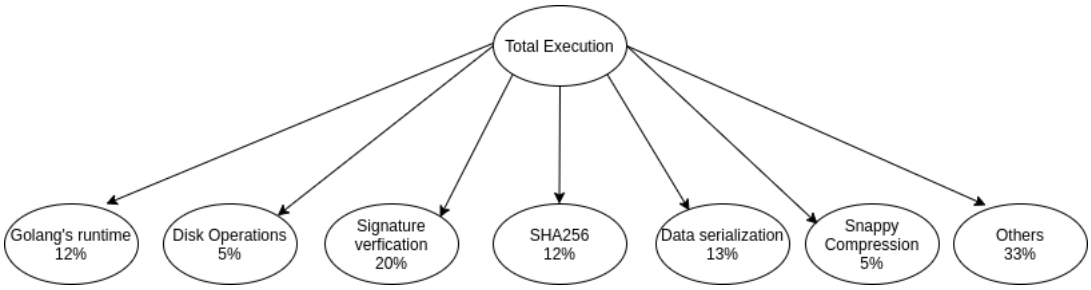


Figure 5.1: Break down into primitive operations for peer process

Voting systems are composed of a transactional component and an analytical component. Transactional component determines the velocity and volume of votes the system can ingest whereas an analytical component determines the capacity of the system to tally the votes. If we are to ensure that the system can handle the volume and velocity of incoming votes of the election day, it is important to analyze the performance, find bottlenecks and address them. The performance metrics we are concerned about are *throughput* and *latency*. Throughput of a system is the number

of transactions a system can process per second whereas latency is the number of seconds spent to mark a transaction as complete. As part of performance analysis, we set up an experiment to profile the code of the peer node. An experiment was done with 40k transactions such that each transaction performs 1 read and 1 write operation. Except for the profiler running in the background, the experimental setup is exactly the same as the experimental setup explained in chapter 6. The tool that we used to profile is pprof [1], it is a tool that comes natively with golang toolchain. Figure 5.1 shows the breakdown of the total execution time of peers into primitive operations.

5.1.1 Crypto Operations

From Section 5.1, it can be seen that signature verification is the most expensive operation as 20% of the total time is spent on it. The calculation of SHA256 hash takes 12% of total execution time. SHA256 is calculated as part of (i) signature generation and verification process (ii) endorsement of a transaction (iii) data integrity check. Similarly, most of the signature verification is performed as part of (i) endorsement phase: the signature of proposal sent by the client needs to be verified, (ii) commit phase: as part of validation system chaincode (VSCC), the transactions a peer received needs to be validated if they are signed as per the correct endorsement policy and also if the transaction block contains the signature of authorized Orderer nodes. A basic call graph of crypto operations can be seen in figure 5.2

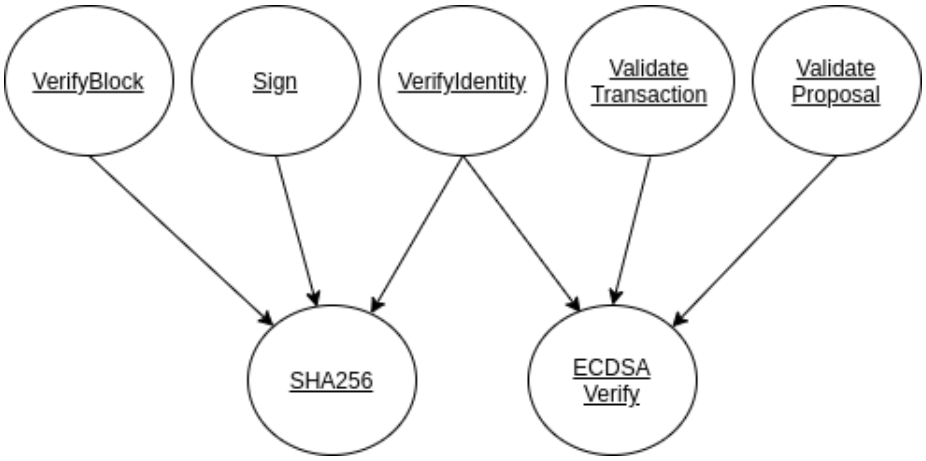


Figure 5.2: Call graph of of crypto operations

[1] https://github.com/google/pprof

Cryptographic operations are heavily involved in the working of blockchain. Irreversibility property of blockchain entirely depends on data integrity checks. It is obvious to assume that cryptographic operations are the building blocks of blockchain and also from code profiling we can see that 32% of total time is spent on performing cryptographic operations. Hence, one of the routes that can lead to better performance of the system is making crypto operation faster.
In order to verify if crypto operations falls in hot path, we modified the underlying signature verify method call in the HLF source code to simply return true instead of performing actual signature verification operations. Theoretically a high improvement in throughput of the system should have been observed but the throughput improved by only 18%. Upon further analysis, it was found that removing the signature verification reduced the total execution time but increased the rate of memory allocation which made the garbage collector more aggressive. The time spent in garbage collection increased by 15% which explains why little improvement was seen, but it was enough to make a claim that on reducing execution time of signature verification, throughput of the system can be improved. Also, as SHA256 calculation is done along with signature verification, it can be implied that SHA256 also falls in the hot path. This analysis taught us two important insights, and they are - (i) expediting an operation might not lead to performance improvement as expected; there are other factors into play (ii) crypto operations fall in hot path so faster alternatives of existing cryptographic libraries should improve the performance of the system; further explained in section 5.3.

5.1.2 Data Serialization and Deserialization

Data serialization and deserialization takes 13% of the total time. HLF uses protocol buffers, a library to serialize structured data, to serialize the blocks of transaction before they are communicated amongst different entities. Data serialization and deserialization mostly involve allocating memory and data encoding so in an application written with garbage collector (GC) based language (Golang) such operations are affected by how memory allocator behaves with respect to the GC. This is because the rate of such memory allocation and allocation size can affect how golang's garbage collector interferes with the execution of the program. Gorenflo [19] has shown how caching the result of data serialization/deserialization can have 3x speed up. Caching such memory intensive operations not only saves CPU cycles from seri-

alization/deserialization operations on the same data block, it also reduces the load on the garbage collector by reducing the number of memory allocation and the size of the memory the garbage collector has to scan.

5.1.3 Disk IO and Snappy Compression

The total time spent on disk IO actually depends on 2 major factors. The behaviour of transaction i.e how many disk IO it performs and the kind of storage system being used. As per the experiment we ran, there is 1 read and 1 write per transaction. Also, because SSD was used instead of HDD, we observed disk IO not to be a major bottleneck.

It can be seen that only 5% of time is spent on disk IO such that 4% is spent on write operation whereas 1% is spent on read. Out of total disk writes, 40% write time comes from garbage collector thread of Leveldb whereas 60% disk write constitute state database writes, history database writes, ledger block file writes and logger writes. Disk reads constitute reads from chaincodes via a state database (Leveldb) and reads from the compaction thread of Leveldb. The task snappy compression, which is a part of Leveldb write operation, takes up 6% of total execution time. Just before Leveldb writes a block to disk, it compresses the block using a compression technique called snappy to reduce the number of bytes to be written to disk and hence enhance the speed of writes. But it turns out that on a SSD, compression operation is as expensive as disk write. As snappy compression is optional in Leveldb, we turned it off and ran the experiment. There was no improvement in the overall performance of the system as the disk write increased by an amount which was equal to snappy compression's total execution time.

We observed that the compaction thread can block the writes of the application. As part of compacting the size of the LSM tree, an indexing data structure which Leveldb maintains, the compaction thread can block the whole database while compacting block files at different levels. So, there might arise a scenario where as HLF is writing the transaction to the disk, the compaction thread blocks it.

5.1.4 Golang's Runtime

Unlike in languages like C++ and C, Golang provides a garbage collector, memory allocator and deallocator to perform memory management. As garbage collector (GC) can intervene and block the whole application, sometimes a high performance system can take a hit and this phenomena can be easily observed in case of HLF.

19% of total execution time is spent on golang's runtime operation like scanning and marking unused allocated memory, managing heap while allocating new memory blocks, etc. Out of total runtime operations, 70% is spent on a method called mallocgc whose main task is to allocate memory and at the same time deallocate unused previously allocated memory. The golang's garbage collector tries to reduce the time for stop the world, a phase of GC where application is blocked, by marking unused memory concurrently with the application thread on a multicore machine. This means the application does not get the full cores at its disposal; instead as per the aggressiveness of the memory allocation behaviour, some of the cores are used to mark the unused memory. This marked unused memory is then sweeped during the mallocgc function call i.e memory allocation. Due to this, allocating memory can be very expensive in a memory intensive application like HLF. This also explains why data serialization and deserialization is expensive and how caching can help reduce interference of garbage collector and improve the performance of the system.

5.1.5 Other Task

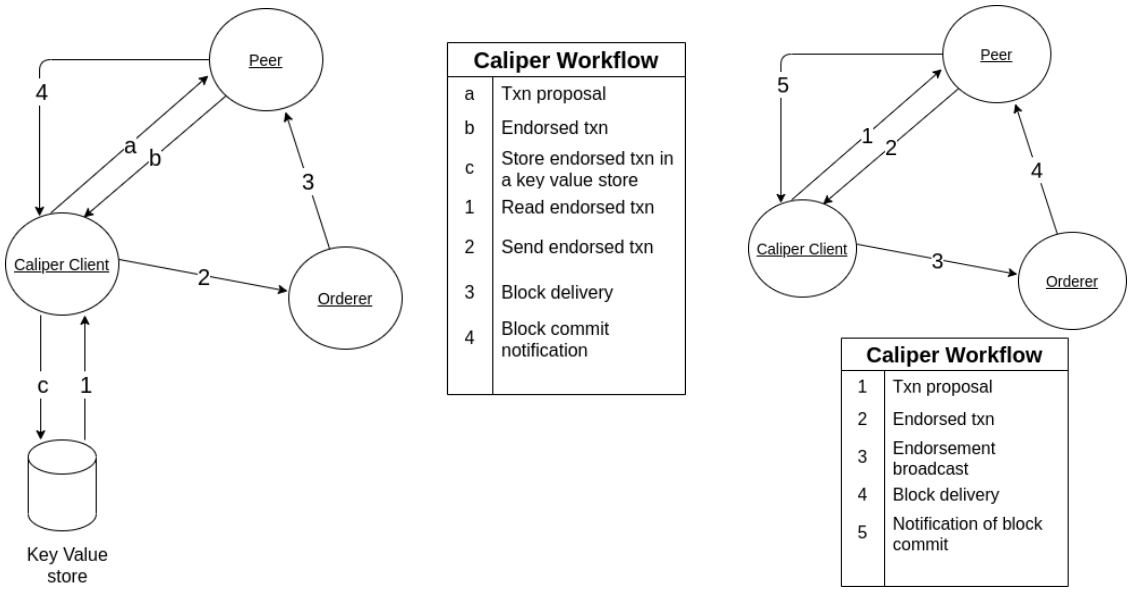
It is quite difficult to categorize all the primitive tasks as per their execution time share which is why they are addressed as others. Some of the other tasks are logging operations (3%), network reads and writes (3%),locking and unlocking mutexes and semaphores (6%), multiversion concurrency check (1%), etc.

5.2 Performance Measurement

In HLF, a peer can both endorse and commit transactions and due to this, the endorser thread and committer thread has to compete for the cpu and memory resource. Such resource contention becomes worse when HLF is deployed in a single machine. This makes it difficult to measure the performance metrics of these phases individually. Even though deployment of HLF in a single machine is resource intensive, testing

code changes in a local machine is convenient and less expensive. For example, in a distributed environment, after making a code change, we would have to build it and transfer the newly built binary to all the machines and deploy them individually. This process can be cumbersome especially when the testing process becomes iterative. Also, as we deal only with the capacity of peers to process transactions which includes operations like signature generation and verification, state database commits, etc, single node deployment is sufficient for use case of benchmarking. This led us to focus our efforts to make the testing process in a single machine less resource intensive.

We chose an open source tool called Hyperledger Caliper to do end to end performance testing. The working of this tool is independent of how the blockchain network is deployed, and it does not handle the resource contention issue when multiple blockchain entities are deployed in a single machine. Hence, we decided to tweak the working of hyperledger caliper and devise a convenient and cost effective way to measure performance of peer node in a single machine. The tweaked workflow can be seen in the figure 5.3.



(a) modified workflow (b) original workflow

Figure 5.3: Contrasting Caliper workflow

The key idea is that in order to measure the performance of the commit phase, we do not need to endorse the transactions. Similarly, to measure the performance of the endorsement phase, we don't need to commit transactions. We introduced new para-

metric flags to the tool such that they determine if the commit or endorsement phase is to be ignored. The way endorsement phase is skipped is by saving endorsed transactions in a key value store like LevelDB in the first run (setup phase). After the first run is complete, the stored transactions can be re-used instead of sending proposals to the peer node for endorsement. In the above figure, alphabetical notation denotes the interaction involved in storing endorsed transactions (setup phase) whereas numerical notation denotes the interaction involved in re-using endorsed transactions and skipping endorsement phase. Hence, depending on the flags passed to the tool, it can decide if it should read the key value store, re-use the endorsed transaction and skip the endorsement phase. The logic to skip endorsement works only if the peer nodes and Orderer nodes have not changed their public keys and certificates from the time when endorsed transactions were stored in a key value store. Similarly, in order to skip the commit phase, Caliper client has to simply avoid sending endorsed transactions to the Orderer node. Hence, these modifications allowed the development of HLF in a single machine with reduced resource contention.

5.3 Replacement of Crypto Libraries

Thakker et al [38] parallelized the transaction validation phase (VSCC) instead of making the constituent crypto operations faster which makes a case that this path is unexplored. Hence, this led us to work on minimizing the total execution time spent on signature verification and hashing by using faster algorithms or faster implementation with the same level of security. The security of the whole system is tightly coupled to the cryptographic algorithm that is used for operations like signature verification and hashing so we decided not to look out for other algorithms but instead replace the existing implementation with a faster one.

HLF uses golang's natively written libraries for calculating SHA256 hash and ECDSA based signatures. The golang package crypto/sha256 is used to calculate SHA256 hash whereas the package crypto/ecdsa is used for both signature generation and verification. The package crypto/ecdsa implements elliptic curve digital signature algorithm on a prime elliptic curve p256 as per the guidelines from FIPS 186-3 [21] and the equivalent implementation was found as part of OpenSSL library, a fast and battle-tested crypto library in C. The package crypto/sha256 implements SHA256 hash algorithm following the guidelines from FIPS 186-4 [33]. The equivalent faster

SHA256 implementation we used was minio/SHA256 [20].

5.3.1 Benchmark comparison of SHA256 Libraries

We benchmarked the native golang's SHA256 implementation and minio/SHA256 implementation. The benchmark consisted of 1k iterations. It was found that the mionio/SHA256 was about 14% faster than the native implementation. Some of the metrics observed using perf tool [2] can be seen below:

Metrics	minio/SHA256	Native Golang	% Change
Instruction Per Cycle	3	2.88	+7.16%
Branch Misses	20M	24M	-16.6%
L1 dcache misses	92M	96M	-4%
Vectorized 128b single	128K	128K	0
LLC Load Misses	6.5M	8M	-18.75%
Total execution time	5.7s	6.6s	-13.6%

Table 5.1: Benchmark result for SHA256 (1k iterations)

It can be seen that the third party implementation has higher instruction per cycle and less branch misses. By analysing the source code, it was found that third party implementation had only 247 MOV instruction and 4 jump instruction whereas golang implementation had 1317 MOV and 9 jump instruction. As MOV and jump instruction are higher it explains the lower IPC and higher branch misses with native golang implementation. We performed several benchmarks to verify the results; the distribution of execution time can be seen in figure 5.5

5.3.2 Benchmark comparison of Signature Verification Libraries

We benchmarked the native golang's ECDSA signature verification implementation and OpenSSL implementation. The benchmark consisted of 1k iterations. It was found that the OpenSSL's implementation was about 16% faster than the native implementation. Some of the metrics observed using perf tool can be seen below:

As OpenSSL implementation has lower branch misses, cache missed and higher IPC, it explains why OpenSSL implementation is better than golang's native imple-

[2] https://github.com/brendangregg/perf-tools - it is Linux based profiler that can count hardware and software events

Metrics	OpenSSL	Native Golang	% Change
Instruction Per Cycle	3	2.7	+11.1%
Branch Misses	6.9M	9M	-23.3%
L1 dcache misses	11B	10B	+10%
Vectorized 128b single	110K	110K	0
LLC Load Misses	246K	390K	-36%
Total execution time	5.9s	7s	-15.7%

Table 5.2: Benchmark result for Signature Verification (1k iterations)

mentation. We performed several benchmarks to verify the results; the distribution of execution time can be seen in figure 5.4

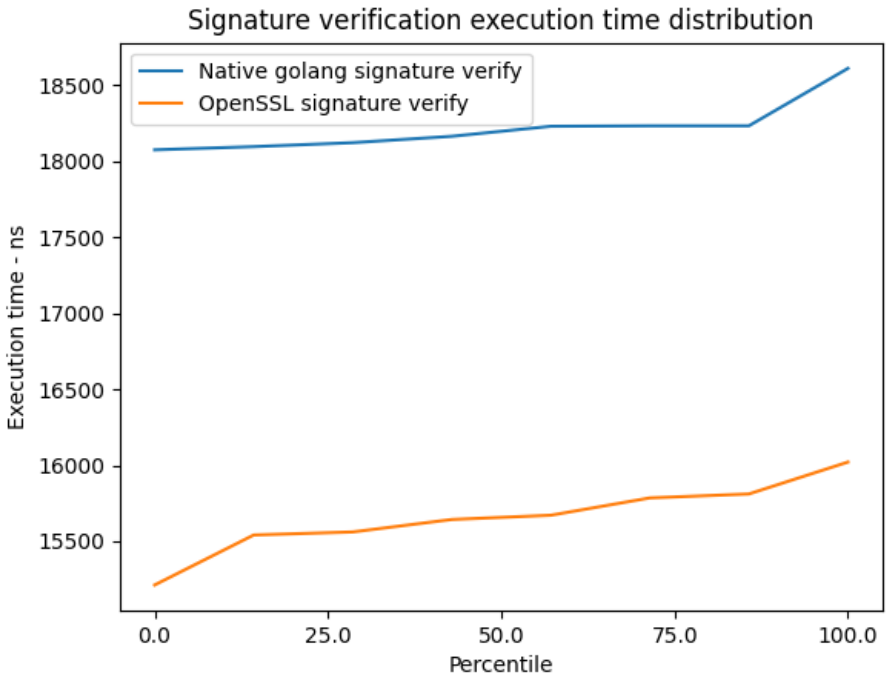


Figure 5.4: Distribution of ECDSA verification time

5.4 Tuning Concurrency Control Mechanism

Unlike public blockchain like Bitcoin, Ethereum which use order-execute architecture for transaction processing, HLF uses execute-order-commit architecture for increased concurrency. In HLF, as ordering service is reaching consensus regarding the order of

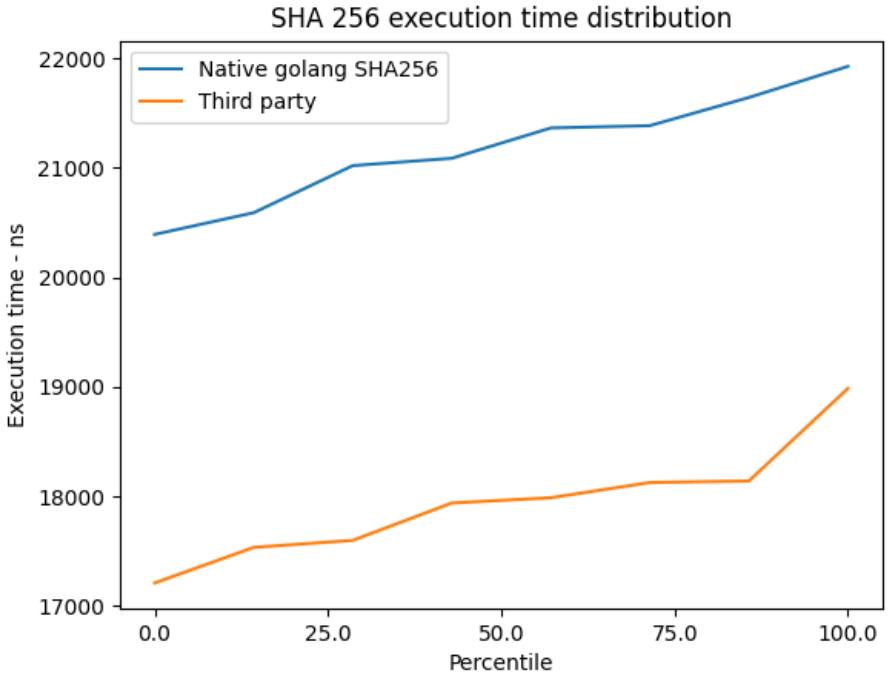


Figure 5.5: Distribution of SHA256 execution time

transactions, a peer node can execute and commit transactions concurrently which is why permissioned blockchain like HLF are supposed to have better performance than blockchain based on order-execute architecture.

As transactions are processed concurrently, HLF uses lock based transaction manager along with multiversion concurrency control (MVCC) check to ensure isolation of transactions (repeatable read isolation). When a block of transaction is to be committed, the commit thread obtains the shared RW lock. Similarly, as part of transaction endorsement, before a transaction reads the state database, the transaction manager obtains a shared read write (RW) lock to get a consistent view of the database. The interaction between these phases of transaction can be seen from the figure 5.6

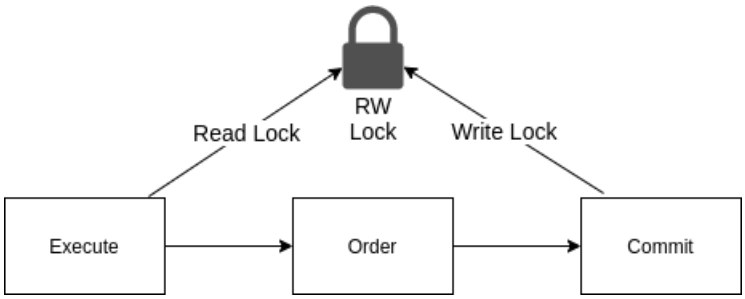


Figure 5.6: Lock contention between different phases

Algorithm 3: Endorse and Commit

```
Function Endorse(txn):
    RWLock.lock()
    keys = txn.read()
    r = executeChaincode(keys)
    RWLock.unlock()
    return r
Function Commit(block):
    validate(block)
    RWLock.lock()
    writeDB(block)
    RWLock.unlock()
```

Hence, to avoid aforementioned inconsistent behavior and ensure repeatable read isolation, a lock is acquired just before reading the database. The issue with locking is that it can cause the commit thread to wait until the transaction finishes its execution or it can cause the endorser thread to wait until the commit thread finishes committing the block. An experiment was done to study the lock contention behavior of endorser threads and commit thread. Experiment was set such that each transaction ran for 200ms and at the same time the commit thread received blocks to commit. The objective of this experiment was to calculate the commit time while transactions were in contention with each other. The commit time distribution can be seen in the figure 5.7. It can be observed that the first half of the blocks have commit time greater than 200ms whereas later blocks have commit time less than 100ms. This is because for initial blocks, the endorser thread and commit thread are concurrently running and the commit thread has to wait for a minimum of 200ms until the endorser thread releases the RW lock. This makes a case that a lock based transaction manager can degrade the performance of the system.

The key insight we observed is that we don't need a lock based transaction manager to ensure repeatable read in specific cases. If a transaction reads and writes a single key, it does not need to lock the database because only concurrent reads and writes of multiple keys can result in inconsistent view of the database. For example, assume there exists a transaction that reads two keys x and y. Similarly, assume a commit thread is about to commit a block of transactions that update the state of the keys x and y. If there is no locking mechanism, there might exist a scenario where the transaction reads the old value of x and new value of y. Without lock, in the described scenario, there's no isolation. But if a transaction is associated with a single key, the aforementioned inconsistent behaviour does not arise. Even if a commit thread updates the single key after a transaction finishes reading the same key, the transaction will get aborted as part of MVCC check. Detailed proof for the correctness is present in section 7.1. Hence, in cases where a transaction is associated with a single key, lock based transaction manager is not required for repeatable read isolation. In the section 4.2, we have designed the voting protocol such that the data writes and reads to blockchain is associated to a single key. Each ballot is stored as a key value pair where key is the hash of public key of voter and the value is ballot data. The decision to design the chaincode logic was based on this optimization in the hope of better throughput and latency. Hence, we explored the source code of HLF and modified the transaction manager to be lock free and designed transactions for the voting application to use a single key.

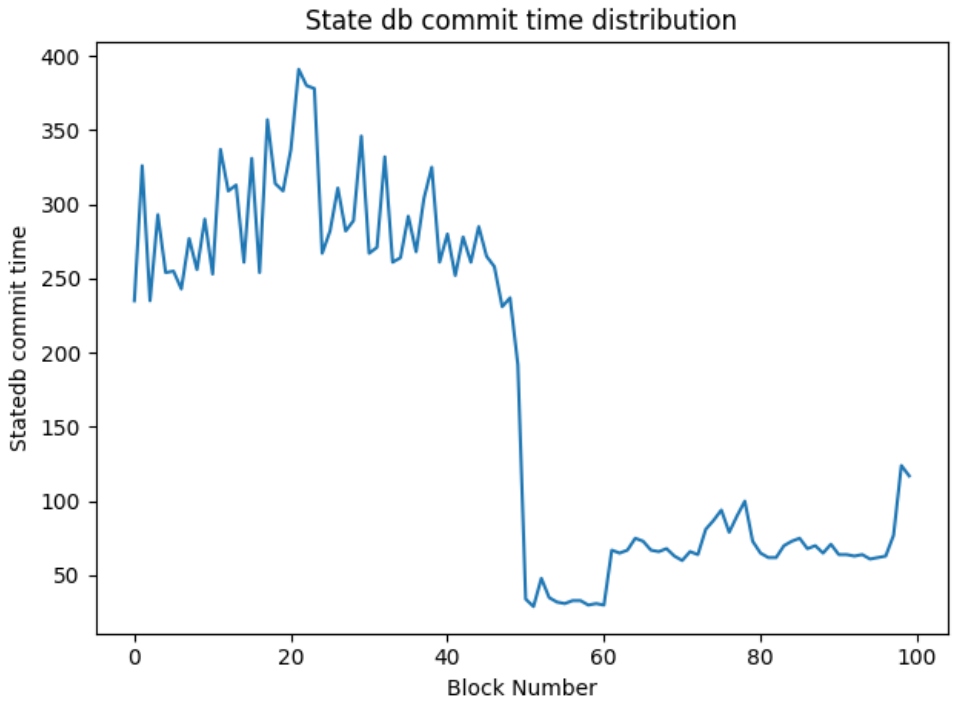


Figure 5.7: Commit time distribution

Chapter 6

Experimental Results

In this chapter, we first discuss the experimental setup we had to test our proposed optimization. We then show results we observed for each proposed optimizations. We also show how the individual optimization add up on each other to give about 30% speedup.

6.1 Experimental Setup

The main objective of our experiments is to find out how the proposed optimizations improve the throughput and latency of the system. As part of the experimental set up, our main tasks were to write chaincode that implements the voting logic i.e. verification of ballots, deploy a blockchain network and set up a performance measurement tool for end to end performance testing. [38] [19] created a multi-node deployment and wrote custom tools for workload generation and performance testing whereas we used Hyperledger caliper for performance testing and used a single node deployment for simplicity. We also added a feature to Hyperledger caliper to improvise the single node deployment which has been explained in section 5.2. All the experiment were conducted in a machine with following configurations:

- **Processor:** Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz with 4 cores
- **Memory:** 12 GB RAM 2400MHz
- **Storage:** NVMe Kingston SSD 256 GB

In order to measure the throughput and latency of the system, we need to simulate a scenario where ballots are sent to the blockchain network at a high rate. Ballots are

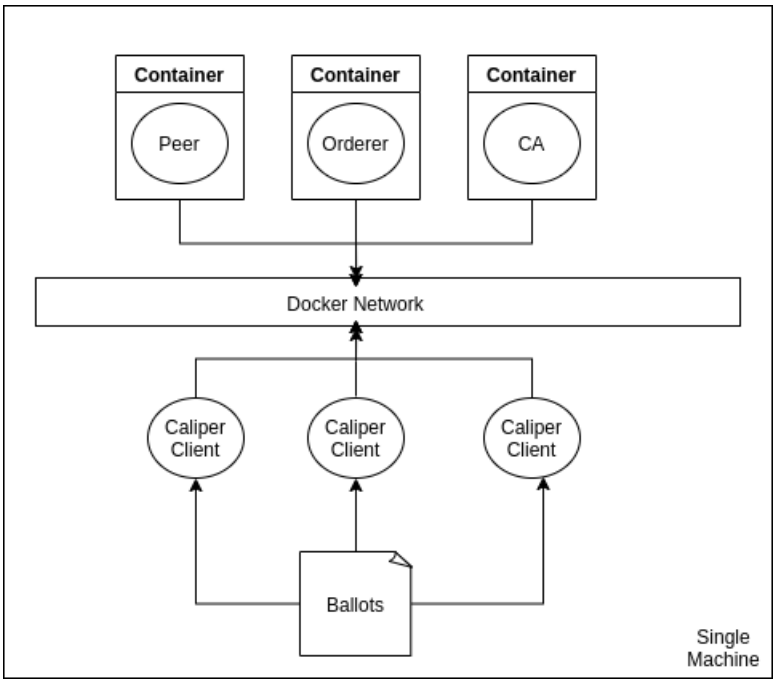


Figure 6.1: Deployment Architecture

basically encrypted voter choice along with proof of encryption. We used a python library [14] that implements Paillier encryption scheme to encrypt the voters choice and also implemented methods for generating and verifying the proof of encryption. The ballot verification is written in Golang as part of chaincode execution. We run a batch job to generate 60k encrypted ballots and store them in a file. This file is split and read by caliper clients. Basically, these caliper clients simulate the voting process for a voter by forming transactions and sending them at a fixed rate to the blockchain network. Multi-peer deployment on multiple nodes is resource intensive and expensive which is why we used modern isolation technologies like Docker containers [1] to create hyperledger network in a single machine. Docker containers are basically isolated set of programs running in a specified setting. Ideally a blockchain network has multiple peers belonging to different organizations for decentralization and transactions are ordered via consensus. As in this thesis we deal only with the capacity of peers to process transactions which includes operations like signature generation and verification, state database commits, etc, we deploy a single peer with a single orderer and measure its performance. Entities like peer, orderer, certificate authority were deployed in separate containers where all the communication between

[1] `http://www.docker.com`

containers and caliper clients is done via software based virtual network interfaces which is quite fast and reliable compared to physical networks. The underlying key value store used to store the world state is Leveldb and the blocks of transactions are stored directly in files.

6.2 Results

In this section, we show the results of all the experiments we performed. The objective of all these experiments is to measure performance metrics i.e throughput and latency. Throughput is measured in transaction per second (TPS) and latency is measured in time units.

6.2.1 Baseline

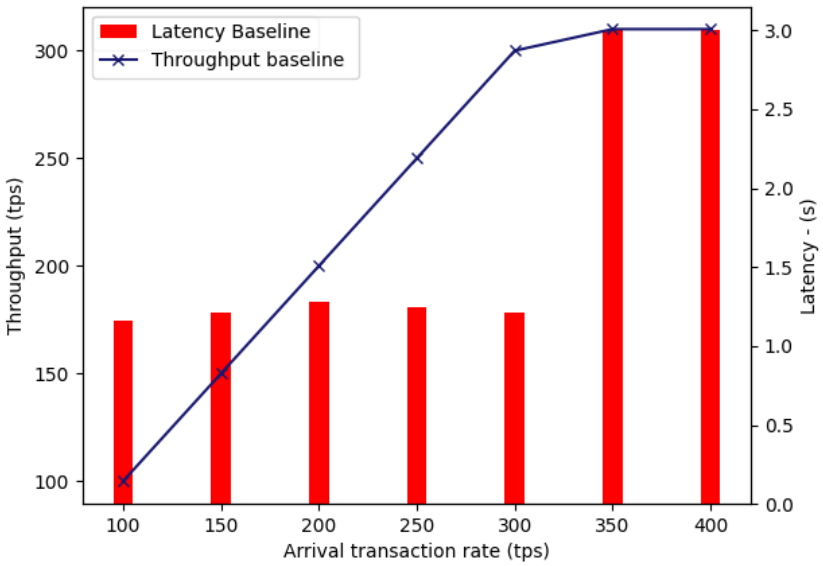


Figure 6.2: Baseline metrics

We first created a baseline to which we compared the result of our optimization. As part of the experiment we varied the arrival transaction rate by fixating other variables like block size, chaincode execution time and measured throughput and latency of the system. We ran 7 trials for each arrival transaction rate and calculated the mean of throughput and latency. From figure 6.2 we can see that as the arrival

transaction rate increases, the throughput of the system increases linearly till the saturation point. Arrival rate of 300 TPS seems to be the saturation point for the deployment we had set up. The latency of the transactions started increasing after the saturation point as the transactions are queued up before the commit phase. The main objective of optimization we have proposed is to push the saturation point to the right.

6.2.2 Native SHA256 Replacement (O-I)

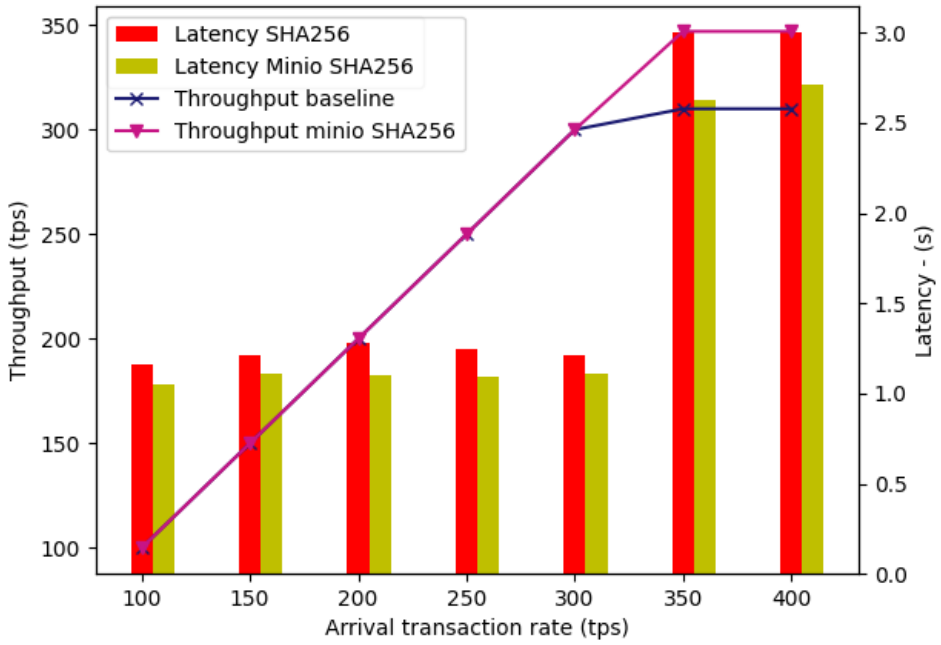


Figure 6.3: Speedup due to efficient SHA256 implementation

The figure 6.3 shows the metrics after we replaced golang's native implementation of SHA56 with minio/SHA256. Just like section 6.2.1, we ran 7 trials for each arrival transaction rate and calculated the mean of throughput and latency. With the new SHA256 implementation, we shifted the saturation point from 300 to 350 TPS. Compared to baseline metrics, at saturation point, there was about 12% improvement in throughput and about 10-15% improvement in latency.

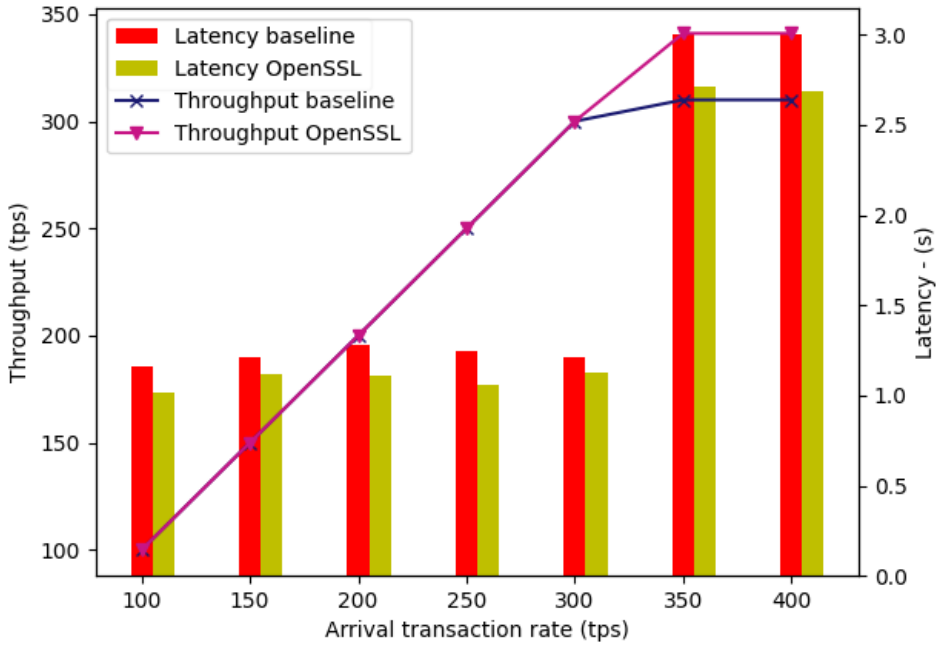


Figure 6.4: Speedup due to OpenSSL implementation for signature verification

6.2.3 Native ECDSA implementation Replacement (O-II)

The figure 6.4 shows the metrics after we replaced golang's native implementation of ECDSA with OpenSSL. Just like section 6.2.1, we ran 7 trials for each arrival transaction rate and calculated the mean of throughput and latency. With the OpenSSL for signature verification, we shifted the saturation point from 300 to 350 TPS. Compared to baseline metrics, at saturation point, there was about 10% improvement in throughput and about 10-14% improvement in latency.

6.2.4 Removal of Read-Write Lock (O-III)

Chaincode Exec time (ms)	TPS with lock	TPS no lock	% Change
50ms	340	367	+8%
200	316.4	359.7	+14%
500	285.2	340	+20%

Table 6.1: Effect of lock on TPS

Chaincode Exec time (ms)	Latency with lock	Latency no lock	% Change
50	2.56	2.23	-12%
200	3.6	2.9	-18%
500	5.56	4.12	-25%

Table 6.2: Effect of lock on latency

In the section 5.4, we show how lock free concurrency control can be used to achieve repeatable read isolation. We removed the shared RW lock and measured the performance metrics of the system. We varied the chaincode execution time and observed the performance improvement introduced after making the concurrency control mechanism lockless. It is obvious that as chaincode execution time increases, throughput decreases and latency increases. From table 6.2, 6.1, it can be seen that as chaincode execution time increases, the extent of performance improvement also increases. For chaincode execution duration 200ms, which is close to our implementation of voting logic, there was 14% improvement in throughput and 18% improvement in latency. As the endorser thread and commit thread does not have to wait on acquiring the read write lock, the statedb commit time decreases and the CPU utilization also increases which results in performance improvement. The improvement in CPU utilization and statedb commit time can be seen from the graphs 6.6 and 6.7. The legitimacy of the aforementioned performance improvement can be corroborated by the result observed in [29]. Even though detailed descriptions of the experiment were not given, [29] showed that removing lock can result in 8x performance improvement of the system. As per our observations we were not able to achieve 8x improvement, but we observed similar trend as [29]; further speed up analysis is done in section 7.1.1. Thus, it can be said that we were able to leverage the benefits of lockless transaction isolation mechanism by designing the chaincode to use single key.

6.2.5 Combined Result

The optimization we have proposed does not conflict with each other; each optimization's performance benefit adds up to each other. The SHA256 calculation, signature verification and wait on lock are all independent tasks falling in the hot path. Therefore, we conducted an experiment where all the modification to fabric was put together and performance metrics were measured. 40K transactions were sent at 450 TPS and the single orderer was set to produce blocks with 500 transactions. End to

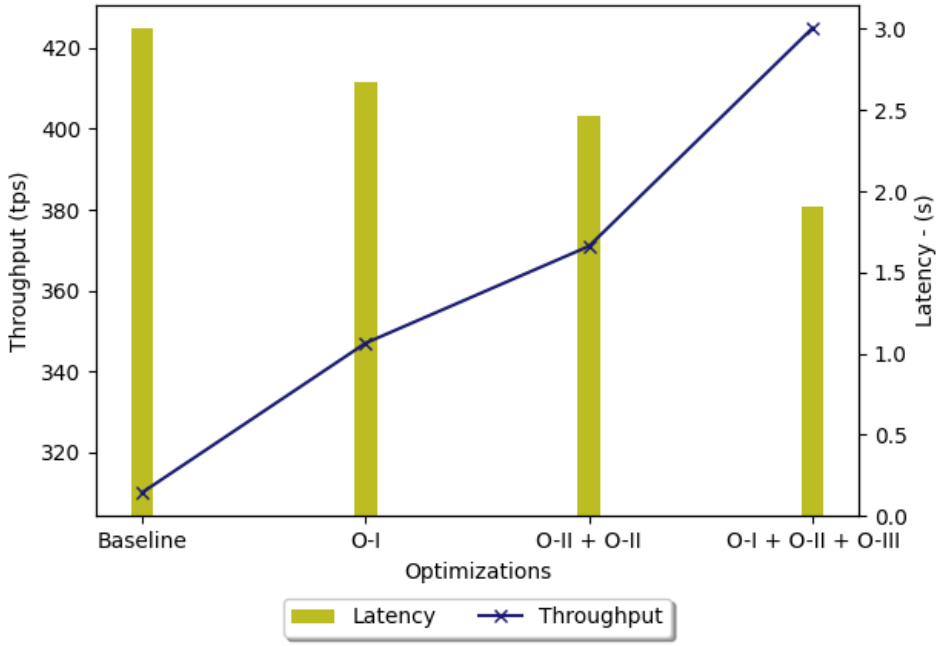


Figure 6.5: Metrics for combined optimization

end throughput of the system increased from 310 TPS to 420 TPS - 37% increase. Average latency of the transactions decreased from 3 seconds to 1.91 seconds - 36% decrease (fig: 6.5).

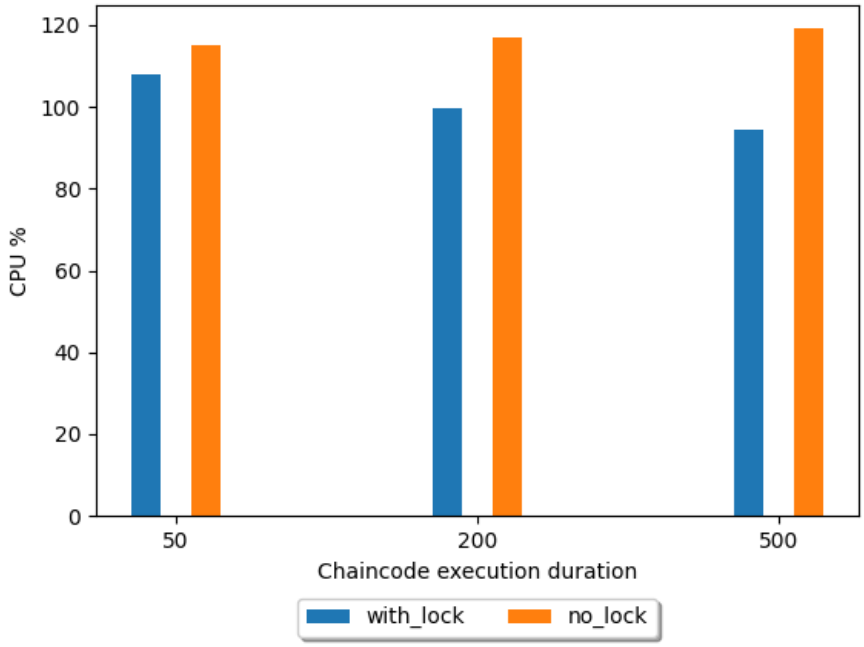


Figure 6.6: CPU Utilization

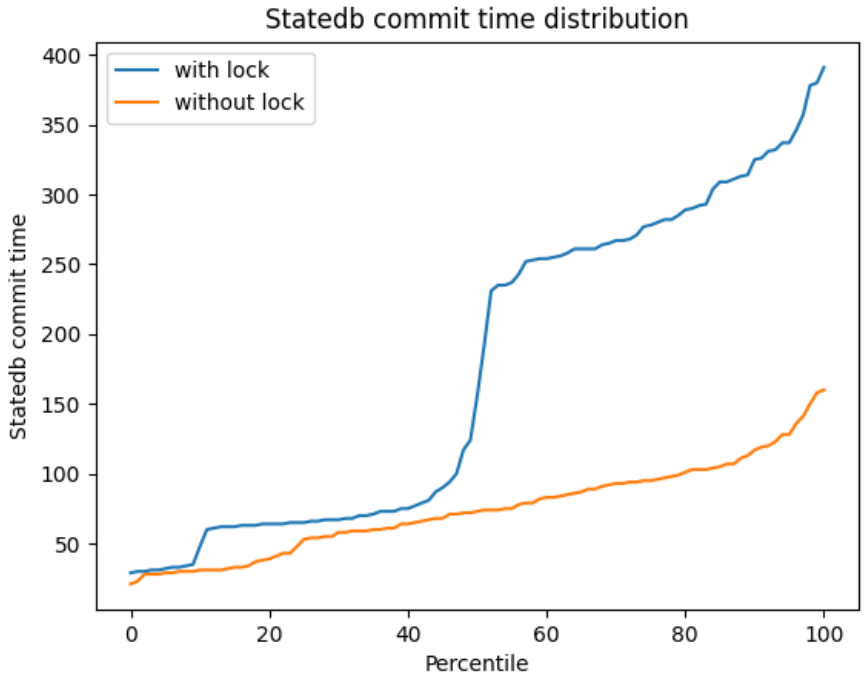


Figure 6.7: Improvement of statedb commit time

Chapter 7

Evaluation, Analysis and Comparisons

This section analyzes cost benefit of performance optimization of the underlying blockchain, the characteristics of blockchain based voting systems and advantages of permissioned blockchain based voting system over public blockchain based voting system.

7.1 Analysis of read-write lock removal

As we proposed to remove the read-write lock to improve performance specific to our use case, it can be argued that the correctness of the transaction manager gets violated. Also, a speedup analysis is required to know about the expected speedup. Hence, in this section, we provide a proof for the correctness of transaction manager and speedup analysis for the change we made in transaction manager.

7.1.1 Speedup analysis

From figure 5.7, we can observe that when there's lock contention the time taken for block commit can range from 230 - 390ms with average time being 285 ms whereas when there's no lock contention the time taken for block commit can range from 40 - 130 ms with average time being 85 ms. After the lock removal, we can expect the speedup of block commit time to be 3.3x. But the total speed up in the throughput of the system should be less than 3.3 times of the baseline throughput.

Theorem 1. *In HLF, if lock removal results in speed up for the block commit time by a factor of p then the improved throughput of the system is less than p times of baseline throughput.*

Proof. Let T be the time taken to process a block of transactions with lock whereas T' be the time taken to process a block of transactions without lock. T can be written as $T = x + y$ where x is the time taken to commit a block and y is rest of the total time; y is composed of network latency, unmarshalling operations, garbage collector stop the world phase, etc. From figure 5.7, we can claim that without the lock, the commit time i.e. x becomes x/p. Thus, $T' = x/p + y$. Let S be speedup of throughput (TPS) and N be the number of transaction in a block. S can be written as $S = \text{TPS}_{\text{without lock}} \ / \ \text{TPS}_{\text{with lock}} = N/T' * T/N = T/T'$. This can be further reduced to $S = (x + y)/(x/p + y)$. Unless y is negligible, $S < p$.

From profile data we found out that unmarshalling operation takes total of 10s for 40 blocks which implies that one block takes 250ms during the unmarshalling operation. For simplifying the analysis, we ignore all other cost for y and only consider unmarshalling operation and write $y = 250ms$. The average value x is 285ms with lock and 85ms without lock. Hence, $S = (285 + 250)/(85 + 250) = 1.59$ which is less than 3.3 times speed up observed in the commit time. Unless y is very small relative to x, the speed up observed in the throughput of the system will be less than speed up observed in the commit phase. □

From the above proof and our observation of profile data, we also believe that it is difficult to achieve speed up of factor 8 after lock removal as mentioned by [29].

7.1.2 Correctness analysis

Algorithm 4: MVCC validation steps (HLF)

```
Function MVCCValidation(block):
    updates = dictionary()
    foreach txn ∈ block do
        if txn.readSet ∈ updates then
            markAbort(txn)
        else
            updates[tx.readSet] = tx
            value, version = DB.Get(tx.readSet)
            if version > tx.readSet.version then
                markAbort(txn)
            end
        end
    end
```

Definition 1. *Let transaction T be represented by $\langle R, W, O \rangle$ where $R(T) = \{X_1, X_2.X_n\}$ is the set of key it reads, $W(T) = \{X_1, X_2.X_n\}$ is the set of key it writes, $O(T) = n$ is the number of read operations and $V(X_n, T) = v$ is the version of key X_n*

Theorem 2. *For algorithm 3, in a system where transactions read and write single key and $O(T) = 1$, locks are not required to ensure repeatable read isolation.*

Proof. We use proof by exhaustion to prove the theorem. We consider two transactions $T1$, $T2$ and $R(T1) = R(T2) = \{X_1\}$, $W(T1) = W(T2) = \{X_1\}$. These transactions can be in three scenarios and we prove that in all scenario a lock is not required to achieve transaction isolation.

- **Transaction $T1$ and $T2$ are both in same block**: If $T1$ and $T2$ are in the same block, then as part of MVCC validation Alg 4, the transaction that is ordered second in the block gets aborted. This means that if transactions have overlapping readwrite set, except for one all get aborted. Hence, in the time of conflicting reads and writes, only one transaction is marked valid.

- **Transaction $T1$ and $T2$ are not in the same block**: There are two ways $T1$ and $T2$ are in different block and they are-

- $T2$ **reads before** $T1$ **commits**: This implies that $T1$ is in a block in commit phase, say B and $T2$ is in endorsement phase where it read value from the state database. Assume $T2$ reads X_1 before $T1$ writes new value for X_1. Let $V(X_1, T2) = v1$ and after $T1$ is committed, version of X_1 becomes $v2$ such that $v2 > v1$. When $T2$ goes into commit phase, MVCC validation for $T2$ fails (Alg. 4) as $V(X_1, T2) < v2$. This means if two transaction has overlapping readwrite set and concurrent updates, the transaction which reaches commit phase first will be committed whereas the following transaction is marked aborted.
- $T2$ **reads after** $T1$ **is committed**: Assume after $T1$ is committed, the version of $X1$ becomes $V(X_1, T1) = v2$. As $T2$ starts reading after $T1$ is committed, the version of $X1$ it observes is $v2$. When $T2$ reaches commit phase, it is marked valid as the version of X_1 it reads is equal to version observed at commit phase.

It is important to note that isolation is achieved by aborting transactions when there is overlapping readwrite set and concurrent updates. Hence, we can claim that for single keyed transaction, the locks used in Alg.3 can be removed to achieve repeatable read isolation. □

7.2 Cost benefits of performance improvement

In this section, we will do basic cost calculation by making assumptions for things for which we don't have data. We will compare the cost of US election with and without performance optimization.

In the 2020 US election, there were 239 millions eligible voters. Assume voters are expected to cast their votes in a single day and say the voting starts from 7 in the morning to 7 in the evening i.e 12 hours. The workload distribution of the servers are never uniform and since we don't have any historical data on it, we'll make an assumption that peak workload will be 10 times of average workload.

Required throughput of the system (rt) = 10 * 239000000 / (12 * 60 * 60) = 55324 tps. A single blockchain network won't be able to achieve such a high transaction rate, so depending on the size of constituency, shards of blockchain network needs to

be created so that that transaction rate can be accumulated to achieve such a high transaction rate.

As per the baseline performance metrics, the system can process 310 transactions per second with 3 seconds average latency which means we need to deploy N = rt / 310 = 178 instances of blockchain networks. Assuming each blockchain network has 4 organizations, we would have a minimum of 4 peer machines 1 Orderer machine in the network. Assuming each machine costs $1000, total cost to deploy blockchain network = 178 * (4+1) * 1000 = $890K

As per the performance metrics after optimization, the system can process 420 transactions per second with 2.5 seconds average latency which means we need to deploy N = rt / 420 = 131 instances of blockchain networks. Assuming each blockchain network has 4 organizations, we would have a minimum of 4 peer machines 1 Orderer machine in the network. Assuming each machine costs $1000, total cost to deploy blockchain network = 131 * (4+1) * 1000 = $655K which is 26% less than the cost without optimization. Depending on the scale of the election, the difference in cost would rise and 26% cost savings can be very important.

7.3 Comparison with Public Blockchain

Voting systems can be implemented with both public and permissioned forms of blockchain. The aim of both forms of blockchain is to achieve decentralization but they differ on the level of decentralization they offer. There are 2 main advantage of using permission blockchain over public blockchain and they are explained in the following subsections.

7.3.1 Security

To achieve full decentralization, in public blockchain like bitcoin and ethereum, the data and computation is replicated in all nodes of the network. As the validation logic is run in every node, this increases the attack surface as anyone can be the validator and receive ballots during the election on the fly. With permissioned blockchain, only the trusted parties like political and civil organization can participate and restrict the validation role and data flow. Even though not everyone can validate the ballots on

the fly, they can rely on the organization they trust to act in a non-dubious manner, which basically forms a chain of trust and therefore decentralizes the authority. Also, in a public blockchain, it is possible for adversary nodes of the network can create partitions in the network and act in an unpredictable way. For example, take long to run a smart contract and make the whole network unpredictable. Thus, a voting system being a complicated system can have unexplored security vulnerabilities and as everyone can be part of the network in public blockchain, it would be easy to exploit the system compared to permissioned blockchain. In other words, the attack surface of public blockchain is greater than private blockchain. This makes a compelling argument to use permissioned blockchain to implement voting systems.

7.3.2 Cost and Speed

It is obvious to think that the system which replicates the computation and data in all the nodes in a network is slower and expensive compared to the system where such replication happens on a smaller number of nodes. To make the above argument more believable, we compare an ethereum based voting system [27] with the one we proposed. As per the paper, gas consumed to cast one ballot is 3M and cost of 1 gas is 50 gwei, a unit of ethereum gas. Total gwei for 1 ballot cast is 150M gwei or 0.15 ether (1 gwei = $10^{-}9$). Similarly, gas consumed for voter registration is 15M gwei or 0.015 ether. As of 30th June 2020, the price of 1 ether is 310$ which implies that -

- Cost to cast all ballot = 210M * 0.15 * 310 = $9.7 Billion
- Cost to register all voters = 210 * 0.015 * 310 = $0.97 Billion
- Total cost = $10.67 Billions

As per the section 5.1, we showed it would require only $890K to form a permissioned blockchain network based on HLF. Also, the transaction rate of ethereum is quite low which is 6-10 transactions per second. Hence, public blockchain like bitcoin and ethereum have higher cost of computation and lower throughput compared to permissioned blockchain which makes a compelling reason to implement voting systems in permissioned blockchain like HLF.

7.4 Analysis of System Properties

7.4.1 Decentralization

One of the main reasons for blockchain being used for designing voting systems is that it provides decentralization of authority. Ordering of the transactions and validation of business logic requires participation from all the organizations involved in the network. This means that the authority of the election committee over ensuring the correctness of the election is distributed over multiple organizations and chances of detecting foul play will be relatively better. Such decentralization is fruitful only when the competing organizations do not have common interest. The fact that the blockchains are considered irreversible is because any illegal data change can be detected by consulting the majority of the peers in the blockchain. Also, looking at the log of consensus protocol, it should be possible to find out which organization tried to sabotage the election. If the competing organizations have common interest, there might arise a scenario where the majority of the organizations might conspire, change the data in blockchain and change the trusted truth of the network and hence change the outcome of the election. Thus, if the competing political parties and civic organizations are kept as the peers of the network then it is safer to assume that they won't have common interest which makes the network to be trustful.

7.4.2 Transparency and Verifiability

As the ballots stored in the blockchain are encrypted, anyone can be given access to the data without the possibility of leaking ballot data. The authenticity and integrity of every operation in blockchain can be checked via hash and signature verification. Anyone with access to the data can verify the final result of the election by tallying themselves. As the data is encrypted with homomorphic encryption, anyone can perform the sum operation on the encrypted ballot data to calculate the final tally of the election. The final tallied value can be submitted to a election authority server which can provide decrypted value of the election with a zero knowledge proof of decryption which can verify the authenticity of the decryption performed by an entity without leaking anything about the private key. Not only can the voters verify the integrity of the tally process, they can also verify if their ballot was casted as intended. They can compare the encrypted content of the ballot at the time of voting with the encrypted ballot that is present in the blockchain. This way voters can verify if there was error

or malign during elections.

It is to be noted that provision of access for transparency does not mean anyone with the wrong intention can change or destroy the data as the blockchain ensures irreversibility. They would have to take control of all the participating organizations to do so. Hence, in blockchain and homomorphic encryption based voting systems, the correctness of the election can be verified by anyone resulting in increase of public trust in the election procedure.

7.4.3 Privacy

Even Though the ballots are publicly available, the privacy of the ballots are still maintained as they are encrypted with a private key maintained by election authority. Given that the private key of the election authority is not leaked, we can assume that the privacy of the ballot can be maintained. Ballots are recorded as key value pairs where key is a pseudo identity of the voter whereas value is the encrypted ballot. The mapping of pseudo identity and real identity of the voter is maintained by the election authority. If such mapping is securely maintained then it is also possible to privatize whether a voter has voted or not.

7.4.4 Robustness

As the computation and data is replicated, blockchains are high availability systems. Even if a node goes down or minority of the nodes malfunction, the system can still function. For a permissioned blockchain with endorsement policies like HLF, the configuration of endorsement policy determines how many node failures the system can resist. If the endorsement policy is set such that x endorsements are required then in a system with n nodes, $n - x$ failures can be resisted.

7.4.5 Feasibility

As blockchain is slower than traditional data stores like relational databases, it is important that the blockchain can process the volume of transaction that is expected in the production environment. The major objective of this thesis is to improve the feasibility of usage of blockchain for the voting application. With more than 30% improvement in latency and throughput after the exploration of characteristics of

HLF, we believe we have made the usage of blockchain more feasible in terms of cost and performance when compared to other blockchain based voting systems [17], [22], [27].

Chapter 8

Conclusions

8.1 Conclusion

Even though the use of blockchain makes voting systems more transparent and trustable, implementing a secure remote voting system is a complex task as there are social, political and technological factors to be taken into consideration. We found that as the throughput of blockchain is low, it can be a major bottleneck in implementing a practical voting system. We made an attempt at analyzing the performance of one of the most popular blockchain platforms, Hyperledger fabric. The performance profile data suggested that most of the time during transaction processing is spent on cryptographic operations and golang's garbage collector operations. As garbage collectors have complex behaviour and are difficult to understand, we concentrated our optimization effort on other aspects of transaction processing. We replaced the slower native golang cryptographic library with faster alternatives and observed more than 20% improvement in the throughput and latency. We also analyzed the concurrency of transaction processing and found that for single keyed transaction read write locks are not required to ensure repeatable read isolation. Like database systems, we were able to tune the concurrency mechanism and gained more than 10% improvement in the throughput and latency. Even though the initial idea was to improve the performance of the remote voting system, we ended up providing optimization on Hyperledger fabric that could also be used in other areas like supply chain management, smart property management, etc. Thus, we believe that the goal of improving the performance of the election system was achieved by the optimizations we proposed.

8.2 Future Work

There are 2 directions that can be followed to further extend the scope of this thesis – (i) make voting protocol more practical (ii) further analyze and improve the performance of underlying blockchain. Some of the task that can done as part of future work are listed below:

1. Even if the cryptographic primitives for encryption, signature verification, etc are secure, if the platform i.e. hardware or operating system on which voting client application is running is not secure, the system cannot be claimed to be secure. Ensuring the client application, operating system or the hardware being used to run the client application is secure is quite difficult. For example, in cases where the operating system is infected, the attacker might be able to tamper the voting process for the user without user being able to know about it. Therefore, it is dire to be able to ensure that the voting client application is running without any interference. One way to do so is by using trusted computing base technologies like Intel SGX [10] which can ensure that the application runs without any interference. Thus, it would be quite interesting to see how the practicality of the voting protocol can be improved by the use of trusted computing base for implementing client application.

2. From performance analysis we found blockchain and voting protocol are highly dependent on the encryption scheme and signature scheme. We believe performance of the whole system could be improved if faster algorithms or the efficient implementations are used. As part of this thesis, we opted for faster implementation of the existing cryptographic algorithm. One of the tasks that can be done as part of future work is investigating the suitability of faster algorithms for encryption schemes and signature schemes.

3. As part of this thesis, we set up an experiment where all the peer nodes of the blockchain network had negligible latency but in reality when the peer nodes are deployed across data centers, network behavior can affect the characteristics of the system. As part of the future work, the effects of the network on the performance of blockchain network can be studied.

4. For programmer's ease Hyperledger fabric is implemented using Golang which has automatic memory management. From our performance analysis we found

that the task done as part of automatic memory management or in other words, garbage collector, affects the working of Hyperledger fabric quite a lot. The garbage collector can be tuned and its impact on the performance of the system could be studied. Also, it would also be quite interesting to see performance of the Hyperledger fabric had it been implemented in a language like C++.

5. As the number of votes in a national election is quite large, there will be a requirement for a fast analytical component to tally the votes. It can be interesting to see how distributed computational framework like Apache Spark can be integrated with the blockchain to develop a fast analytical tool.

Bibliography

[1] Ben Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

[2] Ben Adida, Olivier De Marneffe, Olivier Pereira, Jean-Jacques Quisquater, et al. Electing a university president using open-audit voting: Analysis of real-world use of helios. *EVT/WOTE*, 9(10), 2009.

[3] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[4] Chris Baraniuk. Bitcoin's energy consumption 'equals that of switzerland'. *British Broadcasting Corporation*.

[5] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 274–283, 2001.

[6] Stefano Bistarelli, Marco Mantilacci, Paolo Santancini, and Francesco Santini. An end-to-end voting-system based on bitcoin. In *Proceedings of the Symposium on Applied Computing*, pages 1836–1841, 2017.

[7] Henry E Brady and John E McNulty. Turning out to vote: The costs of finding and getting to the polling place. *American Political Science Review*, pages 115–134, 2011.

[8] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[9] Michael R Clarkson, Stephen Chong, and Andrew C Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 354–368. IEEE, 2008.

[10] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[11] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International conference on financial cryptography and data security*, pages 106–125. Springer, 2016.

[12] Jason Paul Cruz and Yuichi Kaji. E-voting system based on the bitcoin protocol and blind signatures. *IPSJ Transactions on Mathematical Modeling and Its Applications*, 10(1):14–22, 2017.

[13] Ivan Damgaard, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

[14] CSIRO's Data61. Python paillier library. `https://github.com/data61/python-paillier`, 2013.

[15] Kevin Driscoll, Brendan Hall, Michael Paulitsch, Phil Zumsteg, and Hakan Sivencrona. The real byzantine generals. In *The 23rd Digital Avionics Systems Conference (IEEE Cat. No. 04CH37576)*, volume 2, pages 6–D. IEEE, 2004.

[16] Laure Fouard, Mathilde Duclos, and Pascal Lafourcade. Survey on electronic voting schemes. *supported by the ANR project AVOTÉ*, 2007.

[17] Sebastian Gajek and Marco Lewandowsky. Trustless, censorship-resilient and scalable votings in the permission-based blockchain model. *IACR Cryptol. ePrint Arch.*, 2019:617, 2019.

[18] Nicole J Goodman. Internet voting in a local election in canada. In *The internet and democracy in global perspective*, pages 7–24. Springer, 2014.

[19] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. Fastfabric: Scaling hyperledger fabric to 20 000 transactions per second. *International Journal of Network Management*, 30(5):e2099, 2020.

[20] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. Building high throughput permissioned blockchain fabrics: challenges and opportunities. *Proceedings of the VLDB Endowment*, 13(12):3441–3444, 2020.

[21] Timothy A Hall and Sharon S Keller. The fips 186-4 elliptic curve digital signature algorithm validation system (ecdsa2vs). 2010.

[22] Fririk Hjalmarsson, Gunnlaugur K Hreiarsson, Mohammad Hamdaqa, and Gsli Hjlmtsson. Blockchain-based e-voting system. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 983–986. IEEE, 2018.

[23] Heng Hou. The application of blockchain technology in e-government in china. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–4. IEEE, 2017.

[24] Nitin Gaur Jai Singh Arun, Jerry Cuomo. *Blockchain for Business*. Addison-Wesley Professional, 2019.

[25] Robert Künnemann and Graham Steel. Yubisecure? formal security analysis results for the yubikey and yubihsm. In *International Workshop on Security and Trust Management*, pages 257–272. Springer, 2012.

[26] Yi Liu and Qi Wang. An e-voting protocol based on blockchain. *IACR Cryptol. ePrint Arch.*, 2017:1043, 2017.

[27] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In *International Conference on Financial Cryptography and Data Security*, pages 357–375. Springer, 2017.

[28] Michael McDonald. 2020 general election early vote statistics.

[29] Hagar Meir, Artem Barger, Yacov Manevich, and Yoav Tock. Lockless transaction isolation in hyperledger fabric. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 59–66. IEEE, 2019.

[30] Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. A survey of blockchain from the perspectives of applications, challenges, and opportunities. *IEEE Access*, 7:117134–117151, 2019.

[31] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[32] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[33] Shirley M Radack. Updated digital signature standard approved as federal information processing standard (fips) 186-3. 2009.

[34] Maximilian Schiedermeier, Omar Hasan, Lionel Brunie, Tobias Mayer, and Harald Kosch. A transparent referendum protocol with immutable proceedings and verifiable outcome for trustless networks. In *International Conference on Complex Networks and Their Applications*, pages 647–658. Springer, 2019.

[35] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*, pages 148–164. Springer, 1999.

[36] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *Proceedings of the 2019 International Conference on Management of Data*, pages 105–122, 2019.

[37] Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[38] Parth Thakkar, Senthil Nathan, and Balaji 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)*, pages 264–276. IEEE, 2018.

[39] Mischa Tripoli and Josef Schmidhuber. Emerging opportunities for the application of blockchain in the agri-food industry. *FAO and ICTSD: Rome and Geneva. Licence: CC BY-NC-SA*, 3, 2018.

[40] Melanie Volkamer, Ammar Alkassar, Ahmad-Reza Sadeghi, and Stefan Schulz. Enabling the application of open systems like pcs for online voting. In *Proc. of Workshop on Frontiers in Electronic Elections*, 2006.

[41] Samuel Weiser and Mario Werner. Sgxio: Generic trusted i/o path for intel sgx. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 261–268, 2017.