Performance analysis of hyperledger fabric 2.0 blockchain platform

Conference Paper · November 2020

DOI: 10.1145/3417310.3431398

3 authors, including:

Julian Dreyer
Osnabrück University of Applied Sciences
15 PUBLICATIONS **81** CITATIONS
SEE PROFILE

Ralf Tönjes
Osnabrück University of Applied Sciences
165 PUBLICATIONS **1,782** CITATIONS
SEE PROFILE

Performance Analysis of Hyperledger Fabric 2.0 Blockchain Platform

Julian Dreyer, Marten Fischer, Ralf Tönjes
Lab for RF-Technology and Mobile Communications
University of Applied Sciences Osnabrueck
Osnabrueck, Germany
j.dreyer@hs-osnabrueck.de
m.fischer@hs-osnabrueck.de
r.toenjes@hs-osnabrueck.de

ABSTRACT

Hyperledger Fabric is currently one of the most popular business Blockchain platforms. With its included functionality of executing custom smart contracts, Fabric has become one of the most widely used frameworks, e.g. for industry 4.0 applications. Though, most applications require a previously known data throughput. For a potentially interested developer, it is not trivial to decide, whether a given Fabric network configuration will meet the required expectations in regards to performance. Thus this work shows a performance analysis for Hyperledger Fabric v2.0 and focusses on the evaluation of throughput, latency and error rate, together with the overall scalability of the Fabric Blockchain platform. The results show, that Fabric v2.0 outperforms previous versions in almost any regard in addition to an improved smart contract lifecycle.

CCS CONCEPTS

• **Networks → Network performance analysis**.

KEYWORDS

Blockchain, Hyperledger Fabric v2.0, Performance analysis, Scaling, Distributed Ledger Technology, throughput, latency, error rate

ACM Reference Format:
Julian Dreyer, Marten Fischer, Ralf Tönjes. 2020. Performance Analysis of Hyperledger Fabric 2.0 Blockchain Platform. In *Cloud Continuum Services for Smart IoT Systems (CCIoT '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3417310.3431398

1 INTRODUCTION

The Blockchain Technology offers countless opportunities to increase the data security and redundancy of modern information systems. With its introduction by Nakamoto [5], the first generation of Blockchain systems came up. Its main focus lied on the cryptocurrency domain. With its inherent transparency and tamper-proof data structure, the Blockchain offered new ways for digital payment. With the introduction of Ethereum by Buterin in [1], the second generation of Blockchain technology arrived. The most notable difference to the first Blockchain generation is the introduction of so-called "smart contracts". Software Developers use these to implement their own functionality and execute their custom programs on the Blockchain and thus interacting directly with it. Using this novel paradigm, countless opportunities for custom applications are possible.

Hyperledger Fabric makes use of the smart contract paradigm and offers a fully operational private Blockchain platform, which is currently one of the most established publicly available Blockchain solutions. Firstly introduced by IBM and later handed to the open-source community, Hyperledger Fabric published the official version 2.0 in February 2020. Recent work gave a broad evaluation of the performance e.g. of Hyperledger Fabric v0.6, v1.0 and v1.4 [4]. The importance of the performance evaluation is given because the real-world applications require in-depth planning of the network configuration. A smart contract developer needs to know in advance, whether the requirements in regards to throughput and delay of the Blockchain network can be fulfilled.

Thus, this paper addresses the performance of Hyperledger Fabric v2.0 in regards to the most common evaluation characteristics, namely *throughput*, *latency* and *error rate*. According to the authors knowledge this is the first evaluation of the Hyperledger v2.0 framework. The particular focus lies on the performance analysis of varying network parameters such as the number of peers, orderers, organizations and block size. This paper therefore provides a large performance result set for various Hyperledger Fabric v2.0 network configurations.

The rest of this paper is organized as follows: Section 2 provides an overview of the previously conducted studies in regards to Hyperledger Fabric performance for previously released versions. It also gives an overview of recent publications that address the performance and improvements of Hyperledger Fabric in general. The particular components and improvements within Hyperledger Fabric v2.0 are described within section 3. It also introduces the test strategies and key performance indicators (KPI), which will later be used to evaluate the performance of the Blockchain network. Said evaluation is being described in section 4 and evaluated in section 5. Finally, Section 6 concludes the paper.

2 RELATED WORK

Various performance evaluations of the Hyperledger Fabric Blockchain platform exist in general. One central problem which applies to all of the performance evaluations is the objectivity of the smart contracts. To evaluate the performance, most often a custom smart contract is developed and executed multiple times. Depending on the concrete implementation, the test results may vary by a large extent. Thus the objectivity and the comparability of the results are limited.

Nasir et. al conducted a study by utilizing the previously described methodology in [6]. They developed and deployed a custom smart contract, which is capable of transferring virtual money. The overall goal of their work is to evaluate the performance of one single Blockchain peer node without the inclusion of consensus algorithms. For the evaluation, they also used the common KPIs of "execution time", "latency" and "throughput". The results show, that previous versions of Hyperledger Fabric, namely v0.6 and v1.0 improved in regards to throughput and latency. With an increasing number of incoming transactions, the performance diminishes rapidly. With 10,000 incoming transactions per second (tps), the execution time of a single transaction lies well above 60s for v0.6 and 59s for v1.0 respectively.

Kuzlu et. al evaluated the performance of Hyperledger Fabric v1.4 by varying the incoming workloads in terms of transaction speed, number of transactions and transaction types [4]. The network was set up without regard to the consensus mechanism and no network configuration has been given. For the benchmarking operation, Hyperledger Caliper [10] was used since it is an officially provided benchmarking tool for Hyperledger Fabric v1.4. The results show that the average throughput lies below 200 tps for an "Open" operation when applying non-simultaneous transactions. The latency also increases rapidly after applying a transaction rate above 200 tps. When considering simultaneous transactions the throughput does not exceed 50 tps overall.

Pongnumkul et. al investigated the differences in regards to the performance of Ethereum and Hyperledger Fabric v0.6 [8]. They describe a basic principle for evaluating the performance of the respective Blockchain platform by requiring a common smart contract functionality and scaling the network afterward. For the evaluation, they also used the common performance indicators throughput and latency. The deployed smart contracts revolve around the creation of a money transfer infrastructure. While Fabric exceeds its maximum throughput at 100 incoming transactions at 299.85 tps, Ethereum only reaches a maximum throughput of 38.93 tps. By increasing the incoming transactions per second, the latency also increases linearly in case of Fabric and flattens out at 400s in case of Ethereum.

Various attempts to improve the performance of Hyperledger Fabric have been performed e.g. by Gorenflo et al. [3]. By introducing different improvements to the fundamental architecture of Hyperledger Fabric, the authors managed to scale the throughput performance from 3,000 tps to almost 20,000 tps while also decreasing the latency.

Thakkar et al. [9] investigated multiple performance bottlenecks Hyperledger Fabric v1.0 which can be improved. By making use of aggressive caching or parallelizing the verification of endorsement policies, the authors were able to scale the performance of Hyperledger Fabric v1.0 by a factor of 16. Said improvements were later included into later versions of Hyperledger Fabric.

3 TECHNOLOGIES AND TEST METHODS

Hyperledger Fabric offers a a range of functionalities in regards to customization and possible use cases. It enables a developer to set up a custom Blockchain business network, consisting of heterogenous clients with different roles assigned to them. It also allows to create, deploy and execute smart contracts on the platform. Fabric ensures a flexible configuration by deploying an *execute-order-commit* transaction flow. An endorsement peer node will first execute a transaction and determine its *read-write set*. After every endorsement peer executed the transaction, the results get sent to the ordering service, which orders and sorts the transactions, depending on their timestamps and hierarchical order. Finally, the ordered results get broadcasted via the gRPC Protocol to all peers within the Fabric Blockchain channel [12]. Each one will then commit the results to their own copy of the ledger and thus achieving consensus. The specific roles involved in the transaction flow will be described in the following.

1. Peers

A peer within the Fabric Blockchain network can either be an endorsement or a committing peer. An endorsement peer is responsible for endorsing the transactions which is sent to them by the ordering service. By executing the invoked smart contract code, a read-write set is created which is unique to the particular transaction. A committing peer on the contrary will not execute the smart contract at all. It will only receive the ordered transactions from the orderers and commit them to its copy of the ledger. Any peer can be chosen to be an anchor peer, which is then responsible for the communication with the ordering service or other organizations within the same channel. There must exist at least one anchor peer per organization.

2. Organizations

Every Fabric Blochcain channel can be separated into several organizations, each of which owns a certification authority (CA). There can be multiple peers within each organization that execute and validate the transactions. Though, a single peer can be assigned to one organization only. It has to issue specific keys and certificates from the CA of the organization to communicate with the other participants of the network.

3. Orderers

The ordering service is the central component to ensure that the transactions within the Fabric Blockchain channel get sorted correctly. Fabric v2.0 recommends the use of the "Raft" ordering service and thus replaces the now deprecated Kafka ordering service [11]. Each orderer node will order and append the endorsed transactions to a new block for the Blockchain. After either the upper "block size" limit is reached, which describes the maximum number of transactions a single block can hold, or the "block timeout" is met, the block is being validated and appended to the Blockchain. This is done by broadcasting the block information to all anchor peers of each organization, which then broadcast the information to all peers within the particular organization internally.

4. Fabric v2.0 innovations

With the v2.0 release of Hyperledger Fabric, new features and improvements were introduced [13]. Most notable in regards to performance is the new lifecycle management of the peers. New smart contracts, which are developed against the Fabric v2.0 SDKs are now packaged in a common tape archive (TAR) format, unlike previous versions of Fabric which used a proprietary packaging format for every language. Every organization is now responsible for approving the metadata of the to-be-installed smart contract, after it has been loaded onto every endorsement peer within the channel. If every organization approved the smart contract, it gets committed to the ledger of each peer and is then executable. By utilizing specific configuration parameters within the configuration files, the approval process can be customized.

5. Test methods

The performance of a Hyperledger Fabric business network can be measured by issuing multiple transactions in parallel which the network has to validate. By measuring the number of transactions that get validated per second, the "throughput" can be identified. This metric can be primarily used to evaluate the superficial performance of a network, by comparing it to other Blockchain platforms like Ethereum. Such a comparison has been described by Pongnumkul in [8]. Figure 1 depicts the general concept of the throughput metric.

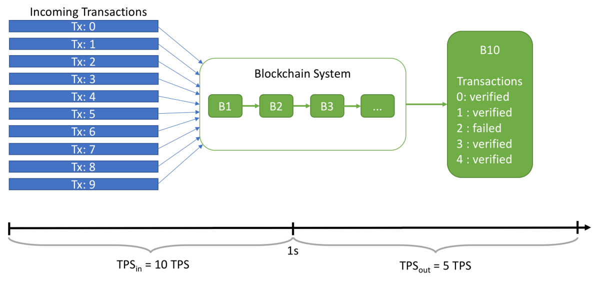

Figure 1: Throughput performance metric

Another metric is the latency/delay of the Blockchain network. It gives a concrete measurement, when the data, which gets added to the Blockchain, is available for query. If the latency was too high, the data could not be read within a short time interval after adding the data to the Blockchain, thus resulting in errors.

The last metric which gets measured within this work is the error rate. There are multiple ways for a transaction to fail within a Blockchain network. This paper focusses on measuring the software-side errors, meaning that external errors due to signal interference or packet loss get excluded from the overall error rate. The software errors mainly revolve around measuring the number of race-conditions, which cause a validation error due to an inconsistent read-write set. Other software errors like validation errors are also included in the overall error rate.

Listing 1: Java Interface for the test smart contract

```
class DefaultContract implements ContractInterface {
    @Transaction()
    public boolean defaultAssetExists(String _id);

    @Transaction()
    public boolean createDefaultAsset(String value);
}
```

Though, all these performance indicators are also depending on the smart contract, which gets executed to evaluate their performance. If the complexity is too high, the comparability to other performance evaluations, e.g. of previous Fabric versions, is not given. Thus, for this paper a custom smart contract has been created with the interface in listing 1, using the Fabric Java Chaincode SDK.

The `DefaultAsset` class is only containing a single `value` attribute, which holds the particular random data of the object itself. Each `DefaultAsset` object also contains a unique `id` object for querying. The sole functionality of the smart contract is to implement a basic Create and Read (CR) scheme, which can be easily compared to existing performance evaluations. Future work can also implement a smart contract based on this scheme and use it for comparable testing. Considering the smart contract, which has been used by Pongnumkul in [8], the complexity has been higher thus making it not comparable with the smart contract interface used in this paper.

While the Kafka ordering service is now deprecated since Fabric v2.0, the Raft ordering service should be used. Fundamentally, the ordering algorithm has not changed, thus resulting in no theoretical impact on the performance of the network itself.

4 PERFORMANCE ANALYSIS

The purpose of this work is to get a definitive evaluation of how the scaling of different network participants influences the performance of the Fabric Blockchain network. In particular, the scaling of the number of peers, organizations, orderers and block size has been considered, since the number of each component is configurable. A custom tool [2], purposely made for Fabric 2.0, has been developed to automatically generate the required business network containing the desired number of individual network components. It also installs the test smart contract automatically. Using this tool, a broad spectrum of custom made network configurations can be generated in an instant.

After the setup is complete, the network can be used and tested. By utilizing the generated `invoke` and `query` scripts, the network can be directly addressed and interacted with. To measure the different impacts on the performance metrics, four different incoming transaction rates, ranging from one transaction per second (tps) to 1000 tps, have been considered. The incoming transaction rate will be called TPS_{in} in the following and the transaction rate of the network itself will be called TPS_{out}. The transactions are sent in parallel to generate a real-world application scenario where multiple nodes issue different transactions at the same time. All transactions will call the `createDefaultAsset(String value)` method of the installed smart contract and pass a random value as an argument, in order to append new data. All transaction types will be *invoke* since only invoke calls will trigger the ordering and consensus process within the network, thus generating the required load. For every transaction rate, each external client will issue 100 transactions. The network will then validate the transactions and append new blocks to the Blockchain, thereby generating the performance metrics.

The performance metrics cannot be measured directly and have to be derived by measuring the execution time of the transactions, the commit times of the individual blocks and the error rate. The following formulas can be used to get the performance metrics from these measurements:

t_{tx_i} : $i-th\ Transaction\ time$ $\quad$ F : $Failed\ Transactions$

t_{b_i} : $i-th\ Block\ commit\ time$ $\quad$ N : $No.\ Transactions$

$$Throughput = \frac{1s}{\frac{\sum_i^N tx\_time_i}{N}} [TPS] \qquad Error\ rate = \frac{F}{N} \cdot 100\%$$

$$Latency_i = t_{tx\_i} + t_{b\_i}\ [ms]$$

Said measurements are directly reported for each individual transaction and logged by every peer that executed the smart contract for the particular transaction. After the specific test is complete, these measurements can be extracted from the logs and be used to derive the performance metrics afterward.

In order to scale a Fabric network to multiple numbers of peers, orderers and organizations, enough hardware resources are required for the virtualization. For this work an Ubuntu 16.04 system with two Intel Xeon E5-2690 v3 CPUs, 128GB 2133MHz RAM and Docker 19.03.8 was used. By using the officially provided Fabric 2.0 Docker Containers, the evaluation can be directly compared to other studies, since Docker allows for a platform- and hardware independent compatibility.

Multiple Fabric network configurations have been considered. The goal of the performance evaluation is to get a measurement of the impact when scaling the peers, orderers and organizations. Additionally, the impact of varying block sizes has been tested as well. The basic network configuration consists of one single organization containing two endorsement peers and one single orderer with a block size of ten transactions per block. The configurations follow an abbreviation scheme: `<organizations>-<peers` ↪ `>-<orderers>-<kafka_nodes>`. The individual test schemes for each transaction rate also make use of the following notation: `bs<blocksize>-<tps>tps`. When considering the basic network configuration and testing a transaction rate of 100 tps, the notation will be: `1-2-1-1, bs10-100tps`. The evaluation considers the scaling of each component individually. The following Table 1 gives an overview of all network configurations that have been tested. The maximum number of 16 peers per organization was provided by Nguyen in [7].

Table 1: Test network configurations

Organizations	Peers	Orderer/Kafka nodes
1	2,4,8,16	1
2	2,4,8,16	1
4	4,8,16	1
8	8	1
2	2	2,4,8
4	4	2,4,8

Each test network configuration has been evaluated against the previously described transaction rates, which are summarized in Table 2. The following section will describe the evaluation results in detail.

5 EVALUATION

To derive a complete evaluation of the different performance impacts of the various network components, each of them will get analyzed in detail. Generally, all test scenarios described in Tables 1 and 2 have been executed under the same conditions thus allowing a mutual comparison.

Table 2: Test cases

TPS_{in}	Ext. Clients	Chaincode	Test Runs
1 TPS	1	Default	100
10 TPS	10	Default	100
100 TPS	50	Default	100
1000 TPS	100	Default	100

1. Impact of the peer and organization count

In these particular test scenarios, the number of peer nodes shows how many peers will execute a given smart contract because all of the configured peers are assigned to be endorsement peers. This decision is crucial for a fair comparison because unlike the endorsement peers, the committing peers would not create an equal amount of load. For each test scenario, the amounts of orderers, kafka nodes and organizations were fixed to a value of one, creating a baseline test environment for the scaling of the peer nodes. Those have been scaled beginning with only two peers up to 16 peers per organization. Figure 2 shows the execution times of the different

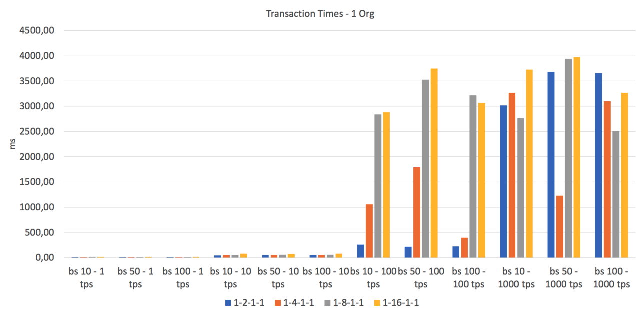

Figure 2: Transaction times at one organization and scaling peer counts

transactions graphically. Notice that the x-Axis also includes the scaling of the block size and tps. In particular, the scaling of the peers can be best seen within batches of four bars beginning with the blue bar on the left and ending with the yellow bar on the right. Each bar shows which effects and impacts a doubling of the peer count has on the execution times. By increasing the number of peer nodes within a given organization, the execution time of the transaction increases significantly. With higher TPS_{in} rates above 100 tps, the difference between the lowest configuration containing two peers and the highest configurations with 16 peers is as large as 3535ms in the scenario bs50-100tps. This behavior is expected since the introduction of more peers leads to a higher communication und synchronization overhead within the network. When analyzing the same scenario with two organizations, the overall tendency in regards to the performance impact continues. Figure 3 shows the same evaluation as in figure 2, but now includes two organizations. All the peers are equally distributed among all organizations. In the particular scenario for 16 peers, each organization therefore

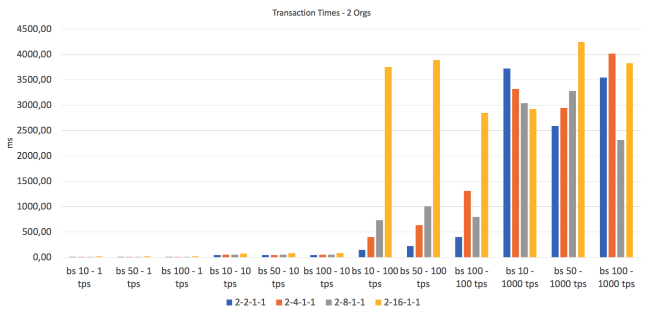

Figure 3: Transaction times at two organizations and scaling peer counts

possesses eight individual peer nodes. The results depicted in figure 3 also show that the performance decreases rapidly with an increasing number of peers within the network. When comparing the execution times for one and two organizations with each other, the configuration with two organizations produces lower transaction times on average. Though, this particular trend is not continued when considering the same configuration with four organizations. The performance impacts though are also present with four organizations. The results are generally more similar to the results of figure 2, thus hinting towards an optimization of transaction times for two organizations. This trend has also been discovered by Nasir et. al [6] for previous Fabric versions. Generally, the transaction times for two organizations are significantly lower when compared to test scenarios with other organization counts.

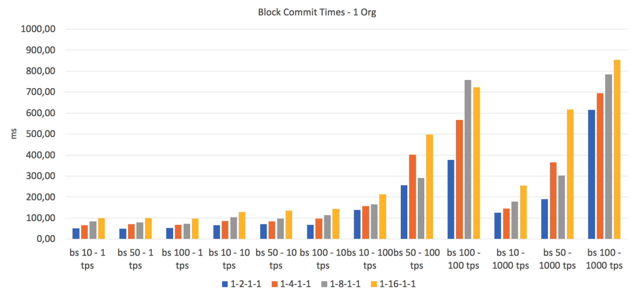

Figure 4: Block commit times at one organization and scaling peer counts

The scaling of the peers generally has a negative impact on the performance as the transaction times increase steadily. When considering the block commit times for different peer counts, said negative impacts are also present. Figure 4 shows the particular block commit times for each test case containing one organization. A steady increase within each batch of four bars can be seen, thus indicating that the scaling of peer nodes also impacts the block commit times.

Regarding the error rate, no particular trend can be extracted other than a generally rising error rate with higher TPS_{in} numbers. Figure 5 depicts the particular error rates for each test case containing one organization. Though, when focussing on one bar

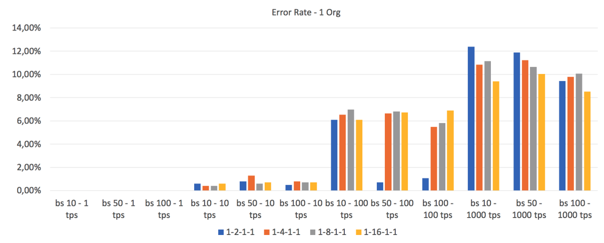

Figure 5: Error rates at one organization and scaling peer counts

color only, e.g. the blue bars for two peers, a bigger block size has a positive effect on the error rate on average. This hints towards a larger block size causing fewer errors at runtime.

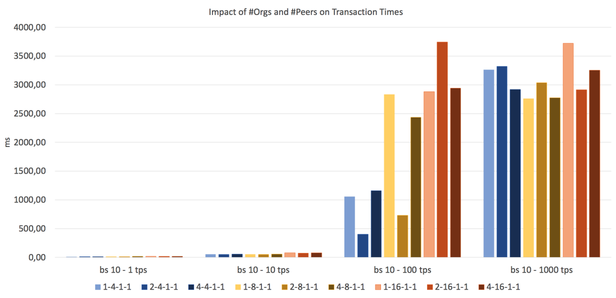

Figure 6: Transaction rates with scaling peers and organizations with block size 10

Figure 6 shows a concrete comparison between the different transaction times for an increasing number of organizations. The previously mentioned optimization for two organizations can be seen again. The shades of different colors such as blue symbolize an increase in organization count. The different colors show varying peer counts. This can also be seen when comparing the different block commit times of each organization count with each other. In regards to the error rate, no particular scheme or trend can be extracted.

2. Impact of the orderer count and block size

The ordering service is primarily responsible for generating an ordered set of transactions, thus reaching consensus within the network. Since the orderers will not execute any transactions issued from external clients, it is to be expected that the scaling of orderer nodes will hardly impact the transaction times. Though, the block commit times should be influenced by the number of orderers, since the blocks get validated by the orderers. With a scaling of the ordering nodes, more synchronization among the individual nodes is required, thus creating an impact on the performance.

The results for the transaction rates in figure 7 do not show a generalizable result such as the scaling of peers. There is no definitive in- or decrease in performance apart from a few single cases, that show an increase in transaction times (s. figure 7 bs10-1000tps).

Also, when graphically visualizing the results of the block commit times for different amounts of ordering nodes, no clear impact

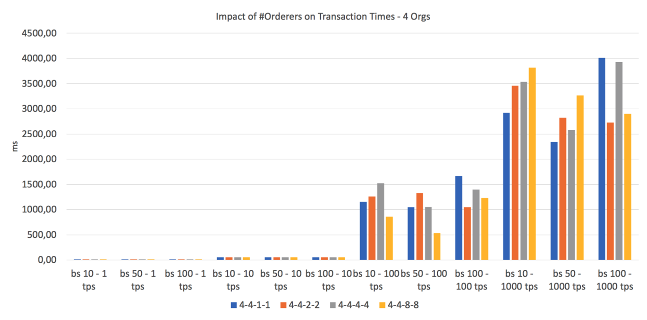

Figure 7: Transaction times with scaling orderer nodes with four organizations

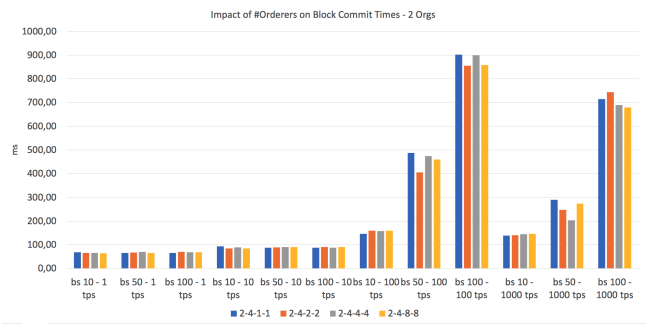

Figure 8: Block commit times with scaling orderer nodes with two organizations

is derivable (s. figure 8). Also the error rate does not show any clear notable impacts when scaling the orderer nodes.

The previous figures for the block commit times also hint towards a performance impact which is caused by the block size. A higher block size will lead to a higher block commit time. This is due to the block including more transactions and thus requiring more validation time. So the block size stands in direct correlation with the overall latency of the system.

3. Performance metrics

By applying the formulas of section 4, the impact of the peers in conjunction with a high TPS_{in} rate gets emphasized even more. Figure 9 shows the throughput TPS_{out} in dependency of the TPS_{in} rate. The throughput is highly dependent on the number of peers, as can be seen from the transaction times. With more endorsement peer nodes being active within the network, less throughput is to be expected. On average the throughput reached 22,071 tps among all test cases, including those with high TPS_{in} rates of over 100 tps.

The latency is further dependent on the block size, as well as on the number of endorsement peer nodes. Smaller block sizes generally allow a significantly lower latency. Though, it is to be noted that the block size should be selected for the expected TPS_{in} rate, as a fast filling of the block may result in a frequent validation process thus causing additional latency.

The previous subsection already discussed the error rate for each particular component. Generally, only the block size and the TPS_{in} rate are highly influential to the error rate. The number of peers,

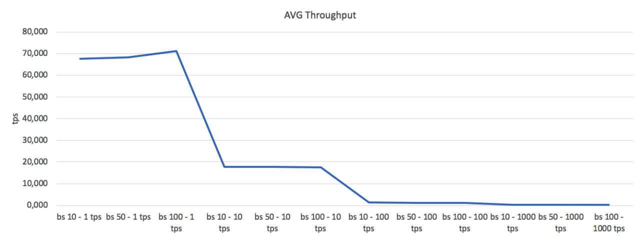

Figure 9: Throughput of different configurations

orderers or organizations does influence the error rate only within the margin of error. The results show, that a larger block size will result in less errors than a smaller block size.

Nasir et al. showed in [6] a performance comparison between Fabric v0.6 and v1.0. Those results can be directly compared with the transaction times from this work. Thus, figure 10 extends the figure 8 of [6] with the measurements for Fabric v2.0. The comparison shows, that Fabric v2.0 improved the overall performance of the system even more.

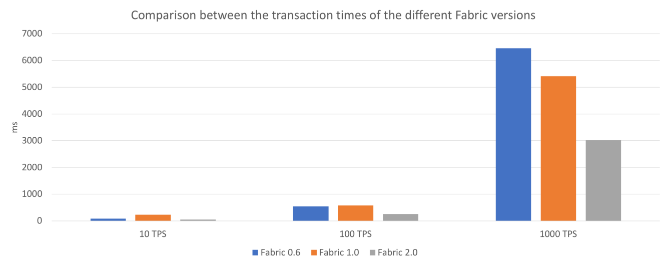

Figure 10: Transaction times of Fabric v0.6, v1.0 and v2.0

6 CONCLUSION

The performance of Hyperledger Fabric has been further improved with the release of version 2.0. The conducted performance study in this work showed, that the internal improvements to v2.0 of Fabric lead to an increase in the overall throughput of the network. The measured performance factors such as latency and error rate also experienced an improvement in comparison to previous versions of Fabric. With the new lifecycle management of Fabric 2.0, the validation of transactions is faster than before. Also, the probability of errors occurring during transaction validation is also lowered.

This work also showed that by scaling the different components within the network such as peers, orderers and organizations, the performance varies by a large factor. It is not trivial to decide which configuration is fitting for a given requirement in regards to throughput expectations.

Future work may focus on examining the real-world performance of Fabric e.g. in an Industry 4.0 appliance. A guideline for an ideal configuration of the network might be provided. This work did not include the performance of *query* calls, which are generally faster, as they do not require any changes to the ledger. Thus future work may focus on evaluating the overall query performance of Hyperledger Fabric v2.x or above.

REFERENCES

[1] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014).

[2] Julian Dreyer. 2020. Fabric 2.0 Configurator. https://github.com/JulianD267/Hyperledger-Fabric2-0-configurator Access: 12.10.2020.

[3] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. 2019. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. 455–463. https://doi.org/10.1109/BLOC.2019.8751452

[4] Murat Kuzlu, Manisa Pipattanasomporn, Levent Gurses, and Saifur Rahman. 2019. Performance Analysis of a Hyperledger Fabric Blockchain Framework: Throughput, Latency and Scalability. In *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 536–540.

[5] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008).

[6] Qassim Nasir, Ilham A Qasse, Manar Abu Talib, and Ali Bou Nassif. 2018. Performance analysis of hyperledger fabric platforms. *Security and Communication Networks* 2018 (2018).

[7] Minh Quang Nguyen, Dumitrel Loghin, and Tien Tuan Anh Dinh. 2019. Understanding the scalability of Hyperledger Fabric. *BCDL VLDB* (2019).

[8] Suporn Pongnumkul, Chaiyaphum Siripanpornchana, and Suttipong Thajchayapong. 2017. Performance analysis of private blockchain platforms in varying workloads. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–6.

[9] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. 2018. Performance benchmarking and optimizing hyperledger fabric blockchain platform. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 264–276.

[10] The Linux Foundation. 2018. Hyperledger Caliper. https://www.hyperledger.org/blog/2018/03/19/measuring-blockchain-performance-with-hyperledger-caliper Access: 2.9.2020.

[11] The Linux Foundation. 2020. The Ordering Service. https://hyperledger-fabric.readthedocs.io/en/release-2.0/orderer/ordering_service.html Access: 2.9.2020.

[12] The Linux Foundation. 2020. Transaction Flow. https://hyperledger-fabric.readthedocs.io/en/release-2.0/txflow.html Access: 2.9.2020.

[13] The Linux Foundation. 2020. What's new in Hyperledger Fabric v2.0. https://hyperledger-fabric.readthedocs.io/en/release-2.0/whatsnew.html Zugriff: 9.6.2020.