The DLPS: A New Framework for Benchmarking Blockchains

Johannes Sedlmeir
Project Group Business & Information Systems Engineering of the Fraunhofer FIT
Bayreuth, Germany
johannes.sedlmeir@fit.fraunhofer.de

Philipp Ross
BMW Group
Munich, Germany
philipp.p.ross@bmw.de

Andre Luckow
BMW Group
Munich, Germany
andre.luckow@bmwgroup.com

Jannik Lockl
FIM Research Center
University of Bayreuth
Bayreuth, Germany
jannik.lockl@fim-rc.de

Daniel Miehle
BMW Group
Munich, Germany
daniel.miehle@bmwgroup.com

Gilbert Fridgen
SnT - Interdisciplinary Center for Security, Reliability and Trust
University of Luxembourg
gilbert.fridgen@uni.lu

Abstract

Distributed Ledger Technologies (DLT) promise to revolutionize business ecosystems by permitting secure transactions without intermediaries. A widely recognized challenge that inhibits the uptake of DLT is scalability and performance. Hence, quantifying key metrics such as throughput and latency is crucial for designing DLT-based infrastructures, applications, and ecosystems. However, current benchmarking frameworks for blockchains[1] *do not cover the whole benchmarking process; impeding transparent comparisons of different DLT networks. In this paper, we present the Distributed Ledger Performance Scan (DLPS), an open-source*[2] *framework for end-to-end performance characterizations of blockchains, addressing the need to transparently and automatically evaluate the performance of highly customizable configurations. We describe our new framework and argue that it significantly improves existing DLT benchmarking solutions. To demonstrate the capabilities of the DLPS, we also summarize the main results obtained from a series of experiments that we have conducted with it, giving a first comprehensive comparison of essential scalability properties of several commonly used enterprise blockchains.*

[1]strictly speaking, blockchains are a subset of Distributed Ledger Technologies (DLT), but in this work, we will use the terms interchangeably since all distributed ledgers we investigated are blockchains

[2]the DLPS is available at `https://github.com/DLPS-Framework`

1. Introduction

DLT are expected to play an important role in tomorrow's IT landscape [1]. Nakamoto introduced the first blockchain, Bitcoin, in 2008 and established a Peer-to-Peer (P2P) digital currency without the need for trusted intermediaries such as banks [2]. Buterin et al. then extended the scope of blockchain technology from financial transactions to the execution of general process logic and implemented respective capabilities in Ethereum [3]. This finally realized a vision first communicated by Szabo, where so-called Smart Contracts (SCs) could be concluded digitally and on a P2P basis, without any trusted intermediary [4]. Since then, a large number of blockchain-based use cases have emerged, outreaching the financial sector [5, 6].

To secure a Distributed Ledger (DL) without a distinguished administrator against malicious behavior, storing data and performing operations on the ledger is performed *redundantly* on all participating nodes. A suitable tamper-sensitive data structure (often Merkle-trees) and usage of public-key cryptography make retrospective manipulations easily detectable and allow for secure authentication [7]. A *consensus mechanism*, a mixture of economic incentives and cryptographic methods, ensures that the presupposed benevolent majority agrees on which transactions to operate. Redundancy and the additional overhead caused by the consensus mechanism, however, lead to a significantly decreased performance of DLs when compared to centrally managed databases [7]. This makes building decentralized applications very challenging as established DLT networks usually cannot elastically scale on demand [8]. Therefore, an in-depth understanding of DLT performance becomes essential, as the performance poses a key aspect for the viability of emerging decentralized applications.

To address the performance requirements of enterprise blockchain solutions, permissioned DLs have been developed. They restrict participation, allowing for other types of consensus mechanisms that generally exhibit finality and lower latency. Moreover, in an enterprise scenario, hardware and bandwidth restrictions are less relevant than in a permissionless system. However, enterprise IT-systems must also meet high performance requirements, and throughput of permissioned DLT still lags significantly behind that of their centralized counterparts. Consequently, research considers performance a major obstacle for productive usage of enterprise DLT implementations [9].

Unfortunately, literature only provides limited knowledge regarding the performance of enterprise blockchain solutions, and for the few currently available results, we also discovered quite different performance results. Moreover, blockchain implementations have already become heterogeneous and are quickly evolving, so no generally acknowledged benchmarking tool has been established to comprise all of these particularities. Moreover, existing work on benchmarking does not provide clear definitions of key metrics such as throughput and latency, and do not specify the algorithms that they use to measure these key metrics, which leads to a lack of transparency and reproducibility.

In this paper, we address this research gap by presenting a transparent and highly flexible, open-source framework for obtaining reliable performance data of several enterprise DLT solutions. It was implemented in an iterative approach within an enterprise project that needed reliable performance comparisons to support the choice of enterprise blockchain technology for their use case. We argue that the DLPS covers the deficiencies of existing approaches and allows to measure well-defined quantitative key performance indicators of different DLT with a universal, comprehensive and transparent benchmarking algorithm. We also present and discuss the results of a first systematic scalability comparison of the DLT that we have already integrated to compare our framework with previous solutions and to demonstrate that the DLPS is applicable to a variety of technologies. To the best of our knowledge, the range of investigated DLT and also the variety of network sizes that we tested is, so far, unique.

The remainder of this paper is structured as follows: Sec. 2 gives an introduction to essential background concepts and presents some of the permissioned DLT that we have already integrated into the DLPS. Sec. 3 provides an overview of existing work on benchmarking permissioned DLT and sketches their main shortcomings. We then introduce the DLPS in Sec. 4. In Sec. 5, we present and discuss the main results of our exemplary scalability and performance analyses in order to demonstrate the capabilities of the DLPS. We conclude with a summary and our plans for future work in Sec. 6.

2. Background

2.1. Consensus mechanisms

For permissionless blockchains, which constitute the technology behind cryptocurrencies, the most common consensus mechanisms are Proof-of-Work (PoW) and Proof-of-Stake (PoS). They tie voting power in the system to some scarce resource – energy in PoW and capital in PoS – to defend the system against Sybil attacks. These consensus mechanisms generally exhibit high latency and do not provide finality, implying that even after some nodes have performed a particular transaction, one has to wait minutes to hours before the probability that this transaction will be replaced is sufficiently small [10]. Moreover, PoW is very energy-intensive [11]. For permissioned blockchains, voting-based consensus mechanisms are applicable because participation in consensus is restricted. These consensus mechanisms provide finality and also much lower latency, but are only viable for small-scale networks.

In most voting-based consensus mechanisms, the participants (i.e., nodes) usually agree on a common leader, which proposes new transactions and distributes them to the other nodes called followers. In a consensus mechanism that exhibits **Crash Fault Tolerance (CFT)**, the other nodes will realize a crash of their leader and elect a new leader. However, the followers blindly rely on their leader as long as it is active, so a *malicious* leader can be problematic. Prominent examples for a CFT consensus mechanism are Kafka and RAFT [12]. Since leader election needs a majority vote, a network must be of at least size 2f+1 to handle f crashing nodes.

By contrast, consensus mechanisms with **Byzantine Fault Tolerance (BFT)** can deal not only with crashes but also with arbitrary malicious behavior. Like CFT protocols, an elected leader proposes new blocks, while multiple cross-checks ensure consistency among the non-faulty nodes. Therefore, the communication overhead of a BFT protocol grows faster in the number of nodes than for a CFT protocol. In general, the best case accomplishable is that a network of size 3f+1 can deal with f malicious nodes [13]. Popular implementations are Practical BFT (PBFT) [14], Istanbul BFT (IBFT) [15], and Redundant BFT (RBFT) [16]. RBFT is an advancement of PBFT which offers very reliable performance also under the actual presence of malicious behaviour [16].

Proof-of-Authority (PoA) consensus mechanisms have been implemented to achieve an approximation to CFT or BFT at less overhead. Prominent examples thereof are Clique, and Aura [17]. They generally use a simplified leader election and leave out different steps thereafter as compared to PBFT protocols. In [17], the authors question whether these consensus mechanisms are adequate for blockchains because they cannot guarantee data consistency among all non-faulty nodes (known as *safety*).

A large variety of other consensus mechanisms exists, but so far, these have had only little adoption. One that should be mentioned in this paper is Proof of Elapsed Time (PoET), which uses trusted hardware (the Intel SGX) to establish tamper-proof random number generation for nodes, which then determines who may publish the next block. It claims to offer solid performance even in permissioned networks with a larger number of validators [18].

2.2. Permissioned Blockchains

We now give a short overview of the permissioned blockchain systems currently integrated in the DLPS. All of them are open-source, and – apart from Indy – provide means to implement Turing-complete transaction logic, also know as SCs. Table 1 summarizes these DLT and some of their characteristics at the time that we conducted our experiments presented in Sec. 5.

DLT	Consensus	SC Languages	Version
Eth. (Geth)	PoA (Clique)	Solidity	1.9.8
Eth. (Parity)	PoA (Aura)	Solidity	2.5.10
Fabric	Solo, Kafka, RAFT	Go, Javascript	1.4.4
Indy	RBFT	-	1.12.0
Quorum	RAFT, IBFT	Solidity	2.3.0
Sawtooth	RAFT, PBFT, PoET,	Go, Python, . . .	1.2

Table 1. Comparison of the DLT that we integrated in the DLPS and evaluated in the experiments.

Ethereum was the first public blockchain which supported SCs, enabling guaranteed and tamper-proof execution of program code [19] in the so-called Ethereum Virtual Machine (EVM). It is developed by the Ethereum foundation. While the popular public chain currently uses PoW, **Private Ethereum Networks** have been developed on which one can capitalize on other consensus mechanisms. The two most popular Ethereum clients for private networks on which we focus in this work, Geth and Parity, use the PoA consensus mechanisms Clique and Aura respectively [17].

Fabric is a framework for building private permissioned blockchains. Fabric is special among other DLT architectures for one main reason: Most blockchains (both permissionless and permissioned ones) use a so-called validate-order-execute paradigm [20]: They first check whether a transaction is legitimate (*validate*), then agree on the transactions and their sequential arrangement in the next block through consensus (*order*), and finally apply the transactions on their local ledger through the blockchain's state transition function (*execute*). By contrast, Fabric entails an execute-order-validate paradigm: At first, according to a so-called *endorsement policy*, a subset of the nodes simulates the outcome of performing a transaction and signs it (*execute*). The client collects the required endorsements (specified by the SC that, for example, three out of five nodes need to agree) and hands them to the *ordering service*, which collects the transactions, checks whether the endorsement policy is satisfied, puts them in blocks, and distributes the blocks to all nodes (*order*). Finally, when nodes apply the transactions to their ledger, they have to check for read-write collisions as simulations are not necessarily ordered (*validate*). By this design, the degree of redundancy can be customized according to the needs of each SC [20]. Fabric currently offers 3 different kinds of consensus: Solo (i.e., a single ordering node, mainly intended for testing purposes), Kafka, and RAFT. It also supports different databases for the transaction log and the current world state, namely, LevelDB and CouchDB [21].

Indy is a public permissioned blockchain. Participation in consensus is thus restricted while read access is not. Indy is developed mainly for specific purposes in identity management and hence optimized for reading operations because they will occur much more frequently. Therefore, all transactions are signed by all nodes and include a timestamp such that querying from a single node is still sufficient to rule out undetected malicious answers. Indy runs Plenum as a consensus mechanism, which is a slightly adapted version of RBFT. In contrast to all the other blockchains presented here, Indy does not support arbitrary SCs, but only a basic set of transactions related to identity management [22].

Quorum is a private permissioned blockchain project led by J.P. Morgan. It originates from Ethereum but aims to allow for business applications that are not feasible on the public main net due to performance restrictions. Quorum supports RAFT and IBFT consensus mechanism [23].

Sawtooth is another permissioned blockchain project similar to Fabric, Sawtooth separates between the application and core system level, allowing using different programming languages for SC development. Sawtooth supports multiple consensus mechanisms, namely RAFT, PBFT, and PoET, which one can even switch at runtime [18].

3. Related Work

The performance of permissionless blockchains can be monitored by analyzing publicly accessible data, and their architecture is not customizable for a specific use case. Consequently, there is only limited need to conduct benchmarking in the context of use case technology selection and optimization. By contrast, benchmarks for permissioned DLT are desperately needed for enterprises in designing decentralized applications. Originally, we intended to collect available performance results or existing benchmarking frameworks to decide for a specific permissioned network in an enterprise project. For this purpose, we conducted a literature research for the search string "(blockchain OR distributed ledger technology) AND (performance OR throughput OR latency) AND (benchmarking OR measurement OR evaluation OR analysis)" on the Google Scholar, ACM DL, and IEEE Explore databases. We found that there are two existing frameworks, and most articles that study the performance of specific blockchains refer to one of these.

The first systematic benchmarking framework for permissioned blockchains was Blockbench. It relies on established YCSB and Smallbank benchmarks and integrates private Ethereum (Geth and Parity), Fabric, and Quorum. The framework is open-source and modular and provides smart contracts to evaluate different workloads. Nevertheless, beyond reacting to some blockchain-related updates, there have not been significant advancements since 2017. Dinh et al. [24] use Blockbench for an in-depth comparison of the performance of Geth, Parity, and a by 2019 outdated version of Fabric.

The other prominent available framework is Caliper. It was originally developed to benchmark only DLT of the Hyperledger project, but now also integrates Ethereum-based DLT. Caliper contains different basic SCs, which trigger particularly CPU- or i/o-heavy transactions. These experiments are valuable for grasping different characteristics of performance.

In our literature review, we found that while there have been valuable performance measurements on various permissioned DLT, the data is highly fragmented over various contributions, and none of the papers we encountered gives a full description of their benchmarking process or setup. Consequently, the lack of a fully transparent description of how performance metrics were obtained leads to a serious lack of comparability across different works. [25] structure some of the related work that we found in our literature review, and already from this subset it gets evident that benchmarking data is highly fragmented across multiple works, the results generally vary significantly, and it is particularly to compare the results. In our opinion, the reason is that yet no benchmarking framework is sufficiently standardized to provide comparability, highly customizable, and at the same time, ease of use. For example, none of the publications from our literature review provides a precise description of the overall definitions and assumptions underlying the measurements of the key performance indicators throughput and latency. Also, they leave the setup of blockchain and client nodes to the user, which both raises the hurdle to start benchmarking and also impairs comparability because different benchmarks happen on different infrastructures. Integrating several DLT in a single framework is a huge challenge because blockchain technologies are quickly evolving, which often implies shortcomings in the documentation, stability issues, and difficulties in getting familiar with the technology and starting a functioning test network. This is particularly important when conceptualizing enterprise blockchain architectures, in which parameters such as the number of nodes or the block-time could be tuned according to requirements, and a tool that allows testing the performance for different choices of parameters would make things much easier for the engineers.

Driven by the motivation to improve DLT benchmarking and build a standardized framework that is easily accessible and useful to a broad community, as well as to obtain reliable comparisons of DLT performance for enterprises striving to adopt DLT solutions, we took this challenge and implemented the DLPS as an end-to-end pipeline with fully automatic node and client setup, benchmarking, and evaluation. This brings the advantage of built-in scalability of the network size in experiments, as well as an easy setup for researchers and practitioners who are not experts in each of the integrated DLT. At every point, we implemented the blockchain network in the way suggested in the respective development repository, with a client-node architecture that seemed close to how one would implement it in reality (e.g., the provided SDK or popular software such as web3 for Ethereum-like blockchains). Inspired by the functionality of Caliper, we have also integrated different workloads. Currently, we offer doNothing (empty workload), writeData (writing a single key-value pair), matrixMultiplication (CPU-heavy workload), and writeMuchData (i/o- heavy workload).

While the benchmarking process itself is standardized, the blockchain, node, and SC functionalities are highly configurable through a single config file, thus providing highly customizable workloads and configurations while maintaining a standardized benchmarking process. By defining and implementing the intuitive benchmarking logic as described in Sec. 4 and using a realistic client-node setup as well as developing representative workloads that capture the characteristics of many real-world use cases, we also naturally adopt standard best practices for computer systems evaluation [26, 27].

4. The Distributed Ledger Performance Scan

4.1. Definition of key metrics

Following most of the related work referenced in Sec. 3, we focus on the key performance indicators *throughput* and *latency*. Since we could not find a clear definition of these metrics in any of the related work, we start with developing precise definitions, on top of which we can implement a generic algorithm to measure them.

Generally speaking, if we send requests at a certain frequency f_{req} to a DL, this will result in a corresponding response rate $f_{\mathrm{resp}} \equiv f_{\mathrm{resp}}(f_{\mathrm{req}})$ of successfully performed transactions. We define *maximum sustainable throughput* $\hat{f}$ as the maximum reachable rate f_{resp} which the blockchain can reliably process over a longer period (e.g., one minute) when we try different request rates f_{req}:

$$\hat{f} := \max\{f_{\mathrm{resp}}(f_{\mathrm{req}}) : f_{\mathrm{req}} \geq 0\}. \qquad (1)$$

Latency l is generally defined as the average time between sending a request and receiving confirmation that it was operated on a sufficient number of nodes (e.g., on at least 2/3 of all nodes). This quantity may depend on the load which the system is currently facing, so $l \equiv l(f_{\mathrm{req}})$. We define *latency (at stress)* as latency at precisely the request rate at which we attain maximum sustainable throughput (see (1)):

$$\hat{l} := l(\hat{f}_{\mathrm{req}}) \qquad \text{where} \qquad f_{\mathrm{resp}}(\hat{f}_{\mathrm{req}}) = \hat{f}. \qquad (2)$$

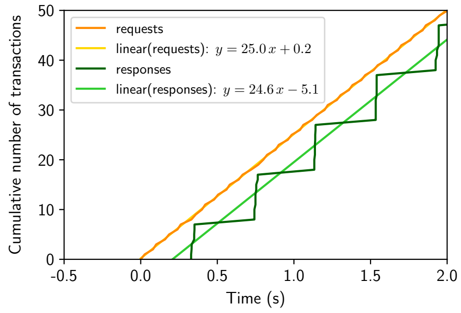

Figure 1. Sending requests to a DLT and getting responses.

The approach which we have just sketched relies on a few assumptions. For example, we assume that throughput depends only on f_{req}, while a real system will exhibit time-dependent fluctuations. Furthermore, $\hat{f}_{\mathrm{req}}$ is not well-defined if f_{resp} does not have a unique maximum. However, the results of our experiments, which we describe later, suggest that actually, the reality is quite close to our simplification, and – not surprisingly – that $\hat{f} \approx \hat{f}_{\mathrm{req}}$. The highest f_{resp} is thus achieved when the request rate matches the maximum sustainable throughput.

To measure $\hat{f}$, we implemented the following algorithm: We start sending requests from some clients to some blockchain nodes at a fixed total frequency f_{req} for a certain duration. For each of the asynchronous requests, we record both the time the client sends it and the time the associated response confirming its execution arrives at the client. Fig. 1 displays data obtained from sending requests to a small Fabric network. It illustrates that by plotting the cumulative number of requests resp. responses against time, we can define f_{req} and f_{resp} as the slope of their corresponding linear regressions: Indeed, differences Δy on the y-axis in a period Δx correspond to the number of transactions sent resp. completed in this period, so the slope $\frac{\Delta y}{\Delta x}$ is precisely the corresponding request resp. response rate. This definition is very robust because it is insensitive to a few outliers. In the picture, we observe that $f_{\mathrm{req}} \approx f_{\mathrm{resp}}$. Note also that in Fig. 1, responses are received in batches of around 10 transactions, representing new, confirmed blocks.

As long as the linear regressions of request and response curves are parallel (i.e., $f_{\mathrm{resp}}(f_{\mathrm{req}}) \approx f_{\mathrm{req}}$), the average time between sending and receiving a transaction is given by the shift between the intercepts of the two regressions with the x-axis (this is a short computation). In Fig. 1, latency is therefore given by approx. 0.2 s. However, if $f_{\mathrm{resp}}(f_{\mathrm{req}}) \not\approx f_{\mathrm{req}}$, due to a potentially growing queue, the former definition of latency in terms of average delay depends on the duration of the experiment while the latter does not. We therefore define latency $l(f_{\text{req}})$ as the shift between the x-intercepts of the regressions $y_i = f_i \cdot x + t_i$ where $i \in \{\text{req}, \text{resp}\}$:

$$l(f_{\text{req}}) := \frac{t_{\text{req}}}{f_{\text{req}}} - \frac{t_{\text{resp}}}{f_{\text{resp}}} : \qquad (3)$$

As long as the blockchain can handle the rate of requests, the two regressions will stay approximately parallel. $f_{\text{resp}}(f_{\text{req}})$ is monotonically increasing and close to the diagonal of the first quadrant. However, successively increasing f_{req}, at some point, the blockchain will not be able to keep up with the rate of requests anymore. Due to overload or congestion, we then expect that further increasing f_{req} will decrease f_{resp}. We can approximate $\hat{f}$ experimentally by successively increasing f_{req} until the regression slopes f_{req} and f_{resp} diverge, and, by (1), obtain an approximation for $\hat{f}$ by taking the maximum value for f_{resp} over all these trials. Fig. 2 shows one example for such a *ramping series* of measurements where we successively increase f_{req} by 25 tx/s. As expected, after a range where $f_{\text{resp}} \approx f_{\text{req}}$, throughput first stagnates and then declines. Further increasing f_{req} causes a drop in efficiency, the ratio of successfully operated transactions. Therefore, we can reasonably state that the maximum sustainable throughput $\hat{f}$ is approximately 200 tx/s in Figure 2.

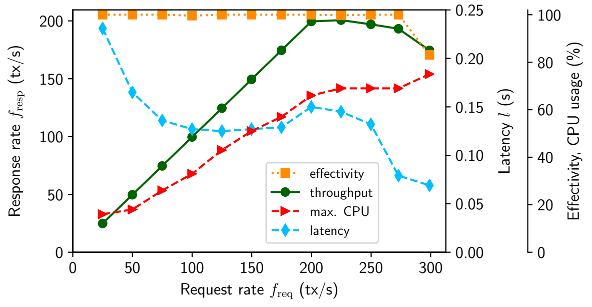

Figure 2. Ramping and localization logic

To find the bottleneck responsible for the bound on throughput, we monitor the most relevant resource stats on nodes and clients. Here, for simplicity, we only discuss CPU usage on blockchain nodes. As expected, CPU usage is increasing in f_{req} and, therefore, with stress posed on the blockchain, it might well be the limiting resource in this case. On the other hand, the chart depicting latency seems surprising at first: One might assume that latency is also increasing in f_{req}. However, since responses are bundled in blocks, the creation of which is normally triggered by timeouts or reaching a certain number of pending transactions, it seems reasonable that for higher f_{req}, more blocks are produced per time and therefore the green staircase in Fig. 1 gets finer and moves closer to the request curve. Obviously, single transactions show a latency of only 0.1 s, and in Fig. 2, we can see that this is also close to the minimum overall latency at $f_{\text{req}} \approx 150$. If we further increase f_{req}, the stress on the system dominates and – in line with intuition – latency increases until we reach $\hat{f}_{\text{req}} \approx 200$. For $f_{\text{req}} \geq 200$, we then see an unexpected drop in latency. However, we relativize this since, due to congestion and growing instability resulting from significant overload, the assumption that the response curve is close to a straight line turns out to be wrong. Hence, the derivation of latency is no more meaningful in this regime.

The heuristics and observations described above suggest an intuitive algorithm to efficiently determine $\hat{f}$ and $\hat{l}$ for a given DLT system. We have implemented this algorithm in the DLPS and display a simplification of the flowchart in Fig. 3. Before starting our experiment, we define parameters that are fixed throughout the whole benchmarking process, such as the $\mathrm{duration}$ of a single measurement period. We also specify an initial request frequency base and a step (25 tx/s in Figure 2) by which we increase f_{req} whenever the blockchain kept pace in the last trial. We have multiple criteria based on which we can decide whether or not a blockchain kept pace. The most crucial one ensures that f_{resp} may not significantly deviate from f_{req}:

$$\left| \frac{f_{\text{resp}}}{f_{\text{req}}} - 1 \right| \leq \delta, \qquad \text{for reasonably small } \delta. \qquad (4)$$

Moreover, we ensure that the response curve is actually close to a straight line by requiring that the coefficient of determination for the response curve is close to 1. To account for fluctuations in the system, we repeat a single trial multiple times if the blockchain cannot keep up according to our decision rule, because we do not want pure coincidence to stop a series as depicted in Figure 2 as long as we have not reached $\hat{f}$. We also require that a few (e.g., more than 3) successive increases of f_{req} have happened during the ramping series because since we want to measure maximum *sustainable* throughput, we also have to rule out that by coincidence, the system managed to deal with a very high rate for the $\mathrm{duration}$.

Finally, when we have completed a ramping series, we run another ramping series with base and step suitably adjusted. We distinguish *localization runs*, in which we choose base and step to get a higher resolution in the region around $\hat{f}_{\text{req}}$ from the last

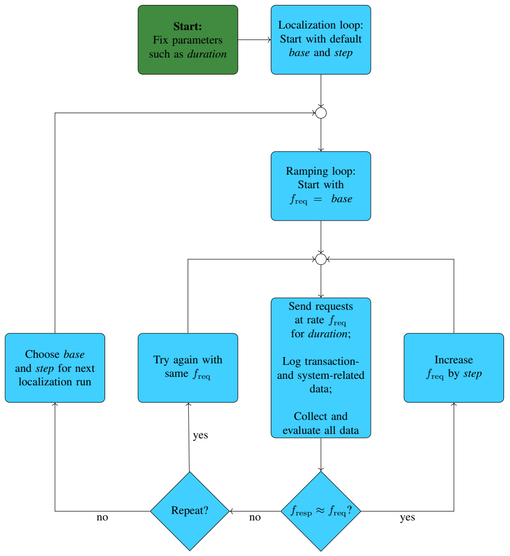
Start:
Fix parameters such as duration
Localization loop: Start with default base and step
Ramping loop: Start with $f_{\text{req}} = base$
Choose base and step for next localization run
Try again with same f_{req}
Send requests at rate f_{req} for duration;
Log transaction- and system-related data;
Collect and evaluate all data
Increase f_{req} by step
yes
Repeat?
no
no
$f_{\text{resp}} \approx f_{\text{req}}$?
yes

Figure 3. The benchmarking process flowchart.

ramping series (and significantly smaller values in kind the last series failed), and *repetition runs* in which we use the last value for base and step from a series of localization runs multiple times to obtain a statistically valid result. During all measurements, the DLPS uses established software for monitoring resources on both nodes and clients such as overall and single-core CPU usage (mpstat), memory (vmstat), disk utilization (iostat), network latencies (ping), and network traffic (ifstat).

4.2. Technical architecture

The DLPS framework consists of three Python packages to coordinate nodes and clients, trigger benchmarking functionalities, and aggregate and structure corresponding data. Fig. 4 shows the technical architecture of the DLPS.

The package BlockchainFormation contains configurable and fully automatic startup-, restart-, and shutdown scripts for different permissioned blockchains. Since we want to offer highly customizable benchmarks, our NodeHandler launches and connects

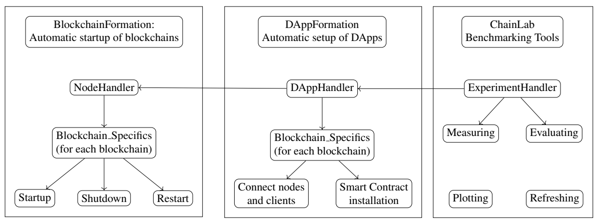
BlockchainFormation: Automatic startup of blockchains
NodeHandler
Blockchain_Specifics (for each blockchain)
Startup
Shutdown
Restart
DAppFormation Automatic setup of DApps
DAppHandler
Blockchain_Specifics (for each blockchain)
Connect nodes and clients
Smart Contract installation
ChainLab Benchmarking Tools
ExperimentHandler
Measuring
Evaluating
Plotting
Refreshing

Figure 4. Architecture of the DLPS.

to virtual machines in Amazon Web Services (AWS) such that we can create and benchmark DLT networks of arbitrary size. However, the functionality could easily be extended to other cloud service providers or computing clusters, since the DLPS only relies on running ssh and scp-commands on the nodes. We have already implemented private Ethereum (with Geth and Parity client), Fabric, Indy, Quorum, and Sawtooth. Our NodeHandler integrates all these specifics and orchestrates the network startup.

The second repository, DAppFormation, consists of the required client-side functionalities. They serve to implement the blockchain-specific parts of the client setup, namely connecting client and blockchain networks. Moreover, for every blockchain, the associated clients wrap SC requests such that we can trigger the sending of requests at a specific rate f_{req} with a single script which is independent of the underlying blockchain.

All immediate functionalities for performance evaluation, in turn, are then integrated in ChainLab. Due to our wrappers in DAppFormation, we can implement the benchmarking logic (see Fig. 3) as well as the evaluation of our measurements in a blockchain-agnostic way. This design makes our benchmarking framework applicable to general DLT with a client-node architecture. Integrating another blockchain merely requires setting up startup scripts for both nodes and clients as well as the wrapper which serves to make the SC method calls from the benchmarking script blockchain-agnostic. The modular approach also allows for applying changes or extensions to the current frameworks at minimal effort and immediate impact on all tests within the framework. Finally, to make the data tidy and accessible, we stick to Wickham [28] and provide a method to aggregate all measurements and their corresponding setup parameters into a single CSV file.

5. Performance Characterization

5.1. Experimental setup

To illustrate the universal applicability of the DLPS, we applied our framework to ten different network architectures for the five DLT presented in Sec. 2. We investigated network sizes of 1, 2, 4, 8, 16, 32, and 64 nodes, and used 32 clients distributed equally among the nodes in each setup. As workload, we chose a simple fundamental functionality on DLs, namely writing a single key-value pair into the ledger. This is the writeData basic workload mentioned in Sec. 3, with a key space of size 10^4 and a value space of size 10^7.

We generally used cloud instances from the AWS m5 series because they balance CPU, memory, and network capabilities, all of which we consider relevant for a DLT node. For the nodes, we decided to use m5.2xlarge instances in AWS in all experiments, which have 8 vCPUs and 16 GiB of RAM. For the configuration of the generic benchmarking process flow, we generally specified the following settings (see Sec. 4). However, in some cases we had to make minor modifications to account for particularities of the setup, but these adaptions do not bring any bias to the results to the best knowledge of the authors.

- The duration of a measurement with fixed f_{req} was 20 s.
- To decide whether $f_{\text{req}} \approx f_{\text{resp}}$, we chose $\delta = 0.05$ in (4). We also specified a minimum coefficient of determination R^2 of 0.98, except for cases (generally, low throughput or large blocks) where the staircase was naturally very coarse. Generally, an R^2 value below our threshold occurred only rarely.
- We allowed for two retries in case $f_{\text{resp}} \not\approx f_{\text{req}}$ before breaking the ramping loop, and three consecutive rampings were required for a valid ramping series.
- In the localization runs, we used a success base rate of 80 % of the last ramping series' maximum throughput, with a step size of 4 %, and 50 % of resp. 4 % in case of a failure.
- We conducted three localization runs, followed by three repetition runs (see Sec. 4).

All remaining parameters which completely determine the benchmarking process are included in the config files for the benchmarks and available in the DLPS repository, among the results associated with the measurements presented here. Thereby, we address one of the key issues that led to the implementation of the DLPS: The benchmarking process is transparent, and by repeating the experiments with the provided configuration files, our results are completely reproducible.

5.2. Experiment results

With the results from the experiments presented in Sec. 5.1, we can give a thorough scalability analysis for these DLT from a single standardized tool. Also, the range of investigated network sizes and in particular networks with 64 nodes are – to the best of our knowledge – unique in existing literature. Fig. 5 depicts our results for maximum sustainable throughput $\hat{f}$. Only the results of the three repetition runs went into the evaluation of each experiment. Each data point is the mean of these three results, with the shaded area specifying standard deviation.

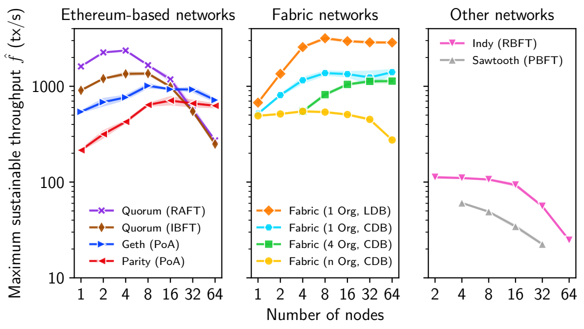

Figure 5. Results of our scalability analysis for different DLT networks

We benchmarked private Ethereum with Geth and Parity PoA consensus and Quorum with IBFT and RAFT consensus. For Geth, Parity, and IBFT, we selected the minimum possible block time of 1 s, and for RAFT the default block time of 50 ms. Overall, Quorum with RAFT and IBFT consensus perform best for small networks, while Geth and Parity scale better for larger networks. The highest maximum sustainable throughputs $\hat{f}$ are 2 363±4 tx/s for 4-node Quorum with RAFT consensus and 1 350±60 tx/s for 4-node Quorum with IBFT consensus. For Geth, we observe maximum $\hat{f}$ of 1 010±50 tx/s at 8 nodes, while Parity reaches its maximum at 16 nodes with 710±70 tx/s. For all network sizes, RAFT has a higher $\hat{f}$ than IBFT, and Geth has higher $\hat{f}$ than Parity. The latter result makes sense because Aura consensus is more complex than Clique consensus in Geth. Our results on throughput for Parity are significantly higher than these of Dinh et al. [24], who obtain a peak throughput of 46 tx/s and a latency of 3 s for Parity on an 8-node 8-client Parity network using Blockbench. On the other hand, Baliga et al.'s [23] results on Quorum using Caliper are similar to our findings with the DLPS: Although they do not reach $\hat{f}$ because they had a limited number of clients and therefore bounded f_{req}, their evaluation suggests that throughput with RAFT is over 2 000 tx/s, and for IBFT around 2 000 tx/s. They also use hardware similar to the m5.2xlarge instances in our experiments, however, their cores had 3.6 GHz while the m5 instances only allow up to 3.1 GHz, indicating that the differences might stem from the experimental setup.

As described in Sec. 2, Fabric separates the consensus layer (ordering service) from the rest of the system and offers customizable consensus options by specifying a SC's endorsement policy. Therefore, no canonical n-node setup exists to compare to the other DLT benchmarked. Consequently, we defined different architectures[3]: (a) 1-Org, where we impose a trivial endorsement policy meaning that only a single node has to simulate the outcome of a transaction, and the SOLO orderer, (b) 4-Org, where, according to our endorsement policy, at least 4 nodes have to simulate and sign each transaction, and a separate 4-node RAFT network as ordering service, (c) n-Org, where all nodes need to endorse every transaction, and the ordering service is an equally sized RAFT network. In our opinion, the 4-Org case might be quite close to what one would implement in production, whereas 1-Org is the ideal case for performance but not really distributed, and n-Org is a worst case for performance and maybe the closest to the other networks in terms of consensus. We generally used CouchDB (CDB) as database because its support for complex queries makes it very suitable for enterprise DLT solutions. To compare it with LevelDB (LDB), we investigated the 1-Org setup in both cases. In the 1-Org settings, we disabled encrypted messaging among the network (TLS), while enabling it in the 4-Org and n-Org case. Our first observation is that throughput of the 1-Org setup with LDB, which peaks at 3 180$\pm$80 tx/s for n=8, is approximately twice the performance of the 1-Org setup with CDB for large n, which reaches a maximum of $\hat{f} = 1\,410 \pm 90$ tx/s for n=64. Spot checks with better hardware and LDB also showed that our results are compatible with the ones of Androulaki et al., who measured more than 3 000 tx/s in these setups with a smart contract similarly complex as the writeData we used. Moreover, for both 1-Org setups and the 4-Org setup, we see that the overall shape of the curve seems like a saturation curve. Indeed, the ordering service does not change when we increase the number of nodes, but the number of potential endorsers increases. Hence, the endorsing tasks are split among more workers. On the other hand, the maximum download and validation rate for a node, which receives the blocks from the ordering service, poses an upper bound on throughput, which is independent on n as long as the ordering service itself is not the bottleneck. In contrast, we do not observe that saturation-like behaviour for the n-Org setup since here, the total endorsement workload increases like the number of nodes. This also explains why for small n, $\hat{f}$ is quite stable.

As highlighted in Sec. 2, Indy does not support arbitrary SCs, so we could not define a writeData workload as straightforward as for the other frameworks. We decided to define issuance of a credential schema consisting of a single, random-number key as writeData transaction because this is a very simple on-chain write operation. For Indy and Sawtooth, we observe that $\hat{f}(n)$ is generally decreasing, which meets expectations for three-round consensus in RBFT resp. PBFT. Indy shows almost constant $\hat{f}$ for n $\leq$ 16, with a maximum of $\hat{f} =$ 112$\pm$2 tx/s for n=2. For n $\geq$16, $\hat{f}$ approximately halves for each subsequent doubling of the number of nodes, suggesting that our results truly display the overhead of BFT-like consensus. The latency of Indy is around 3 s. We found no other benchmarks on Indy to compare with in our literature review and also when explicitly searching for performance results on Indy.

Although we also integrated Sawtooth with PoET and RAFT consensus in the DLPS, we only systematically benchmarked Sawtooth with PBFT consensus as RAFT setup turned out to crash frequently, and spot checking experiments for PoET consensus suggested that $\hat{f}$ is well below 30 tx/s in this case. PBFT consensus in Sawtooth requires at least 4 nodes, and for n=64, we could not manage to set up a network that was running stable. We obtained a maximum of only $\hat{f}$ = 60.5$\pm$0.5 tx/s for n=4, and $\hat{l}$ grows from around 1.6 s for n=4 to 3 s for n=32. We noticed considerable performance improvements alongside the update to version 1.2 in October 2019: With version 1.0, we had never observed more than 8 tx/s, which is close to what Shi et al. [18] measured for version 1.0. Moreover, frequent crashes had made systematic benchmarks almost impossible with this version.

6. Conclusion and Future Work

In this paper, we highlighted the current need for a transparent and universal framework to characterize the performance of DLT. To address this, we designed and implemented the DLPS for determining key metrics and benchmarking applications end-to-end. The framework enables the creation of well-defined benchmarks (see [29]) due to four reasons. It is *transparent*, because we give clear definitions of latency and throughput as well as a description and implementation of the algorithm for measuring these. We offer *configurability* by supporting the automatic benchmarking of highly customizable architectures and parametrizing the benchmarking algorithm. By publishing the results of our measurements as well as our source code, one can easily reproduce the results (*repeatability*). Finally, the DLPS is *extensible* as its modular implementation (see Fig. 4) allows the addition of new DLT as well as adaptions of the benchmarking logic with reasonable effort. To demonstrate the

[3]Org is short for organizations: Due to the execute-order-validate paradigm, there are not only nodes (called peers in Fabric), but also clients and orderers that have an important role. It is intuitive to think about collections of these as orgs, e.g., one org might run an orderer, 4 peers and 8 clients, then they can trust a transaction if only one of their peers endorsed it, and contribute their own orderer to the CFT ordering service

applicability of the DLPS to a large subset of DLT, we conducted an in-depth study of performance and scalability properties of ten architectures for five permissioned DLT on a broad range of network sizes (cf. Fig. 5). We plan to utilize the extensibility of our framework to develop the DLPS further, including the integration of additional DLT, such as Corda, and maintaining support for updates on the DLT that we have already included. We will further extend the parameters that one can choose, and provide further tools for evaluating the gathered data, hence further reducing the hurdle to establishing the DLPS as a standard tool to measure DLT performance.

Acknowledgement

We thank Matthias Babel, Emil Djerekarov, Tobias Guggenberger, Dennis Jelito, and Lukas Müller for their valuable support.

References

[1] M. Iansiti and K. R. Lakhani, "The truth about blockchain," *Harvard Business Review*, vol. 95, no. 1, pp. 118–127, 2017.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. `http://bitcoin.org/bitcoin.pdf`.

[3] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," 2014. `https://github.com/ethereum/wiki/wiki/White-Paper`.

[4] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[5] T. Jensen, J. Hedman, and S. Henningsson, "How TradeLens delivers business value with blockchain technology," *MIS Quarterly Executive*, vol. 18, no. 4, 2019.

[6] A. Rieger, F. Guggenmos, J. Lockl, G. Fridgen, and N. Urbach, "Building a blockchain application that complies with the EU general data protection regulation," *MIS Quarterly Executive*, vol. 18, no. 4, 2019.

[7] B.-J. Butijn, D. A. Tamburri, and W.-J. v. d. Heuvel, "Blockchains: A systematic multivocal literature review," *ACM Computing Surveys*, vol. 53, no. 3, 2020.

[8] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016. `https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf`.

[9] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *International workshop on open problems in network security*, pp. 112–125, Springer, 2015.

[10] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 SIGSAC conference on computer and communications security*, pp. 3–16, ACM, 2016.

[11] J. Sedlmeir, H. U. Buhl, G. Fridgen, and R. Keller, "The energy consumption of blockchain technology: Beyond myth," *Business & Information Systems Engineering*, vol. 62, no. 6, pp. 599–608, 2020.

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

[13] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[14] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, pp. 173–186, 1999.

[15] R. Saltini, "Correctness analysis of IBFT," 2019. `https://arxiv.org/pdf/1901.07160`.

[16] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "Rbft: Redundant byzantine fault tolerance," in *33rd International Conference on Distributed Computing Systems*, pp. 297–306, IEEE, 2013.

[17] S. De Angelis, L. Aniello, F. Lombardi, A. Margheri, and V. Sassone, "PBFT vs Proof-of-authority: applying the CAP theorem to permissioned blockchain," 2017.

[18] Z. Shi, H. Zhou, Y. Hu, S. Jayachander, C. de Laat, and Z. Zhao, "Operating permissioned blockchain in clouds: A performance study of hyperledger sawtooth," in *18th International Symposium on Parallel and Distributed Computing*, pp. 50–57, IEEE, 2019.

[19] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, 2016.

[20] E. Androulaki *et al.*, "Hyperledger Fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, p. 30, ACM, 2018.

[21] "Hyperledger Fabric repository." `https://github.com/hyperledger/fabric`.

[22] "Hyperledger Indy repository." `https://github.com/hyperledger/indy-node`.

[23] A. Baliga, I. Subhod, P. Kamat, and S. Chatterjee, "Performance evaluation of the quorum blockchain platform," 2018. `http://arxiv.org/abs/1809.03421`.

[24] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K. Tan, "Blockbench: A framework for analyzing private blockchains," 2017. `http://arxiv.org/abs/1703.04057`.

[25] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, "Performance evaluation of blockchain systems: A systematic survey," *IEEE Access*, vol. 8, pp. 126927–126950, 2020.

[26] R. Jain, *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing, Wiley, 1991.

[27] J. Ousterhout, "Always measure one level deeper," *Communications of the ACM*, vol. 61, no. 7, pp. 74–83, 2018.

[28] H. Wickham *et al.*, "Tidy data," *Journal of Statistical Software*, vol. 59, no. 10, pp. 1–23, 2014.

[29] J. Gray, *Benchmark handbook: for database and transaction processing systems.* Morgan Kaufmann Publishers Inc., 1992.