Performance Evaluation of Hyperledger Fabric

Salma Shalaby *, Alaa Awad Abdellatif* †, Abdulla Al-Ali *, Amr Mohamed *, Aiman Erbad *, Mohsen Guizani *
* Department of Computer Science and Engineering, Qatar University, Qatar
† Department of Electronics and Telecommunications, Politecnico di Torino, Italy

Emails: {ss1303998, abdulla.alali, amrm, aerbad, mguizani}@qu.edu.qa, alaa.abdellatif@polito.it

***Abstract*—Blockchain is a distributed secure ledger that eliminates the need for centralized authority to store data. The centralized approach has several limitations as it is a Single-Point-of-Failure and a third-party might be needed. Blockchain, on the other hand, provides decentralized, secure and trustless framework that eliminates the need for a third party and enhances fault tolerance. In this paper, we investigate the potentials of customizing the behavior of Blockchain network based on the applications' requirements. In particular, we conduct several experiments to evaluate the performance of the Hyperledger Fabric (HLF) – a permissioned blockchain framework. Seven different scenarios were tested to depict the Blockchain behavior in terms of end-to-end transaction latency and network throughput. Moreover, in these scenarios, the impact of different parameters, such as the batch-timeout, batch size, and number of endorsing peers, has been studied.**

***Index Terms*—Blockchain, Hyperledger Fabric, Decentralized Applications**

I. Introduction

In 2009, Satoshi Nakamoto came up with Bitcoin [1], which is the first decentralized cryptocurrency. Power of Bitcoin comes from the underlying technology, which is Blockchain. Although cryptocurrency was the first application that utilizes blockchain, it has been used in several applications, such as: banking, healthcare, supply chains and much more [2]. When it comes to storing and sharing data between different entities, the centralized database has some limitations. One of these limitations is single-point-of-failure; if there is an attack, the whole system may fail. Also, because of privacy concerns, it might not be acceptable to store the data at a third-party [3]. One possible solution is to choose a trusted entity to store the data. However, since these entities have different policies, it would be hard to agree on the entity that will be storing the data. By its distributed nature, blockchain overcomes these limitations as there is no centralized authorization; each entity can have a copy of the ledger, and all the entities should agree on the transactions before they are committed to the chain. In blockchain, each block of transactions refers to the previous block by its hash, which guarantees data integrity. Thus, if an attacker tries to modify any block, the change will propagate through the chain and will be recognized. Moreover, blockchain enables the usage of smart contracts. A smart contract or a digital contract is an automated program that controls the transaction logic and does not require an intermediary to run it [4].

In the literature, there are mainly two types of blockchains: permissioned and permission-less blockchains. The main aim of the the latter is to provide public accessibility and transparent transactions, thus, eliminates confidentiality [5]. To store sensitive data, e.g., for medical applications, confidentiality is a core requirement. Therefore, permissioned blockchain emerged to solve this issue. Unlike permission-less blockchain that allows anyone to participate, permissioned blockchain enables data sharing and access to specific trusted entities/users (see Table I) [3]. However, these entities have to obtain a consensus between each other in order to identify any unauthorized data manipulation. In Bitcoin, Proof-of-Work (PoW) consensus scheme is used in which miners compete to solve a computationally intensive puzzle and once a miner solves this puzzle it broadcasts the new block. One of the limitations of PoW is that it is vulnerable to 51% attack which happens if a single entity owns more than 51% of the computational power of the blockchain enabling them to take control of the whole network [6]. Peercoin [7] proposed Proof-of-Stake (PoS) to decrease the computational overhead of PoW. PoS depends on the amount of time that the minter holds a certain amount of currency. This approach uses coin-age which is the number of days of holding the currency times the amount that has been held. Based on the coin-age, the miners are chosen; the probability of being chosen as the next block miner increases as the coin-age increases until it is maximized when the coin-age is 90 days. Unlike PoW, this approach does not consume huge amount of resources. Also, it is not vulnerable to 51% attack as the attacker needs to own more coins than the rest of the network; causing an increase in the coin price, which turns the attacks very costly and almost impossible [8]. PoW and PoS are two of the most common consensus techniques in permission-less blockchains that guarantee trust, however the mining process is time consuming. On the contrary, permissioned blockchain leverages faster protocols to achieve consensus [9].

Amongst the most common platforms for implementing blockchains are Ethereum [10], and Hyperledger fabric [11]. Ethereum was established in 2015 and eventually it became one of the biggest programmable blockchains framework. It is used for permission-less blockchain to exchange Ether (ETH) cryptocurrency. However, Ethereum can offer more than this as it is programmable and open source. Thus, it is used also for the permissioned blockchains to develop customized

TABLE I
COMPARISON BETWEEN PERMISSIONED AND PERMISSION-LESS BLOCKCHAINS

Permission-less blockchain	**Permissioned blockchain**
Open - anyone can access it	Access is guaranteed to certain users
Fully decentralized	Partially decentralized
Slow	Fast

decentralized applications [12]. While Ethereum is more light-weight and easier to use, it is highly not customizable, making it hard to change system parameters such as the consensus algorithm, the number of transactions per block, etc.

This paper focuses on the performance evaluation of Hyperledger Fabric (HLF) for building a secure healthcare system. In healthcare applications, it is crucial to process the acquired data from the patients before sharing or storing it [17]. Moreover, it is important to enable secure data sharing scheme to prevent accessing the acquired data by untrusted users and violating users' privacy [18]. We tackle this challenge by implementing a blockchain-based data sharing scheme that provides medical data access, processing, and sharing between big healthcare entities, i.e., hospitals, pharmacies, medical institutions, and so on. In this context, a basic framework is built using HLF, where a smart contract is designed to control the transaction logic and to enforce policies on who can access the network. The rest of the paper is structured as follows: section II presents some important concepts in Hyperledger Fabric, section III introduces the environmental setup, section IV presents the experimental results, and finally section V concludes the paper.

II. HYPERLEDGER FABRIC

Hyperledger Fabric (HLF) is an open source permissioned blockchain established by the Linux Foundation that is mainly used to serve enterprises. What distinguishes HLF from other platforms is its new transaction architecture called Execute-Order-Validate architecture. This new architecture replaces the traditional order-execute one used by all of the existing platforms. In order-execute architecture, first, transactions are ordered based on the consensus protocol. Then, in the execution phase, each peer executes transactions sequentially in the same order. This execution phase has a negative impact on the performance of the network as the peers have to go through all the transactions in the block and execute them which increases the latency. Another point that differentiates HLF is that it supports general-purpose programming language smart contracts or chaincodes as HLF call them. It is obligatory that smart contracts in platforms that follow the order-execute architecture have to be deterministic; and that is why some platforms require writing the smart contract in a Domain-Specific Language (DSL) to eliminate the non-deterministic operations [13], [14].

A. Channels

Based on the requirements, HLF allows the creation of several channels within the same network. A channel could be referred to as a private communication network within the main network. Each channel has its own ledger, therefore only the peers and organizations that are members of this channel will have a copy of this ledger. These members are defined in the channel policy in the configuration block that also defines the type of ordering service. Whenever any configuration is modified a new configuration block is created and added to the chain [14].

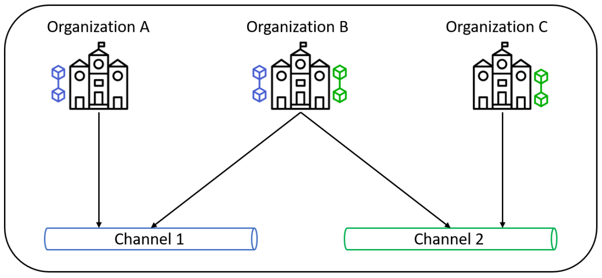


Fig. 1. Clarifying Channels Concept

Fig. 1 clarifies the channel concept, where organization A and organization B have access to channel 1 and organization B and organization C have access to channel 2. Peers in organization 1 have a copy of the ledger that belongs to channel 1 only. Similarly, peers in organization C have a copy of the ledger that belongs to channel 2 only. Since organization B is a member in both channels, its peers have copy of both ledgers. This could be useful in scenarios where there are competing companies; as it allows private communication between the organizations on the same channel.

B. Implementations of Ordering Service

To achieve consensus, HLF introduces orderer nodes. As the name implies, the main responsibility of orderer nodes is ordering the transactions in the blocks and broadcasting them to the peers for validation. HLF offers three different implementations of the ordering service [14]:

- **Solo:** In solo-based implementation there is only one ordering node, which makes it a single-point-of-failure. Consequently, solo-based ordering service is not suitable for production. Nevertheless, it could be applied for testing and for academic usage as it eliminates the administrative overhead in the other implementation schemes.
- **Kafka:** This mode follows leader and follower approach, where the leader orderer sends the transactions to the follower orderer nodes. The choice of the leader is done in dynamic fashion and as long as the majority of the nodes is up, the system is Crash Fault Tolerant (CFT). This scheme utilizes zookeeper coordination service [15] for managing Kafka clusters. Zookeeper is a service used by decentralized applications in order to maintain configuration information, help in tasks coordination, provide distributed synchronization and cluster membership. Although this scheme was the only option that supports multiple orderers since HLF v1.0, the setup of Kafka-

based ordering service is challenging and it requires experts to deploy it [14].

- **Raft:** Recently, HLF added Raft ordering service that is based on Raft protocol. It is similar to Kafka, as it has the advantage of being CFT and it follows the leader and follower approach. From the functional point of view there are no differences between Raft and Kafka, however, Raft is easier to setup.

C. Transaction Flow

In order to achieve consensus in HLF, each transaction goes through Execute-Order-Validate process summarized in Fig. 2, the process starts once the transaction is proposed and ends when it is committed to the ledger [14]:

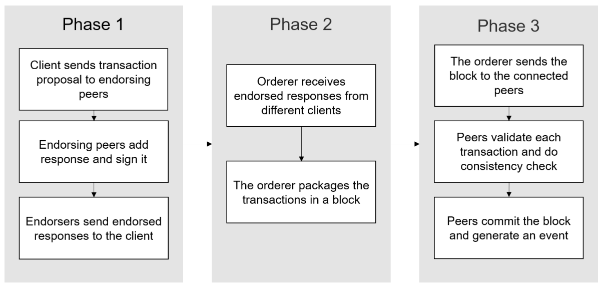

Phase 1
Client sends transaction proposal to endorsing peers
Endorsing peers add response and sign it
Endorsers send endorsed responses to the client
Phase 2
Orderer receives endorsed responses from different clients
The orderer packages the transactions in a block
Phase 3
The orderer sends the block to the connected peers
Peers validate each transaction and do consistency check
Peers commit the block and generate an event

Fig. 2. Execute-Order-Validate process.

Phase 1: Proposal (Execute)
The main aim of this phase is endorsing the transactions. This phase starts when the application client sends a transaction proposal to the endorsing peers. The set of the endorsing peers chosen is determined by the endorsement policy, each of these peers checks the following:

1) The proposal is well structured
2) The application is not trying to duplicate a transaction that already exists
3) The signature of the issuer is valid
4) The issuer is allowed to perform the proposed operation

Then, each endorser executes the chaincode individually and generates a transaction response based on the execution results, and then it signs the response. Finally, the signed transaction proposal response is sent back to the application. Depending on the number of endorsing peers defined in the endorsement policy, the client waits until it receives a certain number of endorsements, which marks the end of the first phase.

Phase 2: Ordering and packaging (Order)
This phase is concerned with packaging the transactions into blocks, which is the orderer's main responsibility. First, the client sends the endorsed transaction proposal responses to the orderers. Then, the orderers package the authorized transactions into a block in a strict order. It is worth noting that the number of transactions per block is decided by the channel configuration, and it affects the overall latency of admitting the transaction into the blockchain. The orderer does not bother itself looking into the content of the transactions, unless it is a configuration block.

Phase 3: Validation and committing (Validate)
The last phase starts when the orderer sends the block to the peers connected to it, peers that are not connected to the orderer will eventually receive the block by gossiping. Each peer on the channel will validate the transactions in the block separately but in a deterministic way, since all the peers validate the block in the same way, each peer will have an identical copy of the ledger. The process of validation includes ensuring that the transaction is endorsed by the required endorsers according to the endorsement policy. To commit the block to the ledger, the peers perform a consistency check to ensure that none of the assets was updated by any other transaction when phase 1 and 2 were taking place. After that, the peers commit the block to the ledger, adding another field to each transaction to indicate whether it is valid or not.

III. Environment Setup

In this work, a total of seven experiments are conducted to evaluate HLF performance in terms of the end-to-end latency and network throughput. Building the HLF framework is divided into: network infrastructure and an application layer.

A. Network Infrastructure

In this work, HLF v1.4 is used to build the network. As shown in Fig. 3, healthnet network consists of one channel that connects two peer organizations: org1 and org2 and one orderer organization, each of the peer organizations has 2 connected peers. The ordering service utilizes the Solo-based design as it will not be used for production.

The channel configuration is defined by two main parameters: Batch-Timeout and Batch Size:

- **Batch-timeout:** It defines the waiting time before creating a block.
- **Batch size:** It controls the number of transactions per block; it is defined by three variables:
  - Maximum transaction Count: The maximum number of transactions per block.
  - Absolute Maximum Bytes: The maximum number of bytes per block that cannot be exceeded.
  - Preferred Maximum Bytes: The preferred number of bytes per block.

The aforementioned parameters have big impact on the performance of the network. The transactions are batched as a block whenever one of the limits is reached; meaning that if

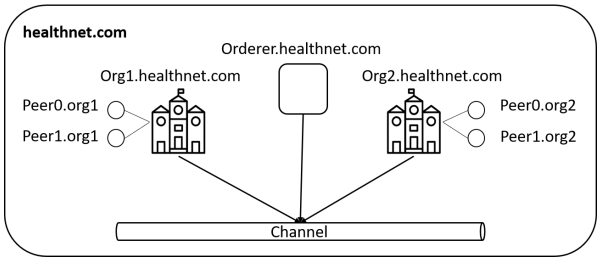


Fig. 3. : High-level architecture of the network.

the batch-timeout is reached but the number of transactions is still less than batch size then the orderer will batch the transactions into a block as discussed in section II-C.

B. Application Layer

In this work, Hyperledger Composer is used to ease the process of implementing the application layer. It helps in modeling the business network that is being packaged to an archive (.bna file) to be used on top of HLF infrastructure. The business network is defined using three main files: Model File, Script File and Access Control File [16]:

- **Model File:** A .cto file that defines all the assets, participants and transactions. It is written in Hyperledger Composer Modeling Language.
- **Script File:** A .js file that is considered as the smart contract where the transactions logic is implemented, it is written in JavaScript.
- **Access Control File:** In HLF, users do not have the same access level. A .acl file defines the access control rules that states the CRUD (Create, Read, Update and Delete) operations a user can perform in the business network like creating assets, participants or transactions. For example; some users can only read data, others can read, write create and delete.

TABLE II
THE BUSINESS MODEL

Assets	Participants	Transactions
Case	Doctor	CreateDoctor
	AdminStaff	CreateCase
		TransferCase

Our application is designed for sharing medical data between healthcare participating entities. The model file defines one asset, two participants, and three transactions as shown in table II. It is worth noting that in such scenarios the business model should be more complicated, however these were only created for evaluating the network. Regarding the access control rules, only the Network Admins are allowed to add new assets, participants, and transactions. In real networks, the access rules have to be more complicated. For example, only the Network Admins can add new Staff, e.g., new doctors using the CreateDoctor transaction. Regarding the assets, only the doctors will be allowed to add a new Case using CreateCase transaction. Both Admins and doctors will be allowed to transfer a case from a doctor to another doctor. Each of the cases is defined by some attributes like the name of the patient, age, vitals, description, a supervising doctor, etc. It should be noted that all the experiments in this study were done using the createCase transaction to guarantee that all the transactions have the same size. The createCase transaction operates as follows:

1) Create a new Case instance
2) Assign the attributes to the case instance
3) Assign a doctor to the case
4) Add this case to the doctor's list
5) Update Case Registry
6) Update Doctor Registry

C. Experiments

Seven different experiments were conducted to show how HLF performance is affected by changing the batch-timeout, batch size (by changing the maximum transactions count) and the number of endorsing peers. Table III shows the exact setup, in experiments 1, 2 and 3 the batch size is fixed to 45 transactions (Tx) and the number of endorsers is fixed to 4 to examine the effect of changing the batch-timeout. Experiments 4, 5 and 6 study the effect of the batch size by fixing the batch-timeout to 200s and the endorsers to 4 and changing the batch size only. In experiments 1 and 7 the batch-timeout and batch size are set to 2s and 45 Tx, respectively, and only the number of endorsing peers changes. For each of the experiments the average end-to-end latency and the average throughput are measured by taking the average of 10 trials while having different number of parallel transactions. To study the effect of the batch-timeout and the number of endorsers, we started from 1 transaction up to 30 parallel transactions, however, while studying the effect of the batch size we started from 10 parallel transactions and not from 1 transaction; because if there is only one transaction then the batch size will never be reached and the delay will be caused by the batch-timeout which is is set to 200s.

IV. RESULTS

The experiments done are divided into three sets, the first set shows the effect of changing the batch-timeout, the second set studies how the block size affects the performance and the third set focuses on the number of endorsers.

A. Batch-timeout

In experiments 1, 2 and 3 we study how the end-to-end latency and throughput are affected by changing the batch-timeout as the number of concurrent transactions increases to 30 while fixing the batch size to 45 Tx which is big enough to avoid reaching the maximum batch size before the timeout and to accommodate all the transactions in one block. Fig. 4 shows that the latency increases as the number of parallel transactions increases. It also shows that it increases by increasing the batch-timeout. Fig. 5 shows that the throughput increases as the number of parallel transactions increases; this happens

TABLE III
EXPERIMENTAL SETTINGS

	Batch-timeout	Batch Size (Max. Transaction Count)	Number of Endorsers
Exp. 1	2 Seconds	45 Tx	4 endorsers
Exp. 2	5 Seconds	45 Tx	4 endorsers
Exp. 3	8 Seconds	45 Tx	4 endorsers
Exp. 4	200 Seconds	2 Tx	4 endorsers
Exp. 5	200 Seconds	5 Tx	4 endorsers
Exp. 6	200 Seconds	10 Tx	4 endorsers
Exp. 7	2 Seconds	45 Tx	2 endorsers

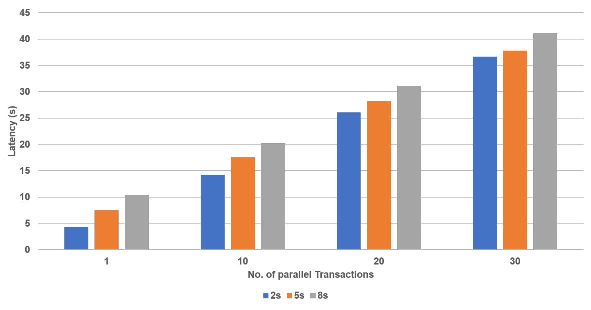


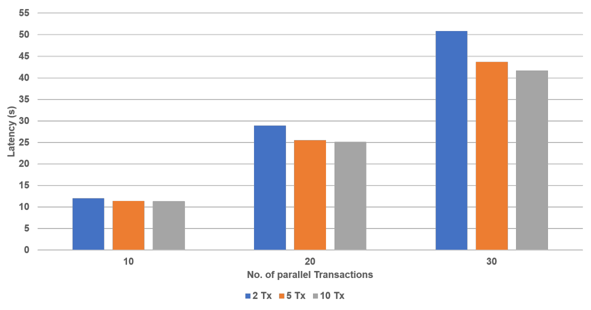
Fig. 4. Average latency with varying Batch-timeout

because the number of transactions within the block increases meaning that more transactions will be processed at a time. It is also observed that as the batch-timeout increases the throughput decreases as a result of increasing the delay.

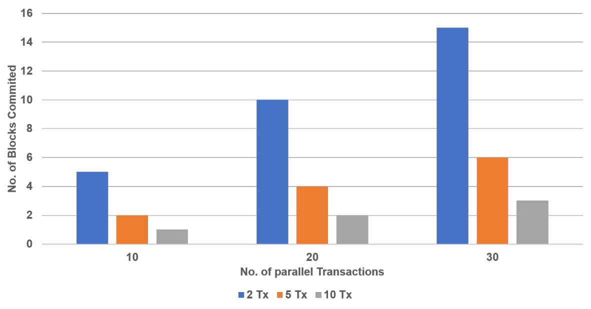
B. Batch Size (Max. Number of Tx per Block)

In experiments 4, 5 and 6 the batch-timeout is fixed to 200 seconds and the batch size varies by changing the maximum number of transactions per block to study how it affects the end-to-end latency and throughput. The batch-timeout was set to a high value (200 seconds) to ensure that the blocks will never timeout before reaching the block size. Fig. 6(a) shows that the latency almost doubles when the number of transactions increases. It also shows that increasing the batch size decreases the latency; this change is clear when the batch size increases from 2 Tx to 5 Tx, but as it increases from 5 Tx to 10 Tx the change is negligible. The reason behind that is illustrated in fig. 6(b) that shows the number of blocks in which the transactions were packaged and committed; the difference between the number of blocks committed is small when the batch size increased from 5 Tx to 10 Tx when it is compared to the difference between the number of blocks when it increased from 2 Tx to 5 Tx. This is reflected on the latency as it increases as the number of blocks increases because there are more blocks to be validated as discussed in section II-C.

Fig. 7 shows that the throughput decreases as the number of parallel transactions increases. On the other hand, it demonstrates that the throughput increases as the batch size increases



(a) Average latency with varying Batch Size



(b) Number of Blocks Committed

Fig. 6. Effect of Changing Batch size

because there are more transactions per block and eventually the number of blocks decreases.

C. Number of Endorsers

In experiments 1 and 7 we examine how the number of endorsing peers affects the latency. In both experiments the batch-timeout is set to 200 seconds and the batch size is set to 45 Tx. From Fig. 8, it is observed that the latency increases as the number of parallel transactions increases. It also shows that increasing the numbers of endorsers leads to a slight increase in the latency; because with increasing the number of endorsers the client has to wait for more endorsed transactions responses before sending to the orderer in the second phase of the transaction flow.

V. Conclusion

This paper has investigated the performance evaluation of Hyperledger Fabric for healthcare applications. Several

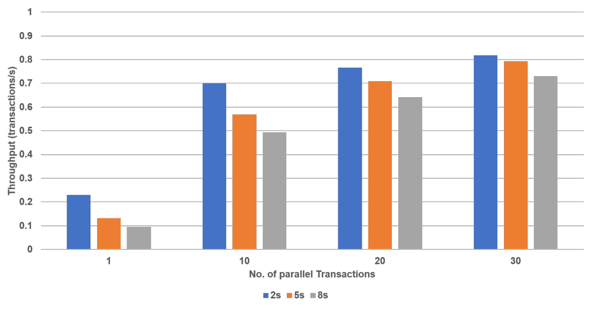


Fig. 5. Average throughput with varying Batch-timeout

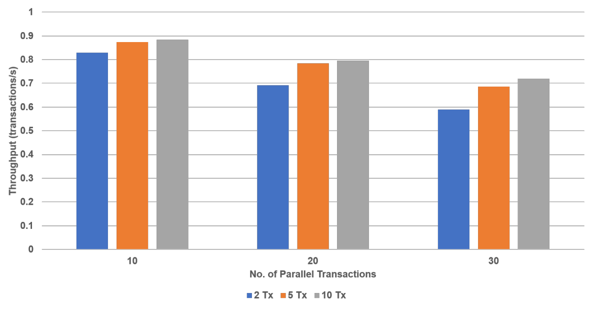


Fig. 7. Average throughput with varying Block Size

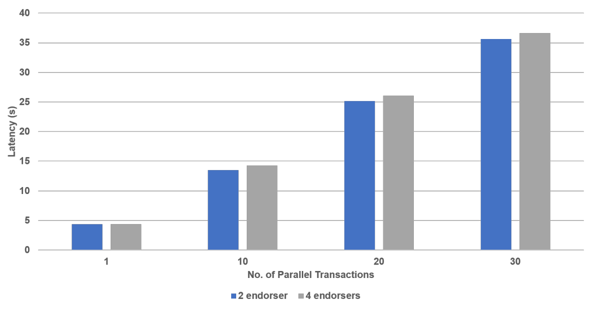


Fig. 8. Average latency with varying number of Endorsing Peers

scenarios were conducted to study the average end-to-end latency and throughput. The parameters that were considered in this work are the batch-timeout, block size and the number of endorsing peers, while varying the number of parallel transactions. Our results reveal that the latency increases as the number of transactions and batch-timeout increase. Also, we show that the number of generated blocks and number of transactions per block have an impact on the obtained throughput. Indeed, the throughput increases as the block size increases, because more transactions in one block means that more transactions will be validated at the same time. It is also observed that increasing the batch-timeout leads to an increase in the latency because each block has to wait for the timeout even if it has received all the transactions. Thus, we recommend that: (i) for the applications with large number of parallel transactions, the batch-timeout and block size should be large in order to maintain high throughput; (ii) for the application with urgent transactions, the batch-timeout and block size should be limited in order to obtain low latency (especially, in case of emergency). For the future work, using the implemented framework, more parameters will be studied, such as number of orderers. Also, Kafka and Raft ordering services will be implemented in order to mimic a realistic scenario, as Solo ordering service should not be used in production. Furthermore, large size network will be considered with increasing number of organizations and parallel transactions.

VI. Acknowledgement

This work was made possible by Qatar University Grant QUHI-CENG-19/20-1. The findings achieved herein are solely the responsibility of the authors.

References

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” p. 9.

[2] B. A. Tama, B. J. Kweka, Y. Park, and K. H. Rhee, “A critical review of blockchain and its current applications,” in 2017 International Conference on Electrical Engineering and Computer Science (ICECOS), 2017, pp. 109–113.

[3] X. Xu et al., “A Taxonomy of Blockchain-Based Systems for Architecture Design,” in 2017 IEEE International Conference on Software Architecture (ICSA), 2017, pp. 243–252.

[4] V. Singla, I. K. Malav, J. Kaur, and S. Kalra, “Develop Leave Application using Blockchain Smart Contract,” in 2019 11th International Conference on Communication Systems Networks (COMSNETS), 2019, pp. 547–549.

[5] Z. Alhadhrami, S. Alghfeli, M. Alghfeli, J. A. Abedlla, and K. Shuaib, “Introducing blockchains for healthcare,” in 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA), 2017, pp. 1–4.

[6] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press, 2016.

[7] S. King and S. Nadal, “PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake,” p. 6.

[8] “Peercoin University.” [Online]. Available: https://university.peercoin.net/. [Accessed: 11-Nov-2019].

[9] M. E. Peck, “Blockchain world - Do you need a blockchain? This chart will tell you if the technology can solve your problem,” IEEE Spectrum, vol. 54, no. 10, pp. 38–60, Oct. 2017.

[10] “Home — Ethereum,” ethereum.org. [Online]. Available: https://ethereum.org. [Accessed: 12-Nov-2019].

[11] “Hyperledger Fabric.” [Online]. Available: https://www.hyperledger.org/projects/fabric. [Accessed: 09-Nov-2019].

[12] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, “Performance Analysis of Private Blockchain Platforms in Varying Workloads,” in 2017 26th International Conference on Computer Communication and Networks (ICCCN), 2017, pp. 1–6.

[13] E. Androulaki et al., “Hyperledger fabric: a distributed operating system for permissioned blockchains,” presented at the Proceedings of the Thirteenth EuroSys Conference, 2018, p. 30.

[14] “Hyperledger-fabricdocs master documentation.” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/orderer_deploy.html. [Accessed: 21-Oct-2019].

[15] “Apache ZooKeeper.” [Online]. Available: https://zookeeper.apache.org/. [Accessed: 09-Nov-2019].

[16] “Introduction — Hyperledger Composer.” [Online]. Available: https://hyperledger.github.io/composer/latest/introduction/introduction. [Accessed: 23-Oct-2019].

[17] A. Awad, M. Hamdy, A. Mohamed and H. Alnuweiri, ”Real-time implementation and evaluation of an adaptive energy-aware data compression for wireless EEG monitoring systems,” 10th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, Rhodes, 2014, pp. 108-114.

[18] A. A. Abdellatif, A. Mohamed, C. F. Chiasserini, M. Tlili and A. Erbad, ”Edge Computing for Smart Health: Context-Aware Approaches, Opportunities, and Challenges,” in IEEE Network, vol. 33, no. 3, pp. 196-203, 2019.