

# Improving Hyperledger Fabric

## Authors:

Ebtesam Al Haque (160041077)

Fabiha Iffat (160041072)

Novera Tasnuba Aura (160041041)

## Supervised by:

Prof. Muhammad Mahbub Alam, PhD

Professor

Department of Computer Science and Engineering (CSE)

Islamic University of Technology (IUT)

**A thesis submitted to the Department of CSE  
in partial fulfillment of the requirements for the degree of  
Bachelor of Science in Computer Science and Engineering**



**Department of Computer Science and Engineering (CSE)**

**Islamic University of Technology (IUT)**

**Gazipur, Bangladesh**

March 2021

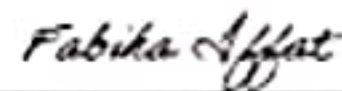
## Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and simulations carried out by Ebtesam Al Haque, Fabiha Iffat and Novera Tasnuba Aura under the supervision of Prof. Muhammad Mahbub Alam PhD, Professor, Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Gazipur, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

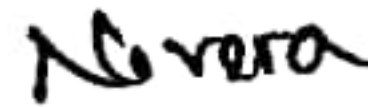
*Authors:*



Ebtesam Al Haque  
160041077

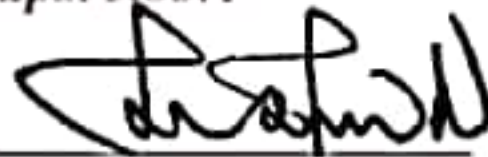


Fabiha Iffat  
160041072



Novera Tasnuba Aura  
160041041

*Supervisor:*



Prof. Muhammad Mahbub Alam, PhD  
Professor  
Department of Computer Science and Engineering  
Islamic University of Technology (IUT)

## Acknowledgement

We would like to express our grateful appreciation to Prof. Muhhamad Mahbub Alam, PhD, Professor, Department of Computer Science Engineering, IUT for being our advisor and mentor. His suggestions and insights for this thesis have been invaluable. Without his support and mentorship this research would not have been possible. His valuable opinion, time, constructive criticism and input provided throughout the thesis work made this research possible. We are truly grateful to him.



## Abstract

Hyperledger Fabric is a popular open-source blockchain platform used by several projects around the world. However, several bottlenecks exist in the current system which limit its performance. One of the best attempts to improve Hyperledger Fabric was FasftFabric, which modifies the existing architecture to improve the throughput of Hyperledger Fabric. However, it does not provide an insight into the bottlenecks that could potentially exist in the orderer algorithm itself. In our work, we compared the existing ordering algorithms with Hashgraph to potentially improve Hyperledger Fabric's throughput. Our proposed solution does not require a lot of system resources and also provides Byzantine Fault Tolerance with higher throughput.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Background Study . . . . .	5
1.1.1	Blockchain . . . . .	5
1.1.2	Drawbacks of Blockchain . . . . .	6
1.1.3	Hyperledger . . . . .	7
1.1.4	Hyperledger Fabric . . . . .	8
1.1.5	ChainCode . . . . .	9
1.1.6	Components . . . . .	10
1.1.7	Assigned Roles of Hyperledger Fabric . . . . .	10
1.1.8	Architecture . . . . .	12
1.1.9	Orderers . . . . .	14
1.2	Problem Statement . . . . .	15
1.2.1	Limitations . . . . .	15
1.2.2	Improvement Scopes . . . . .	16
1.3	Thesis Objectives and Contributions . . . . .	17
1.4	Thesis Structure . . . . .	17
<b>2</b>	<b>Literature Review</b>	<b>18</b>
2.1	FastFabric . . . . .	18
2.1.1	Performance analysis of Ordering and Peer Improvements . . . . .	25
2.1.2	Tangaroa . . . . .	26
2.1.3	Limitations . . . . .	26
<b>3</b>	<b>Proposed solution</b>	<b>27</b>
3.0.1	Hashgraph . . . . .	27
<b>4</b>	<b>Experimental Analysis</b>	<b>28</b>
4.1	Experimental Setup . . . . .	28
4.2	Experimental Results . . . . .	28
<b>5</b>	<b>Conclusion</b>	<b>29</b>
5.1	Summary . . . . .	29
5.2	Future Work . . . . .	29

# 1 Introduction

## 1.1 Background Study

Hyperledger Fabric [1] is a project of Hyperledger which has similar drawbacks of limited number of transactions/second like Blockchain. This was improved by the proposal of FastFabric. Our main focus is to work on the improvement of throughput of FastFabric furthermore and reintroduce byzantine fault tolerance without affecting scalability.

### 1.1.1 Blockchain

A blockchain is a distributed network of software that acts as both a digital ledger and a system that enables assets without an intermediary to be transferred securely. Blockchain is a technology that facilitates the digital exchange of units of value, much like the Internet is a technology that facilitates the digital flow of information. On a blockchain network, everything from currencies to land titles to votes can be tokenized, stored, and traded. Blockchain technology holds tremendous promise to transform centuries-old business structures, paving the way for higher levels of government legitimacy and generating new opportunities for ordinary people to thrive, regardless of the type of blockchain protocol that is implemented.



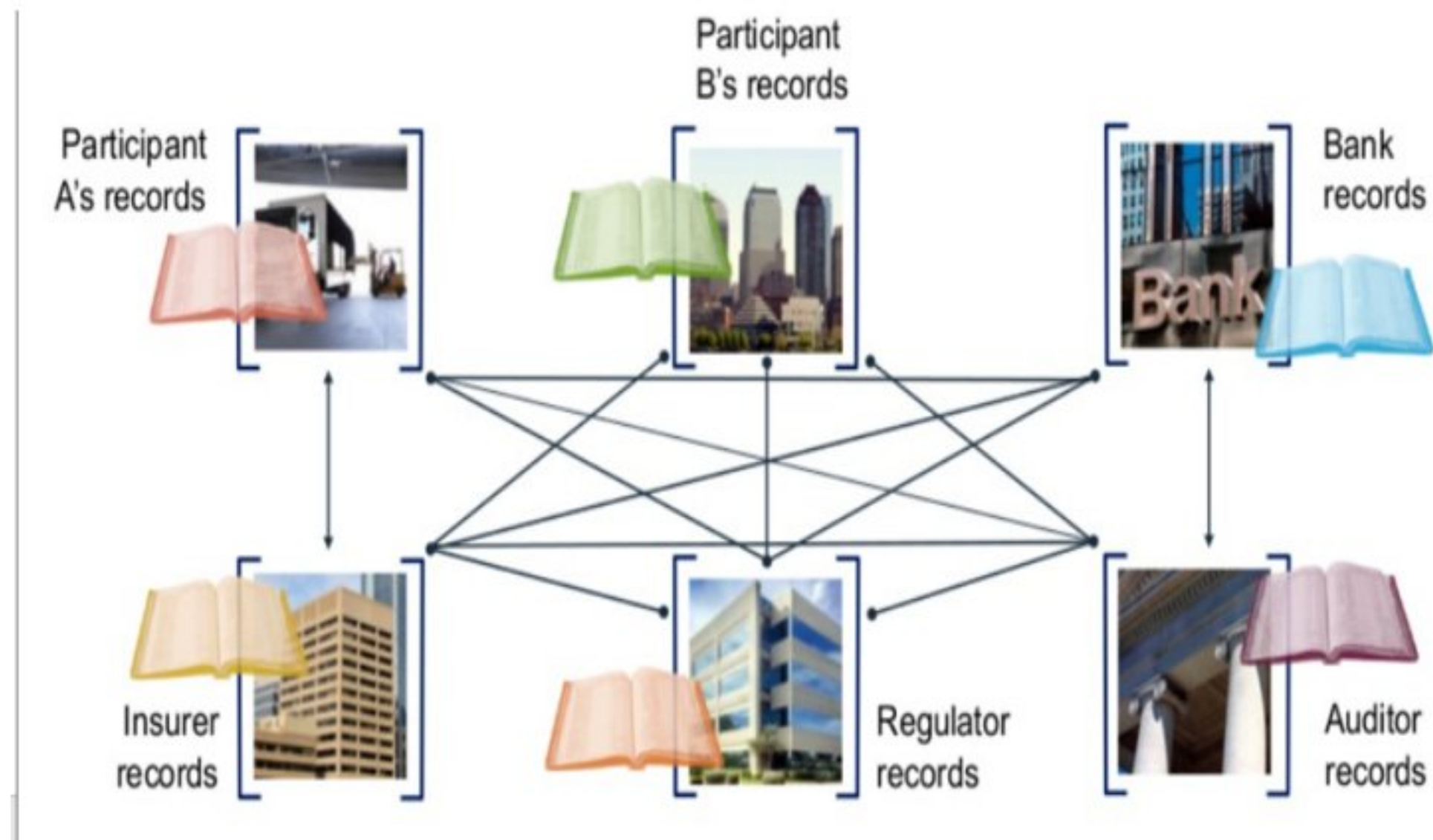


Figure 1: A scenario of a Public Blockchain

### 1.1.2 Drawbacks of Blockchain

Though blockchain has brought a revolutionary change in cases of secure transactions, there are some major drawbacks in blockchain. The drawbacks [2] are discussed below:

- Violation of Privacy :** In blockchain, every member of a network shares the same ledger which means every transaction that takes place in a blockchain is known to every node in the network (Fig: 1). This protocol violates the privacy and confidentiality of the nodes.

- Scalability:** The bitcoin scalability issue is the restricted pace at which transactions can be processed by the bitcoin network. It is linked to the fact that records in the bitcoin blockchain are limited in size and frequency (known as blocks). The blocks of Bitcoin include transactions on the bitcoin network.

In order to get rid of these problems, many proposals have been made. Amongst them, hyperledger works the best in the case of maintaining privacy of the nodes.

### **1.1.3 Hyperledger**

Hyperledger [3] is an open source project under Linux Foundation mainly for businesses. As we know we have Ethereum which runs on a generalized protocol for everything that comes on the network. But we know every business is unique and applications should be personalized. Keeping this thing in mind, the goal of the project was to make blockchain more accessible to the world as a technology. In short, we can say that hyperledger is a software that can be used by anyone to create or maintain one's personalized blockchain service.

As mentioned earlier, in blockchain every node in the network will have the same ledger having the same records. This is the point where the activities of Hyperledger mainly differ.

In hyperledger, nodes can create their own channels and only permitted nodes or peers can join that network. All the nodes present in the channel will have the same ledger which won't be shared with anyone else outside the channel. This ensures the confidentiality of transactions within the permitted nodes only. Since not all the ledgers need to be updated for a single transaction, the rate of throughput increases efficiently.



Under the domain of hyperledger, there are many projects- Hyperledger Sawtooth, Iroha, Hyperledger Fabric and Hyperledger INDY. Our focus point is Hyper ledger Fabric.

#### 1.1.4 Hyperledger Fabric

Hyperledger Fabric is basically a Private/Permissioned Blockchain which has a feature of getting subnets, known as Channels.

There are some special additions or terms associated with Hyperledger Fabric. Those are described below:

- **Assets:** Assets are defined as things with monetary values ( food or car or anything else). In Blockchain, we needed to transfer cryptocurrencies. But here we can use any sort of assets to meet the purpose of the transaction.
- **Members:** Members can define asset types consensus protocol needed for their network. Existing members of a network can also set permissions on who can join the network.
- **Modular Design:** Hyperledger Fabric has a special design called Modular Design which means that businesses can plug into different functionalities. Also they have the privilege to set their assets by themselves.

In Hyperledger Fabric, one peer can be a part of many channels which means they can have several ledgers. The number of ledgers indicates how many channel one peer is attached to.

The initiation of transaction needs to be determined. Though an application is there with the network it does not have any power. So programming is needed in this case. This is where smart contracts come into play. Using Smart Contracts, applications can send a request of transaction or initiate the transaction of assets. In Hyperledger Fabric, Smart Contracts are written with the help of Chain Code.

#### **1.1.5 ChainCode**

ChainCode is a piece of software that describes assets and Transaction Logic, specifically in the sense of asset modification. ChainCode is used to update, add, and move all of these properties. Chaincode is used by representatives of each permissioned network to communicate with the ledger.

There's also the Member Identity Program, which keeps track of user ids and authenticates them. An 'Access Control List' is used as an additional layer of authorization once again.

### 1.1.6 Components

In blockchain, we have seen one kind of ledger only but in Hyperledger, there are two kinds of ledgers used.

- **Transaction Log/Business Log:** This ledger stores immutable sequenced records of transactions in blocks. In this ledger, all the records from its creation are being stored.
- **State Database/World State:** This ledger maintains Blockchain's current state. When a transaction takes place, it erases the previous record from its memory and writes the recent state.

It is proved that the addition of a State Database in Hyperledger improves the speed of transaction. In the Bitcoin Blockchain, current state is calculated by going through all the transactions in the ledger which is quite time consuming.

### 1.1.7 Assigned Roles of Hyperledger Fabric

In Hyperledger Fabric, the nodes are assigned with some responsibilities to do their respective jobs. At first, all the nodes are divided into two categories- Peer Nodes and Ordering Nodes. Peer Nodes are again divided into Committers and Endorsers. In most cases, all the Peer nodes play the roles of committers which means that Endorsers need to play both of the roles as committers and endorsers.

The roles of the nodes are shown in Fig 2:



## ASSIGNMENT OF ROLES

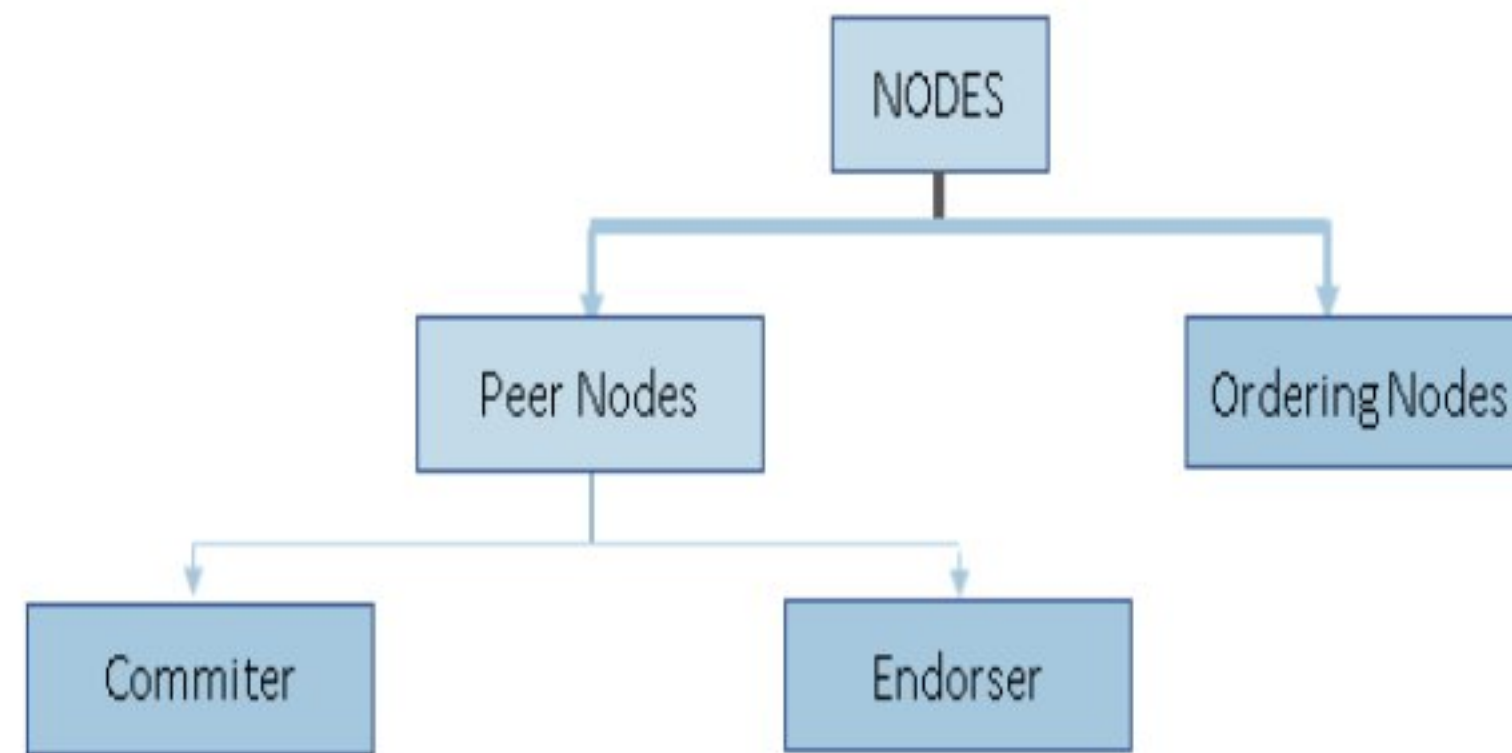


Figure 2: Assigned Roles of Hyperledger Fabric

- **Orderer Nodes:** Orderer Nodes are given the responsibility to order propagate correct history of events. The orderers first come to a consensus about the order of incoming transactions and then segment the message queue into blocks.
- **Committers:** Blocks are delivered to peers by Orderer Nodes, who then validate and commit them.
- **Endorsers:** Each endorser uses various business rules to validate the correctness of the transaction. The client waits for a sufficient number of endorsements and then sends these responses to the orderers, which implement the ordering service.

### 1.1.8 Architecture

The Hyperledger Fabric used to work with the BFT consensus algorithm earlier. Due to the slow throughput rate, it started using Kafka as a Consensus Algorithm. Hyperledger Fabric works with the help of above mentioned nodes and Kafka.

Let's have a look at the architectural diagram of Hyperledger Fabric step by step:

- A participant from the network firstInvokes a transaction request through the client application
- Client application then broadcasts transaction invocation request to the Endorser peer
- Endorser peer checks the certificate details and others to validate the transaction. Then it executes the Chaincode which is the Smart Contract. After these tasks are done, Endorser Peer returns the Endorsement responses to the Client
- After client receives the response from Endorser, he sends the approved transaction to the Orderer peer

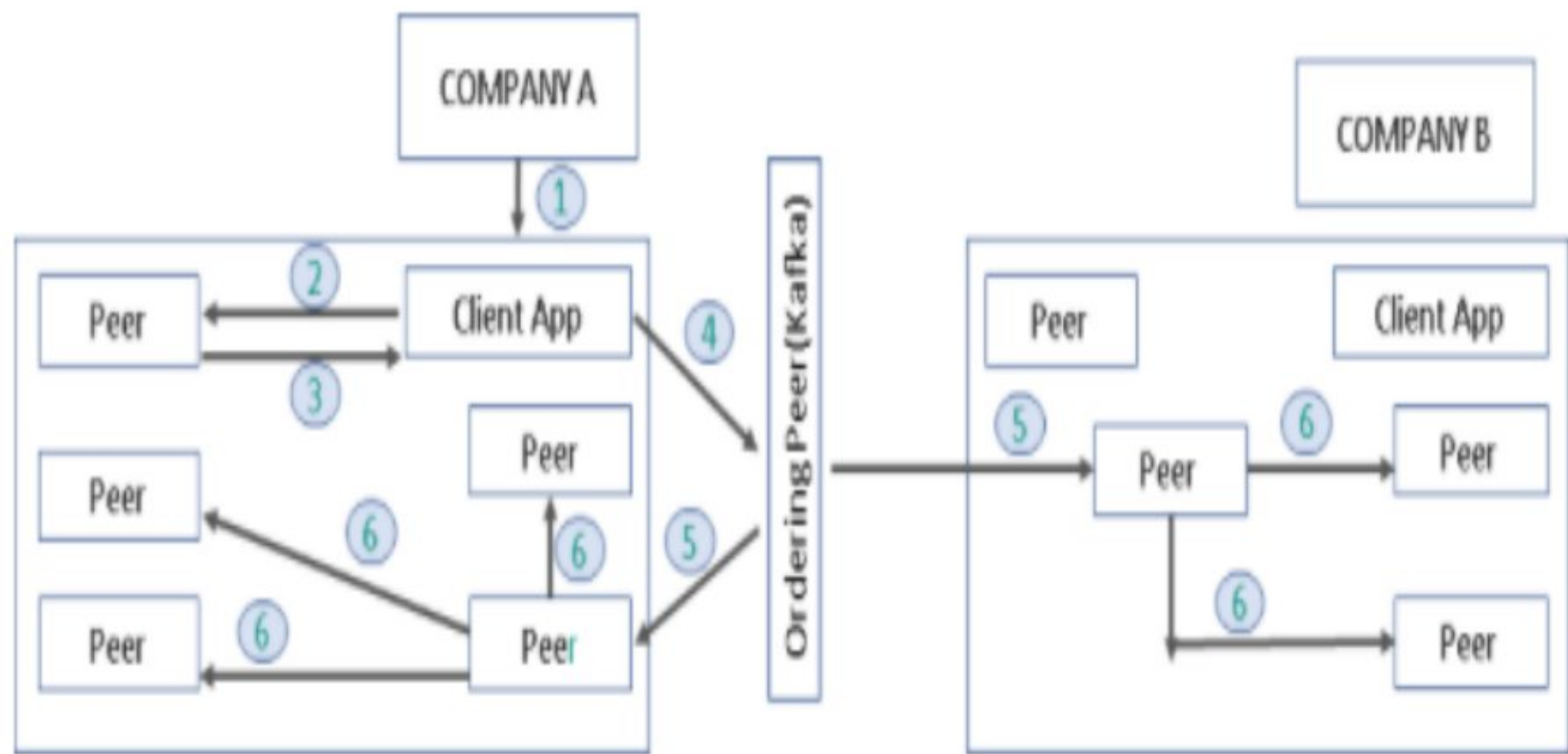


Figure 3: Architecture of Hyperledger Fabric

- Orderer node then includes the transaction into a block and forwards the block to the Anchor nodes
- Anchor nodes broadcast the block to the other peers inside their own organization
- These individual peers update their local ledger with the latest block. Thus all the network gets the ledger synced

The architecture of Hyperledger Fabric is shown in Figure 3:



### 1.1.9 Orderers

The application of the two existing ordering service node (Kafka, Raft) implementations varies substantially, despite the fact that they use the same consensus mechanism. First, Kafka was built for smaller networks. Even though the Blockchain network has many peers and nodes, the mechanism behaves as if it were one. It varies from Raft, which uses ordering service nodes directly as a replication state machine (Raft Consenter) so that each entity can have its own ordering service node and make the Blockchain network more decentralized, Consenter uses ordering service nodes indirectly as a replication state machine. Another key difference lies in their dependencies. Kafka requires Zookeeper to function which makes the set up process more complex than Raft, which is relatively straightforward.

## 1.2 Problem Statement

### 1.2.1 Limitations

Though Hyperledger Fabric 1.2 has been known as the fastest available open-source permissioned blockchain there are some limitations which can be focussed to work on for further improvements. The limitations of Hyperledger Fabric are described below :

- The consensus layer in Fabric receives whole transactions as input, but only the transaction IDs can be used to decide the transaction order. This results in a decreased throughput.
- All the tasks of endorsers and committers are done in a separate time base whereas if some of the tasks could have been parallelized then it would take much less time than it is taking now.
- In Fabric 1.2, block committing blocks are also the responsibility of endorser peers. Endorsement is quite an expensive operation. Although concurrent transaction processing on a cluster of endorsers can boost application efficiency, the extra work needed to duplicate commitments on each new node effectively negates the benefits.

### 1.2.2 Improvement Scopes

We can increase the throughput in Hyperledger Fabric in 3 ways mainly:

- Better Consensus Algorithm:** Existing Byzantine-Fault-Tolerant (BFT) consensus algorithms have extensive overheads resulting in performance bottleneck. Our goal is to find a better consensus algorithm with lesser overhead.

- Improvement to ordering service:** Improving the performance in the orderer peer will result in increased throughput as the orderer performs a bunch of operations like ordering and propagating the correct history of events

- Improvement to peering service:** Peers perform tasks like committing, certifying and permitting a transaction or event. So finding better ways to complete these tasks gives us a better throughput



### **1.3 Thesis Objectives and Contributions**

The objective of this thesis is to overcome the existing limitations of Hyperledger Fabric in order to further optimize it. We explored different consensus mechanisms and found Hashgraph to offer a higher throughput under constrained system resources and also offers a Byzantine Fault Tolerant solution with minimal overhead in contrast to enhanced Raft.

### **1.4 Thesis Structure**

In this section we discussed our background study on Blockchain and elaborated on Hyperledger Fabric and its limitations. We also stated the limitations we identified and our hypothesis to improve the performance of Hyperledger Fabric. In the next section we discuss the existing published works that have been done to improve Hyperledger Fabric. In Section 3 we discuss our proposed method. Section 4 shows the results and comparative analysis of successful implementation of our proposed method. The final segment of this study contains all the references and credits used. We then conclude our thesis and discuss future prospects of our work.

## 2 Literature Review

### 2.1 FastFabric

FastFabric [4] is basically an improved version of Hyperledger Fabric. It increases the throughput of Hyperledger Fabric by two improvements in the ordering service and 5 improvements in the peer services. Let us discuss those improvements in details

Improvements are done on the services of orderers and peers.

There are mainly two improvements in the ordering services:

- Orderer Improvement I: Separate transaction header from payload
- Orderer Improvement II: Message Pipelining Detailed discussion on the improvements are discussed next.

#### •Orderer improvement I: Separate transaction header from payload

In Fabric 1.2, the entire transaction is sent to Kafka for ordering by the orderers. Transactions can be many kilobytes in length, resulting in high overhead contact affecting overall efficiency. This impacts overall performance and throughput.

Only transaction IDs are needed to achieve agreement on the transaction order, so sending only transaction IDs to the Kafka cluster will result in a substantial increase in orderer throughput. As the consensus on the transaction requires only the TrIDs, so if we send only the IDs to the Kafka, we obtain a significant improvement in the throughput. Orderer extracts the transaction ID from the header and publishes this

ID to the Kafka cluster on receiving a transaction from a client.

●**Orderer improvement II: Message pipelining:** One transaction couldn't be processed before the previous transaction's channel is identified, validity checked against a set of rules and forwarded to the consensus system. Which was time consuming and gave very small throughput.

So a pipelined mechanism is proposed that can process multiple incoming transactions concurrently, even if they originated from the same client. They maintain a pool of threads that process incoming requests in parallel, with one thread per incoming request. A thread calls the Kafka API to publish the transaction ID and sends a response to the client when successful.

Pipelined mechanism:

- Process multiple incoming transactions concurrently
- Maintain a pool of threads that process incoming requests in parallel, with one thread per incoming request

**New orderer architecture:**

- Incoming transactions are processed concurrently
- TxID is extracted from header
- The TransactionID is sent to the Kafka cluster for ordering



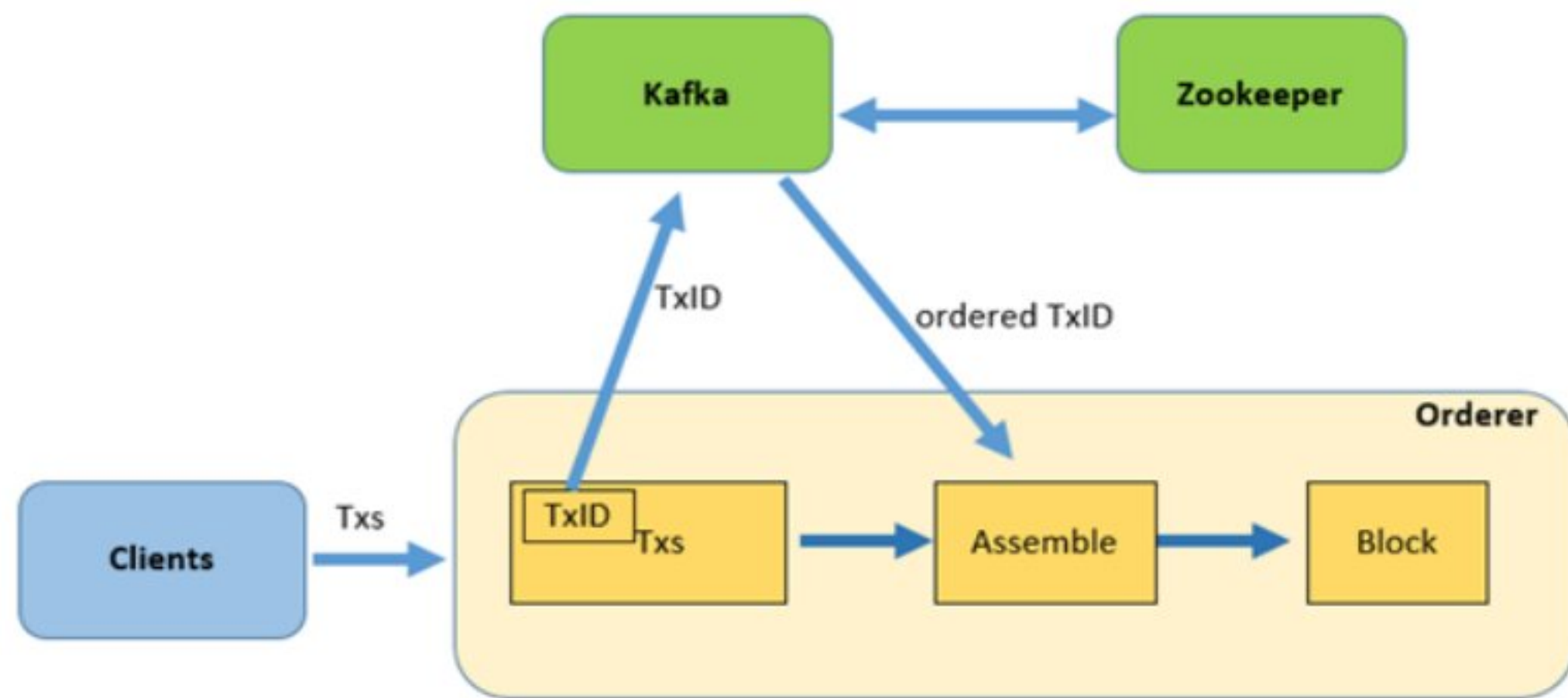


Figure 4: Architecture of the new Orderer

- When receiving ordered TransactionIDs back, the orderer reassembles them with their payload and collects them into blocks

The whole architecture of the new orderer is shown on the Figure 4.

**Improvement in the peering service:** The peer does a number of tasks like:

- Verify that the received message is valid.
- For each transaction in the block, validate the block header and each endorsement signature.
- Validate collections of transactions to read and write
- In either LevelDB or CouchDB, change the world state
- Store the blockchain log, with the corresponding indices, in the LevelDB file system.

Multiple improvements are made to these peer nodes to increase the throughputs. Like:

- Peer Improvement I: Replacing the world state database with a hash table
- Peer Improvement II: Store blocks using a peer cluster
- Peer Improvement III: Separate commitment and endorsement
- Peer Improvement IV: Parallelize validation
- Peer Improvement V: Cache unmarshalled blocks

Detailed discussion on the improvements are discussed below:

**●Peer Improvement I: Replacing the world state database with a hash table**

For each transaction, the world state database must be looked up and checked sequentially to ensure continuity across all peers. It is also important for changes to this data store to occur at the highest possible transaction rate. The world state is basically small relative to the large amount of data. We need a solution so that billions of data can be stored in memory. So, an in-memory hash table is proposed instead of LevelDB/CouchDB to store world state. The hash table:

- Eliminates hard drive access when updating world state



•Eliminates cost of unnecessary database redundancy item•**Peer Improvement II: Store blocks using a peer cluster**

Blocks are permanent. So they are ideally suited for stores of append-only data.

We can imagine several types of data stores for blocks and world state backups by decoupling data storage from the remainder of a peer's tasks, including a single server storing blocks and world state backups in its file system, as Fabric currently does.

They suggest the use of a distributed storage cluster for optimal scaling. Notice that with this approach, only a fraction of the chain is included in each storage server, which motivates the use of distributed data processing tools such as Hadoop MapReduce or Spark5.

•**Peer Improvement III: Separate commitment and endorsement**

Endorser peers also act as committing blocks. Endorsement and commitment are both expensive operations. Although parallel transaction processing on a cluster of endorsers may theoretically boost application performance, additional work to replicate commitments on each new node effectively nullifies the benefits.

To improve this, they propose to split the roles, where a committer peer executes the validation pipeline and then sends validated blocks to a cluster of endorsers who only apply the changes to their world state without further validation. This step allows us to free up resources on the peer. Here it is important that an endorser cluster, which can scale



out to meet demand, only splits off the endorsement role of a peer to dedicated hardware.

•**Peer Improvement IV: Parallelize validation** Both block and transaction header validation are highly parallelizable, including verifying sender permissions, implementing endorsement policies, and syntactic verification.

They expand Fabric 1.2's concurrency efforts by adding a full validation pipeline. Specifically, for each incoming block, one go-routine is assigned to the shepherd via the block validation process. Each of these go-routines subsequently allows use of the go routine pool that already exists for transaction validation in Fabric 1.2. Therefore several blocks and their transactions are tested for validity in parallel at any given time.

•**Peer Improvement V: Cache unmarshalled blocks** Fabric 1.2 does not store previously unmarshalled data in a cache, so this work has to be redone whenever the data is needed. To solve this problem, a temporary cache of unmarshalled data is proposed.

Blocks are stored in the cache when in the validation pipeline and whenever necessary, retrieved by block number. When every portion of the block has been unmarshaled, it is processed for reuse with the block. We implement this as a cyclic buffer that is as wide as the pipeline of validation.

A new block may be admitted to the pipeline once a block is committed and immediately overwrites the current cache location of the committed

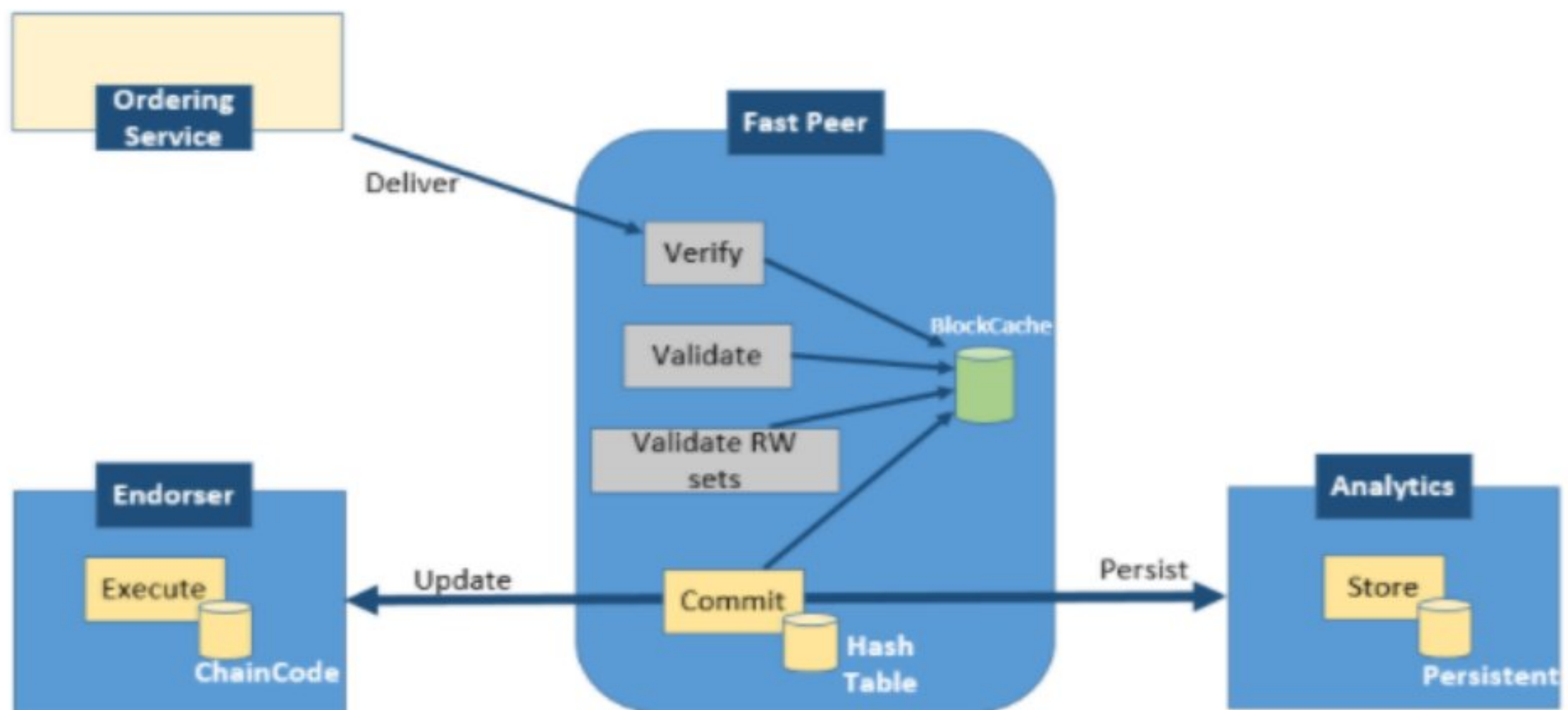


Figure 5: Architecture of New Peer

block. Since the cache is not needed after commitment and a fresh block is guaranteed to arrive only after an old block leaves the pipeline, this is a safe process. Remember that unmarshaling only adds, never mutates, information to the cache.

### **New peer architecture:**

- The fast peer uses an in-memory hash table to store the world state
- The validation pipeline is completely concurrent, validating multiple blocks and their transactions in parallel
- The endorser role and the persistent storage are separated into scalable clusters and given validated blocks by the fast peer
- All parts of the pipeline make use of unmarshaled blocks in a cache



### 2.1.1 Performance analysis of Ordering and Peer Improvements

#### **Orderer Improvement:**

- When we send only the TrID to Kafka, it triples the average throughput (2.8x) for a payload size of 4096 KB.
- Adding optimization O-2 leads to an average throughput of 4x which increases orderer performance from 6,215 transactions/s to 21,719 transactions/s, a ratio of nearly 3.5x.

#### **Peer Improvement:**

- By using a hash table for state storage (Opt P-I), they doubled the throughput of a Fabric 1.2 peer from about 3200 to more than 7500 transactions/s
- Parallelizing validation (Opt P-II) adds an improvement of roughly 2,000 transactions/s
- If we consider all the peer optimizations together, the increase in a peer's commit performance is 7x from about 3200 transactions/s to over 21,000 transactions/s

**End-to-end throughput:** This is the overall improvement in throughput combining all the improvements and the results show:

- Fabric 1.2 used to have about 3185 transactions/s
- FastFabric gives about 19112 transactions/s



### **2.1.2 Tangaroa**

A team at Stanford developed a byzantine fault tolerant Raft [5] algorithm by making significant changes to the existing algorithm. Nodes in Tangaroa send a hash value rather than the real data chunk before the transaction is committed, reducing message overhead for aborted transactions significantly. Hashing ensured that the local states of two distinct nodes remained consistent, which in turn ensured that transactions were handled in the same order.

### **2.1.3 Limitations**

Existing methods that offer higher throughput are not Byzantine Fault Tolerant and require extensive system resources. The Tangaroa has a rather nonoptimal amount of overhead that make it unideal for Hyperledger Fabric.

## 3 Proposed solution

### 3.0.1 Hashgraph

Hashgraph [6] is an asynchronous Byzantine Fault Tolerant consensus algorithm that uses gossip about gossip and virtual voting to achieve consensus. This means that attackers will not be able to alter the order in their favor as long as they do not have over 33.33% of the stake. Hashgraph containerizes data in forms of events. Each event contains timestamp, two hashes of two events below itself, self-parent, other-parent, transactions, and digital signature. Hashgraph uses gossip about gossip to reach consensus. A gossip sync is the synchronization of information between two participants using the gossip protocol. The data collected at the end of each gossip sync is stored in events which is essentially a data structure containing, a timestamp, an array of zero or more transactions, two parent hashes, and a cryptographic signature. It then uses virtual voting to reach consensus. In this step, it sorts the timestamps for each transaction and selects the median timestamp as the consensus timestamp.

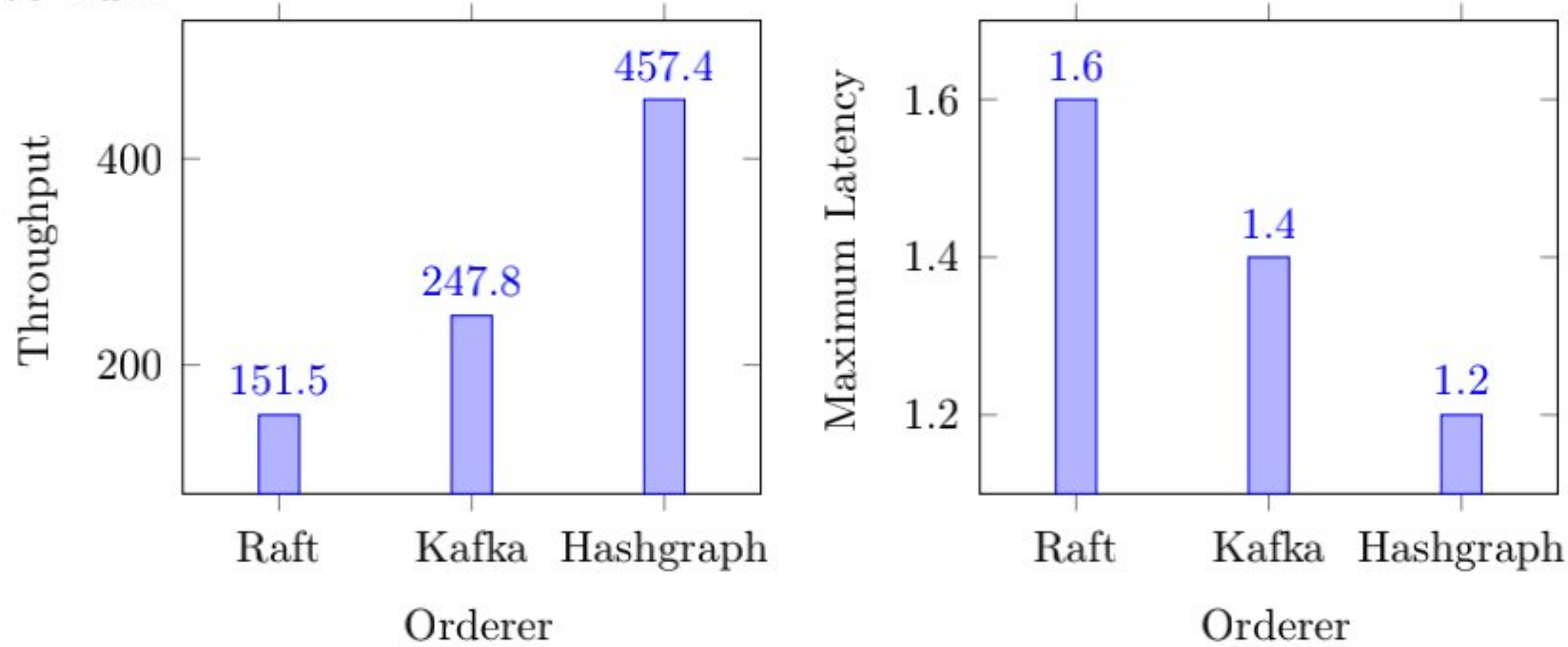
## 4 Experimental Analysis

### 4.1 Experimental Setup

We used Hyperledger Caliper on a virtual machine running Ubuntu Linux with Intel Core i7 10th Gen CPU, 4GB RAM and 50 GB storage. We conducted the experiment for querying transactions on a small network consisting of two organizations and a peer node. We obtained a set of five values for various performance metrics for each orderer mechanism by running the experiment over a period of thirty seconds each round.

### 4.2 Experimental Results

Our results show that there is an 80% increase in the throughput of Hashgraph in contrast to Kafka. There is also a slight decrease in maximum latency. FastFabric requires more extensive system resources to run.





## **5 Conclusion**

### **5.1 Summary**

Our proposed solution is ideal for small scale blockchain networks on Hyperledger Fabric as it does not require a lot of system resources. It also provides Byzantine Fault Tolerance with minimal overhead and a higher throughput.

### **5.2 Future Work**

In the future we will explore the existing overheads in the authentication phase of Hyperledger Fabric to further optimize it. We will also expand our experiment by testing it under various system configurations and networks to derive more insights.

## References

- [1] 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.
- [2] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.
- [3] Vikram Dhillon, David Metcalf, and Max Hooper. The hyperledger project. In *Blockchain enabled applications*, pages 139–149. Springer, 2017.
- [4] 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.
- [5] Christopher Copeland and Hongxia Zhong. Tangaroa: a byzantine fault tolerant raft, 2016.
- [6] Leemon Baird, Mance Harmon, and Paul Madsen. Hedera: A public hashgraph network & governing council. *White Paper*, 1, 2019.