

ACTA DE APROBACION DE TESIS

Hierarchical Reference Shard Cross-Shard Transaction Processing for Blockchains

Por: Marco Vinicio Acuña Vargas

TRIBUNAL EXAMINADOR

FRANCISCO
JOSE TORRES
ROJAS (FIRMA) Digitally signed by
FRANCISCO JOSE
TORRES ROJAS (FIRMA)
Date: 2023.07.18
14:54:14 -06'00'


Dr. Francisco Torres Rojas
Profesor Asesor

Firmado por JOSE RAFAEL CASTRO MORA (FIRMA)
PERSONA FISICA, CPF-01-0653-0432.
Fecha declarada: 29/08/2023 02:20 PM
Esta representación visual no es fuente
de confianza. Valide siempre la firma.

Dr. José Castro Mora
Profesor Lector

LEONARDO
ANDRES ARAYA
MARTINEZ (FIRMA) Digitally signed by LEONARDO
ANDRES ARAYA MARTINEZ (FIRMA)
Date: 2023.06.29 14:45:28 -06'00'

M.Sc. Leonardo Araya Martínez
Lector Externo

 LILIANA SANCHO CHAVARRIA (FIRMA)
PERSONA FISICA, CPF-03-0257-0983.
Fecha declarada: 02/10/2023 09:19:24 AM

Dra.-Ing. Lilliana Sancho Chavarría
Presidente, Tribunal Evaluador Tesis
Programa Maestría en Computación



19 de junio, 2023

Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Programa de Maestría en Computación



**Hierarchical Reference Shard Cross-Shard Transaction
Processing for Blockchains**

Presented on November 2022, thesis project
to opt for the degree of **Master Scientiae in Computing with emphasis in
Computer Science.**

By
Marco Acuña Vargas
201022750

Supervised by Francisco Jose Torres Rojas, PhD.

I hereby declare that this dissertation is entirely my work, except as indicated in the text.

Signature: _____

Date: ____/____/____

This graduation thesis proposal has been accepted by the evaluation board of the School of Computing engineering of Tecnológico de Costa Rica and was also reviewed as a partial requisite to opt for the master degree.

Hierarchical Reference Shard Cross-Shard Transaction Processing for Blockchains

Members of the evaluation board

Francisco J. Torres-Rojas, Ph.D.
Thesis Director

Lilliana Sancho Chavarría, Ph.D.
Computer Science Graduate Coordinator

José Castro Mora, Ph.D.
Computing Engineering School Professor

Leonardo Araya Martínez, M.Sc.
External Professor

Marco Acuña Vargas
Student

Acknowledgments

I want to say thanks to my family, without whom this effort would not have been possible and would have no sense.

Thanks Mom and Dad, thanks for always believing in me.

Thanks to my girlfriend Fer, thanks for being here.

Marco Acuna Vargas Cartago, November 4th, 2022

Summary

In the present work it is proposed to study hierarchical implementations for a reference shard in sharded blockchain environments. Classic sharded blockchains suffer from low cross-shard transaction throughput, given that common cross-shard protocols such as two phase commit, compromise throughput to prioritize consistency. The results of this study could lead to the creation of better sharded blockchain systems with enhanced cross-shard throughput with no compromises to system consistency. This research will use simulations with real main net transactions with the intent of analyzing the yield in performance that having a hierarchical structure of reference shards could have in cross-shard transaction throughput and latency. The results of this experiments will be statistically analyzed (ANOVA) to determine that there could be a significant advantage of using a hierarchical implementation against using a single reference shard implementation.

Keywords: Blockchain, Sharding, Cross-Shard Transactions

Contents

List of Figures	x
List of Tables	xi
1 Introduction	1
1.1 Problem Statement	3
2 Hypothesis and Objectives	6
2.1 Hypothesis	6
2.2 General objective	6
2.3 Specific objectives	7
3 Theoretical Framework	8
3.1 History of Blockchain	8
3.2 The first steps of Blockchain: Bitcoin	9
3.3 Solutions to scalability issues	12
3.3.1 Blockchain Layers	13
3.3.2 Blockchain consensus layer solutions	13
4 Implementation	19
4.1 ShardSim design	20
4.1.1 ShardSim classes and simulation details	21
4.2 Generalizing shard algorithm	22

4.3	Experiment design	25
4.3.1	Initial notions for experiment design	25
4.3.2	Experiment setup	28
4.3.3	Experiment simulation parameters	29
5	Results	31
5.1	Intra Shard Throughput and Latency	32
5.2	Cross Shard Throughput and Latency	35
6	Conclusions	39
7	Bibliography	41

List of Figures

3.1	Distributed blockchain ledger	10
3.2	Bitcoin timestamp server [1]	10
3.3	Transactions per second of widely used payment systems, derived from [2]	12
3.4	Sharded blockchain [3]	15
3.5	Practical Byzantine Fault Tolerant algorithm [3]	16
3.6	Client Driven Two Phase Commit algorithm [3]	17
3.7	Rivet framework Cross Shard protocol [4]	18
4.1	ShardSim high level overview	21
4.2	Types of shard in a hierarchical topology of shards	23
4.3	Hierarchical shard blockchain example	27
5.1	Intra Shard throughput across different architectures	33
5.2	Intra Shard latency across different architectures	34
5.3	Cross Shard throughput across different architectures	36
5.4	Cross Shard latency across different architectures	37

List of Tables

3.1	Current solutions for blockchain scalability [5]	14
4.1	Experimental architectures	28
4.2	Experiment Blockchain Parameters	30
5.1	Intra Shard Throughput Anova Results for Shards=10	33
5.2	Intra Shard Latency Anova Results for Shards=10	35
5.3	Intra shard throughput and Latency Variation with each architecture configuration	35
5.4	Cross Shard Throughput Anova Results	37
5.5	Cross Shard Latency Anova Results	38
5.6	Cross shard throughput and Latency Variation with each architecture configuration	38

1. Introduction

In 2009, a person or persons under the pseudonym of Satoshi Nakamoto published the white paper of the first successful digital currency, Bitcoin [6]. This work presented the idea of using a distributed network of computers to maintain a ledger that could record transactions in a totally decentralized and trust less fashion. This network would have nodes where every node would maintain a local copy of a data structure called blockchain and these nodes would be rewarded for keeping the network safe and behave according to an established protocol.

The blockchain was proposed as a timestamp server [1] composed of a succession of blocks that could hold transactions as payload and where each block would be linked to its predecessor by using a hash pointer.

This technology proved to be very secure in part due to its distributed nature and to the fact that recording the historical log of transactions in a linked list of hash related blocks made it tamper proof. This means that if any malicious actor tries to modify any of the past transactions confirmed in the network, it would have to change the whole chain of hashes in the rest of successive blocks, which in turn would be noticed as an unmistakable sign of tampering with the system.

Bitcoin implementation also solved the so called double spending problem, that was an unresolved challenge in the field. By using the blockchain data structure, transactions were recorded in an unchangeable order, meaning a person could not

spend the same token twice since the origin of each token could be tracked since the beginning, making impossible to spend something that was already spent.

After bitcoin's success, there were a number of other projects that by harnessing the benefits of blockchain technology were able to put together cryptocurrencies with enhanced functionalities. For instance, Ethereum was created as a cryptocurrency that leveraged blockchain to record transactions, but abstracted away the idea of transaction execution and generalized it into a general purpose code execution engine, which gave birth to smart contracts [7].

Very soon many computer scientists began to observe that despite the strong security properties of cryptocurrencies implemented using blockchain, it would be hard to reach mass adoption due the lack of capacity of these systems to scale. The amount of transactions that these networks were capable of processing is very low, for instance bitcoin can only process as much as 15 tx/s, which is very low when compared to other payment systems like Visa that is capable of processing 65000 transactions per second [5].

Blockchain technology is normally modeled as 3 layer stack, where layer 0 is related to data, layer 1 is related to consensus and layer 2 is related to off chain solutions [3]. So far there have been many different attempts in all of these layers. Scalability is likely to be achieved by a combination of solutions from all three layers [5].

The consensus layer has been a layer where many computer scientists have focused, since improvements in this layer would be the ones that would rebound the horizontal scalability capacity of blockchain systems. One of the areas that has re-

ceived most of the attention in recent years is sharding [8]. Sharding is a divide and conquer approach, where the responsibility of maintaining the state of the system is divided among the nodes of the network.

Sharding technique basically divides the network into smaller groups of nodes. This exploits the locality of transactions to avoid the need of involving the whole set of nodes. This means transaction could be processed only in the nodes that maintain the state related to that transaction. Nevertheless, this techniques has very important drawback, such as reduced tamper resistance and poor cross-shard transaction throughput [4].

In a sharded blockchain system cross-shard transactions are of crucial importance as lacking the ability to connect the different subgroups that compose the system, would degenerate into a set of blockchains working independently.

1.1. Problem Statement

Blockchain systems are the backbone of today's cryptocurrencies. Cryptocurrencies are one of the most promising technologies to transform global economy [9]. This technologies will allow the democratization of currency creation through decentralization. Nevertheless, they have not been able to reach wide mass adoption due in part to their poor scalability.

Since their inception, blockchain systems have suffered from poor scalability properties [5]. The consensus protocols that make these systems successful in terms of security have undermined their ability to process a large number of transactions per second.

The most popular and widely used blockchain systems today can rarely process more than a 1000 transactions per second. When these systems are compared against wide adopted credit card networks like Visa (65000 tx/s), it is evident that there is a long way in blockchain development so that one day these systems will be able to take over cash payments.

A number of solutions have been proposed to alleviate this system weakness. Sharding, is one of such solutions and is a well-known approach to increase throughput in databases [8]. When sharding is applied to a distributed ledger running a blockchain protocol, a big monolithic network is split into several subgroups with each of the subgroups maintaining its own blockchain.

Since the states stored into each of the blockchains are disjoint, a mechanism is needed to allow transactions to happen between different shards. Transactions that traverse many shards are called *Cross Shard Transactions*. The mechanisms to implement this transactions needs to guarantee atomicity and prevent double-spending [3].

In the sharding space there have been a number of approaches to implement the cross shard mechanism, for instance most sharded blockchain use an approach called *two phase commit*. This approach basically blocks value in the origin shards and then transfers that value to the destination shard [8]. The problem with this common approach is that the protocol needs to wait until all the shards that are involved in a transaction confirm value has been locked, which in turn compromises the protocol throughput.

So far the community has been trying different alternative approaches. One

of these approaches is by using a reference (central) shard [4], where worker shards continually publish their state and cross shard transactions are processed on top of the states that worker shards publish to reference shards. This approach uses optimistic assumptions and doesn't rely on performing busy waiting until value is locked in origin shards.

Implementations with reference shards were proven to perform better than other implementations using *2pc* [4]. Nevertheless, this approach has not been tested while using a hierarchy of reference shards, which could in turn yield better performance and allow this type of protocols to scale better. So, basically it is unknown if whether or not using a hierarchy of reference shards these type of protocols would improve their cross shard transaction throughput and latency while intra shard transactions (transactions that do not cross shards) throughput and latency remain unaffected.

2. Hypothesis and Objectives

2.1. Hypothesis

There have been many sharded blockchain implementations that take advantage of using a reference shard pattern, for instance rivet, polkadot (Relay chain), ethereum 2.0 (beacon chain) and they have proved to be efficient, although no study has tested that using a hierarchy of reference shards could increase throughput. So, the hypothesis for this research is the following:

A sharded blockchain (like Rivet framework) implemented using a hierarchical structure of reference shards will yield an increase of at least 10%¹ in cross-shard transaction throughput, while having the same cross-shard transaction latency and without affecting intra-shard transaction throughput and latency² when tested against a blockchain using a single reference shard using organic main network transaction benchmarks (Ethereum transactions).

2.2. General objective

Determine if a hierarchical structure of reference shards implementation of a sharded blockchain system (rivet framework) yields an advantage in terms of cross shard trans-

¹Given that performance gain of a framework like Rivet against 2PC is of 20% and that experimental throughput was 80% of input rate when using a single reference shard.

²Given that a framework like Rivet experimental intra-shard latency is 5 times the one of 2PC it would be undesirable to have an increase in this metric

action throughput against a single reference shard implementation.

2.3. Specific objectives

1. Design the algorithms for cross shard transaction processing with hierarchical reference shards.
2. Extend a python based simulation framework (BlockSim [10]) to test both hierarchical and single reference shard implementations.
3. Explore which topologies yield best results for the hierarchical implementation.
4. Analyze how intra-shard transaction confirmation latency is affected in a sharded hierarchical setup.
5. Explore how the number of worker shards affect the structure of the hierarchical setup.

3. Theoretical Framework

This chapter presents all the fundamental theoretical concepts that are directly or indirectly related to the development of this research.

3.1. History of Blockchain

The first mention to digital money happened back in 1983, David Chaum was the first to describe digital money [11]. Then, in the 1990's there were companies like paypal who implemented web-browser based payment systems that allowed individuals to pay one another by using their emails as identities. In 1998 Wei Dai, proposed B-money [12], which shares many similarities to what Nakamoto proposed [1] and which eventually became the worlds first successful cryptocurrency. After B-money it came Bit Gold, which was another digital currency which had its own ***Proof-of-Work (PoW)*** mechanism through which the solutions were cryptographically signed and broadcasted to the public [11, 13]. In 1997, Adam Back proposed ***Hashcash*** as a denial of service counter measure for services such as email [14]. Hashcash also made use of a PoW algorithm for generating new coins and faced many of the same problem of its predecessors.

In 2009, Bitcoin was presented in [1] by a person under the pseudonym of Satoshi Nakamoto. This work proposed the use of a peer to peer network that would achieve consensus through proof of work algorithm (PoW). Also, one of the innovations intro-

duced by Nakamoto was the use of a data structure called blockchain as a ledger to store transactions.

3.2. The first steps of Blockchain: Bitcoin

Bitcoin was proposed as a peer to peer system. In this system the nodes would collect transactions from clients (users) and by using a gossip protocol they would transmit these transactions to the rest of the system. After that, they would compete (through a PoW protocol) to add these transactions to a distributed ledger called blockchain. In figure 3.1, it can be seen the basic dynamic of such distributed ledger implemented through blockchain. The blockchain is a data structure that exist in each member of the network and it is by following a consensus protocol that nodes will be able to update their local copy of the blockchain.

The concept of blockchain entails a timestamp server as can be seen in figure 3.2, this idea works by creating a hash of a block of data to be timestamped and publishing that hash to all the network. This timestamp would in turn prove that the data must have existed at the time. Each timestamp would also include the previous timestamp in its hash, this would form a chain with each new timestamp and the succeeding timestamps would in turn reinforce the ones before [1].

The way of adding new blocks to this timestamp server is through a consensus protocol called Proof of Work. In this protocol, to propose a new block for the blockchain a node also called a miner, creates the new block out of as many transactions (that it has in its transaction pool collected from clients and other nodes) that could be fitted into 1 MB [1] of space [6]. Then, the miner would add a nonce to start

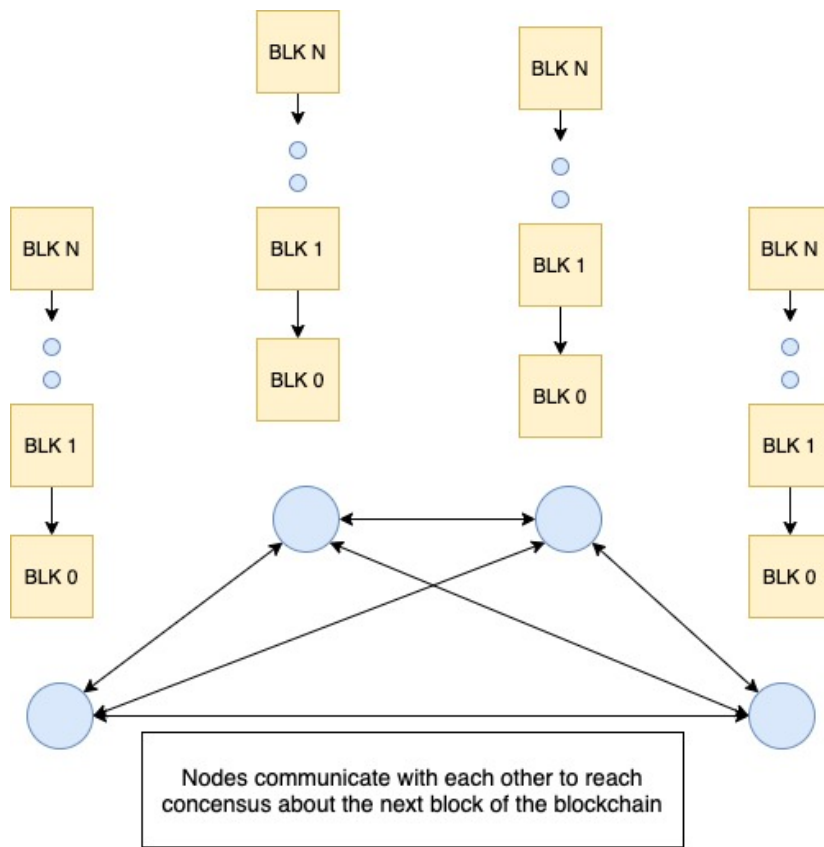


Figure 3.1: Distributed blockchain ledger

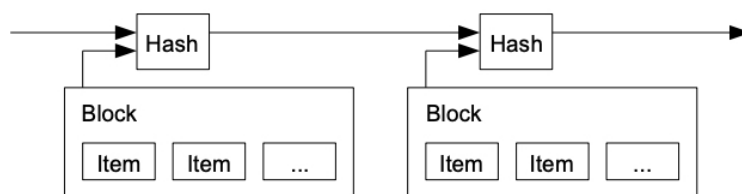


Figure 3.2: Bitcoin timestamp server [1]

calculating a **sha256** hash of the block (transactions + nonce) and verifies if such hash is less than a specific number called **difficulty**. If the hash obtained from such calculation is not less than the difficulty then the miner would increase the nonce by one and repeats until a hash less than the difficulty is obtained [6]. The process of looking for a nonce that satisfies the difficulty is called **mining**.

This means that in order to add new transactions to the blockchain the miner needs to provide a proof that it has made a lot of work (and he will be rewarded for this work). This proof in turn is hard to fake, since the proof of work protocol works with a problem that is very hard to solve but very easy to verify. This means that while a miner could have used thousands of computing cores to solve the problem fast, anyone in the system could verify his solution with commodity hardware [6].

This protocol makes blockchains tamper proof in the sense that if someone wants to deviate from the protocol and introduce forged transactions to benefit from that, it will need to invest a huge amount of computing power (money) to compete in the proof of work protocol. This in turn means that it will need to beat the **hash rate** (computing power) of the rest of the honest nodes in the network. So, if the network has a big hash rate, it will be nearly impossible and unattractive for a rational player to try to perform this type of attack, this is called 51% attack.

Consensus protocols have made blockchains very secure and resistant to tampering, but they have affected their scalability. In blockchain, scalability means the ability to increase the throughput of a network if the amount of nodes is also increased [5]. Due to the use of consensus algorithms like PoW, the mechanism to process transactions is strongly coupled to the consensus algorithm [15]. This coupling in turn makes

throughput heavily dependent on the speed of the consensus protocol which is more likely slow [5]. In figure 3.3, one can see a bar chart with the amount of transactions per second for mainstream payment systems and some popular cryptocurrencies. In this chart, it can be seen that *Visa* which is a widely used credit card system allows 45000 transactions per second. It can be argued that if blockchain systems would take over world's economy one day, they should at least be capable of processing as much transactions as Visa. Furthermore, considering that Visa is not able to handle all of the worlds transactions, cryptocurrencies and blockchain technologies would need to go way beyond 45000 *tx/s* throughput to achieve such goal.

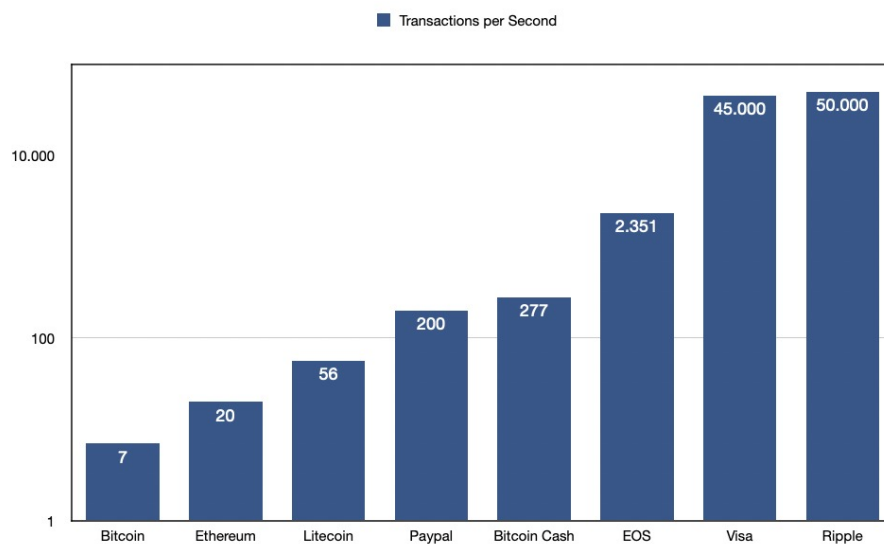


Figure 3.3: Transactions per second of widely used payment systems, derived from [2]

3.3. Solutions to scalability issues

To ease the development of blockchain technology and also analyze the solutions to scalability issues the community has created a model of layers, composed of three layers: **network**, **consensus** and **application** [3].

3.3.1 Blockchain Layers

Network Layer: In this layer nodes join a *peer to peer* p2p network to synchronize information with each other. In p2p networks there is no central communication node [3].

Consensus Layer: the nodes that participate of this layer act as block producers and generate such blocks by following a consensus algorithm.

Application Layer: On top of consensus layers, there may exist applications such as digital assets, smart contracts and decentralized applications [3].

3.3.2 Blockchain consensus layer solutions

The consensus layer is where transactions get processed, thus this is the layer that is directly related with the blockchain throughput [5]. This is the layer that needs to be modified the most to increase the amount of transactions that can be processed. As seen in table 3.1, this layer is also called *Layer 1* is divided into 4 areas. These areas aim towards vertical scalability and then others that aim towards horizontal scalability.

Vertical scalability has been used in projects such as bitcoin cash where the size of each block in the chain has been increased to allow for more transactions to fit in, thus increasing the amount of transactions processed per run of the consensus algorithm. Nevertheless, this approach has been demonstrated [5] to lead to a more centralized network, given that nodes that participate of the protocol should have enhanced capabilities in terms of networking hardware to be able to handle blocks of bigger size. This approach is similar to the *SegWit* in bitcoin project which is a solution

Table 3.1

Current solutions for blockchain scalability [5]

Layer	Categories	Solutions
Layer 2: Non On-Chain	Payment Channels	Lightning Network, DMC Raiden Network, Sprites
	Side Chain	Pegged Sidechain, Plasma liquidity.network
	Cross-chain	Cosmos, Polkadot
Layer 1: On-Chain	Block Data	SegWit, Bitcoin Cash Compact Block relay, Txilm CUB, Jidar
	Consensus	Bitcoin-NG, Algorand Snow white, Ouroboros
	Sharding	Elastico, Omniledger RapidChain, Monoxide
	DAG	Inclusive, SPECTRE, PHANTOM, Conflux, Dagcoin, IOTA, Byteball, Nano
Layer 0	Data Propagation	Erlay, Kadcast Velocity, bloXroute

to transaction malleability but that at the same time increases block size to allow for more transactions to fit into the block [6]. If this kind of solution is extrapolated further that would lead to a blockchain network that is highly centralized, which could in turn lead to increasing risks of the blockchain being hijacked by a group of nodes that collude to control the protocol [8].

The horizontal approach to blockchain scalability aims to modify the mechanisms that are used to process the transactions. In this space the most promising solution is sharding. Sharding is a divide and conquer approach where the fact that a transaction that involves only 2 different parties should not cause all the network to spend bandwidth to process such transaction [8].

In figure 3.4 there is an example of how such a system would work. It can be seen that basically the whole network is divided into groups of nodes called *shards*. Each

one of these groups would process transactions and maintain a local (to the group) blockchain. These blockchains hold disjoint states meaning that each of them has the record of a subgroup of the accounts of the system and if there is a transaction that only involves transactions that exist in the local shard blockchain then that transaction would be processed locally, this transactions are called *Intra Shard transactions*. There may be a case in which a transaction could involve tokens or value that is locked into different shards, this transactions are called *Cross Shard transactions*.

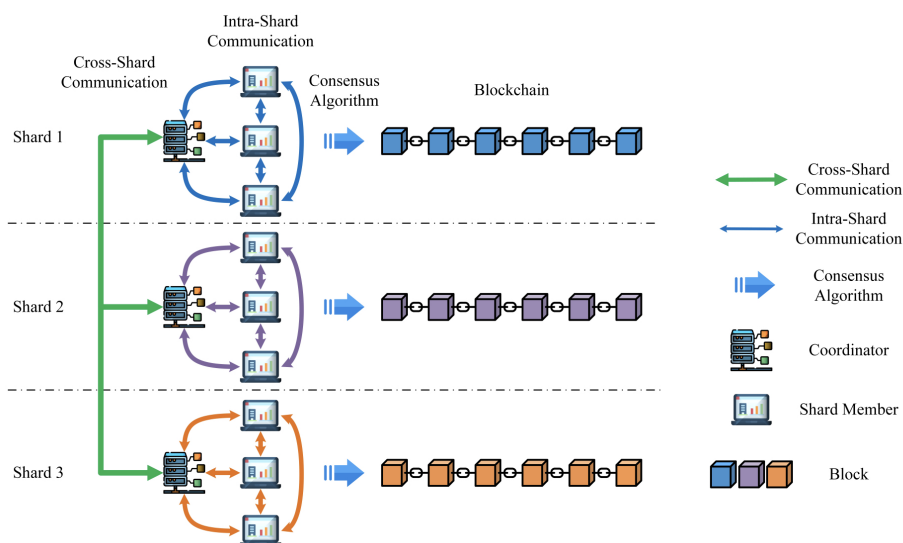


Figure 3.4: Sharded blockchain [3]

There are already many different projects such as Elastico [16], Chainspace [17], Omniledger [18], RapidChain [19] and Monoxide [20] that have leveraged sharding techniques to increase the transaction throughput and lower transaction latency. These protocols are far more complex than bitcoin [3], because they need to have mechanisms for node selection, epoch randomness, node assignment, intra shard consensus, cross shard consensus and reconfiguration.

These protocols use a combination of *Practical byzantine fault tolerance* consensus algorithms for intra shard consensus and then special consensus like *two phase*

commit algorithms for cross shard transaction processing. *PBFT* algorithms allow shards to reach consensus quickly, but as seen in figure 3.5 they rely on several phases of message exchanges. This makes these algorithms to scale badly and make their execution expensive.

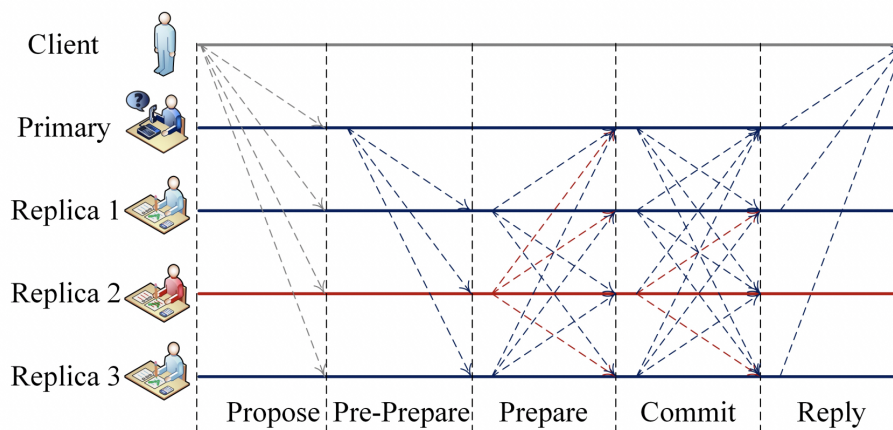


Figure 3.5: Practical Byzantine Fault Tolerant algorithm [3]

Even though these protocols have been able to theoretically increase the blockchain bandwidth, cross shard transaction consensus remains a bottleneck [4]. This happens because of the *two phase commit* algorithm’s reliance on running *PBFT* intra shard consensus algorithm [4].

In figure 3.6, an execution example can be seen of a client driven two phase commit protocol. In this protocol when there is a transaction between shards (cross shard), the values within the input shards need to be locked, so the client sends a lock request to the shards that in turn need to run the intra shard consensus protocol to agree on the lock value. After the values are locked, the input shards will send a certificate signed by the majority of the nodes. The client will then be able to send this certificate to the output shard where the value (or tokens) will be assigned to the account of the transaction recipient. Each time the client sends a message to each of

the shards involved in the transaction, the intra shard consensus needs to run, which makes this approach expensive in terms of message exchanges.

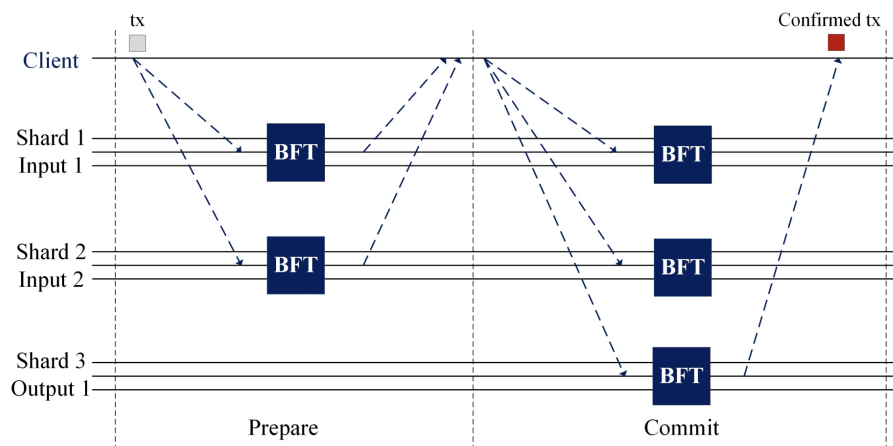


Figure 3.6: Client Driven Two Phase Commit algorithm [3]

The protocols in [17, 16, 18, 19] use two phase commit protocol (2pc), which affects their cross shard transaction throughput. So, even though this solution has proved to be effective processing thousands of intra shard transaction per second the cross shard transaction mechanism is a bottleneck that affects overall system performance [4].

As an alternative to 2PC protocol there are some protocols like ethereum 2.0 (beacon chain), polkadot (relay chain) and Rivet framework which use reference shards. Which are basically central nodes that maintain a main chain where the state of all the worker shards is committed through hashes. In figure 3.7, there is a reference shard that constantly receives the hashes of the state of each of the worker shards, therefore if a cross shard transaction occurs the transaction will be processed on top of the states reported by worker shards. This means worker shards run an optimistic algorithm that whenever there are no cross shard transactions will run in a normal fashion and will be able to do progress each time it commits its state to the reference shard. In case a cross

shard transaction happens, after the transaction is processed in the reference shard it will invalidate state that has not been committed to reference shard. This approach eventually eliminates the need to block value in each shard with the drawback that it will affect the intra shard throughput [4].

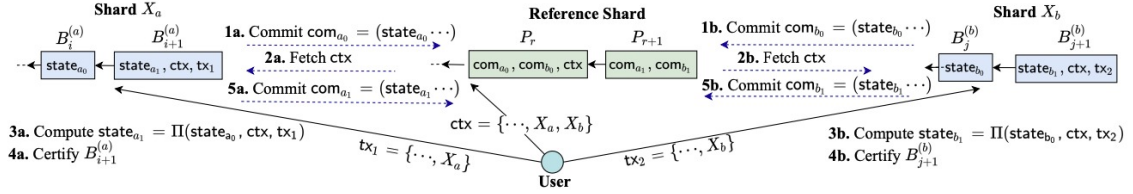


Figure 3.7: Rivet framework Cross Shard protocol [4]

The implementation of sharded blockchain with a central reference shards may yield performance benefits in relation to the performance of sharded blockchains that use 2PC [4]. Nevertheless, having a single reference shard will only allow the network to grow up to a certain size because of the following reasons:

- The reference shard needs to run a PBFT protocol which scales bad.
- The nodes in the reference shard will be able to handle incoming traffic networks up to a certain amount, after which they would start falling behind.

That is why we needed a method in which reference shards could scale to handle more load coming from more clients and more worker shards. A way to achieve such scaling is by using a hierarchical structure of reference shards which could in turn handle more traffic and maybe even increase the performance of the overall network (transaction throughput and latency).

4. Implementation

At the beginning of this research, *BlockSim*[10] was going to be used to conduct all the experiments. This simulator was going to be extended and the ability of shard simulation was going to be added. Nevertheless, extending this simulator proved to be a difficult task.

Since the very beginning it was observed that even when this simulator was used to explore latency and throughput of major blockchains such as bitcoin and ethereum, the network topology was not a major concern of the author.

In this scenario it was decided that the best approach was to use discrete simulation but *BlockSim* was no longer an option to conduct the experiments for this research and this is why the idea of creating a simulator from scratch that would expose an *API* to allow a protocol researcher to use code to represent a topology in a simple way was thought of as very good approach. So, one of the main contributions of this work is *ShardSim*, this simulator unlike *BlockSim* has the topology of the network encoded in its core.

Since there may be other repositories in the community with the same name, I hereby declare that ***ShardSim*** is an entirely original creation of this research.

4.1. ShardSim design

ShardSim was born as an alternative to *BlockSim*, so from a usability perspective it is very different to *BlockSim* as it is designed to represent topologies by using code since the beginning, but it uses the same simulations principles i.e discrete simulation.

In figure 4.1, it can be seen that the simulator receives an stimuli consisting of transactions, this transactions are parsed and put into an event *queue*. This queue is constantly being sorted timestamp wise, so that each time that an event gets drawn from by the simulation *engine*, it gets the most *imminent* event.

In *ShardSim*, events have a type and the engine will handle each event depending on their type, handling an event generally means updating the state of the simulation, which is held at the *blockchain classes*. Updating the state in turn could potentially generate new events that are sent to the queue by the *blockchain classes*.

While the simulation is running there is a special kind of event that will be scheduled periodically for reporting. Each time this event happens while the simulation is running, metrics will be gathered and aggregated by a module called *metrics aggregator*. The *metrics aggregator* will traverse the whole topology in search of incoming transactions to calculate the amount of transactions that the simulated blockchain system is receiving and will also check the amount of transactions that are being settled in the blockchains of each *shard* of the system to calculate the output amount of transactions. Is important to note that *metrics aggregator* is capable of gathering and aggregating such transactions *shard* wise, meaning that it allows to eventually calculate the average *intra-shard throughput*, *intra-shard latency*, *cross-shard throughput* and *cross-shard*

latency. When the simulation finishes the engine will command the metrics aggregator to dump those metrics into file for later analysis.

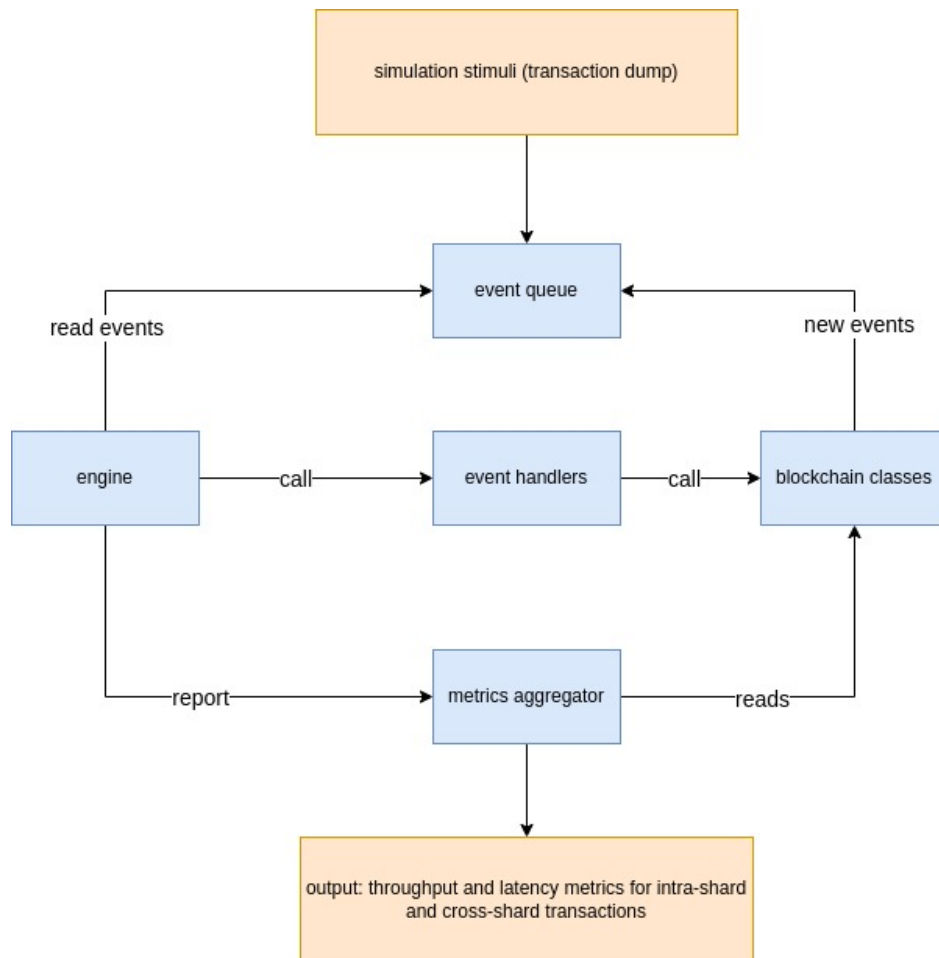


Figure 4.1: ShardSim high level overview

4.1.1 ShardSim classes and simulation details

ShardSim is basically a discrete time simulation system, this means that simulations in ShardSim are driven by events that will carry a timestamp. When these events get processed by the simulation engine, new events will be generated with a different timestamp, this basically represents how much *simulation time* has passed between the initial event and the subsequent event, in other words *simulation time* increments could be little or large depending on what model is used to represent the phenomena

that is being simulated. Also, it is worth noting that *simulation time* in a discrete event simulator is completely decoupled from *host system time*, so it does not matter if the simulator is being used in a slow or very fast computer the results should be the same on both.

In ShardSim, each time that an event gets generated, the timestamp that will be assigned will be the sum of the current *simulation time* and a *time delta* that is generated randomly and has exponential distribution. An exponential distribution is what best describes the delay on a network according to [21].

4.2. Generalizing shard algorithm

In [4], authors use 2 different types of shards, worker and reference shards. This research main intent was to test the performance (throughput and latency) of different shard hierarchies, intuitively what this means is that shards are no longer just reference or worker type, there will be an intermediate type of shard that should have characteristics of both reference and worker shards.

Intuitively, in the *Rivet framework* worker shards are allowed to progress with *intra-shard* transaction execution while at the same time they poll the reference shard and when a *cross-shard* transaction is detected the worker shard will throw away all the blocks that were been created since the last time it committed a block to the reference shard.

Then, if there is no conflict with *cross-shard* transactions the reference shard would have to accept worker shard block commitments and settle those in a blockchain that it maintains.

One of the big questions in this research was how would an intermediate shard behave? That is, a shard that is not a worker and is not the top most shard. In figure 4.2 it can be seen the 3 types of shards that would be present in a hierarchical system. The intermediate kind of shard would in turn need to have a little bit of worker in the sense that it would need to be able to poll its parent shard but at the same time have a bit of a reference shard as it would need to accept commitments coming from its children.

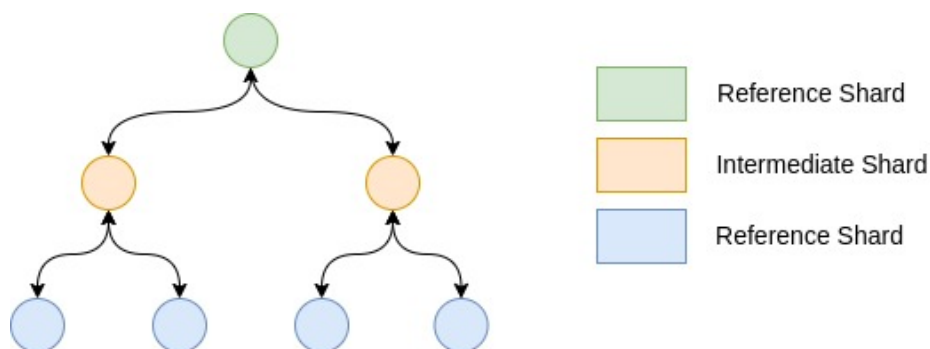


Figure 4.2: Types of shard in a hierarchical topology of shards

Intermediate shards would have to behave as hybrids and that is why a special version of the algorithms proposed in [4] had to be created. Algorithm 1, is a combination of the aforementioned algorithms and it allows a shard to accept commits from its children and then commit the blocks it generates to its parent. In algorithm 1, each time that the shard commits a block it will retrieve the *latest known parent block* P_i , but the parent on its own will do progress that is why the *latest parent block* P_k is mentioned in the algorithm because they can differ depending on the block generation frequency of the parent. The algorithm starts by checking if the latest block in the parent is at a height that is higher than the height of the latest known block (from the previous commit), if the latest block has a lower height that means that its parent has discarded blocks because of an upstream conflict with a cross-shard transaction, so now

the shard will have to look in the parent blockchain for its latest committed block. This block commitment should appear in parent block P_g , with $g \leq i$. The shard will now have to discard all the blocks, that appear after B_g (B_g being the block committed at P_g). Since this type of shard does not maintain the state of accounts and instead only preserves a history of cross-shard transactions, the shard will take all the transactions of all the blocks that appear after B_g and will move them to the transaction pool, so that they are not lost. After this, the shard will retrieve from its parent all transactions that involve any of its children $Q_{g,k}$ and that appear after P_g and will store these in a new block B_{g+1} . This block will also contain all the new cross-shard transactions and commitments done by the shard's children, that are not mentioned in $Q_{g,k}$. On the other hand, if the shard's parent has not discarded any blocks since the last time the shard committed a block P_i height will be greater than P_k and in this case, it will need to ask for transactions that involve any of its children $Q_{i,k}$, then all new transactions and commitments done by the children will be added to the new block, if they do not conflict with the transactions mentioned in $Q_{i,k}$.

It is worth mentioning that this algorithm is a first and maybe naive implementation, I invite other researchers to dig deeper on the optimization of this algorithm. The optimization of this algorithm was not included in this work because it falls out of the strict scope of deciding if using a multi-level hierarchy of shards renders any benefits in terms of throughput or latency.

Algorithm 1 Intermediate shard

```
1: Inputs:
2:  $P_i$ :latest known parent block
3:  $P_k$ :latest parent block
4:  $B_i$ :latest committed block
5:  $B_j$ :latest block
6:  $new_{ctx}$ :New cross shard transactions available
7:  $new_{com}$ :New state commitment transactions available
8:
9: function VALID_COMMITMENTS( $Q, com$ )
10:    $S_{i+1} \leftarrow \emptyset$ 
11:   for each shard  $X_i$  do
12:      $valid_{X_i} = \emptyset$ 
13:     for each state commitment  $com_j = (b_j, h_j)$  from  $X_i$  in  $com$  do
14:       if  $b_j$  extends  $b_{\ell(i)}$  &  $b_j$  is certified &  $Q_i$  is empty then
15:          $valid_{X_i} \leftarrow valid_{X_i} \cup \{com_j\}$ 
16:        $com_j^* \leftarrow \max\{valid_{X_i}\}$ 
17:        $S_{i+1} \leftarrow S_{i+1} \cup \{com_j^*\}$ 
18:        $Q_i \leftarrow \emptyset$ 
19:   return  $S_{i+1}$ 
20:
21: if  $P_{k_{height}} < P_{i_{height}}$  then
22:    $P_g$  parent block with the highest commitment
23:    $B_g$  block committed in  $P_g$  ▷ All blocks after  $B_g$  are dropped
24:    $Q_{g,k}$  transactions that appear between parent blocks  $P_g$  and  $P_k$ 
25:    $q_{g,j}$  children cross shard transactions that appear between block  $B_g$  and  $B_j$ 
26:    $new_{ctx} = new_{ctx} \cup q_{g,j}$ 
27:    $\theta$  checks the addresses in  $Q_{g,k}$  and adds to  $Q_{g,k}$  all txs in  $new_{ctx}$  that do not overlap with the first
28:    $Q \leftarrow \theta(Q_{g,k}, new_{ctx})$ 
29:    $S_{g+1} \leftarrow VALID_COMMITMENTS(Q, new_{com})$ 
30:    $B_{g+1} \leftarrow \langle g+1, hash(B_g), S_{g+1}, Q \rangle$ 
31: else
32:    $Q_{i,k}$  cross shard transactions between parent  $P_i$  and  $P_j$  that include any children shard
33:    $Q \leftarrow \theta(Q_{i,k}, new_{ctx})$ 
34:    $S_{j+1} \leftarrow VALID_COMMITMENTS(Q, new_{com})$ 
35:    $B_{j+1} \leftarrow \langle j+1, hash(B_j), S_{j+1}, Q \rangle$ 
36:
37:
```

4.3. Experiment design

In summary the aim of the experiment carried out during this research was to answer this question: *Would a hierarchical architecture, meaning an architecture with more than 2 levels(worker and reference) be able to yield any benefit in terms of throughput or latency intra or cross shard?*

4.3.1 Initial notions for experiment design

The experiment design process was preceded by an intuitively high-level analysis of why and why not a hierarchical architecture of shards may yield benefits in terms of latency and throughput.

In a blockchain system, the throughput is given by the number of transactions

that can be settled in the blockchain in a given amount of time. The most important bottlenecks in a blockchain system are the frequency by which the blocks are produced and the number of transactions that can fit into a block. That means that the transaction throughput can be computed as stated in formula 4.1.

$$\textit{transaction throughput} = \textit{block production frequency} * \textit{block size} \quad (4.1)$$

Latency, on the other hand, is given by the amount of time that takes from the moment that the transactions arrive at a node in the system to the moment it gets settled into the blockchain. So, the following intuitions were enumerated in favor and against a hierarchical architecture.

First of all, the main reason shards are used in blockchain implementation is to increase the throughput of the overall system by introducing a boundary that segregates the state maintenance among groups of nodes. That means that in sharded blockchain a node will only need to worry about correctly maintaining a subsection of the state, so it will only need to worry about transactions that modify its state. Then, since shards are part of a bigger overall system that needs to maintain consistency, shards need a mechanism to communicate their state to the rest of the shards. In [4] authors mention that worker shards report their state to the overall system by doing a block commit to a central blockchain. In essence, what worker shards do in *Rivet Framework* is to summarize information by committing a hash of their state, so even though the state of the worker shards can not be reconstructed using the information contained

in a block commit it is enough to maintain consistency and safety across the system. Intuitively, what this suggests is that if the amount of levels is increased then upper levels of shards will be able to summarize a bigger amount of information. For instance, if there are three levels in a system like the one shown in 4.3 then the shards that are at the very bottom maintain a blockchain for a set of accounts and they report their state to an intermediate shard level that will maintain a blockchain with block commits and transactions. At this intermediate level state from the bottom shards is already summarized and then finally the intermediate shards will commit their blocks to a higher level where basically blockchain will contain a summary of summaries.

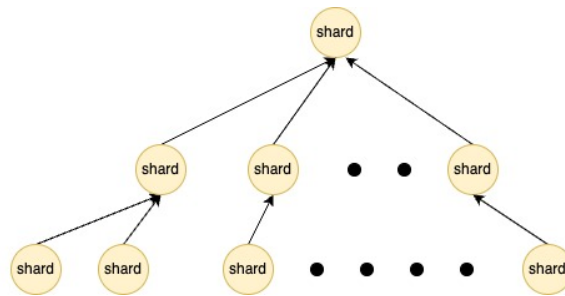


Figure 4.3: Hierarchical shard blockchain example

An argument against the usage of hierarchies in sharded blockchains is that if the state of any of the accounts that are managed by a certain shard changes in between commits that shard will have to throw away all of its progress done since the last commit. This already worsens the *intra shard* latency in a system where only exists 2 levels of shards (workers and reference), but if we have several levels this can easily prevent shards in lower levels from achieving progress, so the effect of the penalty gets amplified in a hierarchical system with more than just the worker and reference levels.

It is worth mentioning that the influence of this effect in the behavior of the overall system is affected by the amount of *cross shard* transactions, that is the number of

transactions that change the state in accounts belonging to different shards.

4.3.2 Experiment setup

After having analyzed the most important characteristics of hierarchical sharded blockchains it was decided that the experiment would have as its input variables, the **number of shards, the number of children per shard, the number of levels in the system, the transaction rate and the cross-shard transaction ratio**. Then, the variables that were going to be measured were **intra shard throughput, intra shard latency, cross shard throughput and cross shard latency**.

The experiment consisted of running discrete event simulations with **16** different architectures using different combinations for the number of shards, children per shard, and the number of levels. In table 4.1 you can see the architectures that were used and the number of shards in each.

Table 4.1

Experimental architectures

	Number of shards	Levels	Children Per Shard
1	5	3	2
2	5	3	3
3	5	2	5
4	10	3	3
5	10	2	10
6	15	4	3
7	15	3	5
8	15	3	10
8	20	4	3
10	20	3	5
11	20	3	10
12	30	4	3
13	30	3	5
14	30	3	10
15	50	4	5
16	50	3	10

At the beginning of the experimentation process, transactions dump were acquired from <https://blockchair.com/dumps> which is a popular site to extract dumps from many popular blockchains like bitcoin and Ethereum. These transaction dumps allowed to run some tests where it was noticed that real transactions were not going to be good for the experiment because **1)** Real transaction dumps included transaction types that were not being studied in the experiment and **2)** Transaction dumps that do not take in consideration sharding, would make the system enter trivial states where for example all transactions were being processed as cross-shard transactions and processing power from lower levels workers was being wasted.

For the purposes of the experiment, it was better to create transaction dumps specifically generated for the number of shards (at the lowest level), transaction rate, and cross-shard transaction ratio of each experimental treatment.

Each of the architectures mentioned in table 4.1 was tested against 3 different transaction rates **100**, **200** and **300** and 3 different cross-shard transaction ratios **10%**, **25%** and **50%** and each these permutations was 5 times to be able to get statistically meaningful results.

4.3.3 Experiment simulation parameters

Although the experiment would be comprised of many different architectures, there were some parameters that were common to all of the systems that were simulated and these parameters are presented in table 4.2.

Table 4.2

Experiment Blockchain Parameters

Parameter	Value	Description
function for delay computaion	Exponential	This function was chosen based on modeling in other work like
block size	300	This was considered a typical value after extensive research on the size of blocks of blockchain like bitcoin and ethereum
block generation frequency	10s	This value for the highest level of every architecture and lower levels would have submultiples of this number.
child blocks per block	20	This number was arbitrarily chosen based on the behavior that was observed during the experimentation process

5. Results

After running the experiment in all the architectures mentioned in 4.1, there was a lot of data that needed to be analyzed and presented for the 4 response variables of the experiment, namely intra shard throughput, intra shard latency, cross shard throughput and cross shard latency. Since the data had many different dimensions like the amount of shards, the amount of children per shard, the amount of levels in the overall architecture, the transaction rate and the cross shard transaction ratio, it was necessary to present and organize the data in a meaningful easy to present way. So, it was decided to center the analysis around the levels of the architectures, given that in turn, it would be the variable that represents the depth of a hierarchy.

Also, it was necessary to choose among all of the data available from all the scenarios that were tested which were the most interesting ones. Throughput the repetitions of the experiment 3 different transaction rates were used, 100 Tx/s , 200 Tx/s and 300 Tx/s . In the data analysis, the transaction rate and cross shard transaction ratio were fixed to 200 Tx/s and 10% respectively for the sake of simplicity and because by analyzing the wide range of data available it was considered that it would represent the behavior of the systems in a meaningful way.

5.1. Intra Shard Throughput and Latency

In figure 5.1, it can be seen the behavior of the intra-shard throughput as the number of shards was increased. For each number of shards (which in turn represents the gross amount of resources), the average of transactions per second using a configuration with 2 different levels is shown. For example, when the number of shards is 5 there is a configuration with 2 and 3 levels depending on the number of children that the shards are allowed to have and the same applies to all numbers of shards. Even though there are a couple of cases (5 and 10 shards) where it is impossible to have an architecture with 4 levels, it can be observed that there is a clear pattern that as the architectures increase their depth the intra-shard throughput will increase.

The reason for this behavior is that as the architecture grows in depth the number of shards affected by a cross-shard transaction is decreased and the penalty that gets a shard on its progress when this happens is less frequent.

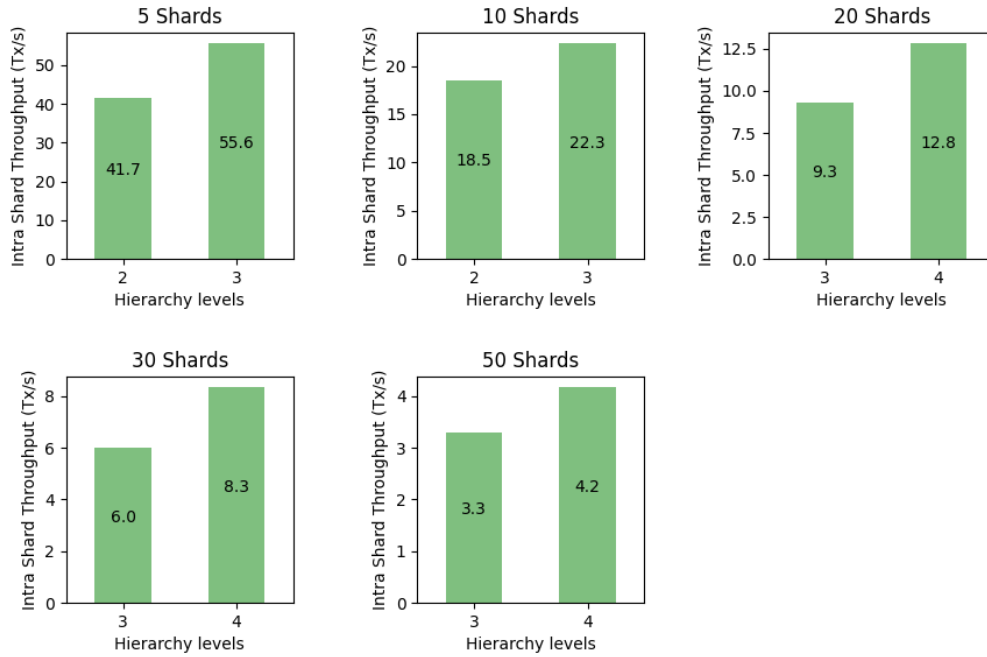


Figure 5.1: Intra Shard throughput across different architectures

To be able to ensure that there is a real difference between using different numbers of levels in any of the architectures of the experiment an *ANOVA* (analysis of variance) was used and it was applied to each number of shards individually. So, in table 5.1, it is shown the case when the number of shards is 10 and that the *P-value* for the *number of levels* variable is small compared to $\alpha = 0.05$, so the throughputs of the configurations for this architecture (2 and 3 levels) are statistically different.

Table 5.1

Intra Shard Throughput Anova Results for Shards=10

	sum_sq	df	F ($F_{crit} = 2.49$)	P-value
number of children	178.603597	2.0	5.640453	9.512041e-03
number of levels	819.344388	1.0	51.751177	1.539658e-07
transaction rate	424.360860	2.0	13.401675	1.108606e-04
cross shard transaction ratio	1319.364466	2.0	41.666645	1.095786e-08
Residual	395.809544	25.0	-	-

In the latency case, it can be observed in figure 5.2, that in most cases it will be lower as the depth of the architecture increases, and this can be attributed again to

the fact that when architectures have more hierarchical levels, lower shards experience penalties less frequently.

In this case, again an ANOVA was performed to be able to establish the statistical significance of the behavior of the latency when the number of levels is changed. In this case the ANOVA was made with the cases where the number of shards is equal to 10 and the P-value for the number of levels variable is considered small when compared to the level of significance so it can be established that statistically, the number of hierarchical levels affect the intra shard latency.

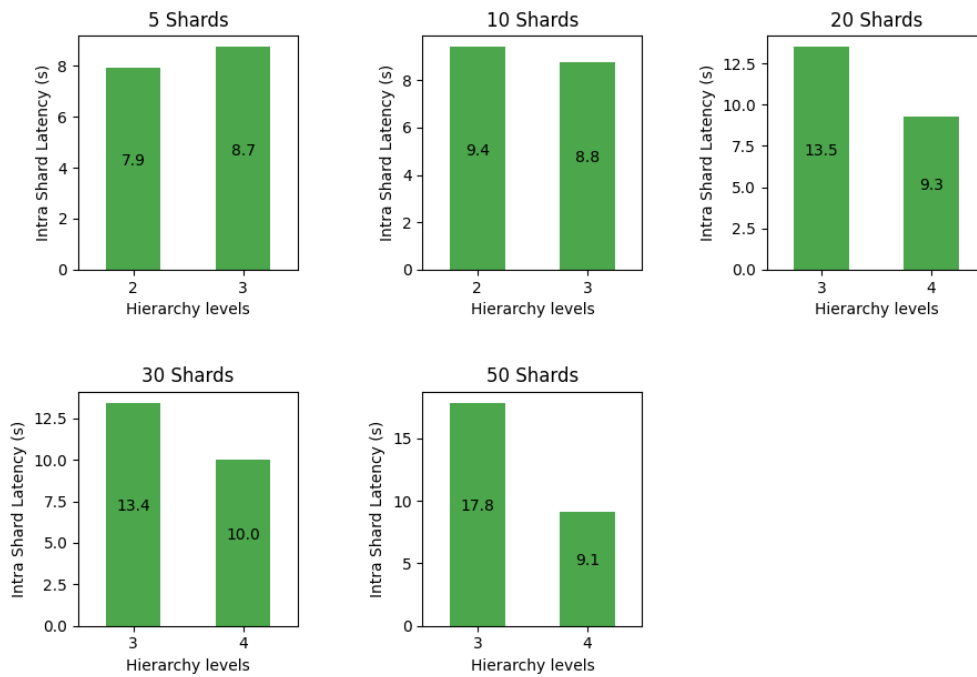


Figure 5.2: Intra Shard latency across different architectures

The variation of the latency when the number of levels is increased can be observed in table 5.3 and it ranges between 6% and 48% which represents a very good gain when the number of shards grows.

Table 5.2

Intra Shard Latency Anova Results for Shards=10

	sum_sq	df	F ($F_{crit} = 2.17$)	P-value
Number of children	800.463134	2.0	1522.167835	5.931622e-90
Number of levels	511.136029	1.0	1943.961661	2.979356e-79
Transaction Rate	0.296066	2.0	0.563001	5.709012e-01
Cross Shard Transaction Ratio	0.756734	2.0	1.439012	2.409724e-01
Residual	33.655711	128.0	-	-

Table 5.3

Intra shard throughput and Latency Variation with each architecture configuration

	5 shards	10 shards	20 shards	30 shards	50 shards
Intra shard transaction Throughput	+33.3%	+20.5%	+37.6	+38%	+27.2%
Intra shard transaction Latency	+10%	-6.3%	-31.11%	-25.3%	-48.8%

5.2. Cross Shard Throughput and Latency

When testing cross-shard throughput and latency, the cross-shard transaction ratio needed to be changed to 25%. In the intra-shard case, 10% ratio was used because this ratio means that the majority of the transactions are going to be directed towards the lower shards, but it means that what is measured in terms of cross-shard behavior is not very significant because it is a ratio that won't be stressing the system. Instead, a higher ratio would allow a fair amount of transactions to be cross-shard and the overall system could be measured in a more meaningful way.

In figure 5.3, the behavior of the cross-shard throughput can be observed across architectures with different numbers of shards and for each number of shards there are 2 number of levels. So, very similar to the intra-shard case the cross shard throughput increased for all of the architectures when the architecture was deeper with variation going from 16.3% to 31.2%, the variation between architecture configurations can be seen in table 5.6. The higher throughput may be due to the reason that was mentioned

in the implementation section, which points to the fact that transactions experience less back pressure to get settled in the blockchain because they are summarized in blocks at a certain level that then get committed to higher levels instead of having to wait in a queue for a time when there is room in the block of the shard where they are being processed.

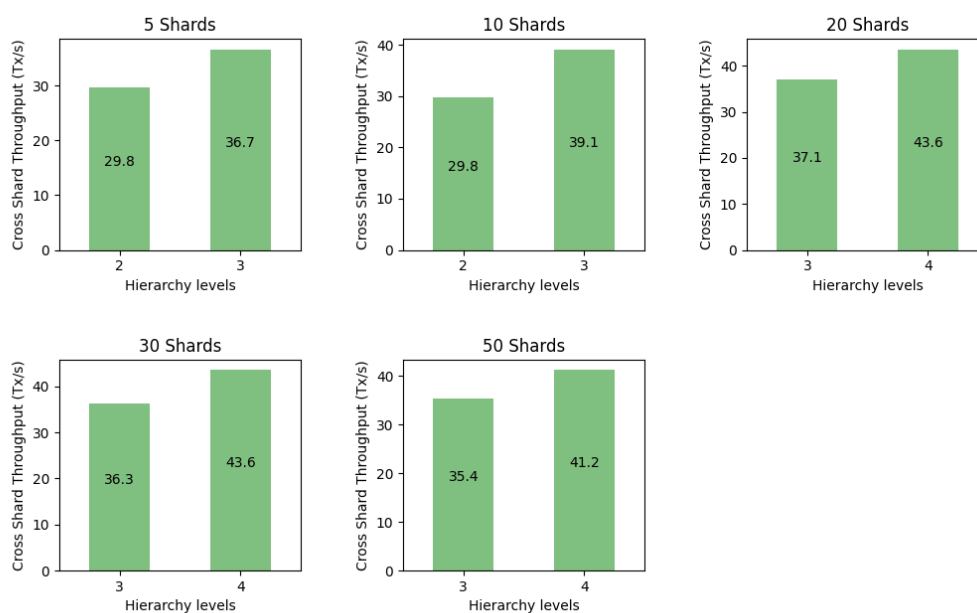


Figure 5.3: Cross Shard throughput across different architectures

In this case, an ANOVA was performed case-wise (according to the number of shards), to provide an example that there is a statistically meaningful difference between the performance that is obtained at different amounts of hierarchical levels when the number of shards is 10.

In figure 5.4 results can be observed for many different architectures and it is clear that latency is lower when the architecture has more hierarchical levels. In the latency case, the descent was up to 61.6%, the variations for each architecture when the levels were increased are shown in table 5.6. The latency decrement gains may be

Table 5.4

Cross Shard Throughput Anova Results

	sum_sq	df	F ($F_{crit} = 2.49$)	PR(>F)
Number of children per shard	58.077666	2.0	1.587235	2.244156e-01
Number of levels	750.761935	1.0	41.035945	1.046155e-06
Transaction Rate	613.246198	2.0	16.759732	2.415438e-05
Cross Shard Transaction Ratio	1712.329853	2.0	46.797174	3.535495e-09
Residual	457.380681	25.0	-	-

due also to a higher capacity of the blockchain due to the summarization of blocks of an intermediate shard level into a higher shard level.

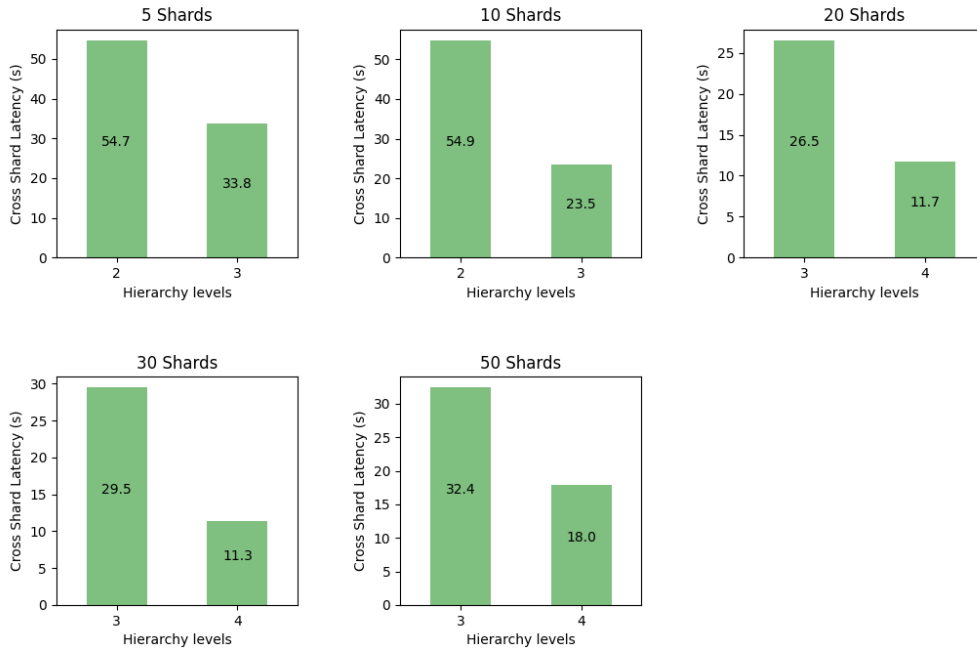


Figure 5.4: Cross Shard latency across different architectures

Cross-shard latency followed the same behavior as the cross-shard throughput and from the ANOVA shown in table 5.5 it can be established that there is a statistically significant difference between the number of hierarchical levels when it comes to cross-shard latency.

Table 5.5

Cross Shard Latency Anova Results

	sum_sq	df	F ($F_{crit} = 1.89$)	PR(>F)
C(children)	1.228106e+04	3.0	7.646355	4.821745e-05
C(levels)	7.004643e+04	2.0	65.417814	4.861786e-27
C(transaction_rate)	3.517519e+05	2.0	328.508427	7.082142e-105
C(cross_shard_transaction_ratio)	1.779614e+06	2.0	1662.018709	1.279252e-285
Residual	4.283019e+05	800.0	-	-

Table 5.6

Cross shard throughput and Latency Variation with each architecture configuration

	5 shards	10 shards	20 shards	30 shards	50 shards
Cross shard transaction Throughput	+23.1%	+31.2%	+17.5%	+20.1%	+16.3%
Cross shard transaction Latency	-38.2%	-57.1%	-55.8%	-61.6%	-44.4%

6. Conclusions

This research met all the established goals and made good contributions to the state of the art in terms of knowledge about sharded hierarchical systems and also tooling that can be used by other researchers.

In terms of the tooling that this research yields, ShardSim was created as a framework that allows simulating through discrete event simulation complex, architectures of blockchain shards, this is a big contribution to the blockchain research community, because until this point there were no alternatives to simulate complex architectures of sharded blockchains.

In terms of the findings that this research yields it is clear that using architectures with more levels of summarization can yield benefits in terms of both latency and throughput and this behavior was observed in both contexts intra-shard and cross-shard.

After having performed many runs of experimentation with many different architectures and configurations it can be concluded that:

- The initial hypothesis was partially confirmed and it was proven that a hierarchical architecture can yield benefits in the intra-shard and cross-shard scope. It is partially true because it was found that these gains depend on how the system is being operated in terms of the ratio of cross-shard transactions.
- The cross-shard transaction ratio is a very important factor in the response of a

sharded system and this fact has been stressed by other researchers but in the literature, it hasn't been stressed the fact that to study these systems different ratios should be used to extract relevant information from the intra-shard and cross-shard response variables.

- This is an initial exploration, it remains a future work to build real distributed systems and use the hierarchies used in this work. There may be unknown effects of these systems that are not yet known. Also, the consensus algorithm may be improved to optimize the way shards communicate with their parents and their siblings.

7. Bibliography

- [1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Dec 2008. Accessed: 2015-07-01.
- [2] Tezro, “Cryptocurrency transaction speeds in 2021.” <https://blog.tezro.com/cryptocurrency-transaction-speeds/>, July 2021. Accessed: 2021-10-29.
- [3] Y. Liu, J. Liu, M. A. V. Salles, Z. Zhang, T. Li, B. Hu, F. Henglein, and R. Lu, “Building blocks of sharding blockchain systems: Concepts, approaches, and open problems,” 2021.
- [4] S. Das, V. Krishnan, and L. Ren, “Efficient cross-shard transaction execution in sharded blockchains,” 2021.
- [5] Q. Zhou, H. Huang, and Z. Zheng, “Solutions to scalability of blockchain: A survey,” 01 2020.
- [6] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O’Reilly Media, Inc., 1st ed., 2014.
- [7] V. Buterin, “Ethereum: A next-generation smart contract and decentralized application platform.” <https://github.com/ethereum/wiki/wiki/White-Paper>, 2013.
- [8] G. Yu, X. Wang, K. Yu, W. Ni, J. A. Zhang, and R. P. Liu, “Survey: Sharding in blockchains,” *IEEE Access*, vol. 8, pp. 14155–14181, 2020.

- [9] B. Leemoon, “Bitcoin valuation framework pub,” 12 2017.
- [10] M. Alharby and A. van Moorsel, “Blocksim: An extensible simulation tool for blockchain systems,” *Frontiers in Blockchain*, vol. 3, Jun 2020.
- [11] A. Elngar, V. Balas, m. Kayed, and S. Panda, *Bitcoin and Blockchain. History and Current Applications*. 04 2020.
- [12] <http://www.weidai.com/bmoney.txt>. Accessed: 2021-11-12.
- [13] N. Szabo, “Bit gold.” <https://nakamotoinstitute.org/bit-gold/>, Dec. 2005. Accessed: 2021-11-12.
- [14] A. B. e mail:, “Hashcash - a denial of service counter-measure.” <http://www.hashcash.org/hashcash.pdf>. Accessed: 2021-11-12.
- [15] G. Wood, “Polkadot: vision for a heterogeneous multi-chain framework,” tech. rep., 2018.
- [16] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, “A secure sharding protocol for open blockchains,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, (New York, NY, USA), p. 17–30, Association for Computing Machinery, 2016.
- [17] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, “Chainspace: A sharded smart contracts platform,” 2017.
- [18] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, “Omniledger: A secure, scale-out, decentralized ledger via sharding,” in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 583–598, 2018.

- [19] M. Zamani, M. Movahedi, and M. Raykova, “Rapidchain: Scaling blockchain via full sharding,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS ’18*, (New York, NY, USA), p. 931–948, Association for Computing Machinery, 2018.
- [20] J. Wang and H. Wang, “Monoxide: Scale out blockchain with asynchronous consensus zones,” in *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation, NSDI’19*, p. 95–112, 2019.
- [21] A. Sukhov and N. Kuznetsova, “What type of distribution for packet delay in a global network should be used in the control theory?,” 07 2009.