On the Performance of Distributed Ledgers for Internet of Things

Runchao Han[a], Gary Shapiro[b], Vincent Gramoli[b,c], Xiwei Xu[c,d]

[a] *The University of Manchester*
[b] *The University of Sydney*
[c] *Data61-CSIRO*
[d] *The Univerisity of New South Wales*

Abstract

As proof-of-work blockchains are inherently energy greedy and offer probabilistic guarantees, permissioned distributed ledgers based on the classic deterministic consensus of the distributed computing literature appear as promising solutions to record securely the information produced by a large amount of connected Internet of Things (IoT) devices. As of today, it remains unclear whether these solutions can scale to large networks.

In this paper, we evaluate the performance of prominent distributed ledgers using classic consensus protocols. We select five mainstream distributed ledgers, namely Ripple, Tendermint, Corda and v0.6 and v1.0 of Hyperledger Fabric, and evaluate both the throughput and the latency of clusters with different numbers of nodes (ranging from 2 to 32) for each of them. Our results show that, while offering sometimes thousands of transactions per second throughput, their performance usually does not scale to tens of devices as it drops dramatically when the number of devices increases. This study motivates the need for alternative solutions that solve the Blockchain consensus problem, a scalable variant of the classic consensus problem dedicated to blockchains.

Keywords: Private blockchains, consortium blockchains, IoT

1. Introduction

The distributed ledger technology gained momentum for its promises to track the ownerships of digital assets through the recording of transactions that transfer assets across accounts. Distributed ledgers traditionally organise transactions into blocks linked with each other via hashes, hence implementing a blockchain [2], however, new variants have recently emerged to implement other abstractions based on the application needs [3]. Variants of this technology rely on different permission models: either *permissionless*, allowing

A short version of this paper that only evaluates Ripple and Hyperledger Fabric appeared in the proceedings of *IFIP NTMS 2018* [1]. The current version extends it by evaluating Tendermint and Corda.

Email addresses: `runchao.han@student.manchester.ac.uk` (Runchao Han), `gsha5095@uni.sydney.edu.au` (Gary Shapiro), `vincent.gramoli@sydney.edu.au` (Vincent Gramoli), `xiwei.xu@data61.csiro.au` (Xiwei Xu)

machines (or nodes) to create blocks without special permissions or *permissioned*, allowing only permissioned nodes to create blocks.

Traditional permissionless blockchains use proof-of-work to guarantee that no malicious nodes can overwhelm the systems with new blocks. These distributed ledger technologies find various applications in the industry allowing typically Internet of Things (IoT) devices to append data in a tamper-proof log. Enterprises have however been reluctant to adopt proof-of-work blockchains [2] in production, because of their need for computational resources and because they offer probabilistic guarantees [4] that made them subject to double-spending [5], especially in a consortium or private context [6]. It has even been shown that an attacker can double spend without a significant mining power in an Ethereum-based blockchain network [7].

By contrast, distributed ledgers for consortium and private networks often solve classic determistic consensus problem [8], hence achieving, in principle, greater security. In particular, the deterministic Byzantine consensus protocols [9, 10, 11] prevent distributed ledgers from reaching inconsistent states that could lead to double spending. As most classic Byzantine consensus protocols [10] inherit from a protocol originally designed for small local area networks [9], it remains unclear whether one can adapt them to scale to numerous blockchain devices connected through a large network as required by the IoT. Note that the number of IoT devices exceed now the world population[1].

In this paper, we evaluate whether distributed ledgers that solve the classic consensus problem can scale to numerous connected devices. Depending on their failure model, these distributed ledgers typically favor consistency over availability in a permissioned consortium context, where participants have dedicated permissions to create new blocks. To this end, these ledgers typically embed, at their core, a classic fault tolerant consensus protocol that guarantees a total order on the committed transactions despite failures of a bounded number of participants, sometimes malicious.

We present an evaluation of some of the prominent distributed ledgers:

- Hyperledger Fabric (HLF) v0.6 with the Practical Byzantine Fault Tolerance (PBFT) consensus protocol [9];
- HLF v1.0 with the Byzantine Fault-Tolerant State Machine Replication (BFT-SMaRt) consensus protocol [10];
- Ripple v0.60.0 with the Byzantine Fault-Tolerant Ripple consensus protocol [12];
- Tendermint v0.22.4 [13] with the hybrid Byzantine consensus combining PBFT and Casper [14];
- R3 Corda v3.2 [3] built upon BFT-Smart [10].

[1]`https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/`

Existing evaluations focuse either on a single blockchain only [15], the performance of the Byzantine fault tolerant consensus [16], or blockchains with different purposes [17]. In particular, some efforts were recently devoted to evaluate blockchains [17], but they aim at comparing the performance of inherently slow proof-of-work blockchains to faster blockchains based on classic consensus and disregard distributed ledgers that are not blockchains, like Corda. Our work targets distributed ledgers designed for enterprise ignoring the inherently slower proof-of-work blockchains.

Our experiments result from the deployment of permissioned ledgers on a distributed set of physical machines. The peak performance we obtained differs quite significantly from one technology to another but in all cases this performance degrade rapidly with the increase in the number of devices. This work opens up new research questions on the scalability of distributed ledgers for the Internet of Things.

2. Related work

Various distributed ledgers have been proposed for different purposes, but only few proposals aim at the Internet of Things (IoT). Applying successfully distributed ledgers to IoT will likely require borrowing mechanisms from existing blockchains, especially their consensus protocols.

Hyperledger Fabric (HLF) v0.6 is a consortium blockchain platform [18] with customizable smart contracts called "chaincodes". The chaincode supports existing programming languages like Go and Java. Hyperledger Fabric v0.6 relies on PBFT [9], the state-of-the-art Byzantine consensus protocol that was designed decades ago for local area network. HLF v1.0 was designed to obtain a throughput that would scale better than v0.6.[2] Unlike HLF v0.6, HLF v1.0 is not secure by default as it does not tolerate Byzantine behaviors. A fork of v1.0 master branch was changed to invoke a BFT ordering service based on BFT-SMaRt. BFT-SMaRt [10] is a high performance BFT-based consensus protocol that predates blockchains using the same leader-based design as PBFT. This branch uses gRPC and supports only timestamp broadcast rather than smart contracts, but its ordering service copes with malicious behaviors. As a full-fledged distributed ledger typically verify signatures, we expect that a full fledged distributed ledger building upon this protocol could be as efficient or slower than the one we test.

Corda [3] is a distributed ledger—but not a blockchain—that aims at being Byzantine fault tolerant to offer trust among financial institutions. In particular, Corda processes and records transactions between financial institutions with privacy guarantee. Unfortunately, the entreprise version of Corda is not secure as it uses Raft [19] and guarantees consensus only when participants behaves correctly or crash. The public version of Corda embeds both the code of Raft and the code of BFT-SMaRt [10]. After few attempts to deploy Corda-BFT-SMaRt, the Corda development team confirmed that Corda with BFT-SMaRt is not stable. So we reverted back to the insecure public version of Corda. As Raft is leader-based

[2]https://www.hyperledger.org/wp-content/uploads/2017/07/Hyperledger-Fabric-V1-Launch-FAQ-FINAL.pdf

similarly to BFT-SMaRt, we expect the insecure but stable version of Corda should give an estimate of the highest performance one could obtain from the secure variant once it will be available.

Ripple is another blockchain platform focusing on financial applications [12]. The Ripple Protocol supports a wide range of transactions for accounts and their XRP currency exchanges, but lacks the smart contract support. It runs its own Ripple consensus protocol, of which the correctness has been debated [20]. Nevertheless, the security proof is out of the scope for this paper.

Tendermint is another blockchain platform with a BFT Proof-of-Stake (PoS) hybrid consensus protocol, called Tendermint BFT [13]. The Tendermint BFT protocol is stated as a high-performance consensus protocol with the same fault-tolerance level as classic BFT-based consensus protocols. Multiple applications are supported on the Tendermint blockchain inherently, such as a key-value store and a distributed counter[3]. Although some researchers demonstrated that the Tendermint consensus protocol was non-terminating [21], fixing the protocol is outside the scope of this paper.

IOTA [22] is a cryptocurrency for IoT. As opposed to the classic Byzantine consensus technologies, it does not totally order transactions but builds a directed acyclic graph (DAG) of transactions, called *Tangle*. To commit new transactions, IOTA requires currently a single coordinator, which may become a single point of failure, as the failure of this coordinator prevents transactions from being executed[4]. IOTA is still under development and expected to be decentralised in the future. Commercial products also rely on a DAG of transactions [23], unfortunately their code is unavailable.

There exist more recent proposals that improve performance. The Red Belly Blockchain was shown to perform well on up to a thousand of nodes [24], however, it relies on a different problem from the classic Byzantine consensus problem, called Blockchain Consensus [25], especially designed for blockchain scalability. A framework optimized for IoT [26] exploits two distributed ledgers: one that is private and typically is managed centrally at small scale (e.g., smart home) and another higher level public blockchain.

3. Evaluating distributed ledgers with classic consensus

In this section, we describe our evaluation methodology. Our evaluation focuses on comparing the performance of mainstream secure distributed ledgers from distributed system perspective i.e. in terms of throughput and latency. We selected the following blockchain platforms:

1. Hyperledger Fabric v0.6 with PBFT consensus [9]. In particular, the reason why we complemented our study with the prototypical HLF version using BFT-SMaRT is because the consensus of HLF v0.6 is known to have unresolved bugs.[5]

[3]`https://tendermint.readthedocs.io/en/master/`.

[4]`https://cryptovest.com/news/iotas-53-hour-network-standstill-heightens-investors-worst-fears`.

[5]`https://jira.hyperledger.org/browse/FAB-707`.

2. Ripple 0.60.0 with the XRP consensus protocol [12]. In Ripple, a validator is usually a trusted node that is authenticated by Ripple Inc. in the public network. To eliminate the network interference, we created our own local validator rather than using an existing public validator.
3. Tendermint v0.22.4 with the Tendermint BFT consensus [13]. All nodes in the Tendermint network are set as validators, as only validators participate in the consensus process.
4. Hyperledger Fabric v1.0, with the BFT-SMaRt consensus [10]. Because this branch only supports timestamp broadcast based on gRPC rather than message types offered by the main branch, we had to adapt our benchmark when using BFT-SMaRt.
5. R3 Corda v3.2 with a centralized Notary node. Corda uses a separate proxy HTTP server for communicating with clients. In Corda, Notary nodes run the Raft consensus protocol to maintain the transaction order. In this paper, we evaluate Corda clusters with a single Notary cluster.

3.1. Software and hardware

We ran some experiments on a distributed system of 32 machines using Emulab, an environment providing machines for distributed system experiments. We use two types of machines for our experiments as follows:

- d710: Two 64-bit E5-2630 Haswell processors with 8 cores running at 2.4 GHz, 64 GB 2133 MT/s DDR4 RAM (8 x 8GB modules), one Intel SATA SSD with 200 GB and 2 x 1 TB 7200 rpm SATA disks.

- d430: One 64-bit E5-2630 Nehalem processor with 4 cores running at 2.4 GHz, with 12 GB 1066 MHz DDR2 RAM (6 x 2 GB modules), one Seagate SATA disk with 250 GB and one 500 GB 7200 rpm Western Digital SATA disk.

We chose the publicly available versions of the code, namely HLF V0.6[6], Rippled v0.60.0[7], HLF BFT-SMaRt[8], Tendermint v0.22.4[9] and Corda v3.2[10]. Clients for benchmarking ran Node.js v8.11.3 on Ubuntu 16.04.

3.2. Workload design and used interfaces

The difficulty in evaluating different distributed ledgers lies in the workload design. We constructed a generic workload that (i) performs the same functions and (ii) can be implemented straightforwardly using ledger APIs.

Basically, our workload performs a transfer from an account A to another account B with the amount 1. This is a general workload, which is easy to implement on ledgers supporting smart contracts (i.e. Hyperledger Fabric, Tendermint and Corda) or ledgers without smart contracts (i.e. Ripple).

[6]`https://github.com/rleonardco/fabric-0.6`.
[7]`https://github.com/ripple/rippled/releases/tag/0.60.0`.
[8]`https://github.com/jcs47/hyperledger-bftsmart`.
[9]`https://github.com/tendermint/tendermint/releases/tag/v0.22.4`.
[10]`https://github.com/corda/corda/releases/tag/release-V3.2`.

3.2.1. The HLF v0.6 API

HLF v0.6 provides operations that maintain key-value pairs i.e. *states*. Therefore, to simulate an account with balance, the client can just put a key as the username as well as a corresponding value standing for the balance with an initial value. The transfer of money is represented as an atomic operation that decreases the balance of A and increases the balance of B.

Operations that change the state through the function `PutState()` are non-blocking operations called *invocation*s. Therefore, the client can know whether the transaction has been executed only by querying the ledger state.

3.2.2. The Ripple API

Ripple specifies the format of an account as a hash value, which is created from a user-defined passphrase. The account is included in the network only with the consent of the root account. Activating the account relies on a transfer from another existing account to it. The transfer transaction is called *payment* in Ripple. Therefore, we designed a workload for Ripple as a *payment* transaction from an account to another account activated in this network with amount 1. Similarly to HLF, the Ripple interface is non-blocking in that a transaction commitment is not acknowledged.

3.2.3. The Tendermint API

Similar to HLF v0.6, Tendermint also supports the key-value storage, so we simulate the payment the same way as HLF v0.6. Similar to Ripple and HLF v0.6, Tendermint provides non-blocking HTTP APIs.

3.2.4. The HLF with BFT-SMaRt API

HLF BFT-SMaRt is an experimental branch of the BFT ordering service for HLF. The current version only supports the timestamp broadcast message type, and was tested in order to make a comparison with PBFT ordering in HLF v0.6. As opposed to HLF v0.6, HLF BFT-SMaRt uses gRPC.

3.2.5. The Corda API

Corda decouples the Corda server and the HTTP server, where the Corda server is the core DLT system and the HTTP server is a proxy between the clients and the Corda server. The HTTP server exposes the transactions provided in the smart contract, by which we can simulate the payment transactions like Ripple and HLF v0.6. By contrast with other DLT platforms, Corda transactions only involve the participants and the Notary node rather than all nodes in order to keep the privacy of each node[11]. Note that the transaction invocations in Corda are blocking operations: The response of a transaction invocation will be given only after the transaction is confirmed by the Notary node. Our client communicates with the HTTP proxy server in order to invoke or query the Corda server.

[11]`https://docs.corda.net/key-concepts-notaries.html`.

Table 1: Parameters of the workload

Parameter	Description
Method	The HTTP method used for the request
HTTP Headers	HTTP Headers of the request
HTTP Data	HTTP Data of the request(can be a JSON string)

Table 2: Parameters of the experimental settings

Parameter	Description
Endpoint	The URL of receiving requests
Concurrency	The level of concurrency
Max Seconds	The duration of sending requests
Workload	The workload definition
Start QPS	The requests per second when the testing starts
QPS Stride	The increase of the QPS to the last QPS
End QPS	The highest QPS of the test
Sleep Period	The period of rest between different QPS testing

3.3. Benchmarking and parameters

We conducted all experiments using the `loadtest` library of Node.js, which is a performance testing toolkit for backend services supporting HTTP(s) or WebSocket.

The `loadtest` library specifies an HTTP request format as a JSON object, and we construct workloads for ledgers as separate JSON objects. Transaction invocations are non-blocking except in Corda. In other words, when submitting a transaction using HTTP, an HTTP response encoding the transaction status is sent by the server, before the transaction is fully processed. In this way, the client cannot know if the transaction has committed successfully unless it queries the server again.

Besides the HTTP content, `loadtest` can also control the HTTP request header. 1 outlines related parameters of our workloads. During the test, we increase the request rate step by step, in order to gather related metrics under different request overheads. The parameters of a single load testing are listed in Table 2.

3.3.1. Requests

HLF v0.6. To customize transactions in HLF v0.6, we deployed a smart contract (i.e. chaincode in Hyperledger) written in the Go programming language. The Hyperledger community provides a chaincode example supporting a simple money transfer transaction, and we adopted it to our case directly.

Ripple. Ripple has a set of well-defined transactions, but lacks the support of customized transactions i.e. smart contracts. It defines the *payment* transaction, and we adopt it to our case directly. As aforementioned, accounts in Ripple are inactive at first. Therefore, to experiment on a private Ripple network, we create two accounts A and B, then pay from account A to account B so that account B will be activated.

Table 3: Parameters of the workloads for HLF v0.6, Ripple and Tendermint

Parameter	Method	MIME Type	HTTP Data
HLF v0.6	POST	application/json	Invoking chaincode
Ripple	POST	application/json	Payment transaction
Tendermint	GET	text/plain	Key-value pairs
Corda Invoke	PUT	text/plain	Key-value and participants
Corda Query	GET	text/plain	Queried key

Tendermint. Experimenting Tendermint follows the same way as HLF v0.6. We define two accounts using key-value pairs, where the key is the account name and the value is the amount of coins. To mimic the money transfer, we commit two transactions concurrently, one for decreasing the value of a key by 1, another for increasing the value of another key by 1.

HLF with BFT-SMaRt. Unfortunately, HLF with BFT-SMaRt ordering only supports the timestamp broadcasting with gRPC. Therefore, testing HLF BFT-SMaRt was done by its inherited performance testing module. For all the experiments on HLF with BFT-SMaRt, we use a single orderer. Changing the number of orderers does not seem to impact performance significantly [27].

Corda. Corda communicates with clients with a proxy HTTP server, which exposes transactions supported by the deployed smart contract. We utilize the reference implementation of Corda's smart contract `Cordapp-example`[12], which is similar to the key-value storage application in Tendermint. We simulated the payment in the same way as HLF v0.6 and Tendermint.

3.3.2. Testing Steps

Based on the aforementioned principles, the testing consists of multiple automated steps:

1. Setting up the cluster
2. Deploying the chaincode (for HLF)
3. Activating accounts (for Ripple)
4. Setting parameters of the workloads
5. Setting parameters of the sending workloads
6. Launching the load testing for 10 times
7. Analyzing the results

Parameters of Step (4) and Step (5) for both platforms are listed in Table 3 and Table 4, respectively.

[12] `https://github.com/corda/cordapp-example`.

4. Results and analysis

In this section, we describe the observed results and we analyze the performances of the chosen platforms. The load was applied over $4, 8, 12, 16, 20, 24, 28$ and 32 independent physical machines except for Corda. We only deployed a 4-node Corda cluster with 4 d430 machines because of its performance and unstability that we discuss later. HLF with BFT-SMaRt consensus is evaluated separately because of its differences. For Tendermint and Corda both the transaction invocation and transaction query were tested.

4.1. Metrics for load testing

4.1.1. Query per second

The number of queries per second (QPS) is a common indicator of throughput of a system. It translates into the number of requests the server can treat within a second [28]. The request refers to the HTTP request in our context. Obviously, the more requests the system processes in a second, the better it performs.

Table 4: Parameters of the experiments for HLF v0.6 and Ripple

Parameter	HLF v0.6	Ripple	Tendermint	Corda
Endpoint	ip:7050/chaincode	ip:5005	ip:26657	ip:10009
Concurrency	10	10	10	10
Max Seconds(s)	10	10	10	10
Start QPS	200	500	10000	30
QPS Stride	200	500	10000	30
End QPS	4000	5000	100000	300
Sleep Period(s)	20	20	10	10

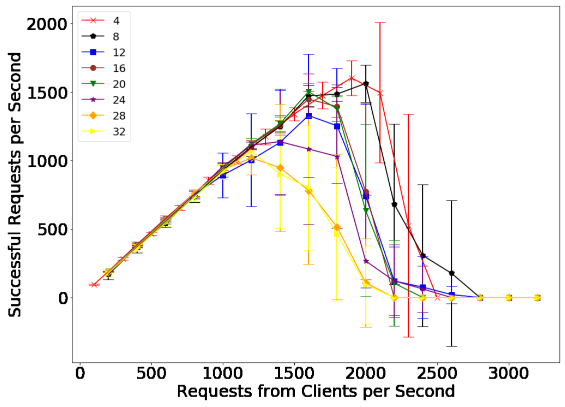
4.1.2. Latency

The latency means the delay of a request or a task that the system processes [28]. A system usually processes multiple requests, so the latency is usually the average value of latencies in a period. As the invocations and queries are non-blocking except for Corda, we did not compute the latency as the time between a transaction is invoked and its commit is acknowledged. Instead, we evaluated the time between a request is sent and the corresponding response is received.

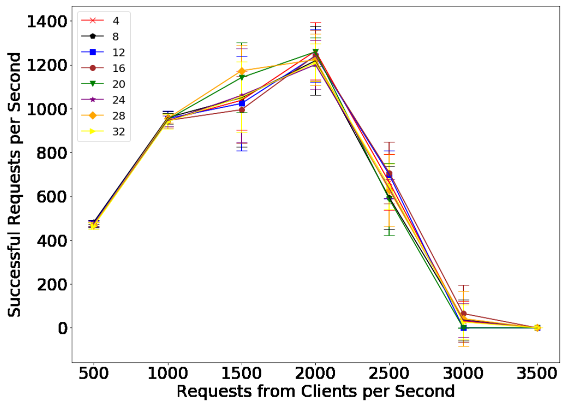
4.2. Testing throughput

The result of HLF v0.6 is shown in Fig. 1a whereas the result of Ripple is shown in Fig 1b. In both figures, we observe that for each cluster, the throughput in either system increases with the growth of the request rate first, and then reaches a peak. However, after reaching the peak the throughput decreases sharply with the increase of the request rate, because the system cannot handle the demand.

In contrast with HLF v0.6 and Ripple, Fig. 2a shows that the Tendermint throughput is more stable with the increasing requests from clients, with shorter uphills and downhills.

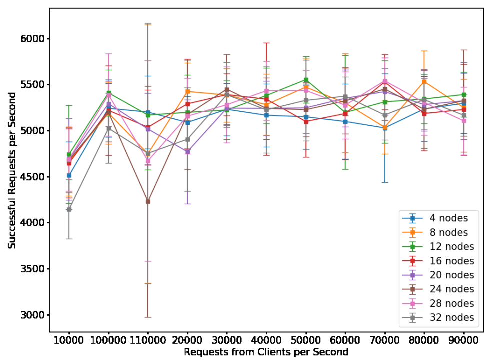


(a) **HLF v0.6.** The QPS first increases but then drops sharply, finally becomes 0.

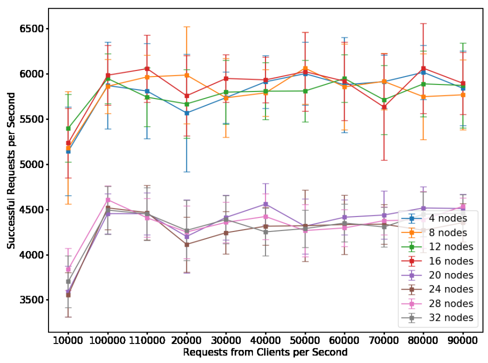


(b) **Ripple.** The trend is similar to the HLF v0.6.

Figure 1: The throughput of HLF v0.6 and Ripple.



(a) **Invoke.** The throughput is fairly stable and high compared to HLF V0.6 and Ripple, but still not scalable.



(b) **Query.** The throughput decreases sharply when the number of nodes reaches 20.

Figure 2: The invoke and query throughput of Tendermint.

Moreover, the number of handled requests is much greater than HLF v0.6 and Ripple with few crashes, but the number of successful requests is similar to HLF v0.6 and Ripple. However, Fig. 2b shows that the query throughput of Tendermint falls sharply when the number of nodes reaches 20. A possible reason is that frequent distributed transactions drain the network bandwidth when more than 20 nodes are active, where all nodes are connected by a central router.

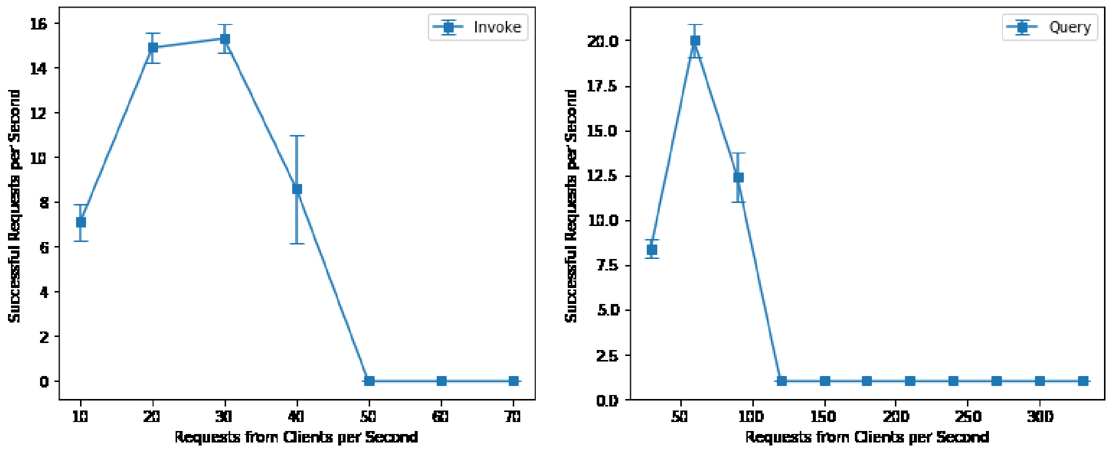


Figure 3: **The throughput of the 4-node Corda cluster.** The throughput is quite low, and the cluster crashes with more than 50 queries per second.

According to Fig. 3, Corda obtained surprisingly low performance on both query and invoke operations on the cluster with 4 nodes. Moreover, the cluster crashed with more than 50 queries per second. This is because of the privacy preserving mechanism and the networking mechanism. Transactions in Corda affect the transaction participants and the centralized Notary node, which orders transactions across the network. However, the transaction invocation is a synchronous blocking operation, making the process highly time-consuming due to the communication with the Notary cluster and the transaction receiver.

Moreover, for each system, the trends of throughputs with different cluster sizes are similar, which means that the performance has no trend of increasing with the increase of nodes. In other words, consensus protocols of our targeted ledgers do not scale. Note that this limitation is well-known for classic BFT consensus protocols because they solve a slightly more restrictive problem than the blockchain consensus [25]. In particular, both PBFT and BFT-SMaRt rely on a consensus leader, which may act as a bottleneck [25].

4.3. Peak throughput with increasing nodes

The results of HLF v0.6 and Ripple consensus are depicted in Fig. 4, whereas Fig 5 depicts the results of Tendermint. Fig 6 shows the results of HLF v1.0.

As expected, the peak throughput decreases with the increase of the cluster size in Ripple. The more nodes involved in a cluster, the more messages are exchanged. For a Ripple private network with only one validator, transactions are batched and sent to the validator, so that

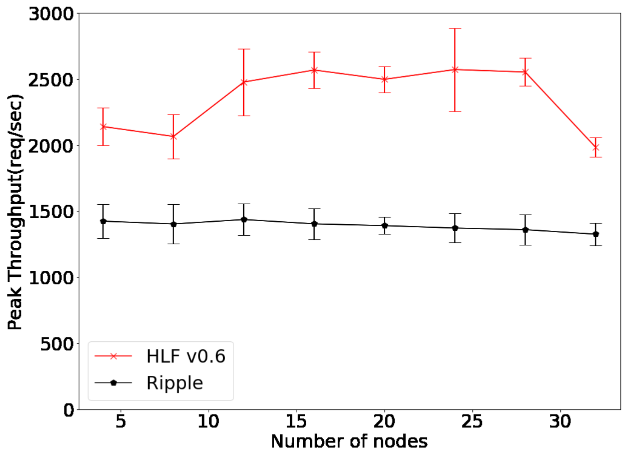


Figure 4: **The peak throughput with increasing nodes in HLF v0.6 and Ripple.** QPS remains stable with increasing nodes for both ledgers.

the validator should process numerous requests. Therefore, if the number of validators and machines running validators remains the same, the consensus does not scale.

Meanwhile, the HLF v0.6 cluster performance fluctuates with the increase of nodes, as well as the HLF BFT-SMaRt and Tendermint. Traditional BFT consensus is used in HLF v0.6 and Tendermint BFT so that the broadcast operation is always involved in the consensus, which creates multiple requests at a time in every node. The bottleneck effect at the consensus leader prevents the consensus protocol from scaling. The sharp decrease of the Tendermint cluster performance proves the limited scalability of BFT-based consensus as well.

As for BFT-SMaRt consensus in HLF, all traffic should go through the centralized nodes called orderers. Though the performance can scale by adding more orderers, it is impossible for the cluster to only have orderers but not have simple clients because of its nature. With the same number of orderers, the performance will not scale if the number of clients increases.

4.4. Latency

With the increase of the request rate, the latencies of both HLF v0.6 and Ripple first increase then decrease sharply, as shown in Fig. 7 and Fig. 8. The increase of the latency is because of the congestion control of the TCP protocol which prevents the computer from serving overwhelming requests simultaneously. The followed decrease is because systems are out of work. With TCP's underlying congestion control, if too much data comes to the server suddenly, the "slow start" mechanism will limit the reception rate on the server side and the "congestion avoidance" will half the congestion window if the network congestion is detected. Therefore, the high request rate will contribute to increasing latency because of triggering the network congestion.

Meanwhile, Fig. 9a and Fig. 9b show that the latency remains fairly stable with the increasing requests. Moreover, with less frequent query requests, the latency remains a small constant value. This result indicates that Tendermint outperforms HLF V0.6 and

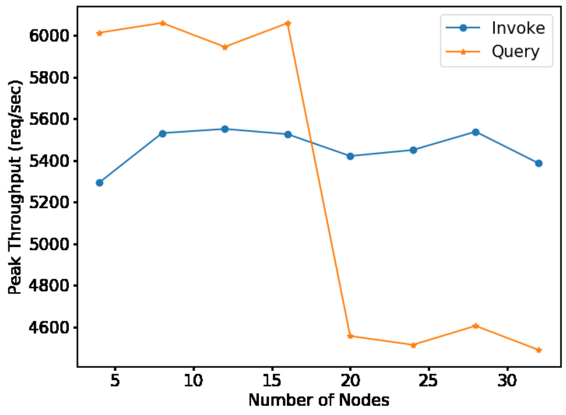


Figure 5: **The peak throughput (invoke and query) with increasing nodes in Tendermint.** The invoking QPS remains stable with increasing nodes. The query QPS remains stable with less than 20 nodes, but drops sharply with more than 20 nodes.

Ripple on both the throughput and the latency.

Fig. 10 shows that the 4-node Corda cluster crashes with more than 50 invocation requests per second, which is the same as Fig. 3.[13] Moreover, the query latency remains very high even when the query is infrequent. This is because the transaction details are retrieved from the Notary node in a synchronous blocking manner.

5. Limitations when setting up HLF v0.6 and Corda

HLF v0.6 depends on Docker containers, where chaincodes are encapsulated and invoked by external requests. HLF clients communicate with chaincodes in Docker containers, which may affect performance, using Unix sockets or HTTP within the machine or endpoint.

The experiment was launched on Ubuntu 16.04 OS. However, Ubuntu 16.04 adopts the `systemd` service manager rather than the previous `initd`, by which Docker starts as a service rather than applying configurations in `/etc/default/docker`. This causes the Unix socket endpoint to remain closed by default. Therefore, starting an HLF client with the same step as on previous versions of Ubuntu will fail on Ubuntu 16.04.

Corda is an enterprise-level DLT platform, which has an emphasis of the privacy of transactions but does not focus on security or high performance. Therefore, setting up the Corda cluster is a complex task with several manual modifications on the configuration file. Firstly, Corda cannot find peer nodes dynamically as the ledger is permissioned. Instead, the peer information is compiled in advance with a network bootstrapping step[14]. Secondly, the peer information contains the IP address of each node. However, only the IP address

[13]We then obtained confirmation from the R3 development team that there is no stable version of Corda that is secure—the BFTSmart support is not stable. And the enterprise version of Corda cannot tolerate Byzantine failures.

[14]`https://docs.corda.net/setting-up-a-corda-network.html#bootstrapping-the-network`.

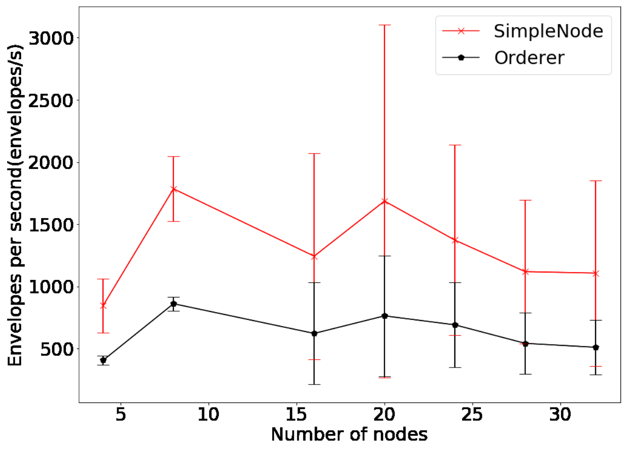


Figure 6: **The peak throughput of a node with increasing nodes in HLF with BFT-SMaRt consensus.** QPS remains stable with increasing nodes.

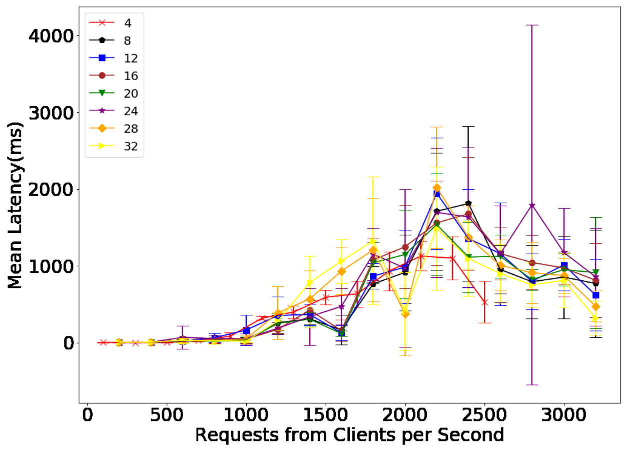


Figure 7: **The latency with the increasing request rate in HLF v0.6.** For all sizes of clusters, the latency increases gradually with increasing request rate, peaks when QPS is approximately 2200 requests per second, then drops.

of a network chip with the highest priority can be recognized, making a private network complex to deploy. In addition, the P2P communication relies on the ActiveMQ Artemis[15]. However, ActiveMQ Artemis does not work well with the Emulab experimental platform which intercepts some ActiveMQ Artemis messages.

6. Conclusion

This paper presents a first evaluation of distributed ledgers that could be adapted for IoT, including Hyperledger Fabric (HLF) with PBFT, Ripple, Tendermint, R3 Corda and HLF with BFT-SMaRt. More specifically, we benchmark selected distributed ledgers running

[15]`https://github.com/apache/activemq-artemis`.

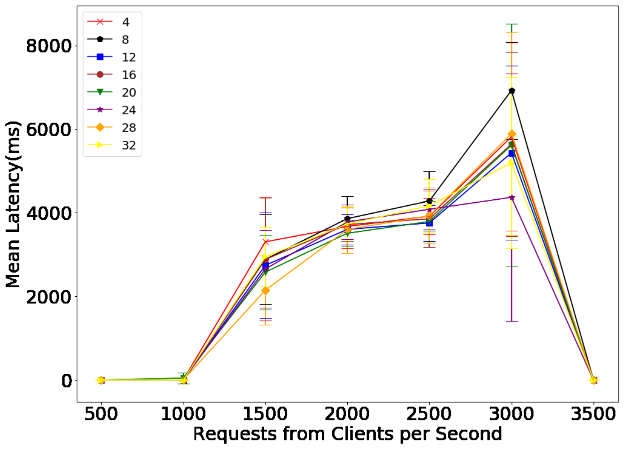
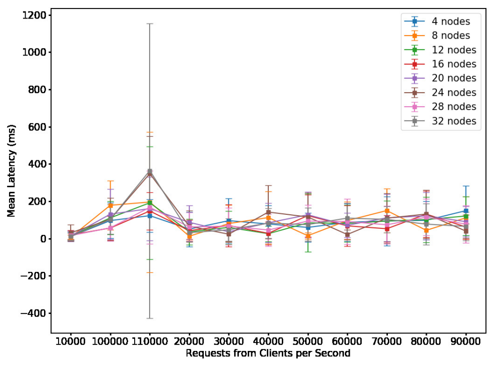
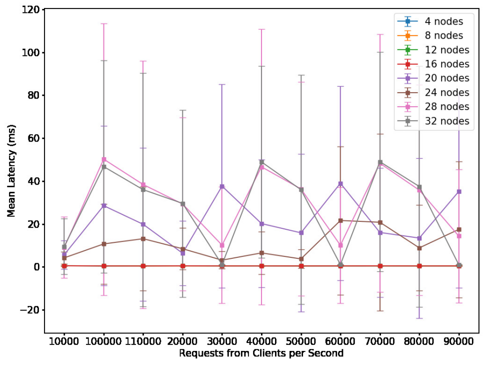


Figure 8: **The latency with the increasing request rate in Ripple.** For all sizes of clusters, the latency increases gradually with increasing request rate, peaks when QPS is 3000 requests per second, then drops sharply to zero.



(a) **Invoke.** For all sizes of clusters, the latency increases gradually with increasing invoke request rate.



(b) **Query.** For clusters with no more than 16 nodes, the latency remains constant and almost zero with increasing query request rate. For clusters with more than 16 nodes, the latency highly fluctuates with increasing query request rate.

Figure 9: The invoke and query latency with increasing request rate in Tendermint.

on clusters with from 2 to 32 nodes, using a unified workload. Our results show that, unfortunately, they cannot scale in terms of throughput and latency on reasonably sized machines. This research confirms empirically the well-known inadequacy of classic consensus to large-scale network. Our future work consists of evaluating distributed ledgers that can scale by solving the blockchain consensus [25] rather than the classic Byzantine consensus. Another interesting direction is to identify IoT applications that do not need consensus but simply a verified reliable broadcast [24].

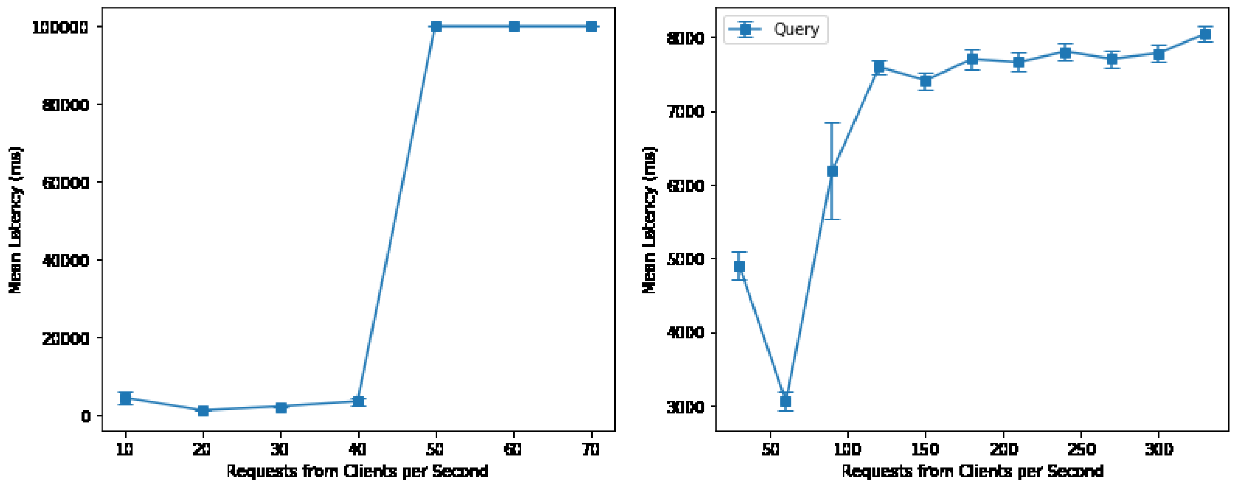


Figure 10: **The latency with increasing query rate of the 4-node Corda cluster.** The cluster crashed with more than 50 invocation requests per second, and the latency remains very high with query operations.

Acknowledgement

We are grateful to Dinh Tien Tuan Anh, Mike Hibler, David Schwartz and Baohua Yang for their help. This research is in part supported under Australian Research Council's Discovery Projects funding scheme (project number 180104030) entitled "Taipan: A Blockchain with Democratic Consensus and Validated Contracts". Vincent Gramoli is a Future Fellow of the Australian Research Council.

References

[1] R. Han, V. Gramoli, and X. Xu, "Evaluating blockchains for IoT," in *Proceeding of the 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018*, 2018, pp. 1–5.
[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[3] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: An introduction," *R3 CEV, August*, 2016.
[4] V. Gramoli, "From blockchain consensus back to byzantine consensus," *Future Generation of Computer Systems*, 2017.
[5] C. Natoli and V. Gramoli, "The blockchain anomaly," in *Proceedings of the 15th IEEE International Symposium on Network Computing and Applications (NCA'16)*, 2016, pp. 310–317.
[6] ——, "The balance attack or why forkable blockchains are ill-suited for consortium," in *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017, Denver, CO, USA, June 26-29, 2017*, 2017, pp. 579–590. [Online]. Available: https://doi.org/10.1109/DSN.2017.44
[7] P. Ekparinya, V. Gramoli, and G. Jourjon, "Impact of man-in-the-middle attacks on Ethereum," in *Proceedings of the 37th IEEE International Symposium on Reliable Distributed Systems*, 2018.
[8] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.
[9] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186. [Online]. Available: http://dl.acm.org/citation.cfm?id=296806.296824

[10] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with bft-smart," in *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on.* IEEE, 2014, pp. 355–362.

[11] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, "DBFT: Efficient leaderless byzantine consensus and its applications to blockchains," in *Proceedings of the 17th IEEE International Symposium on Network Computing and Applications (NCA'18)*, 2018.

[12] D. Schwartz, N. Youngs, and A. Britto, "The ripple protocol consensus algorithm," *Ripple Labs Inc. White Paper*, vol. 5, 2014.

[13] J. Kwon, "Tendermint: Consensus without mining," *Draft v. 0.6, fall*, 2014.

[14] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[15] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," University of Guelph, Tech. Rep., June 2016.

[16] D. Gupta, L. Perronne, and S. Bouchenak, "BFT-Bench: A framework to evaluate BFT protocols," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ser. ICPE '16, 2016, pp. 109–112.

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

[18] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

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

[20] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *International Conference on Trust and Trustworthy Computing.* Springer, 2015, pp. 163–180.

[21] Y. Amoussou-Guenou, A. D. Pozzo, M. Potop-Butucaru, and S. Tucci-Piergiovanni, "Dissecting Tendermint," in *Proceedings of the 7th Edition of the International Conference on Networked Systems (NETYS)*, 2019.

[22] S. Popov, "The tangle," oct 2017.

[23] L. Baird, "The Swirlds Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance," pp. 1–27, 2016. [Online]. Available: http://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf

[24] T. Crain, C. Natoli, and V. Gramoli, "Evaluating the red belly blockchain," arXiv, Tech. Rep. 1812.11747, 2018.

[25] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, "Blockchain consensus," in *ALGOTEL 2017-19èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, 2017.

[26] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for IoT," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, ser. IoTDI '17, 2017, pp. 173–178.

[27] J. Sousa, A. Bessani, and M. Vukolić, "A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform," 2017.

[28] S. Souders, "High-performance web sites," *Communications of the ACM*, vol. 51, no. 12, pp. 36–41, 2008.