An experimental evaluation of the scalability of permissioned blockchains

Stefano Tavonatti, Davaadorj Battulga, Mozhdeh Farhadi, Carlo Caprini, Daniele Miorandi

An experimental evaluation of the scalability of permissioned blockchains

Stefano Tavonatti*, Davaadorj Battulga*†, Mozhdeh Farhadi*†, Carlo Caprini*, Daniele Miorandi*
*U-Hopper
v. da Sanseverino 95
38122 - Trento, Italy
name.surname@u-hopper.com
†Univ Rennes, Inria, CNRS, IRISA

***Abstract*—Permissioned blockchains are decentralized digital systems, which are used to record transactions and which maintain multiple, synchronized copies of the whole list of transactions (i.e., the ledger) on geographically dispersed nodes. Cryptographic operations are used to 'chain' transactions in the ledger, making the system tamper-resistant. In permissioned blockchains, access to the system (in particular in terms of the ability to append new transactions to the ledger) is limited to a specific set of well-identified nodes: this feature puts them apart from the blockchain systems ('permissionless') commonly use to power cryptocurrencies. The ability to control who can operate on the blockchain makes such systems a good choice for implementing use cases like supply chain management, business ecosystems or notarization. Driven by the interest in launching a new digital product for the education market (related to the management of education certificates), we faced some issues related to the scalability of permissioned blockchains. Given the lack of a consistent and comprehensive literature on the subject, we run an extensive experimental testing campaign on a large-scale distributed computing infrastructure (Grid'5000), measuring the performance of a popular permissioned blockchain framework (Hyperledger Fabric) under varying conditions. In this paper, we share the results obtained, which shed light on both scalability bottlenecks and possible approaches for overcoming such limitations in real-world business contexts.**

***Index Terms*—blockchain; measurements; performance; Hyperledger**

I. Introduction

Distributed Ledger Technologies (DLTs for short) are a class of digital storage technologies, characterised by two main features:

- Records are organised into a ledger, i.e., an append-only data structure whereby new data (called 'transactions' in the DLT lingo) are added to the tail of the list including all previous ones. Data is generally not removed from the ledger, but can only be added.
- The system maintains a number of replicas of the ledger, each replica being hosted by a different machine ('peer' or 'node'). Replicas are synchronised through a distributed consensus protocol, which ensures that replicas are eventually consistent (i.e., even if they run out of sync, they will nonetheless re-align after a reasonable time).

DLTs typically use a set of cryptographic primitives to provide tamper-resistance, i.e., to ensure that data written on the ledger cannot be manipulated by malicious nodes or external attackers. DLTs can be classified into two broad categories, depending on who has the right to create/validate new transactions to be added to the ledger. In *permissionless* DLTs, any node can join the network and create/validate new transactions to be added. These systems are radically open, with no controls on the system membership. Of course, being radically open, they require a 'strong' consensus protocol, in the form of a mechanism for ensuring that operations of the system are correct and do not go south in the presence of misbehaving or malicious nodes. Cryptocurrencies such as, e.g., Bitcoin or Ethereum, are powered by permissionless DLTs. In *permissioned* DLTs, as the name suggests, operations such as the addition of new records to the ledger and their validation can be carried out only by nodes that have been previously authorized to do so. These systems cater for use cases in which different actors (represented by different nodes) have partial trust in the other participants. In the case of permissioned DLTs, much lighter forms of consensus protocol may be employed, as there is an a priori selection applied to who can take part in the system operations. Typical applications include supply chain management, business ecosystems and notarization.

Among the broad family of DLTs, a special spot is taken by *blockchains*. Blockchains are a form of DLTs, in which (i) data is organised in blocks, representing a batch of transactions (ii) a new block is cryptographically linked to the previous one in the ledger, thereby 'chaining' the various blocks together. The resulting chain provides robustness against tampering, in that it makes hard to "change the past". Blockchains can be permissionless (such as, e.g., those powering Bitcoin or Ethereum) or permissioned (such as, e.g., that at the core of Hyperledger Fabric[1]).

One use case we are interested in, concerns the management of education certificates and credentials. We do strongly believe indeed that a decentralised system for storing digital versions of education certificates could provide a number of benefits to all the relevant stakeholders. For institutions providing certificates, it would cut the red tape and reduce the costs related to managing the certificates lifecycle. For recipients of certificates, it would put them back into full

[1] https://www.hyperledger.org/use/fabric

control of their certificates (at the moment it is the issuing institution that is in control, think, e.g., on how complicated it is to get a copy of your study certificate [2]. Third, it would allow prospective employers to be able to check in an automated and zero-cost fashion the veracity, or even existence, of an education certificate claimed by an applicant.

Our solution envisages the usage of a permissioned blockchain as the digital infrastructure for storing certificates. This would allow to maintain a control on who (education providers) can add/validate certificates, while putting certificate holders back in control and allowing for an easy verification of the veracity/existence of a given certificate. In the development process, however, we bumped into some issues related to the scalability of permissioned blockchains. An analysis of the state-of-the-art revealed that these issues were only partially covered by existing research literature and technology practise. We therefore designed a set of comprehensive tests for assessing the performance of a permissioned blockchain (in our case: Hyperledger Fabric) along a set of relevant scalability dimensions. We then executed said tests on a research infrastructure, namely Grid'5000[3], a large scale testbed specifically designed to evaluate distributed computing solutions at scale. The results highlighted a number of bottlenecks, and provided insights into potential remedies able to alleviate such scalability problems.

In this paper we describe the experiments devised, how they were executed on Grid'5000, and present the obtained results. The main contribution of our work lie in a comprehensive analysis of the scalability of permissioned blockchains. By adopting a rigorous experimentally-driven testing methodology, we were able to identify a number of scalability issues to be considered by practitioners and by developers of blockchain-based services. Our analysis led to the identification of a restricted number of key parameters and design choices, which can have a major impact on the system performance. To further allow other groups to build upon our results, we released the complete set of raw experimental data collected under an open data license [1]

The remainder of this paper is organised as follows. In Sec. II we revise the state of the art in terms of scalability analysis of permissioned blockchains. In Sec. III we introduce the model underpinning our system. In Sec. IV we describe the methodology used in our study; we first present the set of experiments we designed, and then describe the research infrastructure used for assessing the system scalability performance. Sec. V describes some implementation aspects related to both the system and the execution of tests. Experimental results are presented and discussed in Sec. VI. Sec. VII concludes the paper pointing out directions for improving the scalability performance.

[2]This, e.g., is the procedure to be followed by students from University of Padova, in Italy, to get a copy of their certificate - please note the usage of fax: https://www.unipd.it/en/certificates

[3]https://www.grid5000.fr/w/Grid5000:Home

II. Related work

The scalability of permissioned blockchain networks has been mostly studied as one aspect of the wider topic of blockchain performance.

In [2], the authors refer to the scalability as the number of peers that the blockchain network can handle and stay operational. They performed experiments on Hyperledger Fabric, v.1.4, and showed that the scalability of a blockchain network depends on the hardware configuration, blockchain network design and the complexity of the smart contract's operations.

The authors in [3] define the scalability as the number of peers in the network. They investigate the effect of the network connections bandwidth, geographic distance between the nodes, the communication protocol and the impact of hardware characteristics of the peers (i.e., CPU, RAM) on the scalability of Hyperledger Fabric v.1.1. The authors show that in certain deployment configurations, Fabric can scale to 100 peers and can achieve a throughput of 3,500 transactions per second with sub-second latency.

In [4], the scalability of two different versions of Hyperledger Fabric (v.0.6 and v.1.0) is compared. The authors measure the scalability of the two said platforms by varying the number of nodes (up to 20), while measuring the smart contract execution time, throughput, and latency and conclude that v.1.0 outperforms its predecessor version rather significantly.

Hyperledger provides Caliper[4], a benchmarking tool for evaluating the blockchain implementation's performance. The tool has been used in said articles ([2], [4]) to measure various performance parameters. This tool works with a handful of blockchain solutions such as Ethereum[5], Hyperledger Fabric and FISCO BCOS[6]. Caliper allows to define multiple clients who can inject workloads to the blockchain network. One can configure the test environment through the configuration files of Caliper and measure the parameters of interest. For instance, transaction numbers, transaction rates and type, network settings (i.e. number of peers, channels) are the parameters that one can configure and measure their effect on the performance (scalability, throughput and latency) of the blockchain network.

In [5], the authors introduce BLOCKBENCH, a tool for measuring performance of permissioned blockchains. By observing the effect of increasing the number of nodes and the number of custom workloads on throughput and latency, the authors measured scalability of the resulting system.

Although there are papers studying the scalability of the permissioned networks, they do not measure the consumption of other relevant resources, such as CPU and memory, while increasing the number of peers in the network and/or the rate at which transactions to be added to the ledger are generated. In this paper, we scrutinize the scalability of a blockchain network considering (1) the size of the ledger at the moment

[4]https://www.hyperledger.org/use/caliper

[5]https://ethereum.org/en/

[6]http://fisco-bcos.org/

an operation request is performed, (2) the request rate of the operations, and, (3) the number of nodes, following their resource usage.

III. System Model

Our solution builds upon a two-tier infrastructure. The upper tier exposes a web service, through a set of purposeful REST APIs, allowing clients to:

- Create a new certificate;
- Revoke an existing certificate;
- Verify the existence and veracity of a certificate;
- List the certificates owned by a given user.

To do so, the following resources are exposed by the web service:

- `institution`: a recognised institution who can issue certificates;
- `student`: an individual who is entitled to receiving certificates;
- `certificate`: an education certificate.

Certificates are modelled according to the open IMS standard OpenBadges[7]. The web service requires API calls to be authenticated by means of tokens, to ensure only authorized users (logged in through a proper client) can issue requests. The web service is generally deployed over multiple machines, with a load balancer which forwards incoming requests to the different instances. The requests are generally served in an asynchronous way, i.e., the client requests a computation (by, e.g., making a `POST` request to a service endpoint), receives back an ID and then polls the web server (by making `GET` requests) to fetch the result of the computation requested.

Requests arriving to the web service are validated and then transformed into invocations of chaincode in the lower tier of the system, which is constituted by a network of peers running a permissioned blockchain, in our case Hyperledger Fabric version 1.4.8. Fabric supports a pluggable consensus mechanisms (i.e., the consensus protocol to be used can be defined at configuration time and load dynamically into the nodes); in our system, we use the standard `Raft` [6] crash-fault tolerant consensus protocol, which is loosely based on a leader-follower model.

The smart contracts (chaincode), in our case written in Go, include a set of three assets, mapping one-on-one to the resources exposed by the web service in the upper tier (`institution`, `student`, `certificate`). A set of functions are defined, which correspond to the set of operations allowed on a valid asset. The chaincode includes a set of checks to validate the requests before executing them (e.g., to be revoked, a certificate should exist on the blockchain and not have been revoked before etc.). The results of the chaincode invocation are passed back to the web service, which uses them to cast responses to the web service invocations.

The overall resulting architecture is reported in Fig. 1.

[7] https://openbadges.org/

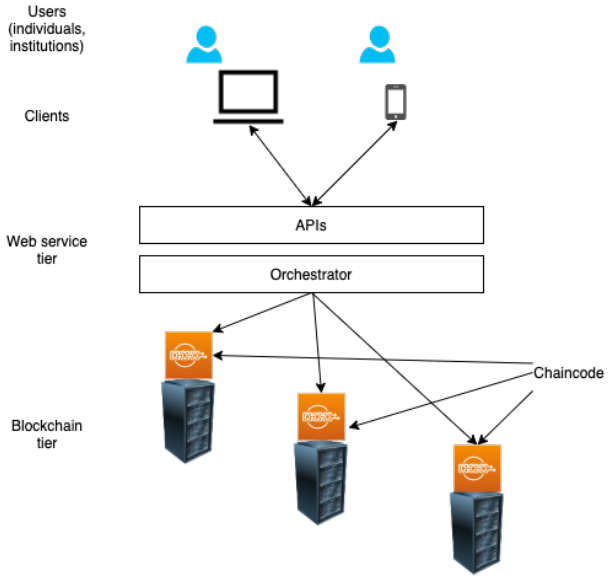

Fig. 1: The high-level system architecture.

IV. Methodology

A. Experiment design

Our system supports four core operations:

1) Registration of a new certificate;
2) Revocation of a given certificate;
3) Verification of a certificate;
4) Listing the certificates owned by a given user.

Out of these four operations:

- Only the first two are impacted by the size of the network (in terms of number of nodes), as ledgers need to be synchronized running a consensus protocol. These are not impacted by the size of the ledger, whereby said length is formally defined as the number of transactions recorded on the ledger.
- Only the last two are impacted by the size of the ledger (in terms of number of records/transactions already recorded in the ledger). These are not impacted by the size of the network (as they can be executed on a single node - just considering the local replica of the ledger).

In terms of relevant key performance indicators (KPIs), we will measure and benchmark:

- *Latency*: the time needed to complete a relevant core operation, from the moment the operation is requested to the web server, until it is completely served;
- *Resources usage*: per node and aggregated, including computing power (CPU), memory (RAM), disk space used, network (traffic in/out).

At the same time, we have three basic *scalability dimensions* along which we can measure the performance of the system:

- the number of nodes in the blockchain network;
- the request rate for a given operation;
- the size of the ledger at the moment an operation request is performed.

With respect to the first scalability dimension (size of the blockchain network), we do highlight the fact that, when new nodes get added to the system, the performance is expected to decrease (as new nodes add resiliency, integrity and availability, but requires replication of data and time to run the consensus protocol).

We can combine said scalability dimensions with the list of core operations outlined above to identify the relevant combinations, which will be subject to experimental performance assessment. The result is a set of eight tests, highlighted in light blue in Fig. 2.

We do remark that some combinations of scalability dimension vs. core operation are not considered relevant, and will therefore not be experimentally tested. To explain why, let us consider the left-bottom case (Initial ledger size and certificate registration). In the case of the certificate registration, a new transaction needs to be appended to the ledger. This involves a set of low-level operations, which we can - with some simplification - reduce to (i) create a new block (ii) cryptographically chain such block to the ledger (iii) push the update through the blockchain to ensure a coherent state. None of such low-level operation depends on the initial ledger size, which is therefore expected to have no (or: minimal) impact on relevant performance metrics.

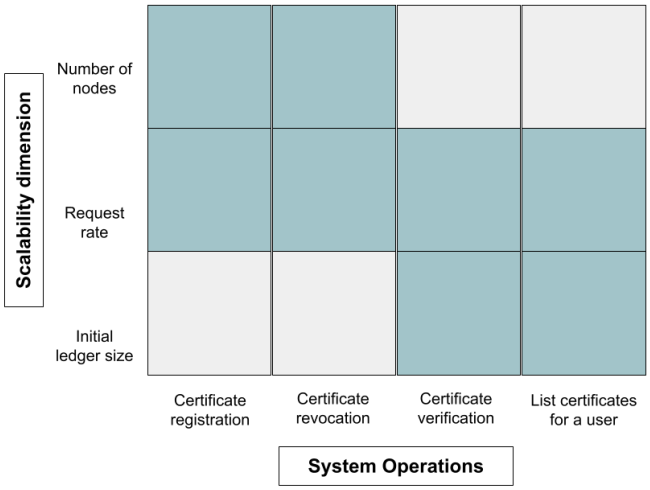

Fig. 2: Graphical representation of the relevant tests in terms of scalability dimensions vs. framework operations. The cells highlighted in light blue represent meaningful experiments.

We can therefore define the following set of tests to be performed:

- **Test-1** Operation involved: certificate registration. Number of nodes: 2, 10, 20, 50. Operation request rate: 100. Initial ledger size: 100.
- **Test-2** Operation involved: certificate registration. Number of nodes: 10. Operation request rate: 1, 10, 100, 1000. Initial ledger size: 100
- **Test-3** Operation involved: certificate revocation. Number of nodes: 2, 10, 20, 50. Operation request rate: 100. Initial ledger size: 10,000.
- **Test-4** Operation involved: certificate revocation. Number of nodes: 10. Operation request rate: 1, 10, 100, 1,000. Initial ledger size: 10,000.
- **Test-5** Operation involved: certificate verification. Number of nodes: 2. Operation request rate: 1, 10, 100, 1,000. Initial ledger size: 100.
- **Test-6** Operation involved: certificate verification. Number of nodes: 2. Operation request rate: 100. Initial ledger size: 100, 1000, 10,000.
- **Test-7** Operation involved: listing certificates for a user. Number of nodes: 2. Operation request rate: 1, 10, 100, 1000. Initial ledger size: 100.
- **Test-8** Operation involved: listing certificates for a user. Number of nodes: 2. Operation request rate: 100. Initial ledger size: 100, 1000, 10000.

B. Research infrastructure

To test our solution, we would need an experimental facility where to run controlled experiments (i.e., replicable and with no influence from external factors like, e.g., cross-traffic) in a systematic matter. The facility should cater for a sufficient large number of nodes (to test scalability along such dimension). Simply put, we would need a *cloud in vitro* [7].

Among the various options available, we discarded commercial cloud options, as they would not provide us with the required level of control (in terms, e.g., of replicability of experiments and full control on deployments). Creating our own experimental infrastructure was out of question, for both cost issues as well as time it would take to set that up.

We finally identified a set of options among those offered by the EC-funded Fed4FIRE+ initiative[8]. Among the experimental facilities federated, we opted for Grid'5000 [9].

Grid'5000 is a large-scale, flexible testbed for experiment-driven research in all areas of computer science, with a focus on parallel and distributed computing including Cloud, HPC and Big Data and AI [8].

Grid'5000 is a large scale, geographically distributed computing platform with over 15,000 computing cores grouped over 8 computing sites in France and Luxembourg. It provides access to a large amount of resources, with a total of 800 compute nodes equipped with the latest technologies and high-speed internet connection. Its architecture is fully customizable, which allows experimenters and researchers to deploy and test from bare-metal deployment features to the higher-level software stack. Grid'5000 has also integrated advanced features like monitoring and measurement features for traces collection of networking and power consumption, which provides a deep understanding of experiments for its users.

All the experiments reported in this paper were conducted on Grid'5000's Rennes Paravance cluster: 72 nodes (2 Intel Xeon E5-2630 v3, 8 cores/CPU, 128GB RAM, 2 x 600GB HDD, 2 x 10Gbps Ethernet)

[8] https://www.ngi.eu/ngi-projects/fed4fire/
[9] https://www.grid5000.fr/w/Grid5000:Home

Grid'5000 provides a set of tools and APIs for *programming and executing* experiments. Out of the available tools, we worked with the following ones. Python-grid5000[10] is a thin wrapper around the Grid'5000 REST APIs. It is a python library, which exposes the main Grid'5000 resources, such as jobs, deployments, storage, vlans and allow their modifications in a friendly manner. We used EnOSlib[11], a library written on top of Python-grid5000, which smooths the experimenter's code by dealing with various platforms. For example, we used EnOSlib to iterate our application deployment and workflow locally before moving to a large scale testbed Grid'5000. EnOSlib targets role-based deployments and allows experimenters to describe their operations using tasks.

V. Implementation aspects

As mentioned above, Grid'5000 has been used as a testbed for running our experiments. For each experiment, a plurality of servers have been used, organized in two groups: the control and compute group.

The control group always contains only one node with an instance of InfluxDb[12] and Grafana[13]. InfluxDB is an open-source time series database and in this work it was used to collect all the raw performance data from the various machines active in the experiment. Grafana is an open source analytics and interactive visualization web application, it is used as a dashboard for monitoring the experiment in real time. In the compute group all the components of our framework are deployed, with the nodes configured in the following way:

- one (1) Orderer node
- two (2) Web service nodes
- from 2 to 50 peer nodes

The Orderer node is the component of the blockchain that orders the endorsed transaction updates into blocks and distributes them to the peers for validation. The peers are the key components of the blockchain, as they compute and validate all the transactions. The Web Services are used as entry points for the blockchain as explained in detail in Sec. III they receive the transaction request from the client and forward them to peers. On each node of the Compute group an instance of Telegraf[14] is also present, which reads the metrics of the system and sends them to the Control node.

All the experiments were carried out in the time-frame January-March 2021. The rather complex experimental setup called naturally for automating the testing process as much as possible. We have used python with the Enoslib library for managing the test life cycle and Ansible (embedded in Enoslib) for the node configuration. First, we used the Enoslib API to book our slots and servers on Grid'5000, then we configured each server and deployed all the blockchain components using Ansible (this includes the definition of roles for each node in the blockchain - peer/orderer, the setup of the local instance of the ledger and the associated CouchDB for the state-of-world, the deployment of X.509 certificates, the deployment and loading of the initial smart contracts etc.). Using Enoslib we also manage the deployment of a TIG (Telegraf, InfluxDB, Grafana) stack for collecting the required measurement data, as described above. A python script is used to generate at the specified rate requests for the desired system operation; Enoslib is used also for the final clearance of the infrastructure after the end of each test. Each test is defined by a python script, which performs the following operations:

1) Reserve the servers.
2) Install and configure docker on each node.
3) Deploy the blockchain components on the Compute group.
4) Deploy the InfluxDB and Grafana on the Control node
5) Deploy an Instance of Telegraf on each node of the Compute group.
6) Initialize the ledger according to the test specifications.
7) Execute the experiment: in this step a series of rest request are performed to the two Web Services and the result and timings are pushed to the InfluxDB on the Control Node
8) Download and pre-process all the data stored in InfluxDB
9) Tear down all the infrastructure.

For each experiment, and for each setting, as defined in Sec. IV-A, ten (10) runs were executed. The analysis of raw data was carried out using purposeful python notebooks.

VI. Experimental results and analysis

For the sake of conciseness and readability, we report here the main (and, in some cases, unexpected) findings of our experimental analysis. The complete set of raw experimental data (8.3GB) has been published under a CC-BY-4.0 on Zenodo [1], to enable independent audits of our findings and further analysis by the research community at large. All the results are presented using standard box-and-whiskers diagrams (often referred to as boxplots), thereby reporting for each performance metric displayed the maximum value, the minimum, the median, the 25% and 75% percentiles, all computed over 10 runs, as stated in the previous section.

The latency decrease when the operation rate increases. In the operations that writes data in the blockchain, the latency decreases when we increase the operation rate (in requests/s). This rather counterintuitive result is due to two parameters of Hyperledger Fabric: the batch timeout and the batch size. In fact the transactions are processed and registered in batches; a batch is closed when a timeout occurs, or when the maximum number of elements (batch size) is reached, whatever happens first. In our test we set the batch timeout to 1 second and the batch size to 10 transactions (standard values). So with a higher request rate we effectively fill the batches faster, as we do not have to wait for the timeout to expire. This highlights the need to tune carefully such parameter in real operational settings, depending on the expected load in terms of request rates. In Fig. 3 we reported the results in terms of latency to

[10] https://pypi.org/project/python-grid5000
[11] https://discovery.gitlabpages.inria.fr/enoslib/
[12] https://www.influxdata.com/
[13] https://grafana.com/
[14] https://www.influxdata.com/time-series-platform/telegraf/

complete the registration of a new certificate for a network of 10 nodes and an initial ledger size of 100. The revocation of a certificate follows the same pattern.

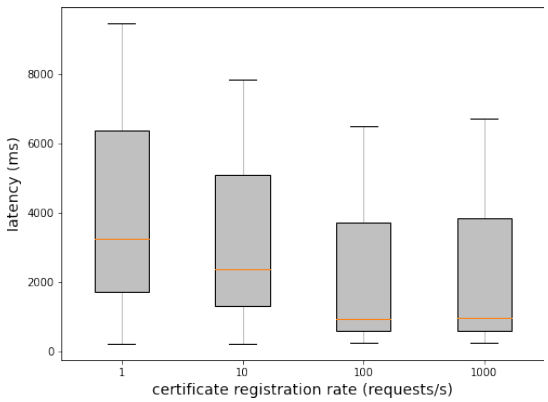

Fig. 3: Latency (in ms) for the registration of a new certificate as a function of the certificate registration rate, blockchain with 10 peer nodes, initial ledger size of 100 transactions.

The latency increases when the number of node increases. In order to be validated, transactions need to be validated through the execution of the consensus protocol (in our case: `Raft`). As this requires nodes to coordinate, it is expected that the time to run it would increase with the number of nodes. In Fig. 4 we reported the results in terms of latency to complete the registration of a new certificate as the number of nodes in the blockchain increases, with 100 requests/s and an initial ledger size of 100 transactions. Again, the revocation of an existing certificate follows the same pattern.

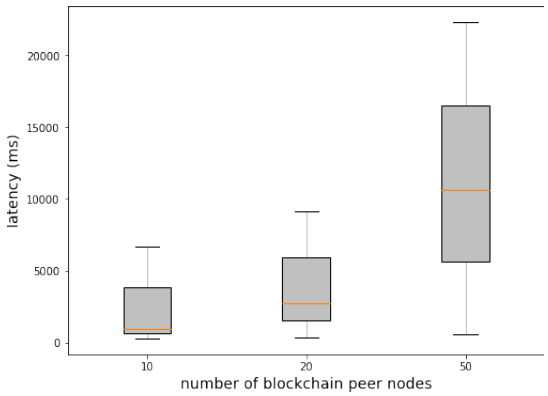

Fig. 4: Latency (in ms) for the registration of a new certificate as a function of the number of peer nodes in the blockchain, 100 requests/s and 100 transactions initial ledger size.

The initial ledger size does not influence the latency. For all operations concerned, it turns out that the ledger size does not significantly impact the latency to complete the requested operation. This is good because it means that the performance will be stable over time, as over time the size of the ledger will increase (but performance will not degrade). In Fig. 5 we reported the results in terms of the latency to complete the verification of a certificate as a function of the initial ledger size, with 2 nodes in the blockchain and a rate of 100 requests/s.

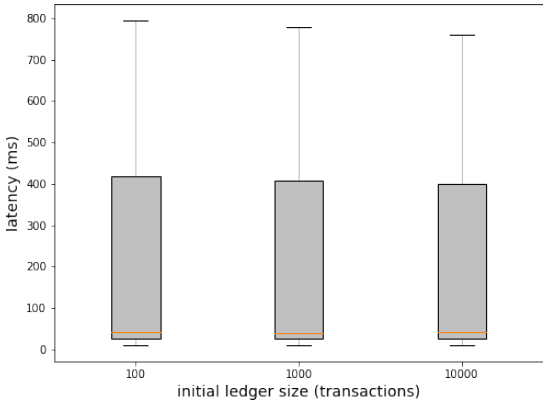

Fig. 5: Latency (in ms) for the verification of a certificate as a function of the initial ledger size, 2 peer nodes in the blockchain, 100 requests/s.

The CPU usage is not significantly influenced by the number of nodes but is influenced by operation per second rate. Experimental results demonstrate that the CPU usage is not influenced in a significant way by the number of nodes but it is influenced by the operation rate. In Fig. 6 we reported the results in terms of CPU usage (in %) as a function of the number of nodes for a setting in which there are 100 requests/s to register a new certificate (initial ledger size of 100). It can be noted that the median value does not increase significantly, whereas the maximum value does; this element of variability is something to be carefully considered when dimensioning servers for running the system in operation. In particular with 50 nodes in the blockchain network we are in one case close to saturating the available CPU. In Fig. 7 we reported the results in terms of CPU usage (in %) as a function of the request rate for the revocation of existing certificates, with 10 peer nodes in the blockchain and an initial ledger size of 10,000 transactions. Here the maximum value does not vary widely, but the median does increase significantly with the operation rate.

Network traffic increases with both the number of nodes and with the operation request rate. As expected, due to the execution of the consensus protocol, the network traffic increases with the number of nodes. At the same time, it turns out that the network traffic is also quite sensitive to the rate at which operation requests arrive to the system. As an example, we report in Fig. 8 the results in terms of incoming/outgoing traffic (averaged over all nodes in the network) for an experiment with an initial ledger size of 100 transactions and a blockchain consisting of 10 peer nodes, plotted as a function of the rate of certificate generation requests. Note the very large variability: while the median values range from approx. 0.1Mb/s (for 1 request/s) up to approx. 2.5 Mb/s (for 1,000 requests/s), our experiments reported a high variability among single runs, in particular in terms of received traffic. (We are currently investigating furtherthe system logs to understand

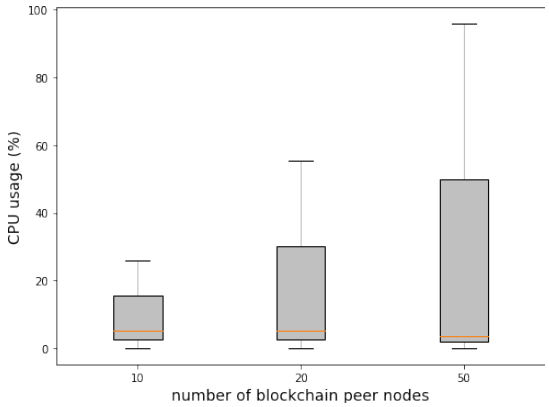

Fig. 6: CPU usage (%) as a function of the number of nodes, initial ledger size 100 transactions, 100 request/s for the registration of a new certificate.

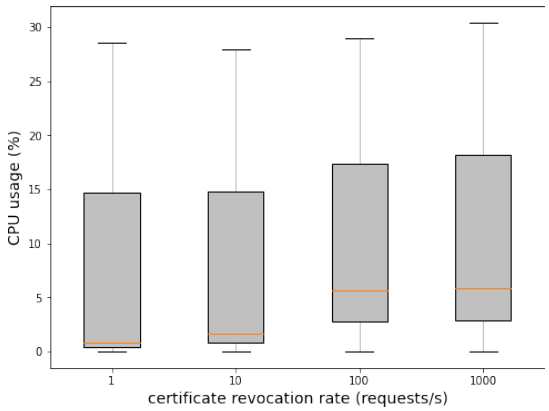

Fig. 7: CPU usage (%) as a function of the request rate for the revocation of certificates, 10 peer nodes in the blockchain, initial ledger size of 10,000.

the reason of such large variance, which we suspect is due to exogeneous factors.) Traffic is very important to consider, as on many public clouds it can soon become the limiting factor (in terms of sustainable economics), due to the typically rather substantial associated costs.

Memory usage is stable. The memory usage does not result very sensitive to either the number of nodes in the blockchain or the request rate, for all operations considered. This is particularly appealing, in that memory is an expensive resource for large-scale cloud applications. We report hereafter two examples. The first one, in Fig. 9 plots the average memory usage (in %) as a function of the number of peer nodes in the blockchain for a setting where there are 100 requests/s to register a new certificate, with initial ledger size of 100. The second one, reported in Fig. 10 reports the average memory usage (in %) as a function of the request rate (for certificate revocation), for a blockchain with 2 peer nodes and an initial ledger size of 10,000 transactions.

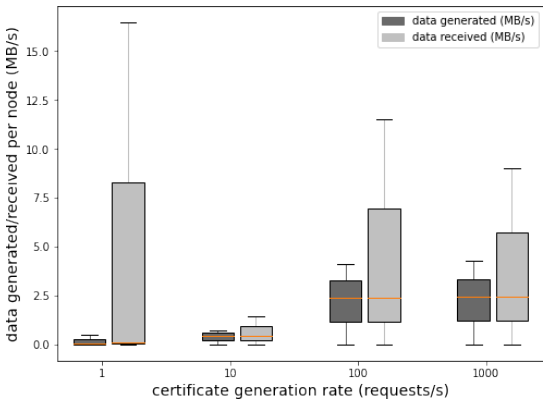

Fig. 8: Traffic generated/received as a function of the certificate registration rate (requests/s); initial ledger size 100 transactions, 10 peer nodes in the blockchain.

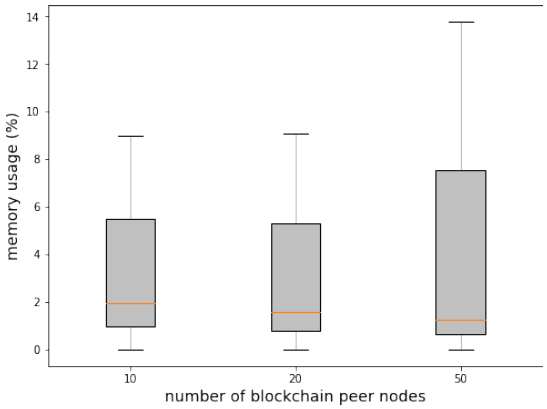

Fig. 9: Memory usage (in %) as a function of the number of peer nodes in the blockchain; 100 certificate registration requests/s, initial ledger size of 100

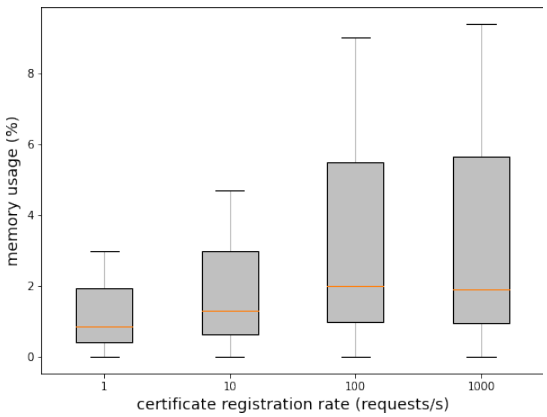

Fig. 10: Memory usage (in %) as a function of the request rate (for certificate revocation); 2 peer nodes in the blockchain, initial ledger size of 10,000.

VII. Conclusion

In this work, we carried out a methodologically rigorous and comprehensive evaluation of the scalability of permissioned

blockchains (in particular, in our case: Hyperledger Fabric), leveraging the capabilities and features of a large-scale cloud computing experimental facility, Grid'5000. The results have been made available as open data to the research community at large and published on Zenodo [1]. The tests performed revealed a number of scalability issues, that can be traced back in the *default* setting of some configuration parameter. These aspects are of high relevance for practitioners and businesses launching digital products based on blockchains, as overlooking them may have detrimental impact on both the performance of the system (in terms of latency) as well as on the economic aspect (related to the consumption of resources).

In particular, four aspects emerged, which should be further analysed and experimentally validated:

- Working with smaller batch size and shorter timeouts should improve the responsiveness of the system, in particular in situations (like in the bootstrapping phase of a new blockchain-enabled service) in which the rate of transaction requests is limited.
- Adding more orderers to the system should improve the scalability in the presence of a rather large number of peer nodes. The presence of a single orderer, while appealing in terms of simplicity and ease of configuration, proved to soon become a bottleneck as the blockchain network size grew. Multiple orderer nodes would also contribute to the robustness and resilience of the overall system, avoid single points of failure.
- In Hyperledger, proposed transaction updates need to be endorsed by a subset of the peers before being validated by the orderer node, which then groups them into a new block. This process is meant to avoid situations in which, e.g., one single malicious organisation in the blockchain network tries to add non-valid transactions. Endorsing peers are usually chosen to represent the majority of the organizations in the network (in most cases: all organizations), so to minimise this risk. *Endorsement policies* define the rules for the orderer node to validate a proposed transaction update. A set of looser endorsement policies, not requiring all organizations in the network to endorse a proposed transaction update, should reduce the latency associated to the creation or revocation of a certificate.
- One relevant parameter concerns the way requests arriving to the web server(s) are forwarded to the blockchain nodes: the *fan-out* defines to how many blockchain nodes an incoming request is forwarded. A larger fan-out provides robustness and resilience against unreachable/faulty nodes, but at the same time leads to a higher consumption of computing resources (as the relevant chaincode is executed on multiple machines). One less obvious drawback of a high fan-out is the fact that concurrent, identical proposed transaction updates are created at different nodes, thereby creating additional (and useless) work for the orderer node, who has to sift through all proposed transaction updates to pick the one to be added next to be the ledger. The tradeoff in the choice of the fan-out parameter clearly influences also the number of orderer nodes which should be included in the system, as discussed above.

VIII. Acknowledgment

This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 765452. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

This work has received also funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732638 (*Open call - F4Fp-SME-08M-Stage2-15 GoldenOwl*).

References

[1] S. Tavonatti, D. Battulga, M. Fahradi, C. Caprini, and D. Miorandi, "Raw data for the experimental evaluation of the scalability of permissioned blockchains," 2021, https://doi.org/10.5281/zenodo.4588504.

[2] M. Kuzlu, M. Pipattanasomporn, L. Gurses, and S. Rahman, "Performance analysis of a hyperledger fabric blockchain framework: throughput, latency and scalability," in *2019 IEEE international conference on blockchain (Blockchain)*. IEEE, 2019, pp. 536–540.

[3] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[4] Q. Nasir, I. A. Qasse, M. Abu Talib, and A. B. Nassif, "Performance analysis of hyperledger fabric platforms," *Security and Communication Networks*, vol. 2018, 2018.

[5] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1085–1100.

[6] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 305–319.

[7] L. Nussbaum, "Testbeds support for reproducible research," in *Proceedings of the reproducibility workshop*, 2017, pp. 24–26.

[8] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jégou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab *et al.*, "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in *The 6th IEEE/ACM International Workshop on Grid Computing, 2005.* IEEE, 2005, pp. 8–pp.