Hyperledger Fabric Blockchain: Chaincode Performance Analysis

Hyperledger Fabric Blockchain: Chaincode Performance Analysis

Luca Foschini, Andrea Gavagna, Giuseppe Martuscelli, Rebecca Montanari

Dipartimento di Informatica – Scienza e Ingegneria, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
{luca.foschini, giuseppe.martuscelli, rebecca.montanari}@unibo.it, andrea.gavagna@studio.unibo.it

Abstract — **Hyperledger Fabric, created and supported by the Linux Foundation and IBM, is one of the most popular open-source blockchain permissioned platforms that has been already used in many industrial scenarios. One of the main characteristics of this platform is that it provides a smart contract system that relies on general-purpose languages instead of an ad hoc one. In fact, a chaincode in the Fabric platform (the equivalent of the Ethereum smart contract) is a software program which encapsulates the business logic for the creation and modification of logical assets in the ledger that can be written in different general-purpose programming languages (currently Java, Go, and Node.js). This paper analyses the transaction performance of the Fabric platform by identifying at a fine-grained degree level the factors that most contribute to the overall overhead. In particular, we focus on how the transaction latency is affected by the programming language adopted for implementing the chaincode and by varying the number of participating endorser peers. Finally, the paper shows a thorough test assessment aimed at evaluating the impact of the different chaincode implementation on performance overhead. As it emerges from our experimental results, Go is the most performing programming language.**

Keywords—blockchain, hyperledger, chaincode, fabric

I. INTRODUCTION

The last few years have been characterized by a significant increase in interest and use of Distributed Ledger Technology (DLT) of which blockchain is the best-known realization both for the number of applications and for the variety of scenarios. The introduction of blockchain technology has opened the doors to new types of applications that allow the sharing and management of data between untrusted organizations and entities in a safe and collaborative manner. Blockchain grants high degrees of non-repudiability, integrity, immutability, and censorship resistance. In particular, it allows untrusted parties to send transactions using a peer-to-peer network without necessarily having an intermediary to guarantee their correctness.

There are several blockchain platforms available nowadays, following different models and visions in terms of application fields. Hyperledger Fabric, created and supported by the Linux Foundation and IBM, is one of the most popular open-source blockchain permissioned platforms, i.e., a blockchain network in which participants need the approval to be part of it, already used in many industrial scenarios. The adoption of a permissioned blockchain is highly suited for enterprises that require authenticated users. Further, enterprise applications need complex data models that can be supported using smart contracts.

Despite the claimed benefits provided by the blockchain technology and by available platforms, there are still several issues to investigate for blockchain to take off as widespread large-scale technology. In particular, there is a strong need for a performance vademecum and established metrics that allow us to clearly evaluate the performance overhead introduced by the deployment of a blockchain platform. Some works have been proposed that analyze the unique performance attributes of blockchains, whereas measuring and comparing performance between different blockchains is still difficult. Existing works have so far been mainly focused on evaluating the Hyperledger Fabric platform, analyzing many aspects that impact on the Hyperledger Fabric performance. Some seminal efforts propose some architectural solutions [9, 11]. With our work, we focus on additional, still not investigated, factors in the Hyperledger Fabric platform that contribute to the overall latency, such as the choice of the programming language used for implementing the chaincode operations.

The contribution of this paper is threefold. First, we analyzed with a fine-grained degree of detail the blockchain consensus algorithm flow of Hyperledger Fabric both for queries and invocation requests isolating the main interval time. Second, we identify the factors which mostly impact the overall latency, such as the programming language exploited for chaincode implementation in relation to the number of network peers. Third, we describe several experimental results we obtained to assess the performance overhead of the different programming languages. The results show that the Go is the most performing one for almost all the tests realized and that in case of update of the ledger the latency follows a linear trend as the number of nodes in the network increase, while in case of query the latency is approximately constant.

The remainder of the paper is divided as follows. Section II provides the background of the blockchain technology and the related works. In Section III, we introduce main performance metrics for our experiments, and in Section IV we use them for performance assessment. Section V draws conclusions and future work.

II. BACKGROUND AND RELATED WORKS

A. Background

Blockchain is a distributed ledger composed of a chain of interconnected blocks containing tamper-proof information. This technology was originally described in 1991 by a group of researchers and was intended to apply a time stamp impossible

to counterfeit digital documents. However, it was not very widespread until Satoshi Nakamoto in 2009 created the Bitcoin cryptocurrency [1]. The blockchain ensures the integrity of the stored blocks using intensively cryptographic techniques, such as hash and digital signature making it secure and non-counterfeit.

The blockchain networks can be divided into two different access models: permissionless and permissioned or, as often referred to in literature, respectively public and private blockchains. In the first model, participation is public and open access: anybody can participate in the network and in the consensus process. In the second model, participation is permissioned: participants have either restriction on writing (validation) rights, or on both reading (access) and writing rights.

The most popular permissionless blockchains are certainly Bitcoin and Ethereum. Bitcoin is Proof-of-Work based blockchain network, giving open access to its transaction logs, whereas Ethereum is an open platform designed to build and use decentralized applications that run smart contracts which are applications that mechanically execute tasks when certain conditions are met.

Within the permissioned blockchains, we can find many solutions, such as the Hyperledger project with Fabric which is mostly contributed by IBM and introduces an important feature that allows nodes to confidentially transact on the same network of peers and Sawtooth which is mostly contributed by Intel and it uses a Proof of Elapsed Time consensus to save energy [2]. Corda is an additional platform created by the software company R3 that leads a consortium of two hundred global financial institutions [3]. Other platforms include Chain Core that is mostly focused on issuing and transferring financial assets within a consortium and Quorum that is a permissioned implementation of Ethereum [4].

Assessing an exhaustive comparison of the different blockchains is very complex since the various platforms rely on different consensus algorithms, follow different data and architecture models where often the roles of nodes are specialized. In particular, in terms of performance comparison between platforms, the main difficulty is to find a way to fairly compare them given the fundamental differences touching to consensus, block structure, P2P behaviors, etc [5].

B. Related Works

Some performance studies are available that focus mainly on the Hyperledger Fabric platform and its performances as explained in the following.

The work in [6] focuses, for example, on the impact on the transaction throughput and latency of specific configuration parameters such as block size, endorsement policy, channels, resource allocation. For example, the work analyses the performance to validate a transaction's endorsement signature (VSCC) varying the endorsement policy and the number of endorsers. Besides, it introduces some optimizations such as aggressive caching for endorsement policy verification and parallelizing endorsement policy verification.

[7] conducts a complete performance analysis of two versions of Hyperledger Fabric, v0.6 and v1.0. The evaluation of the two platforms is performed by varying the workload up to 10.000 transactions. Then it analyses the scalability of the two platforms by varying the number of nodes up to 20 nodes. The results show that the execution times increase as the number of transactions grows and that execution times, throughput and latency for Fabric v1.0 are better than the ones for Fabric v0.6. The results show also that the maximum number of nodes that Fabric v0.6 can have is 16.

Authors of [8] study the throughput and latency characteristics of Fabric by subjecting it to different sets of workloads. Through a suite of benchmarks, they tune different parameters transaction and chaincode parameters such as transaction per block and the time the order waits before creating the block (timeout). They also conduct experiments to study Fabric's performance characteristics while increasing the number of chaincodes, channels, and peers.

[9] re-architects Hyperledger Fabric to increase transaction throughput from 3,000 to 20,000 transactions per second. They focus on performance bottlenecks beyond the consensus mechanism such as the message pipeline, the world state database, and the transaction header and payload. They propose architectural changes that reduce computation and I/O overhead during the transaction.

In [10] a performance model of Hyperledger using Stochastic Reward Nets is presented. From the model, they compute the throughput, utilization and queue length at each peer and critical processing stages within a peer. From their analysis results, they find that time to complete the endorsement process is significantly affected by the number of peers and policies. The performance bottleneck of the ordering service and ledger write can be mitigated using a larger block size, albeit with an increase in latency. For the committing peer, the transaction validation check (using Validation System Chaincode) is a time-consuming step, but its performance impact can be easily mitigated since it can be parallelized.

[11] investigates whether the consensus process using Practical Byzantine Fault Tolerance (PBFT) could be a performance bottleneck for networks with a large number of peers. They model the PBFT consensus process using Stochastic Reward Nets to compute the meantime to complete consensus for networks up to 100 peers.

Our work differs from the above-described studies as it analyses Hyperledger Fabric considering a novel performance aspect, i.e., the impact of the programming language exploited for implementing chaincodes. In particular, the overall transaction latency of chaincode operations is measured by varying the number of nodes) and the programming language used for chaincode and client implementation. In our scenario, all nodes behave as endorser peers

III. Performance Metrics for Hyperledger Fabric

This section gives an analysis of the Hyperledger Fabric platform and defines the set of metrics that we used for comparing the latency of the transaction on Hyperledger Fabric platform [12] and the factors which can affect them: the programming language used for the client and the chaincode

implementation and the number of network peers (endorser peers).

A. Analysis of the peers' role and the consensus algorithm

The Hyperledger Fabric ledger consists of two distinct parts: world state and blockchain. The first is a database that maintains a cache of the current values of the attributes of an object represented by key-value pairs. The exploitation of the world state allows programs to directly access the value of an object without having to traverse the entire blockchain to calculate it. The second is the blockchain transaction log which stores all the changes that led to the current value in the world state collected in blocks hung one in the other to form a chain. In Hyperledger Fabric, each peer keeps a copy of the ledger (world state + blockchain) and the update of the copy is carried out by the peers individually through the consensus algorithm. It ensures that every peer will do the same update and that they will, therefore, have identical copies of the ledger.

Hyperledger Fabric network consists of a set of peers which can assume three distinct roles (see also Fig. 1):

- *Endorser* which receives and executes transactions (transaction proposal) coming from client applications. It is the only type of peer on which a chaincode must be installed and is, therefore, the only one that performs it. They execute the request and reply sending back to the client an endorsed result (endorsed transaction proposal)
- *Orderer* is the peer that deals with creating transaction blocks. It receives endorsed transaction proposals and inserts them in a block together with others in an orderly manner.
- *Committer* which checks the validity of all transactions individually contained in the received block and applies the block to the ledger. All peers take on this role.

The consensus algorithm plays a fundamental role in blockchain technology, as it is the mechanism that allows peers to participate in the network to maintain a consistent version of the ledger. It has a direct impact on blockchain performance. The transactions flow (Figure 1), which is used by the Hyperledger Fabric consensus algorithm can be summarized as follows:

1) The client application sends a transaction proposal to peer, endorsers, i.e. a request to invoke a chaincode method with certain input parameters, with the intent to read and/or update the ledger. Applications use the Fabric SDK to generate the transaction proposal: it packages the proposal in a format appropriate for the gRPC protocol and uses the user's cryptographic credentials to generate the transaction signature.

2) Peer endorsers receive the transaction proposal and verify that the signature is valid. Once the true value is exceeded, the method of the chaincode with respect to the current state of the world you are producing a read set (key-value pairs that have been read by the world state in the execution of the method) and a write set (key-value pairs representing the updating or creation of a new asset). No update is actually carried out on the ledger at this time. The RW set is digitally signed by the peer endorser who produced it and returned as a transaction proposal response to the client application.

3) The application verifies the signatures of endorser peers and, if the transaction proposal is a query, it immediately gets the result. In this case, the transaction is not sent to the orderer since we do not need to execute the consensus algorithm and the protocol ends. If instead the transaction proposal is not a query, the client application sends the transaction to the orderer in order to update the ledger by first determining if the endorsement policy is satisfied and if the answers obtained are identical. The endorsement policy defines the set of peers that must execute the chaincode and signs the result in order to consider valid the transaction and is specified for each chaincode. The application thus sends the transaction proposal response containing the read-write set and the signature of the endorser peer to the order service. The order service does not check the content of the transaction: it receives the transactions, orders them chronologically and creates blocks of ordered transactions.

4) The blocks are delivered from the ordering service to all peers on the channel. Transactions within the block are individually and sequentially validated. This process consists of three steps:

- The first step is called Validation System Chaincode (VSCC) and is carried out in parallel for all transactions in the block. It verifies that each transaction has been approved by the peers requested by the endorsement policy of the chaincode that generated the transaction and that all peer endorsers have generated the same result. The transactions that do not pass these first two checks are marked as invalid.
- For each valid transaction is made a check called Multi-Version Concurrency Control (MVCC) to ensure that the current ledger status is compatible with the status of the system when the transaction proposal (read-write set) has been generated. Indeed, another transaction may have updated the same asset in the ledger, making the transaction no longer valid. Each transaction is compared to the version of the keys in the read-set with that of the same keys in the peer's world state. In case they are not identical the transaction is marked as invalid. This second step is done sequentially for each transaction.
- Once the checks have been carried out on all the transactions individually, the peer adds the block to the blockchain and writes the contents of the write set to the world state for valid transactions. The effects of the transactions resulted invalid are not applied to the ledger, but they are in any case maintained in the same way as the successful transactions. This means that the blocks are the same as received by the orderer.

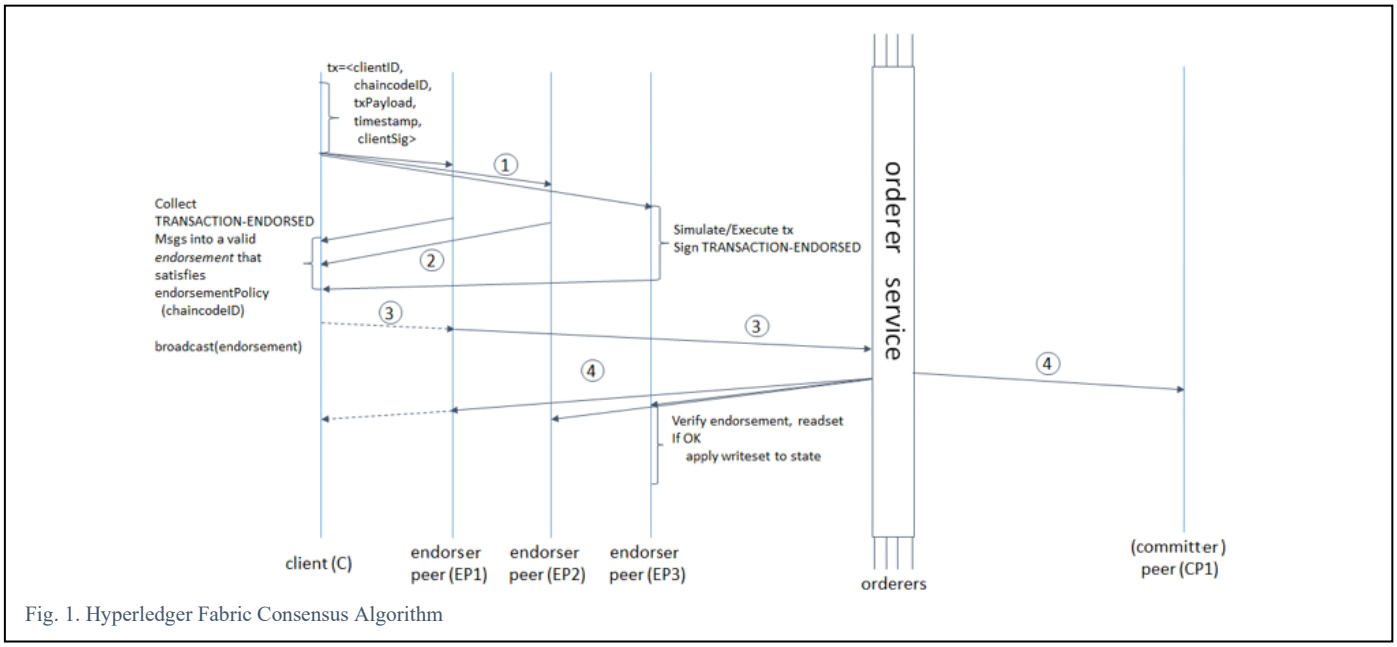

Fig. 1. Hyperledger Fabric Consensus Algorithm

B. Latency details

The transaction latency is the time between the sending of the request and the receiving of the response by the client. In Fabric, we distinguish between two different types of latency, depending on whether we are running a query or updating the ledger. In the first case, in fact, the only interaction between client and a peer occurs, while in the second case the interaction involves several parts (peer endorser, orderer, client application, committer peer) and multiple phases. For these reasons we provide two different definitions:

- Query Latency ***(LpQ)***: the time between when the request is sent, and the response is received by the client.
- Update Latency ***(LpT)***: the time between when the request is sent and when the client receives a confirmation event, which notices that the transaction has been entered in a block and added to the blockchain.

By analyzing the entire flow, we can express query and update latencies in terms of the following components:

$$\mathbf{LpQ} = T_{app} + T_{endorser} + T_{cc} + T_{concurrentTransaction} + T_{worldstate}$$

$$\mathbf{LpT} = T_{app} + T_{endorser} + T_{cc} + T_{orderer} + T_{peer} + T_{concurrentTransaction} + T_{worldstate}$$

where:

T_{app} is the time taken by the client application and depends on the underlying language SDK: Java, Go and Node.js, and on the client machine hardware.

$T_{endorser}$ represents the time needed for a transaction to reach enough endorser peers to satisfy the endorsement policy and to send the response back (endorser peers run the chaincode at run time). It is affected by the number of endorser peers in the blockchain network, the network latency and the hardware of the machines on which the endorsers are running.

T_{cc} is the overall chaincode execution time and depends on the endorsement policy, which defines how many peer endorsers must execute the transaction, the implementation detail, and the chaincode implementation language. In Hyperledger Fabric we can write chaincodes in Java, Go and Node.js.

$T_{orderer}$ is the time needed by the orderer service component to receive transactions and collect them in blocks. This factor depends on which kind of orderer service you use. The orderer service can be implemented in one mode (only one node involved) or Kafka (requires coordination between the various nodes that constitute it). Furthermore, the network latency also has an impact to reach the orderer and the hardware of the machine (or machines) on which it is running. Finally, the maximum block size and timeout parameters (maximum time to create the block) affect latency: the larger a block is or has a high timeout, the more the orderer waits to create the block and therefore the latency increases.

T_{peer} is the total amount of time required by peers to receive the block of transactions, validate all of them individually and append the block to its ledger. It depends on the number of peers in the network (which therefore must be reached by the block in phase 3 of the consensus algorithm), by the network latency and by the hardware available to the various peers, which must validate all the transactions contained in the block

$T_{concurrentTransaction}$ refers to the number of concurrent transactions that are executed: the higher this number is, the more endorsers and peers will be charged and, consequently, the average latency increases.

$T_{worldstate}$ is the time spent in read/write operation on the worldstate and depends on the database used. We can choose between LevelDB and CouchDB influencing the speed of access in reading and writing of information in the world state. For the sake of consistency, in our testing scenario, all peers use LevelDB as a world state.

C. Smart contract Programming Languages

Hyperledger Fabric provides a smart contract system with general-purpose languages instead of an ad hoc language. The chaincode, the equivalent of smart contracts in the Fabric platform, is a software program that encapsulates the business logic for the creation and modification of logical assets in the ledger. It can be written in different programming languages (currently Java, Go and Node.js) and is executed in a separate docker container that isolates it from the other chaincodes and from the other peer processes. For the implementation of the application using the Java language, two ways can be followed: the first involves the use of the SDK Hyperledger Fabric (java low level), the second involves the use of an additional SDK that provides APIs at a higher level of abstraction, called the SDK Fabric Gateway(java gateway). In the following tests, we performed both the Java SDK versions.

IV. Experimental Results

A. Application Use Case and System Deployment

To carry out the evaluation on the latency, we created an application that allows developers to maintain (create, delete, update, query) software projects based on the blockchain. Each software project is characterized by an identification code, a name, a description, an owner, a link to the source code repository and the hash of the code for integrity. The identification code is used as a key for storing it in the world state. Besides, each software project can be modified only by a user with a name that matches the value of the owner field. In order to carry out this control at the chaincode level, each transaction is signed by the sender and the X.509 certificate containing the corresponding public key is incapsulated to check the validity. The owner's name is then obtained extracting the CN (common name) field of the subject (subject) and compared with the value contained in the owner field of the project.

The testing scenario consists of nodes connected to each other in a local network, each peer runs on a different virtual machine (VM) and has the chaincode installed (peer endorser). The order service is implemented in one node only and is running on a single VM, the CA is also present in a separate docker container. The client is running on a separate VM which is not part of the blockchain network but belongs to the local network. The VMs are based on OpenStack-based cloud infrastructure consisting of 4 hosts and have 2GB RAM, 20GB hard disk, 2 CPUs. Each VM has Ubuntu 16.04 LTS as its operating system and uses version 1.4 of Hyperledger Fabric.

B. Performance Assessment

The tests are designed to measure the impact of the programming language used for the client application and for the chaincode and the number of peers on the transaction latency. For this purpose, the chaincode logic has been realized in all three languages supported by Fabric: Java, Node.js and Go. The tests are performed using the languages homogeneously on the client and chaincode and increasing the number of nodes of the network. In particular, tests have been carried out with 1, 2, 4, 6, 8, 10, 12, 14 and 16 nodes considering the worst case, that is when the execution of the transaction proposal is requested to all the peers of the blockchain network to meet the endorsement policy.

The evaluation was carried out both for the latency in writing (IV.C) and reading (IV.D) on the ledger. For each experiment performed a client executes 50 transactions in sequence measuring the time between the sending request and the receiving of the response. To reduce error factor, each experiment has been repeated 33 times and we show average values; we are not reporting standard deviations that are typically rather limited (always below 6% across all tests). In the case of queries, the response is the reception of the result of the operation, while in the case of updates it is the notification that the transaction has been added to a block and then hung on the blockchain. To perform a write, we executed an insertion of software projects while for reading we requested the software project details containing the word "blockchain" in the description.

B.1 Update Latency

Figure 2 shows the latency for a ledger update, in this case, the transactions modify the world state and they are then added to the blockchain replicated on the peers. We can see that the graph follows a linear trend. The Go language (blue) is the most performing for all the tests. It starts from about 100ms of difference with respect to the other languages in the case of a few nodes, growing up to almost 0.3 seconds in a network with 10 peers and 0.5 seconds with 16 peers. Node and Java Fabric SDK have similar performance while Java Gateway SDK turns out to be the worst due to the higher abstraction level.

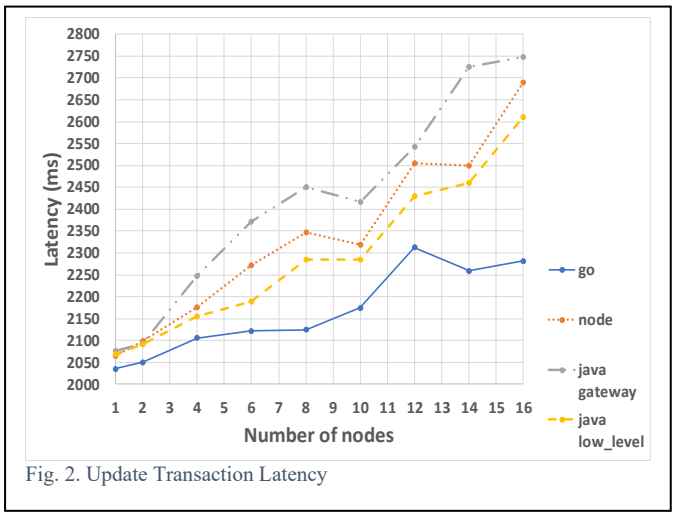

Fig. 2. Update Transaction Latency

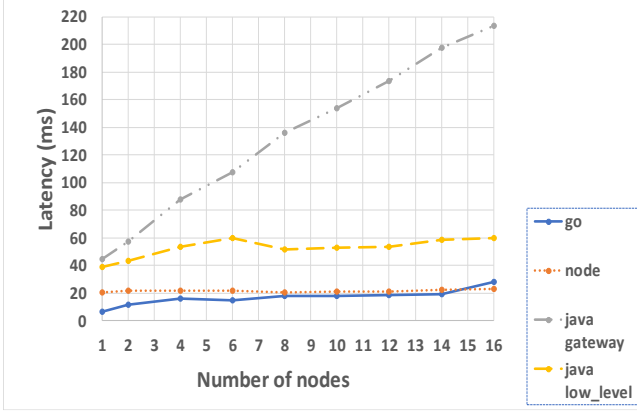

Fig. 3. Query Transaction Latency

B.2 Query Latency

Figure 3 shows the latency in the case of queries which involves only the first phases of the Hyperledger Fabric consensus algorithm without including the transactions in the ledger. In this case, the query is only sent to a single peer instead of having to wait for replies from all peers in the network. For these reasons, as we can see in the graph, the latency is noticeably lower than for an update of the ledger and is constant to vary the number of peers in the blockchain network. In particular, the trend is constant in three cases out of four: Go, Node.js and low-level java, while in the case of java gateway has a linear trend which starts from about 45 milliseconds for a single peer network to almost 220 in the case of 16 nodes. This is presumably associated with the fact that with the Gateway SDK the relative response is expected from all the peers. The Go language, also, in this case, is the more performing in all the tests together with Node.js. The java language, in the case of using SDK Fabric Java, has instead a latency that is about twice that of Go and Node.js with a latency ranging from 40 to 50 milliseconds.

V. Conclusion and Future Works

In this paper, we conducted a study to understand how the latency of Hyperledger Fabric vary, both in case of an update of the ledger and in case of query. In order to do that, first, we identified the main factors that influence the transaction flow. Then our study focused on the impact that the number of nodes in the network and the programming language used for the implementation of the client and the chaincode has on it. As a result of our study, we provided quantitative results that show that the programming language used has an important impact on the latency of the transactions and that Go is the most performing one for almost all the tests performed. Furthermore, we showed that in case of update the latency follows a linear trend as the number of nodes in the network increase, while in case of query it is (as expected) approximately constant.

As a part of future work, our study can be enriched studying the impact on performance due to other parameters, such as the number of concurrent transactions, using the order service in Kafka mode or using CouchDB as world state. In addition, in a real setup of the solution, nodes would be geographically distributed and consequently, the network delay would play a crucial role. Hence, it could be of interest to a study about the network impact on the overall latency of the transactions. Further, CouchDB could be centralized in order to have a shared centralized world state for different peers. This possibility would be very interesting especially in the case of nodes with limited storage capacity and should be deepen studied in terms of feasibility, security and in terms of impact on performance.

Acknowledgment

This research was funded and supported by the POR-FESR 2014-20 (no. E91F18000260009) through CIRI.

References

[1] Nakamoto, Satoshi, "Bitcoin: A Peer-to-Peer Electronic Cash System", 2009

[2] Hyperledger Sawtooth WhitePaper, https://www.hyperledger.org

[3] Corda Enterprise: a next-gen blockchain platform, r3.com

[4] Quorum the proven blockchain solution for business, goquorum.com

[5] M. Belotti, N. Božić, G. Pujolle and S. Secci, "A Vademecum on Blockchain Technologies: When, Which and How," in IEEE Communications Surveys & Tutorials

[6] P. Thakkar, S. Nathan and B. Viswanathan, "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform," 2018 IEEE MASCOTS, Milwaukee, WI, 2018

[7] Nasir, Q. & Qasse, Ilham & Talib, Manar & Nassif, Ali. (2018). Performance Analysis of Hyperledger Fabric Platforms. Security and Communication Networks. 2018

[8] A. Baliga, N. Solanki, S. Verekar, A. Pednekar, P. Kamat and S. Chatterjee, "Performance Characterization of Hyperledger Fabric," 2018 CVCBT, Zug, 2018

[9] Christian Gorenflo, Stephen Lee, Lukasz Golab, Srinivasan Keshav, "FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second" in IEEE ICBC, 2019

[10] H. Sukhwani, N. Wang, K. S. Trivedi and A. Rindos, "Performance Modeling of Hyperledger Fabric (Permissioned Blockchain Network)," 2018 IEEE NCA, Cambridge

[11] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi and A. Rindos, "Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Network (Hyperledger Fabric)," 2017 IEEE SRDS, Hong Kong, 2017

[12] Hyperledger Blockchain Performance Metrics Whitepaper, https://www.hyperledger.org