Article

Latency Analysis of Blockchain-Based SSI Applications †

Tamas Pflanzner , Hamza Baniata and Attila Kertesz *

Software Engineering Department, University of Szeged, 6720 Szeged, Hungary

* Correspondence:keratt@inf.u-szeged.hu

† This paper is a revised and extended version of the conference paper Latency Assessment of Blockchain-based SSI Applications Utilizing Hyperledger Indy, originally presented at the 12th International Conference on Cloud Computing and Services Science—CLOSER. INSTICC, SciTePress, Online, 27–29 April 2022.

Abstract: Several revolutionary applications have been built on the distributed ledgers of blockchain (BC) technology. Besides cryptocurrencies, we can find many other application fields in smart systems exploiting smart contracts and Self Sovereign Identity (SSI) management. The Hyperledger Indy platform is a suitable open-source solution for realizing permissioned BC systems for SSI projects. SSI applications usually require short response times from the underlying BC network, which may vary highly depending on the application type, the used BC software, and the actual BC deployment parameters. To support the developers and users of SSI applications, we present a detailed latency analysis of a private permissioned BC system built with Indy and Aries. To streamline our experiments, we developed a Python application using containerized Indy and Aries components from official Hyperledger repositories. We deployed our experimental application on multiple virtual machines in the public Google Cloud Platform and on our local, private cloud using a Docker platform with Kubernetes. We evaluated and compared their performance with the metrics of reading and writing response latency. We found that the local Indy ledger reads 30–50% faster, and writes 65–85% faster than the Indy ledger running on the Google Cloud Platform.

Keywords: blockchain; Self Sovereign Identity; Hyperledger Indy; latency analysis

check for updates

Citation: Pflanzner, T.; Baniata, H.; Kertesz, A. Latency Analysis of Blockchain-Based SSI Applications. *Future Internet* **2022**, *14*, 282. https://doi.org/10.3390/fi14100282

Academic Editors: Xu Wang, Bin Shi and Yili Fang

Received: 31 August 2022
Accepted: 27 September 2022
Published: 29 September 2022

1. Introduction

Distributed ledger technology (DLT) is a data storage technology managed by multiple nodes in a decentralized way. Blockchain (BC) [1] is a special case of DLT, where blocks contain the data, and every block references the hash of the previous block, which makes it more temper-proof. This technology is utilized for enhancing trust in distributed computing applications and managing distributed ledgers (DL). Applications integrated with permissionless BCs benefit from high levels of security and trust, where BCs provide a fully immutable log of transaction (TX) history without the control of a central authority. Similarly, permissioned BCs utilize a distributed trusted third party for providing services typically requested from a central trusted third party. In applications that require trusted BC nodes, permissioned BCs, typically referred to as Miners, Minters, Verifiers, or Validaters, are used to maintain a consistent and trusted DL. Initially, those participants apply to become BC nodes to a committee that votes for acceptance. Permissionless BC solutions have been designed first for realizing digital cryptocurrencies [2], but nowadays they address a wide variety of environments, such as document management [3], smart applications for eHealth [4], or Internet of Vehicles [5]. In such an access model, any party can join the BC network and collaborate in validating and confirming new pieces of data added to the DL, without a network size limit. For applications that require a distributed yet controlled trusted third party, permissioned BC solutions are suitable, where a limited number of verifiers vote for appending data into the DL and for accepting new verifiers. Examples of applications that require such a BC model include distributed voting [6], Self Sovereign Identity (SSI) [7], and credential management [8].

Specifically, classical internet applications lack an identity layer [9]. Web applications, for instance, use usernames and passwords to identify their clients, yet the use of these attributes is typically limited to the specific contexts of individual applications. Such a framework is considered inconvenient and implies several security issues (e.g., identity theft and fake users) [10]. Global identity [11] providers, such as Google or Meta, have attempted to develop their access framework to enhance usability. However, data leaks are still argued to remain an open issue [12]. Additionally, the loss of clients' control over their private data, which is typically saved on centralized servers, implies trust issues. Some solutions allow users to request the deletion of their data, in order to comply with the European General Data Protection Regulations (GDPR), yet users have to also trust the provider to delete them. SSI concepts enable users to anonymously identify themselves, and verify some private attributes about them, while maintaining full control and permissions over their private data.

Although permissioned BCs have shown unprecedented abilities to maintain DLs with high security and trust measures, compared with non-BC-based DLs, they still suffer from high latency compared with centralized solutions. Furthermore, different permissioned BC solutions are designed differently according to their application and the different layering of the solution. This usually causes a fluctuation of latency measures among different permissioned BC solutions and even for differently parameterized solutions. Such issues mainly appear due to several system properties and limitations, such as the minimum time needed to reach a consensus on a piece of data (i.e., the finality time), the average transmission delay between network entities, the total network size, and the average number of neighbors per miner [13]. This being said, researchers and practitioners tend to model their applications and test several BC solutions for a decision to be made regarding the deployment of a specific BC framework. As many solutions are being continuously proposed, it must be a burden to utilize and test several frameworks before assuring that the application requirements are fulfilled by a selected BC framework.

Blockchain technology is especially useful in applications where the database is managed by a community and it is important for the data to be immutable. One of the most limiting factors is the speed of the blockchain, which can be measured by latency or throughput. A good example is Bitcoin, which is only capable of seven transactions per second. This is remarkably slow compared with the performance of traditional databases, reaching thousands of transactions per second. For a transaction to be confirmed, it takes around 10 min, so it is clear that the performance problem requires improvements to make the technology useful for more applications, and it implicitly requires performance measurements.

As several recent works have utilized Hyperledger Indy [14], and several projects are currently investigating its deployment for their SSI applications, the aim of this work is to analyse Hyperledger Indy in terms of latency. To reach these aims, we propose a deployment architecture using Indy and Aries suitable for SSI applications. We have developed a Python application with containerized Indy and Aries components, building on the official open-source Hyperledger projects. This work is an extension of our previous work [15], where we deployed and examined our application in multiple virtual machines in the Google Cloud Platform. Now, we revise and extend these experiments with detailed performance evaluation and comparison with our locally deployed application in the private Kubernetes Cluster, using different scenarios with a variable number of BC nodes and TX arrival rate. An example of the utilization of these results may be to help decide if it is practicable to use Indy for a latency-sensitive application.

The remainder of this paper is as organized as follows: Section 2 discusses recent related works. Section 3 presents a brief architectural and technical background of Hyperledger Indy, and details the considered architecture and the research methods. Section 4 presents the results we obtained by running our experiments both in a public and private cloud, which are discussed in Section 5. Finally, Section 6 concludes the paper.

2. Related Works

EBSI (https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/EBSI, accessed on 26 September 2022) and BCDiploma (https://www.bcdiploma.com/en-GB, accessed on 26 September 2022) are examples of services built for providing verifiable credential (VC) supportive platforms utilizing decentralized identifiers (DIDs). The BC acts as the trusted third party (TTP) where system entities save their data. However, the standard recommendation of W3C (https://www.w3.org/TR/vc-data-model/#introduction, accessed on 26 September 2022) and the solutions following it did not consider the issuer accreditation issue, assuming any entity should be able to own a DID and issue any type of VCs. The BCDiploma platform directly saves all issued VC hashes on the BC, which means that they can never be deleted, and it uses a PoW-based BC with a linear DL model. Additionally, an issuer accreditation service is not provided. Specifically, EBSI utilizes the general purpose Hyperledger Fabric (https://www.hyperledger.org/use/fabric, accessed on 26 September 2022) which uses the Raft Consensus algorithm.

In [16,17], a BC-based data management system that could be used for the Global Legal Entity Identifier System (GLEIS) is proposed. Towards realizing this, the authors utilized Hyperledger Indy and their previously proposed GraphChain [18]. The implementation and utilization model of the proposed solution were presented, in which some challenges were faced, such as the limited message size of Indy.

Bhattacharya et al. [19] examined certain scenarios of personal data disclosure via credential exchanges between different identities and the risks of man-in-the-middle attacks in a BC-based identity system utilizing Hyperledger Indy. On the basis of the findings, the authors proposed enhancing Indy with a novel attribute sensitivity score model for SSI agents to ascertain the sensitivity of attributes shared in credential exchanges. Additionally, they proposed a method for mitigating man-in-the-middle attacks between peer SSIs. Finally, they proposed a novel quantitative model for determining a credential issuer's reputation based on the number of issued credentials in a window period, which is then utilized to calculate an overall confidence level score for the issuer.

Malik et al. [20] proposed an architecture for decoupling identities and trade activities on BC-enabled supply chains, namely TradeChain. The authors demonstrated the feasibility of TradeChain by implementing a proof of concept on Hyperledger Fabric and Indy. The limited latency evaluations provided in this study are within the ranges we obtained by our experiments.

Prakash et al. [21] proposed that Indy and Aries agents can be implemented and utilized to realize a connected vehicle information network solution. The authors described several challenges they faced when utilizing those two projects, including the high latency and scalability limitations. However, the paper did not provide specific latency measures and settled for a description of the solution.

A detailed and comprehensive study related to different BC platforms, including Indy and Aries, is presented in [22]. Here, well-documented tutorials and implementations can be found, as well as descriptive methods and algorithms for different consensus and block confirmation approaches. Additionally, architectural and low-level details have been differentiated between the studied BC platforms.

Usually, the performance evaluations [23] can be classified into two categories: analytical modelling and empirical analysis. The empirical evaluation includes monitoring, benchmarking, experimental analysis, and simulation.

The diversity of consensus algorithms and APIs makes it impossible to use traditional benchmarking tools for blockchains. The three most popular benchmarking tools for blockchains are Hyperledger Caliper [24], DAGbench [25], and Blockbench [26]. Blockbench, by default, can work with Ethereum, Parity, HLF, and Quorum, but it is possible to implement adapters to use other platforms. It measures throughput, latency, scalability, and fault-tolerance in private blockchains. The main focus of Hyperledger Caliper is Hyperledger blockchains, such as Hyperledger Fabric, Sawtooth, Iroha, Burrow, and Besu. DAGbench is dedicated to benchmarking DAG distributed ledgers such as IOTA, Nano, and Byteball. Scalability, throughput, latency, resource consumption, transaction fee, and transaction data size can all be measured.

Though these works investigate SSI issues with Indy and Aries, none of them has performed such detailed latency analysis with different system settings as we have, concerning network sizes and TX arrival rates.

3. The Proposed Evaluation Architecture Using Indy

Hyperledger Indy [14] is an open-source project, administered by the Linux foundation, which aims to support verifiable credential systems based on the public-permissioned BC approach. The project implementation provides a platform for decentralized identifiers rooted in BCs, or other DLs, so that they are inter-operable across administrative domains, applications, etc., and are usable for validating VCs. The project deploys privacy-preserving mechanisms such as zero-knowledge proofs and digital signatures. Additionally, Indy takes good care of what is saved on-chain, aiming for adherence with the GDPR. The Plenum Consensus Algorithm is used in Indy, which is a special-purpose Redundant Byzantine Fault Tolerance Consensus Algorithm [27]. A live example of an Indy-based solution is the Sovrin BC [28], providing a general purpose, global, DID-supportive platform.

As depicted in Figure 1, Indy requires a small group of miners that serves as a distributed trusted third party. Indy miners, or validators, are computers administered by publicly known parties, added by a voting scheme between miners themselves. The miners' main purpose is to maintain the liveness and consistency of the DL by accepting new TXs. Several types of BCs consist of different types of TXs that need to be saved on-chain. Three of those BCs are depicted in the figure: the Domain TXs BC, the Pool TXs BC, and the Config TXs BC. On these BCs, public data about the admins of miners, TXs meta data, and network configurations are saved, respectively.

Indy miners are added to the system in a decentralized fashion, if they fulfil certain conditions. Indy allows this by utilizing a voting scheme, where available miners accept or decline a new miner application. A steward miner is an authorized Indy node that is needed to be able to add new nodes to the network. Only one node can be added per steward, so an equal number of stewards and miners need to be present. The available sample code at the official Indy repository creates four steward nodes. To add the first four nodes of the network, four wallets and four pairs of PKI keys need to be created. However, a genesis block is created along with the initial stewards' DIDs at the initialization phase of the system. Here, the stewards' data are generated by a seed defined by the developer. Then, NYM requests are sent by the wallet admins to the network of stewards who accept/reject the TXs. Once accepted, a new wallet is created and a block is added to the BC. Once a minimum of four miners are active, the BC network can accept new DID and schema TXs from end-users. Endorsers and agents are end-user entities of the system that should be implemented using the Aries scripts. The main difference between these two types of end-users is that agents are only authorized to read on-chain data, while endorsers are authorized to both read and write.

To realize the deployment of Indy and Aries nodes, we designed and developed a validation architecture, as depicted in Figure 1. We implemented an Aries agent that is capable of connecting to an Indy network, and of generating sample data (i.e., TXs) to be submitted to it. We extracted the initial codes from the Indy-SDK repository, which is implemented using Python. We used the "build_nym_request" method to create a NYM (short for Verinym) request. Verinym is a Hyperledger Indy-specific term that is used for the identification role of the trust anchor. The request can be submitted by the "sign_and_submit_request" method. For the read measurements, we used the "build_get_nym_request" method with the "submit_request" call to obtain the results. Our agent is capable of sending up to 250 TX requests per second, using a multi-threaded approach, to examine the system's read and write processing speeds. The sample TXs are constructed similarly to the standard TX formats generated by an Aries agent. However, the codes available at the Aries official repository allows for only 20 TXs/s; thus, we had to implement our own agent.

We implemented Indy node scripts that are suitable to be run individually on different machines. That is, the available modules we found, at the official Indy repository, run four nodes on a single machine, and they were not ready for production. As a demonstration of the requirement of the network to control different nodes by different admins, we also utilized the Admin UI from the VON network repository, which we configured to connect to the created Indy network.

All our implemented application components are containarized using Docker, which makes them easier to deploy for present and future works. Our implemented application is publicly available at Github (https://github.com/sed-szeged/IndyPerf, accessed on 26 September 2022) (see Supplementary Materials), and its main management scripts are as follows:

- indy_config.py: This file contains the configuration to run the Indy node, i.e., the name of the network and the installation parameters.
- requirements.txt: This file contains system dependencies. We used Flask to create a web service, which displays the genesis file extracted from the container, according to which the admin application could connect.
- indy-node-start.sh: This script starts the server.py in a container to create a BC node.
- create_steward.py: This script can be used to add new stewards, thus adding new nodes to the system.
- test_transactions.py: This script can be used to perform the experiments (to be detailed in the next section). It creates a connection with the BC network, and sends the specified TXs in multiple threads.

We performed several scenario runs using our multi-threaded Aries agent with different TX arrival rates (1–250 TX/s) and different network sizes (4–8). Each scenario run was repeated 10 times, for a total BC height of 10–2500 blocks. This way, an average latency for different heights of the BC, with different network sizes, can be extracted. We measured the elapsed time for a TX to be sent, processed, and responded to. We have collected the minimum and maximum latency measures, and computed their average values. We separately examined the read and write speeds to guarantee the accuracy of our measurements. All nodes were deployed and run individually on different virtual machines, located in different regions of the Google Cloud Platform. We used E2-medium VMs (up to 3.8 GHz, 2 vCPUs, 4 GB memory) running Ubuntu 16.04 OS. Visualizations of the evaluation architecture and installation steps are provided in Figure 2.

We also performed similar read and write latency measurements in a local Docker four-node environment, with increased TX arrival rates (1–500 TX/s), to be able to compare the results of a system with an optimal network performance with a more real-life scenario, where the nodes are in different regions. The architecture of the Indy ledger in the local Docker environment can be seen in Figure 3.

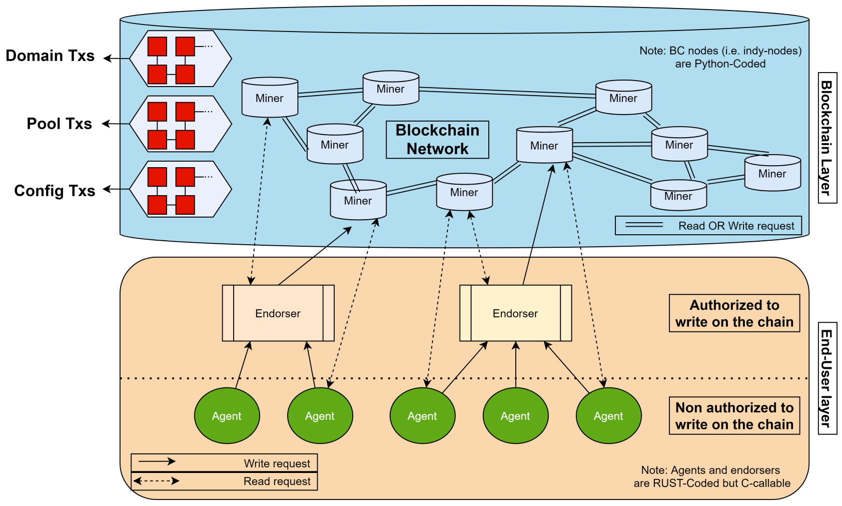
Domain Txs
Pool Txs
Config Txs
Note: BC nodes (i.e. indy-nodes) are Python-Coded
Miner
Blockchain Network
Read OR Write request
Blockchain Layer
Endorser
Authorized to write on the chain
Non authorized to write on the chain
Agent
Write request
Read request
Note: Agents and endorsers are RUST-Coded but C-callable
End-User layer

Figure 1. Our proposed blockchain architecture based on Indy [15].

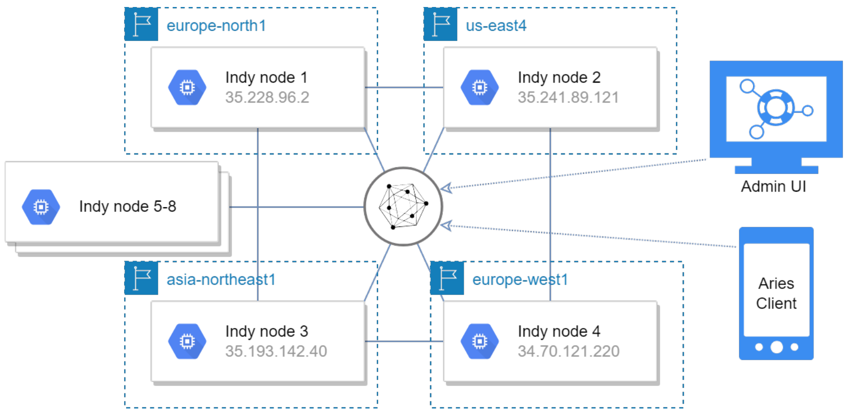

Figure 2. Evaluation architecture in the Google Cloud Platform.

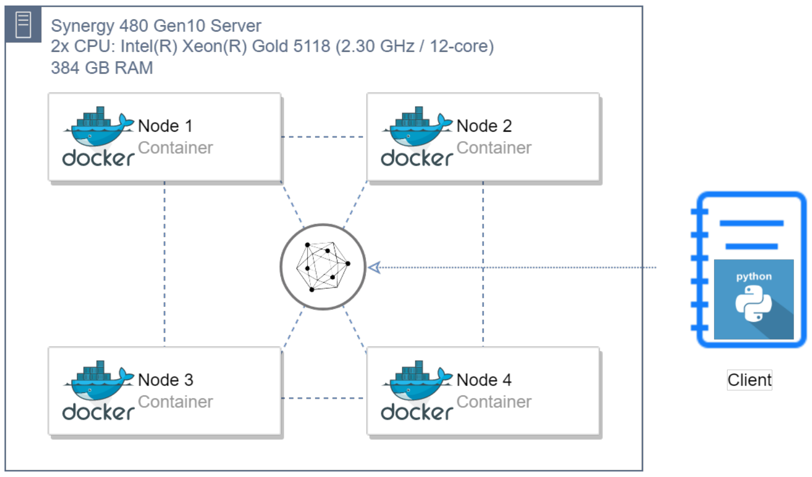

Figure 3. Evaluation architecture in a private cloud.

4. Evaluation Results

In this section, we present the latency measures we obtained by deploying our scripts on multi-regional cloud servers and also on a separate local server.

4.1. Four-Node Measurements in a Public Cloud

The detailed read and write latency measurements are provided in Table 1 (partly performed in our previous work [15]). We computed those measures for an Indy network that consists of four nodes. We found that the size of the dataset stored on-chain does not affect the speed, as the ten measures we collected per scenario were very close to each other. However, we can see a proportional relationship between the TX arrival rate and the average latency for both read and write TXs. It is also worth noting that depending on the TX arrival rate, the average write latency fluctuates from 2.7 to 6.3 s per TX. On the other hand, the average read latency fluctuates from 0.088 to 1.56 s per TX, and also heavily depends on the TX arrival rate.

4.2. Eight-Node Measurements in a Public Cloud

The detailed read and write latency measurements are provided in Table 2. We computed those measures for an Indy network that consists of eight nodes. We also checked if the size of the dataset stored on-chain affected the latency, and came to the same conclusion—that it does not. We can also see that increasing the arrival rate increases the average latency for both types of requests. Depending on the arrival rate, it is observable in the table that the average write latency fluctuates from 2.6 to 10.4 s per TX. The same dependency can be seen for the average read latency, which fluctuates from 0.12 to 2.49 s per TX.

Table 1. Latency measures, in seconds, for read (R) and write (W) TXs where network size = 4.

TX	**Arrival Rate (TXs/s)**					
Type	***1***	***50***	***100***	***150***	***200***	***250***
Min (W)	1.1014	1.1014	1.1014	1.1014	1.1014	1.1014
Max (W)	2.9832	2.9138	2.9490	5.8131	6.8542	13.3811
Avg (W)	2.7388	2.5284	2.2197	4.4763	5.3511	6.3534
TX	**Arrival Rate (TXs/s)**					
Type	***1***	***50***	***100***	***150***	***200***	***250***
Min (R)	0.0108	0.0671	0.1058	0.0841	0.2469	0.0800
Max (R)	0.2160	1.1387	1.4655	2.2378	2.5302	4.7086
Avg (R)	0.0881	0.3961	0.5815	0.8964	1.1408	1.5562

Table 2. Latency measures, in seconds, for read (R) and write (W) TXs where network size = 8.

TX	**Arrival Rate (TXs/s)**					
Type	***1***	***50***	***100***	***150***	***200***	***250***
Min (W)	0.2614	1.7064	1.5032	1.3114	1.2914	1.5289
Max (W)	2.9949	2.9607	5.2011	8.5050	7.4365	16.9094
Avg (W)	2.6118	2.5993	2.5347	4.8506	5.5883	10.4323
TX	**Arrival Rate (TXs/s)**					
Type	***1***	***50***	***100***	***150***	***200***	***250***
Min (R)	0.0092	0.0567	0.0927	0.0667	0.0350	0.0219
Max (R)	0.7147	1.1744	1.4469	1.9189	2.4340	5.6863
Avg (R)	0.1203	0.3718	0.5430	0.8163	1.0652	2.4928

4.3. Four-Node Measurements in a Private Cloud

For the experiments, we used our local, private cloud deployed on a Synergy 480 Gen10 Server, which has two CPUs; both of them are Intel(R) Xeon(R) Gold 5118 with 12 cores running on 2.30 GHz. The server has 384 GB RAM. The Indy ledger consisted of four Docker containers, created from the official Indy-SDK images, using the official

Docker Compose description file. The test script was written in Python and executed in a Jupyter notebook, just like the official Indy demo commands.

Table 3 provides the detailed read and write latency measurements with a four-node network. We also evaluated an eight-node local network; the results can be found in Table 4. We found once again that the size of the dataset stored on-chain does not affect the latency. We can also see that increasing the arrival rate increases the average latency for both types of requests. It is observable in the table that the average write latency fluctuates from 1.8 to 7.1 s per TX in the four-node network, and from 2.0 to 7.7 s per TX in the eight-node network, depending on the arrival rate, as Figure 4 shows. On the other hand, the average read latency fluctuates from 0.03 to 1.4 s per TX in the four-node network, and from 0.03 to 2.4 s per TX in the eight-node network, as shown in Figure 5.

Table 3. Latency measures, in seconds, for read (R) and write (W) TXs in a local Docker environment where network size = 4.

TX	**Arrival Rate (TXs/s)**										
Type	***1***	***50***	***100***	***150***	***200***	***250***	***300***	***350***	***400***	***450***	***500***
Min (W)	0.1011	1.6937	1.8288	1.9917	4.4555	4.8983	5.1942	5.5093	5.7745	6.0929	6.3293
Max (W)	2.0067	2.1450	2.2595	4.3912	5.0304	5.1658	5.6114	6.2726	6.5005	6.9145	10.1340
Avg (W)	1.7733	1.7860	1.9137	3.9115	4.7391	5.0067	5.3868	5.8393	6.1832	6.5251	7.1020
TX	**Arrival Rate (TXs/s)**										
Type	***1***	***50***	***100***	***150***	***200***	***250***	***300***	***350***	***400***	***450***	***500***
Min (R)	0.0178	0.0856	0.2053	0.2155	0.4517	0.6864	0.6904	0.7717	0.8427	1.0742	1.1358
Max (R)	0.0474	0.2229	0.4658	0.6657	0.8918	1.1936	1.2290	1.4729	1.7036	1.9957	2.1096
Avg (R)	0.0266	0.1401	0.2849	0.4032	0.5625	0.7290	0.8534	0.9997	1.1217	1.3038	1.3950

Table 4. Latency measures, in seconds, for read (R) and write (W) TXs in a local Docker environment where network size = 8.

TX	**Arrival Rate (TXs/s)**										
Type	***1***	***50***	***100***	***150***	***200***	***250***	***300***	***350***	***400***	***450***	***500***
Min (W)	0.1004	1.6858	1.4348	2.3036	4.7245	5.1294	4.5748	5.8767	6.3978	6.7831	7.1836
Max (W)	2.0020	2.2518	3.3587	4.8211	5.0167	5.4596	5.9855	6.3402	6.8323	8.5989	7.7109
Avg (W)	1.7919	1.8107	1.7563	3.9873	4.8259	5.2884	5.6173	6.1577	6.6395	7.1475	7.4985
TX	**Arrival Rate (TXs/s)**										
Type	***1***	***50***	***100***	***150***	***200***	***250***	***300***	***350***	***400***	***450***	***500***
Min (R)	0.0143	0.1694	0.2172	0.3529	0.6244	0.5323	0.7636	0.8521	0.9900	1.3708	1.3517
Max (R)	0.0263	0.2425	0.4751	0.7329	0.9451	1.1434	1.3563	1.5009	1.7062	2.0700	2.3891
Avg (R)	0.0214	0.1526	0.2926	0.4009	0.5749	0.7272	0.8616	0.9998	1.1941	1.3387	1.4752

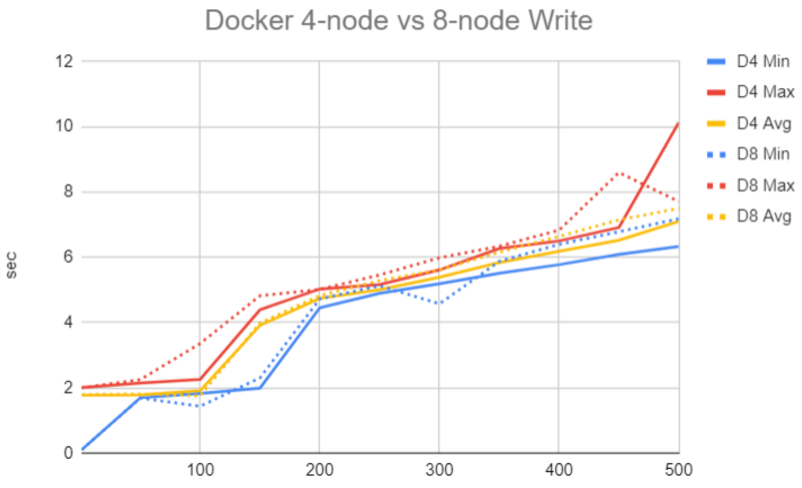

Figure 4. Latency of write TXs (in seconds) measured with a local Indy-based BC and different arrival rates (1–500).

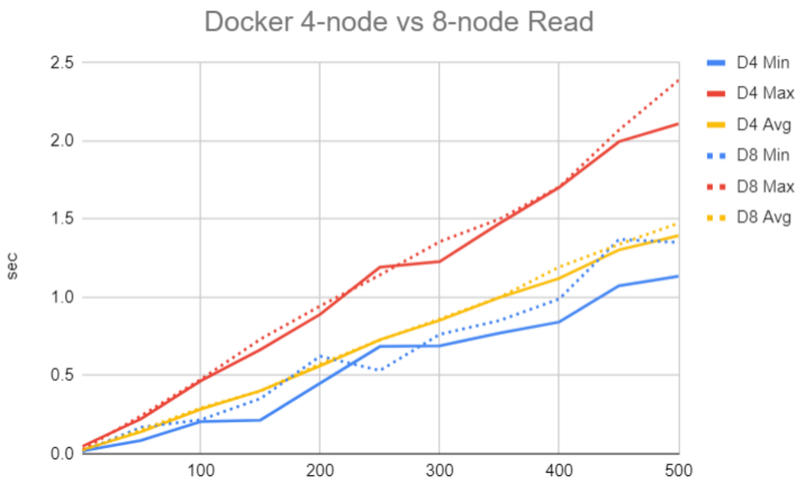

Figure 5. Latency of read TXs (in seconds) measured with a local Indy-based BC and different arrival rates (1–500).

In comparison with the ledger on the Google Cloud Platform, in terms of the performance in the metrics of reading and writing response latency, we found that the local Indy ledger reads 30–50% faster, and writes 65–85% faster than the Indy ledger running on the Google Cloud Platform. The explanation may be that the network latency is much better in a local Docker environment, than in a close-to-real-life scenario, where the different VMs are in different regions in the Google Cloud Platform.

5. Discussion

As we can observe in the results presented in the previous section, increasing the number of Indy nodes and/or the arrival rate increases the average response latency. These measures are important to consider when Indy is considered as a framework for a given BC-based application. The scenarios we tested, and their results, shall help those who tend to utilize Indy and Aries when making the decision as to whether to adopt or exclude this platform from their project. To further clarify the results we obtained, we present our results in Figures 6 and 7.

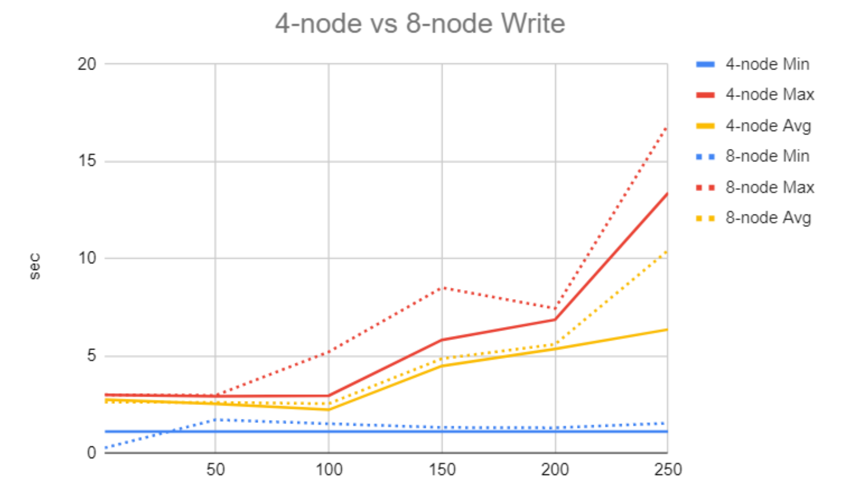

Figure 6. Latency of write TXs (in seconds) measured with different Indy-based BC network sizes and different arrival rates.

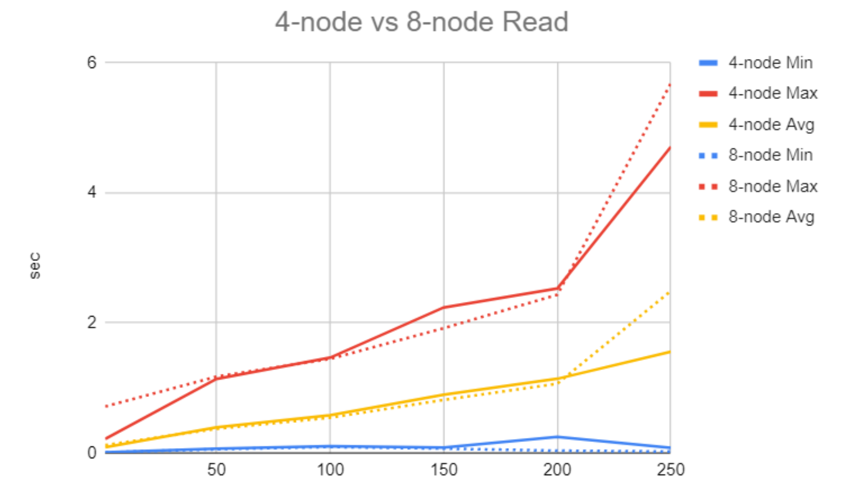

Figure 7. Latency of read TXs (in seconds) measured with different Indy-based BC network sizes and different arrival rates.

In the case of read TXs, we can observe that the average latency measurement is almost linearly proportional to the arrival rate up to 200 TXs/s. After that, the latency starts to increase according to the number of nodes deployed. In the case of write TXs, we can observe that the average latency measurement is almost constant up to an arrival rate of 100 TXs/s. After that, the latency starts to increase depending on the system buffer size and the utilized processing power.

Furthermore, one can see that the average write latency is generally higher when deploying more nodes for different arrival rates. The read latency, on the other hand, is generally lower when deploying more nodes as more servers are available to process the received TXs. An example of the utilization of these results may be a latency-sensitive application that requires an average write latency of less than 2 s per TX, which shall not adopt the Indy framework. Note that this measurement is for the least number of Indy nodes (i.e., four nodes). Adding more nodes shall increase the write latency as discussed above.

If we compare the latency of the local Indy ledger with the same node in the Google Cloud Platform, it is obvious that the local ledger performed better both in writing and reading. The explanation could be the much better network performance between the nodes in the local environment. The dependency of the arrival rate is also shown in both experiments. In Figures 8 and 9, the 30–50% better read latency and the 65–85% write latency can be seen.

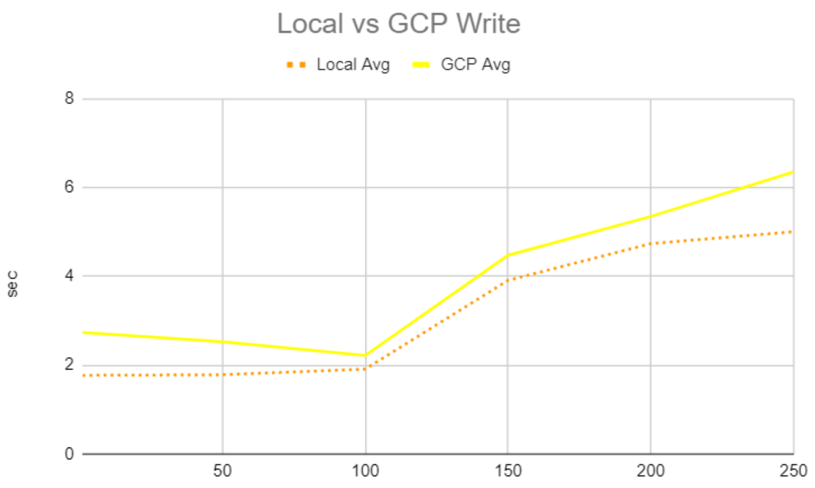

Figure 8. Average write latency comparison of a 4-node GCP and a local Docker Indy ledger.

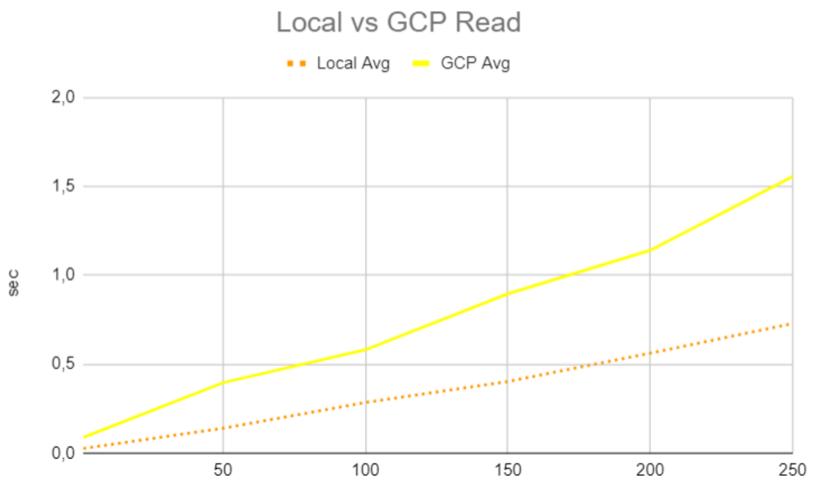

Figure 9. Average read latency comparison of a 4-node GCP and a local Docker Indy ledger.

During the implementation and deployment of Indy and Aries nodes, we faced the following drawbacks/challenges. DLs in Indy are maintained linearly, which implies that high DL consistency depends on a high block finality time [29]. This causes additional write latency on top of the latency required for reaching a consensus. Providing global information regarding what accreditation bodies approve/disapprove, and on which reference criteria blocks are confirmed, cannot be achieved using the currently available open-source code of Indy and Aries. The maximum recommended number of nodes participating as BC miners is 25. Adding more nodes is likely to introduce an inefficient system with very high latency. The open-source code currently available allowed us to perform very limited modifications, and we found that the deployment of an enterprise Indy solution requires deep background knowledge of its technicalities. For example, we tried to deploy the Indy node codes on a recent version of the Ubuntu OS but we could not because of the very restrictive dependencies of the deployed libraries. We have not found a clear and proactive trust model for adding new endorsers or stewards. Thus, adding endorsers does not necessarily mean they are accredited in the associated legacy system.

6. Conclusions

In this paper, we proposed a deployment architecture for SSI applications, and developed a Python application with containerized components for its realization. We evaluated our proposed architecture in terms of latency by deploying its components in the Google Cloud Platform, in different regions with individual VMs. We executed several test scenarios to obtain average latency measures for different system settings, namely different network sizes and different TX arrival rates. For arrival rates between 1 and 250, we found that the write response latency of an Indy-based BC containing four and eight nodes ranges between 1 and 16 s, while the read response latency with similar settings ranges between 0.01 and 5 s. We also compared the results with a local ledger in a Docker environment in our private cloud. In the future, we plan to enhance the scalability of our application by dynamic, automated BC node addition and removal. We have also started to design a solution for some of the mentioned performance issues in a global accreditation and credential verification application, and we have proposed a privacy-aware Fog-enhanced blockchain-based online credential solution in [30].

Supplementary Materials: All our implemented application components are containarized using Docker, which makes them easier to deploy for present, and future works. Our implemented application is publicly available at Github. Available online: https://github.com/sed-szeged/IndyPerf (accessed on 26 September 2022).

Author Contributions: Software, T.P.; Writing—review & editing, T.P., H.B. and A.K. All authors have read and agreed to the published version of the manuscript.

Funding: The research leading to these results has received funding from the national project TKP2021-NVA-09 implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, and from the UNKP-21-4 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

Data Availability Statement: Not applicable.

Acknowledgments: The experiments presented in this paper are based upon work supported by Google Cloud.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nofer, M.; Gomber, P.; Hinz, O.; Schiereck, D. Blockchain. *Bus. Inf. Syst. Eng.* **2017**, *59*, 183–187.
2. Abraham, I.; Malkhi, D.; Nayak, K.; Ren, L.; Spiegelman, A. Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus. *arXiv* **2016**, arXiv:1612.02916.
3. Karasek-Wojciechowicz, I. Reconciliation of anti-money laundering instruments and European data protection requirements in permissionless blockchain spaces. *J. Cybersecur.* **2021**, *7*, tyab004.
4. Krawiec, R.; Housman, D.; White, M.; Filipova, M.; Quarre, F.; Barr, D.; Nesbitt, A.; Fedosova, K.; Killmeyer, J.; Israel, A. et. al. Blockchain: Opportunities for Health Care. Available online: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/public-sector/us-blockchain-opportunities-for-health-care.pdf (accessed on 26 September 2022).
5. Javaid, U.; Aman, M.N.; Sikdar, B. A scalable protocol for driving trust management in internet of vehicles with blockchain. *IEEE Internet Things J.* **2020**, *7*, 11815–11829.
6. Bistarelli, S.; Mercanti, I.; Santancini, P.; Santini, F. End-to-End Voting with Non-Permissioned and Permissioned Ledgers. *J. Grid Comput.* **2019**, *17*, 97–118.
7. Kondova, G.; Erbguth, J. Self-sovereign identity on public blockchains and the GDPR. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, 30 March–3 April 2020; pp. 342–345.
8. Jirgensons, M.; Kapenieks, J. Blockchain and the future of digital learning credential assessment and management. *J. Teach. Educ. Sustain.* **2018**, *20*, 145–156.
9. Tobin, A.; Reed, D. The inevitable rise of self-sovereign identity. *Sovrin Found.* **2016**, *29*, 18.
10. Thomas, K.; Pullman, J.; Yeo, K.; Raghunathan, A.; Kelley, P.G.; Invernizzi, L.; Benko, B.; Pietraszek, T.; Patel, S.; Boneh, D.; et al. Protecting accounts from credential stuffing with password breach alerting. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 1556–1571.
11. Makri, K.; Papadas, K.; Schlegelmilch, B.B. Global social networking sites and global identity: A three-country study. *J. Bus. Res.* **2021**, *130*, 482–492.

12. Boysen, A. Decentralized, Self-Sovereign, Consortium: The Future of Digital Identity in Canada. *Front. Blockchain* **2021**, *4*, 11.
13. Kertesz, A.; Baniata, H. Consistency Analysis of Distributed Ledgersin Fog-enhanced Blockchains. In Proceedings of the European Conference on Parallel Processing, Lisbon, Portugal, 30–31 August 2021.
14. Linux Foundation. HyperLedger Indy. 2020. Available online: hyperledger.org/use/hyperledger-indy (accessed on 26 September 2022).
15. Baniata, H.; Pflanzner, T.; Feher, Z.; Kertesz, A. Latency Assessment of Blockchain-based SSI Applications Utilizing Hyperledger Indy. In Proceedings of the 12th International Conference on Cloud Computing and Services Science—CLOSER. INSTICC, SciTePress, Online, 27–29 April 2022; pp. 264–271. https://doi.org/10.5220/0011082300003200.
16. Sopek, M.; Grakadzki, P.; Kuzinski, D.; Trojczak, R.; Trypuz, R. Legal Entity Identifier Blockchained by a Hyperledger Indy Implementation of GraphChain. In Proceedings of the Research Conference on Metadata and Semantics Research, Limassol, Cyprus, 23–26 October 2018; pp. 26–36.
17. Raclawickie, A. Legal Entity Identifier Blockchained by a Hyperledger Indy Implementation of GraphChain. In Proceedings of the Metadata and Semantic Research: 12th International Conference, MTSR 2018, Limassol, Cyprus, 23–26 October 2018; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2019; Volume 846, p. 26.
18. Sopek, M.; Gradzki, P.; Kosowski, W.; Kuziski, D.; Trojczak, R.; Trypuz, R. GraphChain: A distributed database with explicit semantics and chained RDF graphs. In Proceedings of the Companion Proceedings of the The Web Conference 2018, Lyon, France, 23–27 April 2018; pp. 1171–1178.
19. Bhattacharya, M.P.; Zavarsky, P.; Butakov, S. Enhancing the Security and Privacy of Self-Sovereign Identities on Hyperledger Indy Blockchain. In Proceedings of the 2020 International Symposium on Networks, Computers and Communications (ISNCC), Montreal, QC, Canada, 20–22 October 2020; pp. 1–7.
20. Malik, S.; Gupta, N.; Dedeoglu, V.; Kanhere, S.S.; Jurdak, R. TradeChain: Decoupling Traceability and Identity inBlockchain enabled Supply Chains. *arXiv* **2021**, arXiv:2105.11217.
21. Prakash, N.; Michelson, D.G.; Feng, C. CVIN: Connected Vehicle Information Network. In Proceedings of the 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), Antwerp, Belgium, 25–28 May 2020; pp. 1–6.
22. Mohanty, D. *Blockchain From Concept to Execution: BitCoin, Ethereum, Quorum, Ripple, R3 Corda, Hyperledger Fabric/SawTooth/Indy, MultiChain, IOTA, CoCo*; BPB Publications: Noida, India, 2018.
23. Finck, M. Blockchain and the General Data Protection Regulation. Can Distributed Ledgers be Squared with European Data Protection Law? Available online: https://www.europarl.europa.eu/RegData/etudes/STUD/2019/634445/EPRS_STU(2019)634445_EN.pdf (accessed on 26 September 2022).
24. Caliper, H. Hyperledger Caliper Architecture. *Electronic Article.* 2019. Available online: https://hyperledger.github.io/caliper/docs/2_Architecture.html (accessed on 26 September 2022).
25. Dong, Z.; Zheng, E.; Choon, Y.; Zomaya, A.Y. Dagbench: A performance evaluation framework for dag distributed ledgers. In Proceedings of the 2019 IEEE 12th international conference on cloud computing (CLOUD), Milan, Italy, 8–13 July 2019; pp. 264–271.
26. Dinh, T.T.A.; Wang, J.; Chen, G.; Liu, R.; Ooi, B.C.; Tan, K.L. Blockbench: A framework for analyzing private blockchains. In Proceedings of the 2017 ACM International Conference on Management of Data, Chicago, IL, USA, 14–19 May 2017; pp. 1085–1100.
27. Aublin, P.L.; Mokhtar, S.B.; Quéma, V. Rbft: Redundant byzantine fault tolerance. In Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems, Philadelphia, PA, USA, 8–11 July 2013; pp. 297–306.
28. Windley, P.J. *How Sovrin Works*; Windely.com: Hoboken, NJ, USA, 2016.
29. Baniata, H.; Anaqreh, A.; Kertesz, A. DONS: Dynamic Optimized Neighbor Selection for smart blockchain networks. *Future Gener. Comput. Syst.* **2022**, *130*, 75–90.
30. Baniata, H.; Kertesz, A. PriFoB: A Privacy-aware Fog-enhanced Blockchain-based system for Global Accreditation and Credential Verification. *J. Netw. Comput. Appl.* **2022**, *205*, 103440. https://doi.org/10.1016/j.jnca.2022.103440.