An Evaluation of Blockchain Application Requirements and Their Satisfaction in Hyperledger Fabric

Sadok Ben Toumia, Christian Berger, Hans P. Reiser

An Evaluation of Blockchain Application Requirements and Their Satisfaction in Hyperledger Fabric

A Practical Experience Report

Sadok Ben Toumia[1(✉)], Christian Berger[1], and Hans P. Reiser[2]

[1] University of Passau, Passau, Germany
bentou01@ads.uni-passau.de, cb@sec.uni-passau.de
[2] Reykjavík University, Reykjavík, Iceland
hansr@ru.is

Abstract. Blockchain applications may offer better fault-tolerance, integrity, traceability and transparency compared to centralized solutions. Despite these benefits, few businesses switch to blockchain-based applications. Industries worry that the current blockchain implementations do not meet their requirements, e.g., when it comes to scalability, throughput or latency. Hyperledger Fabric (HLF) is a permissioned blockchain infrastructure that aims to meet enterprise needs and provides a highly modular and well-conceived architecture. In this paper, we survey and analyse requirements of blockchain applications in respect to their underlying infrastructure by focusing mainly on performance and resilience characteristics. Subsequently, we discuss to what extent Fabric's current design allows it to meet these requirements. We further evaluate the performance of Hyperledger Fabric 2.2 simulating different use case scenarios by comparing single with multi-ordering service performance and conducting an evaluation with mixed workloads.

Keywords: Hyperledger Fabric · Distributed Ledger Technology · Application Requirements · Blockchain · Performance · Scalability · Benchmarking

1 Introduction

Since the invention of Bitcoin [24], many people are speculating about how blockchain can revolutionize our daily lives. Several sectors can profit from blockchain, whereas for many other areas it is considered an overkill [35]. Today, a number of industries still struggle with basic concerns like traceability, integrity protection, or privacy [8,16,17]. Competition is higher than ever, which makes certain parties secretive about their transactions. Such issues are often not being sufficiently handled by traditional applications.

Published by Springer Nature Switzerland AG 2022
D. Eyers and S. Voulgaris (Eds.): DAIS 2022, LNCS 13272, pp. 3–20, 2022.
https://doi.org/10.1007/978-3-031-16092-9_1

This raises the demand of a platform that can handle these issues while meeting their standards in respect to resilience and performance [8,16]. Enterprises often need a permissioned blockchain that restricts participation to a consortium of members. Due to competitors being also on the blockchain network, these parties need privacy: not everyone should be able to see all their transactions – instead transactions must be on a need-to-know basis [7].

Hyperledger Fabric (HLF) [1] is an open-source, permissioned blockchain platform that intends to satisfy enterprise application requirements. It presents a modular architecture with pluggable consensus and can achieve high throughput. Previous studies have highlighted the issue of missing support for Byzantine fault tolerance (BFT) [31]. Starting from version 2.0, HLF switched from a Kafka-based ordering service to a Raft [25]-based ordering service. While Raft (like Kafka) does not assume BFT, it could be transformed to do so in future, and could essentially be a step towards implementing BFT in HLF.

We think that it is important to discuss and validate how far design decisions like these, which concern the infrastructure of a blockchain system, match up with the requirements concrete applications impose towards the underlying blockchain infrastructure.

Contribution and Outline. Our main contribution consists in investigating relevant requirements of blockchain applications and discussing how far these are addressed in HLF. In the remainder of this report, we refer to related work (Sect. 2), provide relevant background knowledge (Sect. 3), explain our methodology (Sect. 4) and conduct a requirements analysis for blockchain applications selected from different use-cases (Sect. 5). Further, we analyse design choices HLF makes to match these requirements (Sect. 6) and investigate on the question whether HLF can satisfy performance requirements by conducting experiments for different scenarios (Sect. 7). Finally, we draw our conclusions (Sect. 8).

2 Related Work

Li et al. [21] have recently published a survey paper highlighting Hyperledger Fabric and Hyperledger Composer's use-cases. The paper examined current theoretical and real-life HLF deployment while highlighting how HLF was used as a solution to solve existing enterprise problems. A recently published dissertation [8] has studied the requirements and unresolved issues of supply chains while also proposing architectures based on blockchains to address these issues, the main focus was however restricted to supply chain management.

Several papers have included benchmarks for Hyperledger Fabric [1,3,11,30,33], mainly focusing on the `v1.x` versions of Fabric with FastFabric pushing an impressive 20,000 transactions per second (TPS) [11]. Androulaki et al. proposed the architecture, components and design choices behind HLF and experimentally validated the system performance [1] whereas Thakkar et al. studied how various parameters of the network impacted performance such as number of channels, number of endorsers and world-state database choice [33] - their proposals were incorporated in future Hyperledger Fabric versions.

Further, Guggenberger et al. [12] have recently published a detailed performance report for Hyperledger Fabric v2.0 combining several configurations and testing Fabric's fault-tolerance using DLPS [29]. Their report covers an in-depth benchmark analysis of HLF.

In our report, we focus on discussing *application requirements* of blockchain and how these requirements are met by HLF's design. Our report also includes a performance evaluation that conducts experiments on multi-ordering service performance and mixed workloads (e.g., read-heavy vs. write-heavy) which have yet not been sufficiently studied by previous works but are interesting from an application point of view.

3 Background

Blockchain. The term blockchain is not used consistently in academic literature. Our definition emphasizes that we are referring to a complete system rather than just a specific data structure:

A *blockchain* is a distributed system that manages an append-only and totally-ordered log of immutable transactions (also called the *ledger*) in a replicated fashion. Several nodes hold a consistent copy of the ledger, and several nodes are involved in validating transactions issued by clients. To order transactions, typically a *consensus* algorithm is employed. Further, transactions are usually grouped into *blocks*, which are chained by referencing the hash of the previous block in the block header.

The traceability and immutable history of transactions in a blockchain fundamentally increase the trustworthiness and transparency of the system. As long as a sufficiently large portion of nodes in the system (e.g., determined by quantity, resource allocation, or stake) behaves correctly, the overall system works as intended. There is no need to put trust into the correctness of any single node, thus eliminating single point of failure for blockchain applications. Immutability refers to the property that each block bears also the hash of the previous block, and a modification to a block modifies its hash, which results in the link being broken and thus invalidating subsequent blocks [31,37].

Hyperledger Fabric (HLF). HLF is a highly modular enterprise-grade distributed ledger platform. HLF has plug and play capabilities that allow it to be suitable for a wide range of use-cases. Further, HLF follows an *execute-order-validate* architecture, where transactions are first simulated (this means executed against the current state of the ledger) by endorsing peers, ordered by the ordering service, and then committed by committing peers.

The *transaction flow* consists of the following steps (also shown in Fig. 1) [1]: The *client* sends a transaction proposal to the peers specified in the endorsement policy and the *endorsing peers* simulate the transaction (①), which produces read sets and write sets, without changing the state of the ledger. After that, the client collects the responses of the endorsing peers (②), which contain the read and write sets, checks if the endorsement policy is satisfied, assembles them in an *envelope* and sends that to the *ordering service* (③). Subsequently, the ordering

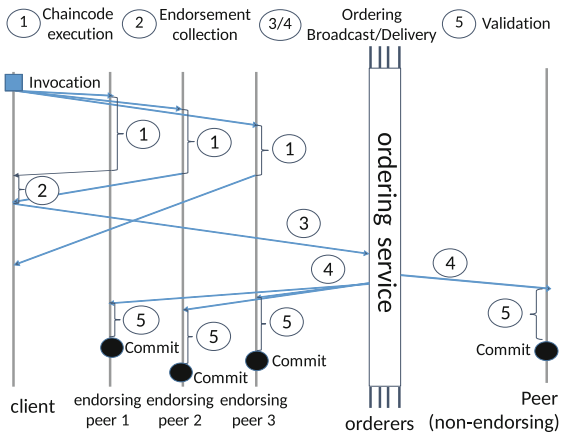


Fig. 1. Hyperledger fabric transaction flow [1].

service orders the transaction without knowing the contents of the envelope. Transactions are batched and once one of the conditions for cutting a block is met, the ordering service sends the block to the committing peers (④). Finally, committing peers validate or invalidate the transactions in the block and the block is eventually appended to the *ledger* (⑤).

4 Methodology

Our evaluation approach covers two dimensions: *performance* and *resilience*. Performance characteristics are quantitative (e.g., transaction throughput) and are used to assert that the blockchain can handle the application's workload. Resilience characteristics are often qualitative and describe if the blockchain can deliver a certain service quality (e.g., tolerating faults, providing confidentiality).

Aspects of Resilience. Resilience is a broad term that encompasses many aspects [5]. We employ the following aspects in our subsequent analysis:

Fault Tolerance Coverage (FTC). A measure of effectiveness for fault tolerance is *fault tolerance coverage* [2]: it encompasses the *error- and fault-handling coverage*, a measure to capture how many of the occurring faults are actually covered by the fault tolerance mechanism (development faults might restrict the intended fault-handling coverage) and the *fault assumption coverage*, which is a measure for reasoning about how closely assumptions of a fault model actually cover reality. In our analysis we employ assumption coverage to indicate which type of faults a blockchain can tolerate.

Fault Tolerance Proportion (FTP). Fault tolerance proportion is an assumed upper-bound that indicates the ratio of faulty nodes a blockchain can tolerate, to total nodes participating in the system (this property is also sometimes called resilience bound). Fault tolerance coverage and proportion are often coupled.

Table 1. Categorization of performance requirements.

	Low	**Medium**	**High**
Scalability	<100 nodes	100 to 1000 nodes	>1000 nodes
Throughput	<100 TPS	100 to 1000 TPS	>1000 TPS
Latency	<3 s	3 s to 10 s	>10 s

Membership (Node Authenticity). In permissioned blockchains a consortium of nodes is defined and a mechanism for managing membership is required. Providing membership information and node authenticity is an important feature for blockchains and blockchain applications might demand the blockchain system to be capable to changing (e.g., expanding) its consortium at run-time.

Confidentiality. There are different *types* of application requirements associated with confidentiality depending whether the content, sender, or other information of a transaction need to be confidential.

Integrity. Integrity is a main motivator towards blockchain adoption. Data integrity in blockchains is achieved by the immutability property of the ledger. Undetected tampering is almost infeasible, as hashes can be used to quickly validate for correctness. We argue that all blockchain applications share the need of this characteristic and will thus not use it in a comparison.

Aspects of Performance. We consider typical blockchain performance aspects that application might demand, in particular:

Scalability. Number of nodes that can participate in the blockchain system.

Throughput. Number of transactions per second (TPS) that can be processed by the blockchain system.

Latency. Time that elapses between a transaction being issued on the client side and being finalized within a block that is appended to the ledger.

For each aspect, we categorize performance requirements of blockchain applications into three categories: *low*, *medium* and *high* as shown in Table 1. This categorization is rough and aligns with performance magnitudes of blockchain systems. Achieving low latency is better and thus means a higher requirement towards the blockchain infrastructure. For the other aspects, higher is better.

5 Requirements Analysis

HLF is currently being used in a number of fields [23], some of them are already in production, but most of them are still in development or proof-of-concept status. These use-cases are high-risk environments with a lot at stake where some parties could be interested in gaining unauthorized access or tampering with the data for personal gain, thus requiring a highly resilient infrastructure.

In this section, we analyse use-cases and derive which of the characteristics (Sect. 4) are required by which application and present a summary in Table 2.

5.1 Electronic Voting (EVote)

EVote [27,28] is an open-source proof-of-concept application for holding an electronic election. The app leverages HLF to meet its needs for immutability and traceability, which in return reduce election fraud. Smart contracts are used to tally up votes, therefore reducing costs of manual work [28]. A voting network to hold an election is a highly adversarial environment that might encourage malicious behaviour of individuals. Therefore, we consider it valuable for such a system to be Byzantine fault-tolerant (and to tolerate up to 33% of participants becoming faulty). The system should be permissioned. It should further provide high confidentiality: When votes need to be checked for their validity (to prevent double voting) they should be untraceable to the voter to prevent any form of coercion. Subnetworks could help to enforce a need-to-know policy for different entities involved in the process.

From a performance standpoint, a high latency is tolerable in such a network, because voting is per user a one-time action. It should not exceed 30 s to maintain a pleasant user-experience. To maintain such latency, the system needs at least a medium throughput, as elections are usually held in a small time period where at peak times, many transactions are issued. In such a use-case, the scalability of the system has another goal other than being able to handle such a traffic and that is transparency and ensuring that not a single entity has more control over the voting process. An approach for this might be to have every election district host a peer node (or more in order to avoid a single point of failure) and as such a medium to high scalability becomes a requirement.

5.2 Supply Chains (IBM Food Trust and GoDirect Trade)

GoDirect Trade [16] is a practical use case for blockchain technology, offering an online marketplace for aerospace parts. The traceability feature of blockchain allows users to access the lifecycle of parts and any associated information required by the government. IBM Food Trust [17,20] is a project by Walmart, IBM, Nestle, and Unilever aimed at improving traceability of products and all their ingredients to the farms and also to access different data about the product to satisfy customer needs and guarantee the safety of foods [17]. IBM Foodtrust and GoDirect Trade both utilize the immutability and traceability aspects of blockchain, in that both are interested in the history and provenance of items recorded on the immutable ledger.

From a resilience perspective of view, IBM Foodtrust could go well with BFT (and have a resilience bound of 33%) whereas GoDirect Trade could benefit from using only CFT (and having a resilience bound of 50%). Contrary to IBM Foodtrust, where participants could bring up their own peers and deploy their own smart-contracts, GoDirect Trade's nodes are in-house [16]. Further, both applications need to be run on a permissioned blockchain, where participants are granted access based on their status on the market. Moreover, both systems require high confidentiality, as trade secrets are at stake here as in both networks competitors are present.

Table 2. Blockchain applications requirements towards the underlying blockchain infrastructure with very high (★), high (●), medium (◓) and low (◶) demands. (Note that, lower latency is better, and is thus considered a higher demand towards the infrastructure, e.g., tolerating a higher latency as in EVote means a lower requirement.)

	Resilience				Performance		
Application	**FTC**	**FTP**	**Membership**	**Confidentiality**	**Scalability**	**Throughput**	**Latency**
EVote	BFT	33%	Yes	★ (sender, content)	◓	◓	◶
IBM FoodTrust	BFT	33%	Yes	● (content)	◓	●	●
GoDirect Trade	CFT	50%	Yes	● (content)	◶	◓	◓
Change Healthcare	CFT	50%	Yes	★ (sender, content)	◶	◓	◓
Visa B2B Connect	BFT	33%	Yes	● (content)	◓	★	●

Contrary to EVote, where users are usually one-time users, supply chains, due to the globalisation of the markets, are usually comprised of a lot of actors and each one of them uses the network multiple times in a small time frame [8]. In terms of throughput and latency, GoDirect Trade requires only a medium throughput and a medium latency whereas IBM FoodTrust requires a high throughput and a low latency. This is due to the number of incoming transactions where supply chains in the context of food generate a lot more requests than supply chains in the context of aviation. In GoDirect Trade, network clients are not allowed to host their own nodes. As stated in [16] the system operates five validating nodes, which indicates that low scalability might suffice. In contrast, IBM FoodTrust subscribes are allowed to host their own nodes, install their own private smart-contracts on private channels to automate transactions, which indicates that it requires a higher scalability than GoDirect Trade and therefore needs a least medium scalability.

5.3 Healthcare (Change Healthcare)

Change Healthcare [15] is a company with the aim to modernize the American health system. Leveraging HLF the company is able to link providers and payers in a trustful environment to facilitate claims.

As an actor in the healthcare industry, Change Healthcare has to be very wary about how data on their network is handled. Providing access to unauthorized persons has serious legal consequences [26], which is why Change Healthcare needs very high confidentiality and private ledgers. Most importantly, transactions, such as financial or patient data, should be on a subnetwork with only participating entities granted access (need-to-know basis), for example a hospital at which a person was a patient in and the insurance company for claims processing. Similarly to GoDirect Trade, Change Healthcare's nodes are in-house and it can benefit from providing only CFT and having a fault tolerance proportion of up to $\epsilon < 50\%$.

The blockchain network has initially run on six nodes in the company's datacenter but now they are looking towards expanding to the cloud. As such, Change Healthcare only needs low scalability due to the nodes belonging to it like in

GoDirect Trade's case. Currently the system can process 550 transactions per second (TPS) but the company is aiming for a higher number in near future [15]. As such, the blockchain system needs at least medium throughput and works best with low to medium latency to maintain a satisfactory user-experience.

5.4 Banking (VISA B2B Connect)

VISA B2B Connect [34] is a project by VISA to facilitate cross-border and cross-currency payments. It leverages HLF to create a secure and trusted network of financial institutions where international transfers do not have to go through intermediate banks, thus drastically reducing both delays and costs.

The current standard for cross-border cross-currency payments and the main system VISA B2B Connect is challenging is SWIFT, which handles approximately 33.6 million transactions per day. VISA B2B circumvents the shortcoming of traditional banking applications by employing an one-to-many architecture, in which VISA B2B is directly linked to several financial institutions, therefore intermediaries can be bypassed. As a result of this centralization and the SWIFT system as a motivator, such a system requires medium scalability, and very high throughput to be capable of handling peak workloads. Typical other banking methods have varying throughput with PayPal having around 450 TPS [13] and credit-card companies such as VISA itself require 50,000 TPS [11].

This centralization also means VISA's nodes are in-house. However, unlike GoDiectTrade and other companies hosting their nodes in-house VISA should employ BFT along with a FTP of up to 33%. The nature of this system makes attacks highly rewarding and insider attacks are a legitimate concern, such as if a participant is compromised or participant himself being dishonest.

6 How HLF Meets Enterprise Requirements

In this section, we focus on the design considerations and features of Fabric that allow it to meet performance and resilience requirements of potential use-cases.

6.1 Resilience Requirements

In the following, we highlight Fabric's features, components and design choices while briefly explaining their role in increasing resilience.

Blockchain Features. Maintaining integrity of the data is a critical aspect of resilience and a priority for businesses. HLF, being an implementation of blockchain, comes with both immutability and traceability. Data is immutable once appended to the ledger, this way users can insure its integrity [1].

Permissions. Fabric is a permissioned blockchain, permissions are maintained by one or more membership service providers (MSP) which use cryptographic identities. Transactions are checked at every step to verify authenticity of requests. This in turn limits unwanted access and increases trust [1].

Channels. Unlike other blockchain implementations, HLF uses *channels*. A channel is a dedicated subnetwork with its own private ledger and a group of channel members that manage a copy of the ledger, thus ensuring that not every peer on the network has access to the ledger, therefore increasing confidentiality.

Endorsement Policy. Channel administrators define the endorsement policy, which specifies which peers (*endorsers*) have to approve a transaction before this is sent to the ordering service. If a client does not fulfill an endorsement policy he has to retry submitting the transaction again [1]. An endorsement policy consisting of multiple peers belonging to different organizations would increase transparency and trust in the system, as no single entity is in full control of endorsing transactions. A single point of failure can be avoided by defining a minimum number or percentage of endorsing peers.

Consensus. An appealing feature of HLF is pluggable consensus. Older versions of HLF use Kafka + ZooKeeper (ZK), while the current default consensus protocol is Raft. Both Raft and Kafka+ZK are crash fault-tolerant. Since consensus is pluggable, developers could opt for a BFT ordering service in future as a new BFT consensus library has been proposed for HLF recently [4]. Raft is embedded into HLF and thus enjoys the direct support of the HLF community whereas Kafka+ZK are supported by Apache. In terms of performance, a published benchmark [10] with v1.4.1 showed that Raft can be much more efficient.

Resilience of the Execute-Order-Validate Design. HLF employs an execute-order-validate architecture to separate these different concerns. A goal of this design is to help withstanding attacks that may target performance degradation or resource exhaustion. In particular, this design can help to circumvent bottleneck situations since it allows for transactions to be processed in parallel and by only a subset of nodes.

Peer Gossip. Peer gossip enables peers' ledgers stay in sync by distributing data to other peers on the channel. This aids resiliency in that peers that have gone offline for sometime are able to have synced ledgers and can endorse transactions again after they are back online.

Records of Invalid Transactions. All transactions in HLF, in contrast to other blockchain implementations, are recorded on the ledger whether they are valid or invalid. This allows dishonest or malicious users to be detected and black-listed from the network which results in a more secure platform [1].

Identity Mixer. HLF supports the use of *identity mixer* (Idemix) to enhance privacy by providing unlinkability and anonymity – this however, comes with limitations such as not being able to endorse transactions. An Idemix entity (issuer) certifies a user's attributes in form of a digital certificate, users are then able to generate a zero-knowledge proof of possession of a certificate while revealing only what they choose to reveal to a verifier.

Hardware Security Module. HLF supports the usage of hardware security modules (HSM) allowing cryptographic operations like signature generation to

be offloaded to them. This has the advantage of letting the HSM manage private keys of peers or orderers, thus protecting the keys from unauthorized reading.

Transport Layer Security. Communication over a HLF network can be secured using TLS. This can be a one-way or a two-way authentication.

Private Data Collections. Channels support data privacy by having only organizations on the channel that are allowed to view these transactions. In cases where a subset of channel members need to conduct transactions between each other while not wanting other channel members to know the contents of these transactions, they could create a new channel. This is however associated with a higher administrative overhead. A solution for this would be the usage of *private data collections*[1]. Private data has a separate transaction flow compared to other data on the channel. Only authorized peers can see private data and it is communicated between them using gossip, all other nodes including ordering nodes only see hashes of this data, non-authorized nodes append the hashes of this private data in their ledgers, so they know a transaction has taken place privately between entities on the channel but they do not know its content. To comply with government regulations, some organizations might need to delete private data after a certain time, this is doable and will leave behind a hash in the peer's ledgers as evidence that some data was there [18].

Chaincode Lifecycle. Introduced in `v2.0`, the new Fabric chaincode lifecycle requires organizations participating in the endorsement process to approve a transaction. Previously, in `v1.x` one organization would define attributes of a chaincode and other organizations choose either to opt-in by installing the chaincode or opt-out and not be able to endorse transactions. The chaincode lifecycle provides equality on a channel by allowing the chaincode to be instantiated only after gathering enough approvals. Chaincode packages also do not need to be identical anymore, different organizations can install different chaincode packages and introduce organization-specific behaviour (for example perform different validations for their interests). This does not conflict with transaction approval as long as endorsement results match [19].

6.2 Performance Requirements

Further, some of Fabric's design choices were made to increase its performance.

The Advantage of Execute-Order-Validate. In HLF execution and ordering of transactions are separated. This allows for better scalability for both phases while increasing modularity and performance because of the decreased amount of work a node has to do [1]. Some blockchain implementations use an order-execute architecture, but this design has its limitations. HLF uses an execute-order-validate approach to allow for parallel execution and eliminate non-determinism of smart contracts (transactions can be processed by a subset of endorsers) therefore increasing throughput and decreasing latency [1].

[1] See https://hyperledger-fabric.readthedocs.io/en/release-2.2/private-data/private-data.html, last accessed 12-22-2020.

How Channels Help Performance. Dividing the network into channels where each channel is serving a purpose and linking a subset of the organizations on the network while having their own endorsers can increase performance due to the decreased workload. This is HLF's version of sharding (HLF can scale up horizontally using channels), which has frequently been proposed to increase performance in blockchains [9,10,36]. Generally, the idea of parallelizing transaction processing is an important scalability technique [6].

Peer Gossip. The optional peer gossip feature allows for better performance. The throughput of the ordering service is limited by the network capacity of its nodes, and adding more nodes can decrease throughput. This service elects a leader per organization that pulls blocks from the ordering service and distributes them to the rest [1]. This reduces the workload of the ordering service.

BatchSize and BatchTimeout. The ordering service in Fabric uses batching and forms blocks out of transactions. A new block is created if (1) the number of transactions in the block is equal to the maximum allowed, (2) the block's size in bytes has reached max, or (3) an amount of time has passed since the first transaction of a new block was received [1]. The parameters *BatchSize* and *BatchTimeout* are customizable, allowing adaptation to the use case. If, however, the wrong values are chosen Fabric's performance can be heavily affected [14].

Supporting Multiple Ordering Services. The ordering service is usually responsible for multiple channels. As the number of channels grows the load on the ordering service grows, scaling the ordering service leads to a performance decrease [1]. In cases where adding more channels would overwhelm the ordering service, a new ordering service instance can be brought up [10].

World-State Database Choice. Recent work [22] has investigated the difference in performance between the supported world-state databases in Fabric. Mostly with lower BatchSizes, LevelDB has shown better performance than CouchDB, but CouchDB offers better functionality through *rich queries*[2]. Applications should again make trade-offs here of whether they want more functionality in a database or a better performance. FastFabric has experimented with an in-memory hash table as a ledger [11] and achieved a large increase in throughput (from 3200 to 7500 TPS).

7 Performance Evaluation

In this section, we aim to examine HLF's performance under different conditions similar to real-world use-cases in terms of setup and transaction loads. For our purposes, we use Hyperledger Caliper[3], a state of the art tool for benchmarking different blockchain platforms such as Hyperledger Fabric and Ethereum.

[2] See https://hyperledger-fabric.readthedocs.io/en/release-2.2/couchdb_as_state_database.html, last accessed 12-22-2020.

[3] See https://www.hyperledger.org/use/caliper.

In the first part of our evaluation, we focus on the ordering service by examining the benefits of operating a secondary ordering service whilst scaling the number of channels as well as the transaction load. In the second part of this evaluation, we investigate how HLF performs under different mixed application workloads in terms of read and write operations, thereby simulating real-world scenarios. To mimic a real application, we exemplary choose a *fabcar* chaincode deployment. The *fabcar* is a simple chaincode that allows users to add or change data (to be concrete: cars and their ownership) on the ledger using the Fabric contract API[4]. This way we can observe the effect of concurrent reads and writes, i.e., users browsing listings and at the same time users creating new listings.

For our purposes, we may employ setups (Fig. 2) with an increasing number of nodes. We are running each node on a 4 vCPU, 6 GB RAM Debian VM running in a private OpenNebula cloud in our university's virtualization farm.

7.1 Multi Ordering Services Performance

We examine the benefits of operating a secondary ordering service. The need for a secondary ordering service could arise when the first ordering service is already operating at a high load and servicing a high number of channels or to include only a certain subset of organizations in the ordering phase for certain channels.

Setup. We experiment with 8, 16 and 28 two-peer channels where each peer is a member of 2, 4 or 7 channels respectively (Fig. 2). For the load generation, we used a suitable number of workers for each workload, since employing too many workers can result in inaccuracies in terms of maintained transaction load, while too few workers may not be able to maintain the desired load.

Method. In this experiment we scale up the number of channels while experimentally controlling the transaction loads with Hyperledger Caliper. Note that, for a multi-ordering service setup, each orderer manages half the channels and processes and as such half the transaction load. Further, we use Caliper for the load generation and performance measurement, where each invocation of the

```
Channels: [variable]
Organizations: 8
Peers per Organization: 1
Peers per Channel: 2
Nodes per Ordering Service: 3
MaxMessageCount: 10
BatchTimeOut: 0.5
Endorsement: 50%
Chaincode: fabcar
```

(a) Multi channels setup.

```
Channels: 1
Organizations: 8
Peers per Organization: 1
Peers per Channel: 8
Ordering Service Nodes: 5
MaxMessageCount: 100
BatchTimeOut: 0.4
Endorsement: 50%
Chaincode: fabcar
```

(b) Mixed workload setup.

Fig. 2. Setups used for the evaluation.

[4] See https://github.com/hyperledger/fabric-samples/tree/master/chaincode/fabcar.

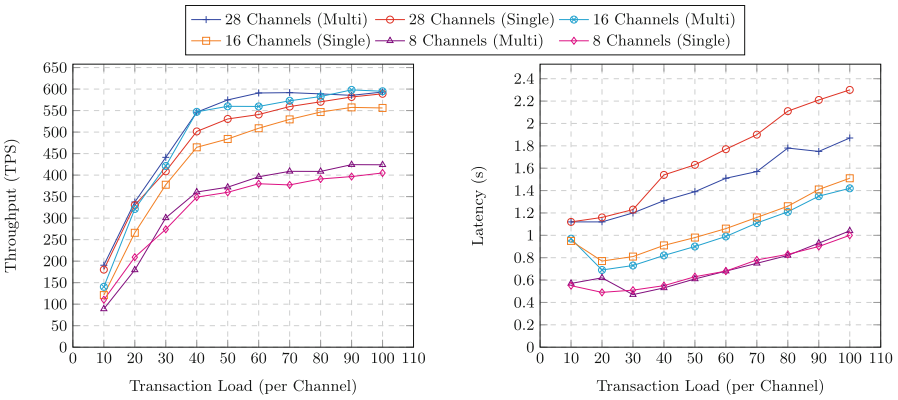
`submitTransaction()` method generates a new transaction per channel thus guaranteeing a fair load distribution among channels and ordering services.

Caliper provides rate controllers to conduct different types of experiments. For our purposes, we decided on the *fixed-load* controller which we slightly modified because it was too inaccurate in terms of maintaining a constant (or minimally oscillating) load. For this, we have overestimated the perceived network throughput in the controller which minimized the delta between the specified *transactionLoad* and the actual load at any time during the experiment.

Observations. We make the following observations:

Observation 1: Figure 3a shows that throughput is continuously increasing as the transaction load increases and converges to approximately 600 TPS for 28 and 16 channel deployments and to 425 TPS for 8 channel deployments. Increasing the number of channels increases throughput, a 100 Requests per Second (RPS) transaction load per channel achieves approximately 425 TPS for 8 channel deployments and 600 TPS for 16 channel and 28 channel multi-ordering service deployments. The same holds for a network level load, a 800 RPS network load (50 RPS per channel for a 16 channel deployment and 100 RPS per channel for an 8 channel deployment) achieves approximately 550 TPS at 16 channels compared to 425 TPS at 8 channels.

Takeaway 1: Increasing the number of channels increases throughput. The difference in throughput between a 28 channel and 16 channel setup is insignificant with both setups reaching a peak of approximately 600 TPS whereas for an 8 channel setup the peak is reached at approximately 425 TPS. Increasing transaction load also increases throughput however the throughput converges after a certain point.



(a) Throughput comparison of multi ordering services and single ordering service.

(b) Latency comparison of multi ordering services and single ordering service.

Fig. 3. Ordering services setups in HLF with variable number of channels.

Observation 2: Figure 3b shows that increasing the number of channels leads to an increase in latency. For 28 channel setups the latency difference between a single ordering service setup and a multi-ordering service setup is somewhat significant with approximately 2.4 s and 1.8 s, respectively.

Takeaway 2: Increasing the number of channels increases latency. This increase is more noticeable in single ordering service setups. Latency has increased by more than 100% between 28 channel and 8 channel setups.

Observation 3: Having a secondary ordering service results in a small throughput increase (¡20 TPS) and a slight latency improvement.

Takeaway 3: A multi-ordering service setup does not seem to lead to a significant performance increase, at least when the ordering phase is not the bottleneck. Several papers have highlighted the validation phase being the bottleneck in HLF [3,11]. In light of this, using a secondary ordering-service solely to improve performance is not beneficial. However, a secondary ordering service makes sense when it comes to separating concerns, i.e., a party that is operating an OSN in the first ordering service and is not involved in channels belonging to the second ordering service, can be excluded from ordering for privacy or security reasons.

7.2 Mixed Workloads

In this experiment we investigate how HLF performs with mixed application workloads. For this reason, we measure performance for different read-to-write ratios, in particular *mostly write* (20/80 read/write), then *mostly read* (80/20 read/write), and also *equal usage* (50/50 read/write).

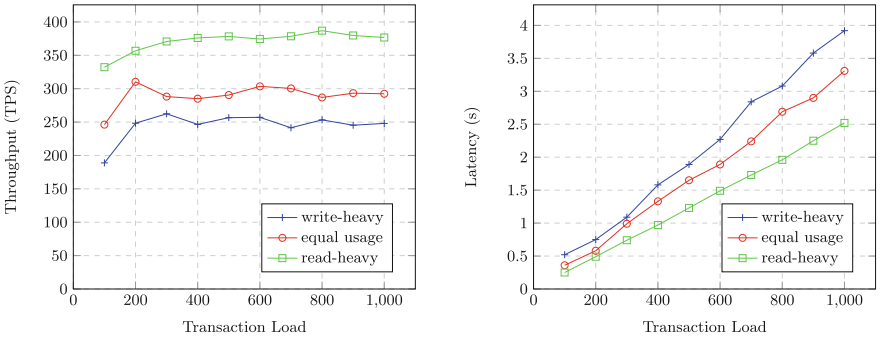
Setup. Our mixed workloads deployment, is similar to our multi-ordering-service deployment in terms of number of organizations and endorsement. We are using five ordering-service nodes for this deployment since this is a more suitable option in practice (Fig. 4).

Method. We evaluate the performance for increasing input rates for which the system is under a transaction load of 100 requests per second to 1000 RPS at any given time depending on the setup. Further, each invocation of the method `submitTransaction()` results in the generation of a single read or write transaction with a certain probability, e.g., for 20/80 read-write ratio, the probability that a read operation is generated equals 20%.

Observations. Overall, we make the following observations:

Observation 1: The read-heavy workload achieves the highest throughput. The difference between a write-heavy and a read-heavy workload is significant with approximately 120 TPS difference. Equal usage achieves a decent throughput of about 300 TPS, i.e., a 50 TPS increase compared to a write-heavy workload and a 70 TPS decrease compared to a read-heavy workload at 1000 RPS.

Takeaway 1: The read-heavy workload results in a noticeable throughput increase when compared to a write-heavy workload with 376 TPS and 248 TPS respectively at 1000 RPS.



(a) Throughput comparison. (b) Latency comparison.

Fig. 4. Employing different mixed read/write application workloads in HLF.

Observation 2: Latency increases with an increased transaction load. The mostly write workload achieves the worst latency with approximately 4 s at 1000 RPS. Note that the difference between the latencies of the individual workloads is more noticeable at higher transaction loads.

Takeaway 2: An increased read-to-write ratio results in a latency decrease with approximately 4 s at a write-heavy workload compared to approximately 2.5 s at a read-heavy workload.

7.3 Discussion

The obtained results indicate that HLF achieves performance of several hundreds of transactions per second even on commodity hardware. It is performance-wise superior to some other blockchain platforms, e.g., Ethereum (as of time of writing). Applying our results to the aforementioned applications, it seems that HLF meets their requirements to a certain extent. For GoDirect Trade and Change Healthcare, HLF proves to be a perfect fit as a platform. For other applications such as Visa B2B Connect and EVote, Fabric lacks BFT support, which is vital in adversarial environments. However, HLF's modularity allows to opt for a BFT ordering service to meet the resiliency requirements of these applications. Such a setup was demonstrated using BFT-SMaRt [31] and SmartBFT-Go [4], respectively. There are also plans for the Mir-BFT library [32] to be eventually integrated into HLF as its ordering service, thus replacing Raft. Further, for VISA B2B Connect and payment settlement in general, HLF could be a bit slow due to the massive workload (in particular of peak loads) such applications bear. Summarizing, HLF, compared to other solutions, already meets most business requirements performance and security-wise with some trade-offs, and future releases could potentially narrow the gap between enterprise requirements and HLF, especially the planned introduction of BFT.

8 Conclusion

Enterprises previously had minimal interest in blockchains due to the scalability and performance issues. This is however continuously changing in the recent years. The use cases discussed in this paper, as shown in Sect. 5, all have different needs which make the modularity, customizability, privacy features and the coinless nature of Hyperledger Fabric attractive. HLF, on its part, tries to meet these needs by mainly diverting from traditional architectures like order-execute and by increasing privacy through the usage of channels and private data collections. Its design also allows it to be integrated easily, in the way, that potential users can setup their own certificate authority or employ their own version of an ordering service. Previous work and our own experiences with HLF show that it is progressing towards being more decentralized while setting new performance and security standards for other blockchain platforms.

Acknowledgements. This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant number 446811880 (BFT2Chain).

References

1. Androulaki, E., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: 13th EuroSys Conference, pp. 1–15. ACM (2018)
2. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. Trans. Dep. Sec. Comp. **1**(1), 11–33 (2004)
3. Baliga, A., Solanki, N., Verekar, S., Pednekar, A., Kamat, P., Chatterjee, S.: Performance characterization of hyperledger fabric. In: Crypto Valley Conference on Blockchain Technology (CVCBT), pp. 65–74. IEEE (2018)
4. Barger, A., Manevich, Y., Meir, H., Tock, Y.: A Byzantine fault-tolerant consensus library for hyperledger fabric. In: International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–9. IEEE (2021)
5. Berger, C., Eichhammer, P., Reiser, H.P., Domaschka, J., Hauck, F.J., Habiger, G.: A survey on resilience in the IoT: taxonomy, classification, and discussion of resilience mechanisms. ACM Comput. Surv. (CSUR) **54**(7), 1–39 (2021)
6. Berger, C., Reiser, H.P.: Scaling byzantine consensus: a broad analysis. In: 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, pp. 13–18 (2018)
7. Cocco, S., Singh, G.: Top 6 technical advantages of hyperledger fabric for blockchain networks (2018). https://developer.ibm.com/technologies/blockchain/articles/top-technical-advantages-of-hyperledger-fabric-for-blockchain-networks/. Accessed 22 Dec 2020
8. Costa, P.M.L.: Supply chain management with blockchain technologies (2018). https://repositorio-aberto.up.pt/bitstream/10216/114335/2/278462.pdf. Accessed 22 Dec 2020
9. Dang, H., Dinh, T.T.A., Loghin, D., Chang, E.C., Lin, Q., Ooi, B.C.: Towards scaling blockchain systems via sharding. In: International Conference on Management of Data, SIGMOD 2019, pp. 123–140. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3299869.3319889. Accessed 22 Dec 2020

10. Ferris, C.: Does hyperledger fabric perform at scale? (2019). https://www.ibm.com/blogs/blockchain/2019/04/does-hyperledger-fabric-perform-at-scale/. Accessed 22 Dec 2020
11. Gorenflo, C., Lee, S., Golab, L., Keshav, S.: FastFabric: scaling hyperledger fabric to 20 000 transactions per second. Int. J. Netw. Manage. **30**(5), e2099 (2020)
12. Guggenberger, T., Sedlmeir, J., Fridgen, G., Luckow, A.: An in-depth investigation of performance characteristics of hyperledger fabric. CoRR abs/2102.07731 (2021). https://arxiv.org/abs/2102.07731
13. Hartnett, S.: When it comes to throughput transactions per second is the wrong blockchain metric (2018). https://energyweb.org/2018/05/10/when-it-comes-to-throughput-transactions-per-second-is-the-wrong-blockchain-metric/. Accessed 22 Dec 2020
14. Hua, S., Zhang, S., Pi, B., Sun, J., Yamashita, K., Nomura, Y.: Reasonableness discussion and analysis for hyperledger fabric configuration. In: International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–3. IEEE (2020)
15. Hyperledger.org: Case study: change healthcare using hyperledger fabric to improve claims lifecycle throughput and transparency (2019). https://www.hyperledger.org/wp-content/uploads/2019/06/Hyperledger_CaseStudy_ChangeHealthcare_Printable_6.19.pdf. Accessed 22 Dec 2020
16. Hyperledger.org: Case study: honeywell aerospace creates online parts marketplace with hyperledger fabric (2019). https://www.hyperledger.org/wp-content/uploads/2019/12/Hyperledger_CaseStudy_Honeywell_Printable_12.12.19.pdf. Accessed 22 Dec 2020
17. Hyperledger.org: How Walmart brought unprecedented transparency to the food supply chain with hyperledger fabric (2019). https://www.hyperledger.org/wp-content/uploads/2019/02/Hyperledger_CaseStudy_Walmart_Printable_V4.pdf
18. Hyperledger.org: Private data (2020). https://hyperledger-fabric.readthedocs.io/en/release-2.2/private-data/private-data.html. Accessed 7 Oct 2021
19. Hyperledger.org: What's new in hyperledger fabric v2.x (2020). https://hyperledger-fabric.readthedocs.io/en/release-2.2/whatsnew.html. Accessed 22 Dec 2020
20. IBM.com: IBM food trust (2019). https://www.ibm.com/downloads/cas/8QABQBDR. Accessed 22 Dec 2020
21. Li, D., Wong, W.E., Guo, J.: A survey on blockchain for enterprise using hyperledger fabric and composer. In: 2019 6th International Conference on Dependable Systems and Their Applications (DSA), pp. 71–80 (2020). https://doi.org/10.1109/DSA.2019.00017
22. Lincoln, N.: Hyperledger fabric 1.4.0 performance information report. https://hyperledger.github.io/caliper-benchmarks/fabric/resources/pdf/Fabric_1.4.0_javascript_node.pdf. Accessed 22 Dec 2020
23. Muscara, B.: Hyperledger fabric use-cases (2020). https://wiki.hyperledger.org/display/LMDWG/Use+Cases. Accessed 22 Dec 2020
24. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2009). http://bitcoin.org/bitcoin.pdf. Accessed 22 Dec 2020
25. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: USENIX Annual Technical Conference (Usenix ATC 2014), pp. 305–319 (2014)
26. Peterson, K.J., Deeduvanu, R., Kanjamala, P., Mayo, K.: A blockchain-based approach to health information exchange networks (2016). https://www.healthit.gov/sites/default/files/12-55-blockchain-based-approach-final.pdf. Accessed 22 Dec 2020

27. Porutiu, H.: Evote (2019). https://github.com/IBM/evote. Accessed 22 Dec 2020
28. Porutiu, H., Bablini, D., Zhang, G., Ryan Bouchard, K.W., Hernandez-Lu, E., Ramamurthy, S.G.: Build a secure e-voting app (2019). https://developer.ibm.com/technologies/blockchain/patterns/how-to-create-a-secure-e-voting-application-on-hyperledger-fabric/. Accessed 22 Dec 2020
29. Sedlmeir, J., Ross, P., Luckow, A., Lockl, J., Miehle, D., Fridgen, G.: The DLPS: a new framework for benchmarking blockchains. In: 54th Hawaii International Conference on System Sciences, p. 10 (2021)
30. Shalaby, S., Abdellatif, A.A., Al-Ali, A., Mohamed, A., Erbad, A., Guizani, M.: Performance evaluation of hyperledger fabric. In: International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), pp. 608–613. IEEE (2020)
31. Sousa, J., Bessani, A., Vukolic, M.: A Byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In: 48th annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 51–58. IEEE (2018)
32. Stathakopoulou, C., David, T., Vukolić, M.: Mir-BFT: high-throughput BFT for blockchains. arXiv:1906.05552 (2019)
33. Thakkar, P., Nathan, S., Viswanathan, B.: Performance benchmarking and optimizing hyperledger fabric blockchain platform. In: 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 264–276. IEEE (2018)
34. VISA.com: Visa B2B connect a network solution for global large-value payments (2019). https://usa.review.visa.com/dam/VCOM/global/partner-with-us/documents/visa-b2b-connect-white-paper.pdf. Accessed 22 Dec 2020
35. Wüst, K., Gervais, A.: Do you need a blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), pp. 45–54 (2018). Accessed 22 Dec 2020
36. Zamani, M., Movahedi, M., Raykova, M.: RapidChain: scaling blockchain via full sharding. In: SIGSAC Conference on Computer and Communications Security, pp. 931–948. ACM (2018)
37. Zheng, Z., Xie, S., Dai, H., Chen, X., Wang, H.: An overview of blockchain technology: architecture, consensus, and future trends. In: International Congress on Big Data (BigData Congress), pp. 557–564. IEEE (2017). Accessed 22 Dec 2020