Preliminary Performance Assessment of Hyperledger Fabric

Attila Klenik, András Pataricza

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: {klenik, pataric}@mit.bme.hu

Abstract—**After the success of bitcoin proved the viability of the distributed ledger technology, other frameworks emerged with the goal of providing a general purpose blockchain platform for businesses to execute smart contracts. The private and permissioned platforms are promising replacements for many current systems in several sectors, such as finance, healthcare, and IoT.**

Such timeliness and throughput critical applications require a framework offering predictable temporal characteristics. Benchmarking is a traditional method for measuring (and later predicting) the performance related extra-functional characteristics of a system. However, blockchain frameworks currently lack standard benchmarks due to the novelty of the domain.

This paper present a preliminary performance assessment of a private and permissioned blockchain framework, Hyperledger Fabric. The evaluation is based on a synthetic load and targets the scalability and performance of different functional parts of the framework architecture. Findings are presented for the different phases of the lifecycle of transactions in the framework.

I. Introduction

Motivated by the success of Bitcoin [1], the interest for blockchain technologies grew rapidly in the past years. The irrefutable way of committing and auditing transactions without a third trusted party makes blockchain a promising new technology for numerous sectors, e.g., finance,[1] business workflows [2], healthcare, the government sector,[2] Internet of Things [3], and Cyber-Physical Systems.

The use cases of distributed, "trustless," Bitcoin-inspired peer-to-peer systems are wider than managing cryptocurrencies. This inspiration gave rise to a large number of continuously evolving, experimental systems. These all modify the "Bitcoin architecture" (and protocol) in fundamental ways, but two aspects do not vary: a) there is a blockchain and peers maintain a copy of it and b) peers participate in a distributed consensus protocol to maintain a consistent state of the system.

The complexity of blockchain systems makes it challenging to ensure certain requirements. Such requirements include temporal properties, like throughput and timeliness; dependability properties, like availability; and security aspects, like confidentiality and privacy. Employing model driven development (MDD [4]) allows the application of model-based simulations and (formal) analyses to facilitate ensuring these requirements. A subset of these models captures performance-related behaviours enabling the performance prediction of complex systems [5]. Model-based performance analysis supports the identification of bottlenecks and execution of sensitivity analysis in the early stages of development, both on platform independent models (e.g., functional architecture) and platform dependent models (e.g., the configuration of system components).

This paper presents a preliminary performance assessment of a pilot reference implementation of a general purpose blockchain platform, Hyperledger Fabric,[3] through a systematic methodology.

II. Architecture Elements

In order to discuss the evaluation methodology and the findings, it is important to first introduce the key concepts of Hyperledger Fabric. The blockchain network consists of a set of *nodes*. Each of them maintains its own replica of the same tamper-proof key-value store – called the ledger – through constant communication. Note, that the concept of nodes above is used in the functional sense without constraining the deployment to physical servers. Nodes can take on two main roles in the system:

- *Peer nodes* store and maintain their own local copy of the ledger. If this is their only responsibility, then these nodes are referred to as *committer* peers. However, the users of the network can deploy executable "smart contracts" – chaincodes in Hyperledger Fabric terminology – to some of the peer nodes.
  In this case these nodes also participate in validating the transaction executions, i.e., the chaincode invocation attempts of the user. This process is called endorsement (detailed in Section III), consequently, these nodes are referred to as *endorsing* peers.
- *Orderer nodes* deliver transactions to the committer peers. The ordering service guarantees atomic delivery of messages (also known as total-order broadcast, or consensus in distributed system theory) and some degree of reliability, depending on the actual implementation of

[1]http://www.reuters.com/article/banking-blockchain-bonds-idUSL8N16A30H

[2]https://www.gov.uk/government/publications/distributed-ledger-technology-blackett-review

[3]https://www.hyperledger.org/projects/fabric

the service. Due to efficiency reasons, the service implementations usually broadcast the ordered transactions in batches.

The previous versions of Hyperledger Fabric (v0.x) had serious limitations regarding scalability that manifested in a relatively low effective throughput: based on previous measurements [6], around 300 transactions per second. The issue originated from the architectural design that first ordered (in batches) then executed the transactions in a sequential manner. The sequential execution of chaincode invocations proved to be a bottleneck in the system even for simple chaincode implementations.

The new architecture of Hyperledger Fabric v1.0 was subject to a fundamental redesign in order to alleviate this problem. The transaction execution phase now precedes the ordering and is performed in parallel. Moreover there is an additional validation step before committing the ledger modifications of the transactions. The life-cycle of a transaction is detailed in the next section.

III. Transaction Life-cycle

The redesigned architecture resulted in a more complicated transaction life-cycle than in the previous versions. A transaction has to go through numerous steps before its effect is reflected in the ledger. These steps are the cornerstones of the current evaluation, so their detailed understanding is a must. The phases of transaction committing are the following:

1) The client initiates a transaction proposal towards a subset of endorsing peers. This step is usually referred to as sending a transaction for endorsement (detailed in the next step). The transaction is considered endorsed if and only if a ”sufficient” number of peers endorse it according to some endorsement policy (e.g., at least 2 out of 3 participants).
2) The endorsing peers (that received a proposal) execute the chaincode with the parameters of the transaction. The chaincode is executed against the current state of the ledger (based on the latest committed transaction block) and every proposed transaction is executed in parallel. This is a significant difference compared to the previous architecture, where the transaction execution was strictly sequential.
   During the execution, the endorsing peer builds a read set and a write set. The read set contains the keys and their versions that were read from the ledger during the execution. The write set contains the keys and their values that were written to the ledger during the execution.
   It is important to note, that the written values are *not* actually committed to the ledger at this point, they are simply gathered by the endorsing peer. Accordingly, this step is also referred to as simulating the transaction.
3) The endorser peers send back a result to the client indicating whether the execution was successful or not, together with the produced read and write set.
4) The client inspects the received results and decides whether to proceed with the next step or not. If the endorsement policy for the chaincode is not satisfied by the received endorsement results, then the transaction will ultimately be rejected as invalid (detailed in step 9), so it is better not to submit it, but try again the proposal. Furthermore, if the produced read sets are different across endorsing peers, then it is recommended to retry the proposal step.
5) The client sends the gathered endorsements along with the read and write sets to the ordering service.
6) The ordering service acknowledges that the transaction was submitted for ordering.
7) The ordering service produces a transaction block from the received transactions, either based on a predefined timeout or on the maximum number of transaction permitted in the block.
8) The ordering service delivers the produced block to the committing peers in the network.
9) The committing peers validate the submitted transaction. At this point the transaction will be definitely included in the blockchain, irrespectively of the validation result. The validation includes checking whether the chaincode policy is satisfied by the submitted endorsements and whether the key versions in the read set match the current state of the ledger. The latter verification is also referred to as multiversion concurrency control (MVCC) in the database domain.
   This means, that the version of every key in the read set must match the version of those keys in the current state of the ledger. If meanwhile an other transaction has already updated a key from the read set, then the transaction will be marked invalid.
   The transaction is considered valid if and only if the endorsement policy is satisfied and the key versions in the read set match the key versions in the current state of the ledger. If this holds, then the write set is committed to the ledger. Note, that at this point, the transaction is not executed again, only its write set is written to the ledger.

The performance of these steps will be the main focus of the evaluation, detailed in Section VI.

IV. Measurement Details

This section presents the goals of the evaluation, the details of the measurement environment (i.e., the used network topology), deployment details and the business logic specific parts of the evaluation, i.e., the chaincode and the client.

A. Evaluation goals

The goal of the current evaluation is not to determine the effective throughput of the platform, but to inspect the scalability and relative performance of the presented transaction life-cycle steps. The main focus is on the chaincode execution times and on the different steps that process the read and write sets produced by the execution.

To this end the evaluation will focus on the response time changes of these steps in case of different ledger update sizes. The sizes of these ledger updates affect the chaincode execution time through the ledger access, and also affect the validation and commitment times of transactions due to sizes of the read and write sets.

B. The environment

The network consists of two participants (referred to as organizations from now on), each maintaining two endorsing peers (for availability reasons).

The ordering service consists of a single orderer node that orders the transactions in a first-come-first-served manner. In production environments it is highly recommended to use a more robust topology for the ordering service (e.g., PBFT [7]).

The entire network is deployed on a single virtual machine using Docker containers. The virtual machine had access to four physical CPU cores (each with 3.1 GHz clock frequency), 8 GB of RAM and was backed by a solid state drive.

C. The chaincode and the client

Due to the lack of standard blockchain platform benchmarks, the deployed chaincode only provides functionality for updating (reading then writing) entries in the ledger. The number and size of the updates are configurable at the time of deployment in order to be capable of approximating the ledger load of real-life chaincodes with different characteristics.

During deployment of the chaincode enough keys were inserted into the ledger at deploy time to provide conflict-free updates for two blocks of transaction, assuming that the block timeout of the ordering service is one second. This means that transactions updating the same entry are approximately two seconds apart, so the updates will be conflict free as long as the committing times of these transactions are under two seconds. As we will see, this is not always the case.

The client that generated the transactions is a general blockchain platform benchmark tool, called Caliper.[4]

V. Evaluation workflow

This section presents the steps of the evaluation and their results. The process followed an iterative workflow (Figure 1, [6]) that systematically guides the evaluation through a sequence of steps with well-defined tasks to perform. This preliminary evaluation only covers the first six steps.

A. Operating envelope

Defining an operating envelope consist of limiting the input space of the system to a subset, that will keep the system within the desired operational conditions throughout the evaluation. Since the goal is to observe the response time changes based on the ledger load, we choose a transaction load rate that minimizes the number of invalid transactions in the system.

Based on some short preliminary test runs, a 100 transactions per second was selected as constant load rate for the rest of the evaluation. Due to the cyclic behaviour of the chaincode, the load duration time was selected to be 2 minutes for each measurement. This means that for each measurement approximately 12000 transactions were submitted and the timing of their steps recorded.

B. Rough functional architecture

The components of the system covered by the measurements correspond to the steps of the transaction life-cycle. Unfortunately, not every step can be measured on its own, at least not without performing additional instrumentation on Hyperledger Fabric.

The platform only provides an event about the commit result of a transaction. This means that the client can only observe the execution times after the endorsement, the orderer acknowledgement, and the commit event. These times will also contain the network latency, but since the network is deployed on a single virtual machine, this time should be negligible. The observed attributes are detailed Section VI

C. Instrumentation and harness

Caliper provides timing data for different steps of the executions that can be processed in an arbitrary manner. The chaincode execution time is logged in the running chaincode and retrieved from the Docker container logs after the measurement is over.

D. Experiment campaigns

The main parameters of the measurements are the number and size of the updates a chaincode invocation will perform. The explored parameter space consists of the combination of the values of the following two variables:

- Number of updates: 1, 2, 4, 8
- Size of updates (in bytes): 8, 32, 128, 512, 2048

These parameters currently have a technical limit, since the underlying remote procedure call library (gRPC[5]) has a maximum message size configured to approximately 4MB. This limits the number and/or size of entries created in the ledger during deploy time, since deployment also produces a read and write set that is sent back to the client.

VI. Exploratory Data Analysis

This section describes the attributes of the collected data and presents preliminary results using exploratory data analysis (EDA). EDA offers a wide variety of tools and methods to intuitively assess larger datasets without detailed knowledge of the measured system. During the measurements the following attributes were collected about each transaction:

- *create time*: The CPU time when the transaction was created, measured with millisecond precision. Unique identifier for transactions.
- *endorse time*: The time it took to send the transaction proposal to the endorsing peers, execute the transaction and receive the endorsements from the peers. Measured with millisecond precision.

[4]https://github.com/Huawei-OSG/caliper

[5]https://grpc.io/

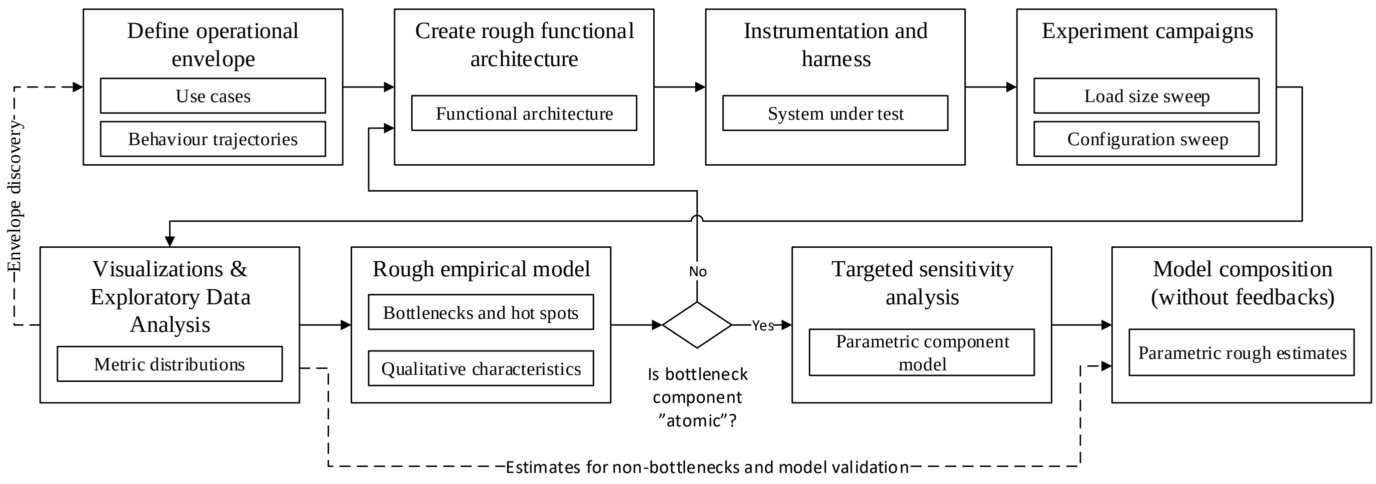
Define operational envelope
Use cases
Behaviour trajectories
Create rough functional architecture
Functional architecture
Instrumentation and harness
System under test
Experiment campaigns
Load size sweep
Configuration sweep
Envelope discovery
Visualizations & Exploratory Data Analysis
Metric distributions
Rough empirical model
Bottlenecks and hot spots
Qualitative characteristics
No
Yes
Is bottleneck component "atomic"?
Targeted sensitivity analysis
Parametric component model
Model composition (without feedbacks)
Parametric rough estimates
Estimates for non-bottlenecks and model validation

Figure 1: Evaluation methodology

- *order time*: The time it took to check the received endorsements, submit them to the ordering service and receive the acknowledgement from the ordering service. Measured with millisecond precision.
- *validation time*: The time it took to order the transaction and validate it at the committing peers and received the event about the result. Measured with millisecond precision.
- *org[1/2] execution time*: The time it took to execution the transaction on the current endorsing peer of the first and second organization. Measured with millisecond precision.

A. Assumptions

Based on a priori knowledge of the platform, the following assumptions were made, which guided the exploratory data analysis:

1) Due to the extra number of entries inserted in the ledger, there should be no conflict between transactions updating the same entries. This means that the transactions should be committed in a two block time range, which is approximately 2 seconds.
2) The execution times of transactions should be symmetric on both endorsing peer, since the same deterministic chaincode is running with the same parameters.
3) The following times should be sensitive for the size of the ledger load: execution time, commit time. Normally, the network communication time would also be in this list, but we omit it now due to the local network communication. Unfortunately the commit time contains the ordering time also, so the pure version validation time cannot be separated in the current evaluation.

B. Assumption checks

During the measurement with the biggest ledger load (8 updates, each 2048 bytes) there were almost 2500 failed transactions. Every other measurement committed every transaction successfully, except for a small transient time period, where the resource utilization of the host machine interfered with the resources of the virtual machine.

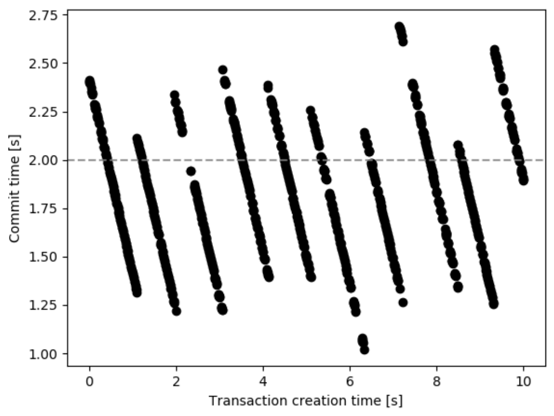

Figure 2: Total execution times of transaction

So the first assumption holds for almost every measurement, except one. Figure 2 shows the total commit time for the transactions initiated in the first 10 seconds of that measurement. The plot shows that numerous transactions had a commit time above 2 seconds. This results in a conflict for the next transaction that tries to update the same value. The second transaction still reads the old value (with the same key version), since the first transaction has not been committed yet. But by the time the second transactions is about to be committed, the first already overwrote that value, resulting in a version conflict.

The second assumption is a reasonable one, since the same code is executed on both endorsing peers in the same way. However, Figure 3 refutes this assumption. If the assumption was true, the observations would lie along the diagonal of the plot. This phenomenon could be explained by the resource constrained environment, but Figure 4 reveals another discrepancy.

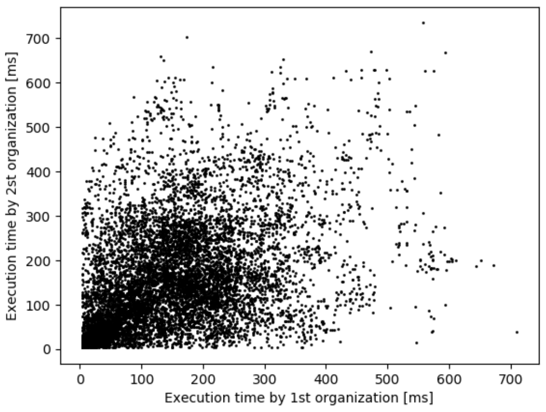

Figure 3: Execution times for the different organizations

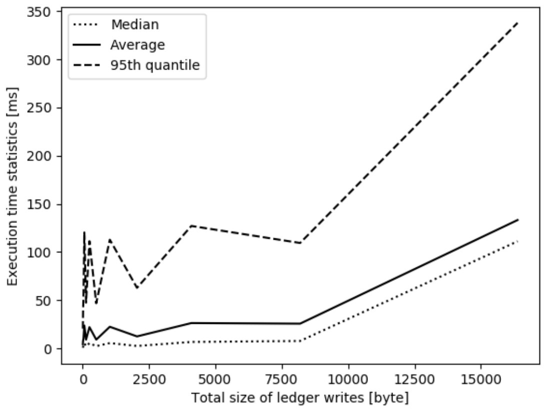

Figure 5: Execution times for the different ledger loads

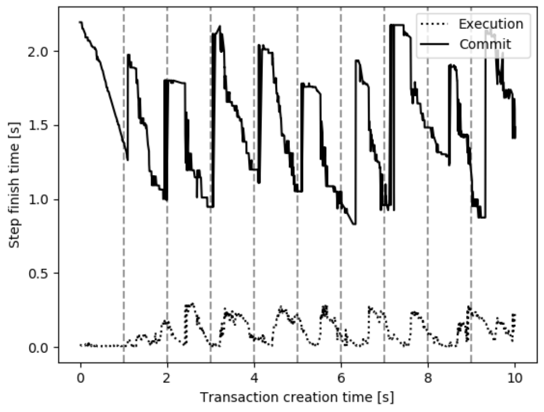

Figure 4: The pattern of execution and validation time of transactions

The vertical dashed lines are denoting the block timeouts, and the sawtooth-like plots approximately fit them. Note, that the transactions have higher execution times when being committed in the middle of a block, except for the first block. This phenomenon requires further, more detailed investigation, possibly through additional instrumentation of Hyperledger Fabric. For this reason, it is outside the scope of this paper.

The third assumption is checked by calculated the mean, median and 95th quantile of the execution times for the different ledger load sizes. This is illustrated in Figure 5. The plot clearly shows the sensitivity of the execution time for the size of ledger writes generated by the chaincode. The sensitivity should be investigated for larger ledger load sizes, but this requires some modification to the chaincode behaviour and Caliper timing data harnessing, consequently it is left as future work.

VII. Conclusion and Future Work

In this paper we presented the detailed working of the new architecture of Hyperledger Fabric, a pilot reference implementation of a general purpose blockchain platform. Then following a systematic methodology, we evaluated the scalability of the platform using different ledger loads. We concluded, that the sensitivity assumption between the ledger load size and the chaincode execution time holds. However, based on current data, it is hard to argue about the timeliness of the platform.

The presented assumption checks showed that the evaluation methodology is capable of delivering important knowledge about the platform. As future work, the evaluation must be repeated in a cloud environment, lifting the current resource constraints, which could be the source of significant noise in the data. Moreover, the ledger load size constraint should be circumvented, and measurements repeated for bigger parameter values. Furthermore, in order to extract network communication delays from the data, a more detailed monitoring or instrumentation is needed, preferably at the transaction lifecycle step level.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
[2] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, “Untrusted Business Process Monitoring and Execution Using Blockchain,” in *Business Process Management: 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, Marcello La Rosa, Peter Loos, and Oscar Pastor, Eds. Cham: Springer International Publishing, 2016, pp. 329–347.
[3] K. Christidis and M. Devetsikiotis, “Blockchains and Smart Contracts for the Internet of Things,” *IEEE Access*, vol. 4, no. 99, pp. 2292–2303, 2016.
[4] B. Selic, “The pragmatics of model-driven development,” vol. 20, no. 5, pp. 19–25.
[5] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, “Model-based performance prediction in software development: a survey,” vol. 30, no. 5, pp. 295–310.
[6] I. Kocsis, A. Klenik, A. Pataricza, M. Telek, F. Deé, and D. Cseh, “Systematic performance evaluation using component-in-the-loop approach,” *International Journal of Cloud Computing*, submitted.
[7] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI ’99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186.