Improving Trust in a (Trans)National Invoicing System

The Performance of Crash vs. Byzantine Fault Tolerance at Scale

Søgaard, Jonas Sveistrup; Eklund, Peter W.; Herskind, Lasse; Spasovski, Jason

Document Version
Final published version

Published in:
Applied Sciences

DOI:
10.3390/app13126941

Publication date:
2023

License
CC BY

Citation for published version (APA):
Søgaard, J. S., Eklund, P. W., Herskind, L., & Spasovski, J. (2023). Improving Trust in a (Trans)National Invoicing System: The Performance of Crash vs. Byzantine Fault Tolerance at Scale. *Applied Sciences*, *13*(12), Article 6941. https://doi.org/10.3390/app13126941

applied sciences

MDPI

Article

Improving Trust in a (Trans)National Invoicing System: The Performance of Crash vs. Byzantine Fault Tolerance at Scale

Jonas S. Søgaard [1], Peter W. Eklund [2,*], Lasse Herskind [3] and Jason Spasovski [4]

[1] Department of Digitalization, Copenhagen Business School, 2000 Copenhagen, Denmark; jss.acc@cbs.dk
[2] School of Information Technology, Deakin University, Geelong, VIC 3220, Australia
[3] Aztec Labs, 161 Rosebery Avenue, London EC1R 4QX, UK; lasse.herskind@gmail.com
[4] ZTLment, Linnésgade 20A, 2.sal, 1361 Copenhagen, Denmark; jason@ztlment.com
* Correspondence: peter.eklund@deakin.edu.au

Abstract: Crash fault tolerance describes the capability of a distributed system to maintain its proper function despite the occurrence of crashes or failures in one or more of its components. When a distributed system possesses crash fault tolerance, it can be further fortified to achieve Byzantine fault tolerance. Byzantine fault tolerance empowers a distributed system to establish consensus among participants, even when faced with faulty or malicious behavior. Consensus plays a critical role in various tasks, including determining the accurate value of a shared variable, electing a leader, or validating the integrity of a business transaction. Compared to crash fault tolerance, Byzantine fault tolerance instills greater trust because it enables consensus even in the presence of malicious entities. This paper focuses on the performance evaluation of two blockchain solutions that exhibit Byzantine fault tolerance, in contrast to a blockchain solution that demonstrates crash fault tolerance. Specifically, the paper investigates the additional performance requirements associated with the enhanced trust resulting from Byzantine fault tolerance in e-business trading on both national and transnational scales. We analyze the resources needed to operate a business-to-business/business-to-government (B2B/B2G) compliance framework in two distinct geographic scenarios. The first examines the national scale, using Denmark as an example, which is the eleventh largest European country by GDP. The second scenario considers the scale of the European Union (EU) with its 27 member states (plus the United Kingdom).

Keywords: distributed ledger; blockchain; national and transnational e-business; distributed network trust; crash fault tolerance; Byzantine fault tolerance; business system scalability

Citation: Søgaard, J.S.; Eklund, P.W.; Herskind, L.; Spasovski, J. Improving Trust in a (Trans)National Invoicing System: The Performance of Crash vs. Byzantine Fault Tolerance at Scale. *Appl. Sci.* **2023**, *13*, 6941. https://doi.org/10.3390/app13126941

Academic Editors: Gianluca Lax and Mirco Peron

Received: 15 April 2023
Revised: 26 May 2023
Accepted: 5 June 2023
Published: 8 June 2023

1. Introduction

1.1. Trust and e-Business

Trust in a (trans)national invoicing system requires a robust and reliable mechanism to handle faults and ensure system integrity for its participants. Traditionally, trust is intermediated by government-regulated third parties, with banks and financial institutions being obvious examples. Increasingly, more decentralised solutions are being sought that circumvent single (or limited numbers of) authorities. These involve the use of blockchain technologies that allow a financial ledger to be distributed and regulated among the participants. A key idea is that the distributed ledger can be universally agreed upon. Two possible approaches to consider regarding distributed systems that share ledgers are crash fault tolerance and Byzantine fault tolerance. Crash fault tolerance assumes that faulty nodes might fail by crashing, in which case they stop participating in the system while the rest of nodes continue to operate. Byzantine fault tolerance, on the other hand, allows for arbitrary fault behaviours, including malicious actions. For this reason, Byzantine fault tolerant systems are more trusted than those which are only crash fault tolerant.

In a large open and public e-business environment, more trust in the shared financial ledger is desirable, so Byzantine fault tolerant e-business systems are preferred because

they provides better and stronger guarantees. This paper addresses the following questions: what is the impact on performance for the increased trust guaranteed by Byzantine fault tolerance? Furthermore, will the additional overhead required allow a Byzantine fault tolerant e-business system to work on a (trans)national scale?

1.2. Blockchain and e-Business

Blockchain systems possess several desirable characteristics for use as decentralized business ledgers with embedded transactional logic [1], making them suitable for business-to-business (B2B) transactions and business-to-government (B2G) reporting and compliance. The introduction of the Blockchain-based Service Network (BSN) in China in 2020 marked a significant development in the adoption of blockchain technology as a (trans)national business ecosystem. The BSN, a prominent state-backed digital infrastructure company [2], has been implemented across more than 200 cities in China so far, showcasing the emerging reality of the integration of blockchain technology into the business landscape.

In Europe, the objective of establishing a decentralised transnational business ecosystem differs from the BSN initiative. Rather than managing or controlling participation in the economy, the European-case focuses around business reporting and compliance. However, both regions encounter similar architectural and performance-related challenges when it comes to establishing a (trans)national e-business trading platform. These challenges form the central topic of this paper, highlighting the importance of addressing these issues for the successful implementation of such platforms.

Most blockchains possess a notable advantage in their resistance to malicious nodes. They have the capability to withstand a defined number of malicious or faulty processes, represented as f out of a total number of nodes denoted as n given that n is equal to or greater than $3f + 1$. This property is referred to as Byzantine fault tolerance (BFT) [3]. BFT serves as an additional layer of trust against malicious activities as well as faults. However, it is important to note that achieving consensus in BFT systems requires increased complexity in the agreement process, called consensus.

As earlier mentioned, in scenarios where the environment is not hostile and the primary fault is machine unavailability, crash fault tolerance (CFT) may be sufficient. CFT allows for a certain number of unavailable nodes, with the maximum limit set at $n/2$. These fault tolerance properties in distributed systems, BFT and CFT, provide varying levels of resilience against faults and help ensure the stability and reliability of the decentralized business ledger.

1.3. Performance vs. Trust

In this study, we conduct a performance comparison of two well-known Byzantine Fault Tolerant (BFT) blockchain architectures, Quorum [4] and Tendermint [5], against the non-BFT Hyperledger Fabric (version 2.2), which utilizes Apache Kafka [6] as the ordering mechanism. Apache Kafka is a centralized, replicated, and distributed database [7] that is employed by Hyperledger Fabric. This version of Hyperledger Fabric is crash fault tolerant (CFT) but not Byzantine fault tolerant (BFT).

To isolate the impact of Hyperledger Fabric, we also examine the performance of deploying Apache Kafka. The objective is to gauge the performance difference between CFT-based protocols and BFT variants, representing the additional performance cost incurred by BFT's enhanced trust. Moreover, it is reasonable to believe that Hyperledger Fabric, the CFT blockchain variant of Hyperledger Fabric, would outperform its BFT rivals. Our findings, later explored in this paper, elaborate on that outcome.

1.4. Transaction Volume

As previously mentioned, our first use-case revolves around a national e-business trading system based on transaction volumes in Denmark. Denmark, as a nation with the eleventh largest GDP in the EU and a well-established digital economy, provides an ideal setting for this study. According to reports by Deloitte [8] and the Danish National

Bank [9], the Danish B2B and B2G market manages approximately 225 to 235 million invoices annually. However, before an invoice reaches its final stage between the buyer and seller, an average of five documents, including offer, quote, purchase order, invoice, acceptance note, and credit note (which we refer to as "related traffic"), are exchanged [10]. For simplicity, we assume that the related traffic documents have the same size as an invoice.

Based on these considerations, a national e-business trading system should be capable of handling approximately 1.12 to 1.17 billion transactions annually. In Denmark, e-invoice standards [11] and EU regulations [12] dictate that the maximum document size exchanged is 50 kilobytes (KB), with an estimated average size of 10 KB [12]. Multiplying this quantity of data by the volume of transactions results in a data storage requirement for a blockchain ranging from 11.5 to 12.0 terabytes (TB) annually, three times larger than the current size of the Ethereum blockchain [13] for Denmark alone.

When considering throughput, the arrival of e-business transactions into the B2B/B2G system exhibits a non-uniform distribution. The majority of transactions occur during business hours, specifically from 8 a.m. to 6 p.m., Monday to Friday. Furthermore, traffic experiences peaks towards the end of the calendar month, end of each quarter, and end of the financial year. In the Danish scenario, the relationship between peak intervals and transaction volume can be described using the formula:

$$\frac{\text{Number of Invoices} \times \text{ Related Traffic} = \text{Total Traffic}}{12 \text{ (months)} \times 20 \text{ (working days)} \times 10 \text{ (working hours)}}.$$

In peak times, the transaction volume is between 130 and 135 transactions per second (tps). In terms of response time, there are no strict requirements as the use-case focuses solely on invoices and related business document traffic. Payment processes are assumed to take place off-chain, utilizing traditional financial settlement systems or contemporary peer-to-peer (P2P) financial settlement systems. It is important to note that real-time responses, typically necessary for credit card settlements, are not necessary for this particular type of e-business document exchange system.

The second scenario entails a transnational e-business trading system spanning the European Union (EU) and the United Kingdom (UK). Extending the use-case to encompass all 27 EU member states, as well as the UK, necessitates a system capable of managing 17 billion invoices [9]. To accommodate this scale, the system needs to handle approximately 2000 transactions per second (tps). However, when accounting for the related traffic documents, the upper range for peak throughput increases to 9000 to 10,000 tps. Given the combined B2B and B2G nature of the use-case, there is an emphasis on governance and compliance particularly regarding the ledger's read and write permissions. Therefore, a permissioned blockchain is a crucial requirement for this scenario.

1.5. Novelty

This work makes two key contributions. Firstly, it offers validation of the performance claims made by some modern blockchain technologies, providing valuable insights into their true performance in real-world scenarios. Secondly, it introduces an innovative approach by employing an "in-the-wild" testing methodology of blockchain technologies, rather than relying on conventional laboratory testing.

In-the-wild testing, as described by Zhuang et al. [14], involves conducting experiments or evaluations in real-world settings, capturing realistic conditions, actual usage scenarios, and various environmental factors. By deploying and testing systems under authentic conditions, valuable insights are gained regarding their performance, behavior, and interaction with the environment. Utilizing an in-the-wild testing methodology, this research goes beyond traditional laboratory-based evaluations and examines how blockchain technologies perform under realistic and dynamic conditions. This approach provides an accurate representation of system behavior and its interaction with the environment, offering a comprehensive understanding of system capability and limitations.

Therefore, the novelty of this work lies in the validation of the performance claims of selected blockchain technologies and its pioneering use of an in-the-wild testing methodology for the investigated blockchains. This enhances the validity and relevance of the findings.

The structure of this paper is as follows. In Section 2, we discuss the experimental set-up, including the selection of candidate permissioned blockchains for the study. We also provide reasons for excluding other blockchains that were considered. Furthermore, we provide a detailed description of the two selected Byzantine fault tolerant (BFT) blockchain technologies and the crash fault tolerant (CFT) blockchain used in our study. In Section 3, we delve into the test protocols, test platform, and specific configuration settings employed for each test run. In Section 4, we present the empirical results obtained from our experiments. Section 5 discusses the key findings and their implications. In Section 6 we look at extensions to our work and its limitations. Finally, in Section 7, we draw our conclusions based on the findings presented in this paper.

2. Experimental Set-Up

2.1. Benchmarking Methodology

Performance studies on blockchains [15–18] typically adhere to standard practices in distributed system testing. These studies aim to maintain consistent parameters to enable fair and comparable comparisons of resource demand, throughput, and transaction latency across different blockchain frameworks. However, this approach can introduce biases towards or against specific blockchain systems, making it challenging to draw conclusive comparisons among them.

To address this limitation, researchers have attempted to develop advanced testing frameworks, such as Blockbench [19] and Chainhammer [20], which aim to provide more sophisticated evaluation environments. However, these frameworks are still considered research in progress [21].

Many existing blockchain performance tests suffer from being largely artificial; they do not accurately reflect real-world deployment scenarios, in terms of network topology, geographic distribution, message lengths, and transaction volume. This paper aims to overcome this limitation by conducting in-the-wild distributed system tests. The specific requirements of a national and transnational B2B/B2G invoicing system serve as the foundation for defining performance expectations.

The main novelty of this paper is its utilization of an in-the-wild testing methodology to assess the performance of crash fault tolerance (CFT) and Byzantine fault tolerance (BFT) blockchain technologies. The testing closely aligns with the transaction volume encountered in a (trans)national e-business platform, employing commodity hardware. This allows a comprehensive evaluation of alternative technologies in realistic scenarios, contributing to a deeper understanding of their capabilities and limitations. To achieve realistic testing, empirical experiments are designed to closely resemble the actual use-case. This involves ensuring that the size and number of transactions entering the system align as closely as possible with the anticipated system load. Furthermore, the number and geographic distribution of nodes and consensus validators are configured to mimic the characteristics of the use-case. Importantly, all systems evaluated are deployed on commonly available hardware and network infrastructure, underlining practicality and feasibility.

By adopting this approach, the paper endeavors to provide a more accurate assessment of blockchain performance in real-world settings, ensuring that the testing methodology aligns with the intended deployment scenario. This enables a comprehensive evaluation that informs performance expectations for national and (trans)national B2B/B2G invoicing systems. To summarize, the test parameters encompass the following features:

- the number of transacting nodes: this refers to the count of nodes actively participating in exchanging transactions within the system.
- the size of individual transactions: denotes the data size or complexity of each transaction processed within the system.

- the number of nodes participating in consensus/ordering: this parameter relates to the quantity of nodes involved in the consensus or ordering process within the tested platforms.
- the volume of transactions over time: this represents the rate at which transactions are generated and processed within the e-business system, typically measured as transactions per second (tps).

The above test parameters provide a framework to evaluate the performance characteristics of the system, enabling an understanding of how these factors impact system performance and efficiency.

2.2. Selection of Platforms

To compile a list of potential blockchain platforms, we conducted a survey and analyzed existing literature sources such as [5,22,23]. We recorded the performance claims of these platforms, summarized in Table 1. Considering the nature of our in-the-wild testing methodology, we chose the popular permissioned enterprise blockchain platform, Hyperledger Fabric, as the performance baseline.

Table 1. Blockchain performance claims, extended from [5,22,23]. The staging of each experiment reporting throughput and response time is different, making it difficult to draw meaningful comparisons between competing solutions. † Hedera has open-source components such as SDKs, but the core consensus algorithm is not open-source due to the patented technology it is built on.

Framework	Type [1]	Consensus Algorithm	Open Source	Throughput (tps)	Response Time (s)
Bitcoin	Open	PoW	Y	3–5	>500
Ethereum	Open	PoW	Y	15–30	360
Kadena	Open	Scalable PoW-BFT	N	10,000	<0.1
Hedera	Open	Hashgraph (aBFT)	(Y †)	10,000	<0.1
IOTA	Semi-open	Tangle	Y	200	N/A
NEO	Semi-open	Delegated-BFT	Y	10,000	15–20
EOS	Semi-open	Delegated BFT	Y	3996	<1
Ripple	Closed	RPCA (Ripple Protocol)	Y	50,000	4
Hyperledger Fabric	Closed	Kafka/Raft	Y	>3500	<1
Hyperledger Sawtooth	Closed	Proof of Elapsed Time (PoET)	Y	>80,000	<1
MultiChain	Closed	PBFT + MultiChain	Y	1000–1500	5–10
Quorum	Closed	Istanbul BFT (IBFT)	Y	600–900	5
Tendermint	Closed	Tendermint BFT	Y	4000–14,000	<1
Red Belly	Closed	Democratic-BFT	N	660,000	2–4

[1] Types of blockchain: Open = public permissionless; Semi-open = public Permissioned; Closed = private permissionless.

To select platforms to benchmark against Hyperledger Fabric, we established three criteria. Firstly, the performance claims of the platforms needed to meet or exceed the transaction volume requirements of the Danish national scenario and ideally the transaction volume of the EU scenario. Secondly, the selected platforms had to employ a Byzantine fault tolerant (BFT) consensus mechanism to ensure resilience against malicious actors, providing

higher trust and better overall fault tolerance compared to Hyperledger Fabric. Lastly, the platforms had to be easily accessible, non-proprietary, and not subject to performance caveats such as throttling.

Based on these criteria, three BFT blockchain solutions were considered: Hedera, Quorum Istanbul, and Tendermint. However, Hedera's consensus service had a predefined limit of 500 transactions per second (tps) which could not be negotiated. Consequently, we focused on the remaining two BFT blockchain platforms: Quorum Istanbul and Tendermint, as they fulfilled the selection criteria and offered promising features for our use-case.

In the above-mentioned criteria, equal emphasis on security and performance is not placed. We reasoned that if a system cannot handle the throughput requirements then security is irrelevant, but the inverse scenario, where a less-secure system exists, performance remains relevant. In the case where the performance requirements for this use-case cannot be meet, we measure the cost of additional trust by comparing CFT vs. BFT. Hyperledger Fabric using Apache Kafka, a blockchain implementation that is not BFT, is selected to identify the baseline of a CFT blockchain that runs on identical hardware and network infrastructure as the BFT blockchain platforms. In our experiments, Hyperledger Fabric using Apache Kafka did not perform as well as expected, so we tested a pure CFT implementation without the blockchain Hyperledger Fabric wrapping.

2.2.1. The Quorum Blockchain

Quorum [4] is an open-source blockchain platform built on top of the Ethereum, designed specifically to address the requirements of financial applications. Unlike Ethereum which is public, Quorum is permissioned which means that participation in the blockchain is limited to a known set of provisioned nodes in the network, a feature that is particularly important in industries such as finance. By default, Quorum provides RAFT consensus [24] and Istanbul BFT (IBFT) [25]. Since BFT is required for our study, the IBFT consensus is chosen.

To understand Quorum, it is important to highlight some specifics about the Ethereum blockchain on which it runs. Ethereum can be characterised as a large distributed finite state machine, where transactions can be viewed as state transitions, i.e., a block is a list of transitions. Transactions (and hence state transitions) should be well-formed (have the right number of values) and carry a valid signature (there are other rules, see the Ethereum whitepaper [26]). A developer can define the specifics of transactions, and these programmable state transitions are known as smart contracts. In other words, a smart contract is a way to encode a state transition. As Ethereum stores the full ordered list of transactions (state transitions), it is possible to re-run the blockchain by replaying every state transition from the original state: this is what occurs when a node is synchronizing. An example of a state transition is shown in Figure 1, this illustrates how a smart contract, located at bb75a980, transitions the State to State'.

When a developer defines a smart contract, they must define where data is stored. In Ethereum, three locations are available that allow data to persist: these are Storage, Memory and Calldata, and each of these locations have their own merits and gas-costs. Variables located in Storage are written into the State for the defined smart contract, i.e., in the case of Figure 1, the name CHARLIE is located in Storage, and therefore written into State'. Variables written into the State in this way persist and can be accessed through a smart contract at a later point in time. Cryptocurrencies built on Ethereum use Storage to keep track of wallet balances. Memory is a temporary location for mutable variables in a smart contract. Variables in Memory do not update the State, and are not later accessible via smart contracts, namely they persist only within the scope of the smart contract execution. Lastly, Calldata, is an immutable version of Memory that can only be passed from an external address as parameters to a function defined with the scope of a smart contract. Recall that Ethereum stores and orders all the transactions that have occurred in the system, so data passed as arguments in any transaction, or used internally as a variable in a

smart contract, remain accessible by examining the transaction, even though they are not retrievable from a smart contract.

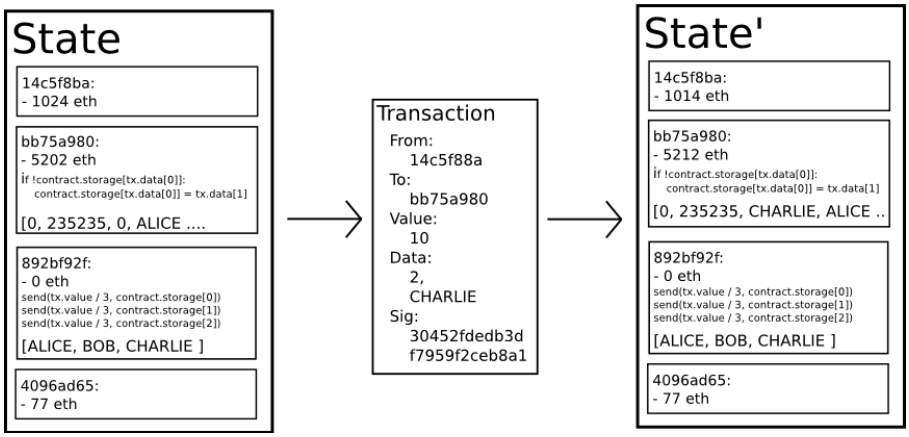

Figure 1. The Ethereum state transition, adapted from the Ethereum whitepaper [26].

In Figure 1, the data used in the central transaction (Memory), or the parameters passed when it was called, Calldata, will still be on-chain without being written to the State. Since the use-case does not require later access via a smart contract, only proof that data was committed by a specific actor, alternatives to recording data in a State are possibilities. This allows one to utilise on-chain data-availability, and use Memory or Calldata to create a persistent record of data by passing arguments into a noop smart contract, thus avoiding writing them into the State. In so doing, a transaction containing the data and a signature is recorded, and this represents proof that data has been committed. In this way, it is possible for anyone to later look at a specific transaction and examine the data committed to the blockchain.

As a final note, the performance figures in Table 1 for Quorum are quoted from a non-peer-reviewed study by Baliga et al. [27]. That study informs our own work, in particular we use the same 30 s time-frame in which to generate transactions.

2.2.2. The Tendermint Blockchain

Tendermint [5] consists of two core components: (i) a consensus engine and (ii) a generic application interface. The consensus engine, called Tendermint Core, ensures that the same transactions are recorded on every machine in the same order. The Tendermint protocol can be configured to support two groups of nodes that comprise the ledger, namely validators and non-validators. Only the validators participate in consensus and non-validators are restricted to read-only ledger access. Unlike many public blockchains, Tendermint does not have a cryptotoken and provides BFT with a claimed throughput of 14,000 tps [5]. The consensus engine utilises a BFT-based voting algorithm that works similarly to classic BFT algorithms such as Practical BFT (PBFT) [28]. This enforces multiple rounds of voting before data is committed. One main difference between Tendermint's BFT-based algorithm and classical algorithms, such as PBFT, is the introduction of voting power, this allows certain validator nodes to have greater decision-making influence within the network. The generic Tendermint API, called an Application BlockChain Interface (ABCI), allows smart contracts to be implemented in any programming language, provided that language implements the ABCI. Smart contracts deployed on Tendermint run on individual nodes, as opposed to Quorum where all smart contracts are run simultaneously across all nodes via the Ethereum Virtual Machine (EVM), thus smart contracts on Tendermint are not decentralised.

2.2.3. Hyperledger Fabric

Hyperledger Fabric is an open-source permissioned blockchain platform and is established under the Linux Foundation [29]. Hyperledger Fabric has a modular and config-

urable architecture that offers a smart contract layer in Java, Go and node.js and supports so-called "pluggable consensus protocols" [6]. The standard implementation of the consensus protocols is delivered in either in Apache Kafka or Raft (up until version 1.4 Kafka was standard where Raft is standard from version 2.0). Apache Kafka and Raft are ordering mechanisms rather than blockchain consensus protocols. Hyperledger Fabric leverages the ordering mechanisms and in so doing does not require a native cryptocurrency to provide incentives for mining or to fuel smart contract execution. Hyperledger Fabric introduces their own terminology defining different key constructs such as peers, organisations, orderers and a Certificate Authority (CA). At a high level, an organization consists of a set of peers that participate in one or more channels that is ordered by an ordering service. A channel is a private blockchain between its participating organizations, meaning that a user can only interact with contracts in channels where their organization participates. For further clarification on the terminology; see the official glossary for Hyperledger Fabric [29].

2.2.4. Apache Kafka Streaming Platform

Apache Kafka is a distributed streaming platform first introduced by LinkedIn engineers and later open-sourced [7]. Apache Kafka consists of three key capabilities: (i) to publish and subscribe to streams of records; (ii) to store streams of records in a fault-tolerant durable way; (iii) to process stream records in the order they occur. The Apache Kafka architecture has three main components, specifically; producers, brokers and consumers. The producer creates data and sends them to a broker who receives, categorises and stores data before the consumer pulls data from the broker. The categorisation of data is processed through topics, and one or more topics are stored in partitions on the broker(s). Records written to the partitions are in the form of (key, value, timestamp) triples which are: immutable, persistent and added to an append-only list preserving message order [30]. The consumers maintain their own state and poll for new data when needed. This allows Apache Kafka to persist a single message independently from the number of consumers, resulting in high-throughput for read and write operations [31].

3. Testing Framework

3.1. Testing Scenarios

The use-case focuses on sharing documents, invoices and related business documents, as described in Section 1.4. Each of the test platforms will be described in detail, but share the following features:

- transactions that simulate compressed average size PEPPOL [12] invoices (2 KB in size) over a testing time of 30 s;
- transactions that simulate non-compressed average size PEPPOL invoices (10 KB in size) and a testing time of 30 s.

3.2. Testing Environments

Transactions for all four platforms (Quorum, Tendermint, Hyperledger Fabric and Apache Kafka) are sent using slave machines. These are virtual machines (VMs) (see details in Section 3.2.1) deployed in the Microsoft Azure cloud and controlled by a master testing machine, also deployed on Microsoft Azure. Each slave machine runs a given test script distributed from the master to the slaves (see Figure 2). All test scripts used in this study contain a method for randomising the order of node addresses, to ensure that transactions are randomly distributed among the 28 nodes for the blockchain platforms, namely not following the same order for every user sending transactions. For the Apache Kafka test, each transaction is sent to one of the five brokers and automatically distributed from there to a partition. The test scripts communicate through web calls to each of the nodes/brokers of the four platforms. Logs of transmission data (time-stamped when the transaction is sent, success status and transaction hash) are collected and stored on the master virtual machine. For the blockchain platforms, after all the transactions have been sent and processed from

the slave VMs, a Python script extracts information (time-of-mining, block number and block size) about the specific transactions using the transaction hash from the transmission data log. These data points make it possible to calculate transactions per second (tps) and further provide the foundation for the visualisations shown in Section 4 (see Figures 3–5).

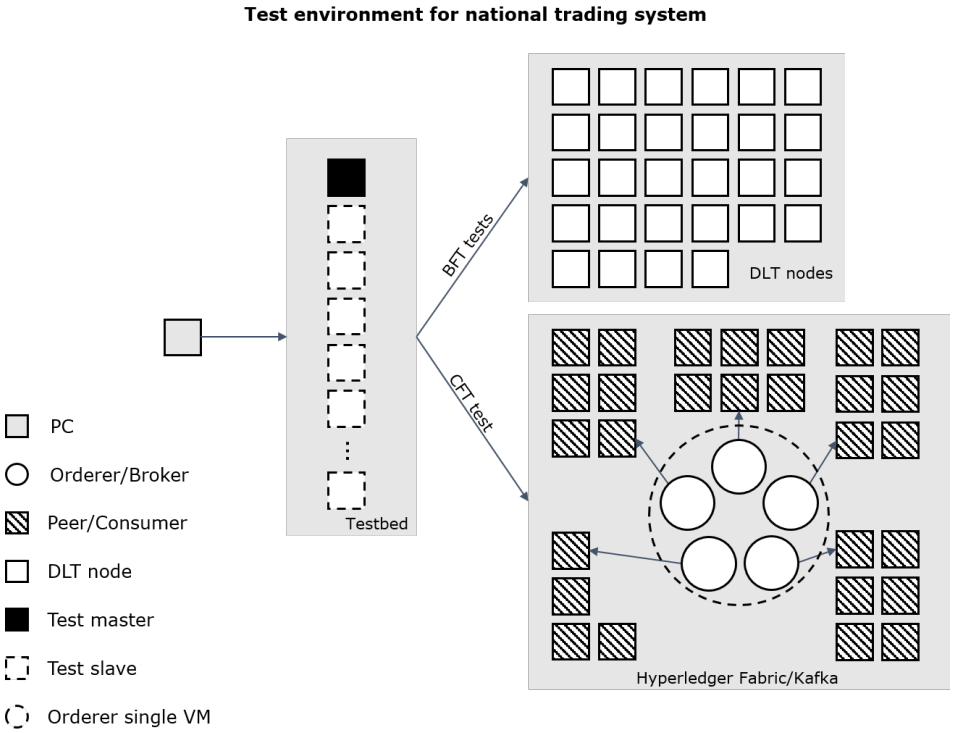

Figure 2. Overview of the testing environment.

Figure 3. Quorum throughput over time in both KB per second (**top**) and tps (**bottom**). All three data storage methods (Calldata, Memory and State) are compared using both 2 KB and 10 KB transactions. Each marker represents a new block.

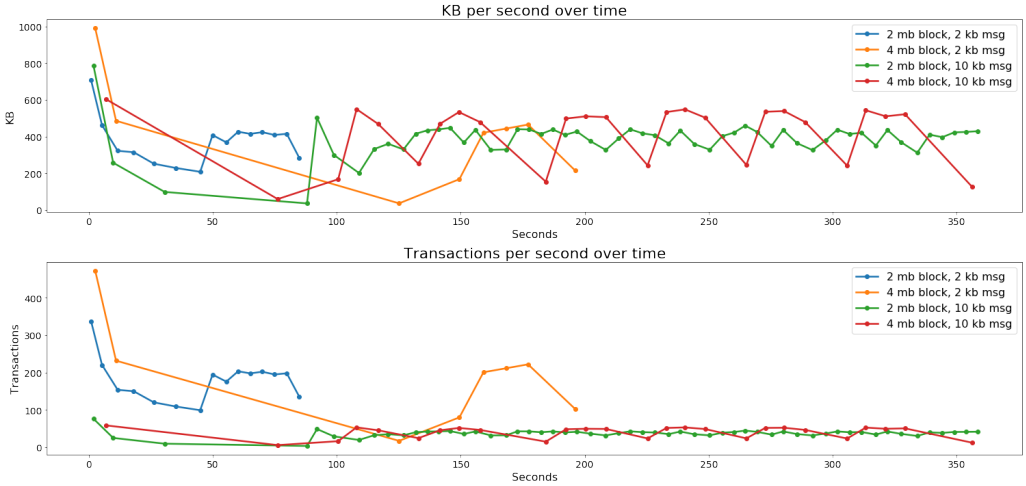

Figure 4. Tendermint throughput over time in both KB per second (**top graph**) and tps (**bottom graph**). The 2 MB and 4 MB block sizes are compared using both 2 KB and 10 KB sized transactions. Each marker represents a new block.

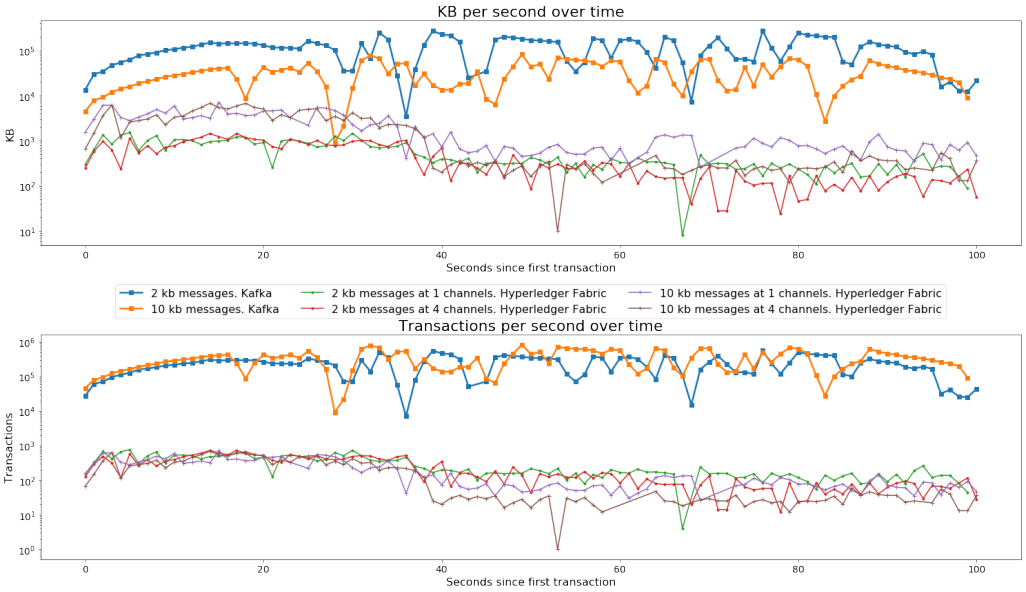

Figure 5. Hyperledger Fabric and Kafka throughput over time, in both KB per second (**top**) and tps (**bottom**). Comparing 2 KB and 10 KB transactions.

3.2.1. Testing Machines

The tests simulate up to 5000 concurrent users over a 30 second period. Such testing, if run on a single machine, induces the Java VM garbage collector (GC) mid-test, an artifact of the use of JMeter being used as the measuring instrument, thus polluting test results. The minimum and maximum heap of the Java VM is set to be 256 MB and 14 GB, respectively. In computer memory management, the young generation, specifically the Eden space, refers to a region in the heap memory used by garbage collectors in generational garbage collection algorithms. The Eden space is therefore maximised to ensure that GC induced by JMeter does not pollute the results, GC will not trigger until Eden starts to get close to full. The verbose GC setting is used throughout all tests to provide all GC information. For this reason, all tests are distributed over five slave test machines, see Section 4 for an overview. All test machines are hosted in Microsoft Azure within the same subnet, located in Western Europe, and the master machine is controlled via an Ubuntu console. All slave machines (and their master) are Azure D4s_v3 with four virtual cores (vCPU) based on 2.3 GHz Intel XEON E5-2673 v4 (Broadwell) processor, 16 GB RAM and

32 GB SSD harddisk [32] (namely commodity hardware when the tests where conducted in 2021). Each of the slave machines runs a Docker image of the slave environment to ensure consistency across the tests [33]. The RAM and CPU of the testing machines are monitored throughout the tests.

3.3. Cloud Computing Configuration

Throughout the tests, 28 nodes are spread over the five geographic regions that Microsoft Azure provides in Europe, at the time of the experiments these were Northern Europe, Western Europe, France Central, UK South and UK West. The geographical spread of the infrastructure replicates a real-world scenario where the nodes are distributed between a consortium of EU-member states (and the UK). The validator nodes are hosted in Microsoft Azure and are likewise run on D4s_v3s, with the same specification described above. The validator nodes run Ubuntu Version 18.04 as an OS and are monitored for CPU utilisation and RAM usage, to ensure that the servers hosting the platform are appropriately provisioned.

3.3.1. Quorum

Quorum has a number of configuration options, see Table 2. Prior to running our tests we experimented with various configurations in order to optimise the setup to best meet the requirements of the use-case. We experimented with different block periods; 1, 3, 5 and 10 s. Maximum throughput is achieved by optimising for the minimum block period where consensus is reached, in this case, a block period of 5 s.

Table 2. Configuration for Quorum, Tendermint, Hyperledger Fabric and Apache Kafka.

Parameter	Value
Quorum	
istanbul.blockperiod	5 s
cache	14 GB
txpool.globalslots	100,000 tx
txpool.globalqueue	1,000,000 tx
txpool.accountqueue	1,000,000 tx
block.gasLimit	50,000,000
Tendermint	
mempool.size	560,000 tx
mempool.cache_size	600,000 tx
mempool.max_txs_bytes	1 GB
mempool.recheck	false
Hyperledger Fabric	
endorsement policy	any
batchsize	300 tx
kafka_default_replication_factor	5
kafka_min_insync_replicas	3
Kafka	
num.network.threads	8
num.io.threads	8
offsets.topic.replication.factor	3
transaction.state.log.min.isr	3

The validator node cache is the amount of memory allocated to internal caching and is set to 14 GB, leaving 2 GB of memory for the OS. In comparison, a default full node on Ethereum's mainnet has 4 GB allocated to cache. The transaction pool for global slots was set to 100,000 transactions (the default is 4096), which is the maximum global number of executable transaction slots for all accounts. The transaction pool global and account

queue is set to 1,000,000 transactions (the default is 1024), which is the maximum number of global and account non-executable transaction slots for all accounts. We experimented using 20,000 transactions for global slots and queue, based on the performance review on Quorum's GitHub page [4], but in our experiments we achieved optimal performance using a value of 100,000. We reserve a single thread for the OS and allocate all remaining threads for each core to validator nodes.

When testing with the default value of 3,758,096,384 as the gasLimit, latency is significant and we received only a few very large blocks that require multiple minutes of processing each. In order to optimize for our use-case (a preference for a steady flow of blocks) we decreased the gasLimit to 50,000,000 (as recommended by Microsoft [34]).

The two scenarios described in Section 3.1 were tested with the three different data storage methods allowed by the Quorum blockchain, State, Memory and Calldata. These three data storage methods are tested through implementation of three different smart contracts. Since Quorum is based on Ethereum, much of the terminology and functionality are the same, e.g., the concept of gas as the computational incentive for smart contract execution. Further, each block has an upper limit on the allowed computational effort (measured in gas). Since the three storage methods have different gas costs, it is important to investigate the performance implications of each.

3.3.2. Tendermint

As with Quorum, Tendermint has configurable settings that impact performance, see Table 2. As our use-case only requires the data to be stored, and not otherwise processed, we utilise Tendermint's built–in noop smart contract. Other optimal settings, such as a minimum block time of 1-second is used, alongside changing the default underlying database (LevelDB) from the GOLANG to the C implementation. We chose to make this swap to the C as it has proven to improve performance under heavy loads [35]. To ensure that the test transactions can fit within the mempool, its size was increased to 560,000 transactions and max_tps_bytes to 1 GB. The cache size is set to 600,000 transactions, which allows for the filtering of up to 600,000 transactions already processed, meaning no two identical transactions in our tests are processed more than once. In order to optimise throughput, we set recheck = false, meaning Tendermint does not recheck if a transaction is still valid after another transaction has been included in a block. The reasoning behind this choice is that all transactions are independent, they do not interact or influence each other.

3.3.3. Hyperledger Fabric

To fit the Hyperledger Fabric terminology to the use-case, 28 organizations were created, so an organization maps to a European country, where each of the organizations had one peer each representing one company per country. ZooKeeper is a component that plays a critical role in coordinating and managing the Apache Kafka cluster. ZooKeeper is used by Kafka as a centralized service for maintaining configuration, tracking the state of the cluster, and managing the metadata associated with Kafka brokers, topics, and consumer groups. We wanted to create an orderer instance containing one Kafka and one Zookeeper in five separate data centers across the Microsoft Azure European data centers. However, multi-hosting an environment across multiple locations is not presented in the official Hyperledger Fabric documentation. This leads us to a single orderer VM containing five Kafka instances and five zookeepers in one data center. This configuration proves to be favourable to overall system performance.

Surveying the literature on Hyperledger Fabric performance tests [36–40] and referring to Hyperledger's own performance benchmark tool, Caliper [41], reveals that there have been no reported peer reviewed papers using in-the-wild performance tests. In fact, every performance study cited above uses large co-located machines. For the Caliper benchmark tests, all of the roles, one orderer and two organizations each with one peer, are hosted on a single machine. This eliminates any variability resulting from network dynamics. This assumption is likewise applied in Thakkar et al. [36] (p. 12) and noted in their conclusion:

"we assumed that the network is not a bottleneck. However, in a real-world setup, nodes can be geographically distributed and hence, the network might play a role". This is verified by Geneiatakis et al. [42] where the effect of bandwidth limitations in a similar use-case across 28 nodes representing EU member states authorities is evaluated.

The batchsize parameter is set to 300 transaction per batch (block size), which was found to be an optimum through our own testing and confirms the findings of Thakkar et al. [36]. The endorsement policy is set to "any", which means that only one of the 28 peers needs to endorse a transaction before it is sent to the orderer. The replication configuration is altered towards the use-case of this paper. Since invoices are business critical documents, the replication of data is important to mitigate data loss. Therefore, we replicate data to all five Kafka instances with replication to a minimum of three, before data are made available to peers. The three-out-of-five Kafka rule creates leeway for downtime in two of the ordering services, without crashing the entire system, hence its crash fault tolerance property. Similar to Tendermint, we deployed a smart contract in each of the channels calling a noop function. We run our tests through one and four channels with all 28 peers enrolled in each of the channels. This serves the purpose of understanding and measuring if channels can be used as a way of scaling [36,40]. The specifications and geographically distribution of the VMs used in the Hyperledger Fabric test consist of 28 VMs each containing a peer, and one VM containing the five orderers in separate docker containers.

3.3.4. Apache Kafka

The purpose of the Kafka test is not to optimise for marginal improvements on the throughput but rather obtain an indicative benchmark from another paradigm running on identical commodity infrastructure. Therefore, the values of the parameters for network and I/O threads shown in Table 2 are chosen with emphasis on the use-case and drawn from configurations from LinkedIn's performance testing [43]. The replication configuration is equivalent with Hyperledger Fabric to achieve CFT. The specifications of the VMs used in the Kafka test are identical to the Quorum, Tendermint and Hyperledger Fabric tests. In total, 33 VMs are used. Five brokers on separate VMs and 28 VMs act as both producers and consumers, to emulate the scenario of companies both sending and receiving invoices. The same geographically distribution applies, leaving one broker per geographic region and five to six producers/consumers in each of the five geographic regions.

4. Experimental Results and Analysis

4.1. Quorum, Results and Analysis

As shown in Figure 3, the throughput for Memory and Calldata with 2 KB messages (blue and orange lines), which replicates the average compressed transaction size, lies primarily around 300 tps, with occasional spikes above 600 tps. Transactions are fully processed after 280 s. For State (green lines), results are around 200 tps, with short bursts of lower throughput. Transactions are fully processed after 440 s. The results show that storage methods that store transactions in history (Calldata and Memory) outperform the storage method that saves data to the State. The figure also shows that Calldata and Memory are almost identical in performance, with only marginal differences.

The 10 KB tests in Figure 3 (red, purple and brown lines) show similar trends to the 2 KB test, where Memory and Calldata significantly outperform writing data to the State. The same pattern is observed for the 10 KB test, however ranging between 70 and 140 tps for Memory and Calldata, while between 20 and 40 tps results for State. When increasing the volume of transactions and the number of concurrent users, the same phenomena is observed with an increase in the difference between average highs and lows. During tests, we observed that the network often requires spin-up time. This is very visible in the results for 2 KB in Figure 3, where the first 20 to 50 s are not used to process transactions. This is because transaction data are passed to the nodes, which create and sign the transactions before they are distributed into the network.

The results of the Quorum tests are presented in Table 3. The conclusion from the Quorum tests is that the results would satisfy the national B2B/B2G use-case for a country the size of Denmark where there is a demand for 130 to 135 tps throughput. However, the requirement from the entire Euro-zone of 9000 to 10,000 tps is not satisfied.

Table 3. Quorum throughput results, averages and maximums for 2 KB and 10 KB transaction sizes.

Test Scenario	Type	Max (tps)	Average (tps)
2 KB	Calldata	622	360
	Memory	620	323
	State	205	182
10 KB	Calldata	139	125
	Memory	139	120
	State	43	39

4.2. Tendermint, Results and Analysis

As shown in Figure 4, the transaction throughput of Tendermint is impacted significantly by the size of the transactions exemplified in the differences in the bottom graph from the 2 KB test (blue and orange lines) and the 10 KB test (green and red lines). However, the throughput (measured in bytes) is unaffected by the block and transaction size, see top graph in Figure 4. Further evident in the graphs, there is an initial spike in throughput across tests, this is due to a commonly known Tendermint performance bottleneck related to indexing transactions when using larger block sizes [44]. When compared to the Quorum tests, Tendermint resembles the Quorum State runs in that writes all data into the State. Nevertheless, we posit that Tendermint outperforms Quorum in cases where transactions are more computationally intensive, due to its smart contracts utilising GOLANG, and running on a single node rather than being executed simultaneously across all nodes on the Ethereum Virtual Machine in Quorum. When comparing the Tendermint test results with the performance claims from Table 1, it is clear that our results and those from Table 1 are widely divergent. This is explained by the transaction size used in the tests carried out by Buchman [5], as tests were never conducted using transactions larger than 250 bytes. To confirm that Buchman's performance claims hold, we tested our Tendermint setup using a transaction size of 32 bytes and obtained a throughout peak of over 10,000 tps. The results from the Tendermint tests can be seen in Table 4 and confirm that the level of performance delivered by Tendermint would satisfy the national B2B/B2G use-case for a country the size of Denmark of 130 to 135 tps, when compressed 2 KB transactions are used. As with Quorum, the requirement for the entire Euro-zone of 9000 to 10,000 tps is not satisfied with Tendermint.

Table 4. Tendermint throughput results, averages and maximums for 2 KB and 10 KB transaction sizes.

Size	Block Size	Max (tps)	Average (tps)
2 KB	2 MB	337	164
	4 MB	472	120
10 KB	2 MB	76	31
	4 MB	58	29

4.3. Hyperledger Fabric, Results and Analysis

Figure 5 shows that the peak and initial throughput of Hyperledger Fabric is more or less indifferent to message size (2 KB and 10 KB) because Apache Kafka is the underlying messaging protocol. However, throughput reduces after the first 35–45 s, struggling under sustained load. Table 5 shows that peak throughput is significantly larger than average

performance, and that Hyperledger Fabric handles large increases in message-size well. As noted by Thakker et al. [36] and Kuzlu et al. [40], Hyperledger Fabric can utilise channels to run multiple "contracts" in parallel and in so doing increase transaction throughput. We increased the number of channels but our experimental results did not reflect this finding. This may be because of limited computing power. To test this assumption, tests were re-run with a larger machine as the orderer and we observed that performance increases 3% for the single channel case, and 18% for the four channel case, indicating that channels have some overhead, but result in increased throughput when used with more computing power. As noted by Thakker et al. [36], the endorsement policy is identified as a performance bottleneck, however, our tests show no significant difference between the policies "majority" and "any". Comparing our results to the literature [36,38–40,42], and Hyperledger's own Caliper tests [45], similar performance is observed. Gorenflo et al. [37] present a study of the bottlenecks in Hyperledger Fabric, which can be summarized as message communication overhead internal to its architecture.

Table 5. Hyperledger Fabric throughput results, averages and maximums for 2 KB and 10 KB transaction sizes.

Size	#ch	Max (tps)	Average (tps)
2 KB	1	763	245
	4	723	142
10 KB	1	714	137
	4	678	130

4.4. Apache Kafka, Results and Analysis

Apache Kafka is expected to be much faster than Quorum and Tendermint and not much faster than Hyperledger Fabric, since it uses Apache Kafka within Hyperledger Fabric. For Quorum and Tendermint this proves to be the case, Apache Kafka is much faster. However, Hyperledger Fabric is not as fast as we hoped. The results, shown in Figure 5, show a significant difference. At peak, for the 2 KB test, Apache Kafka achieves 241,124 tps, and on average around 107,187 tps. The spikes seen in Figure 5 are consistent with the phenomena observed by Le Noach et al. [46] and is the result of re-balancing. The re-balancing feature used in Apache Kafka clients (and/or the Apache Kafka coordinator) allows the formation of a common group, and distributes a set of resources among the members of that group. Re-balancing occurs every time a member joins or leaves a group. In our case, since we cannot start every producer at the exact same time, a script starts sequentially, leaving the producers and consumers to gradually join and leave the group over the period of the test run resulting in re-balancing, and therefore a re-allocation of the resources occurs, hence the presence of spikes in performance.

When comparing these results to other studies of Apache Kafka [46–48], we note that they deploy larger broker machines to manage more transactions. If we had deployed larger machines for the Apache Kafka test, system performance would be even better than we observed.

4.5. Comparison of Blockchain Framework Results

This result is unexpected since Hyperledger Fabric is often marketed as a highly scalable blockchain [49]. In practice this means that consensus is performed by fewer peers, increasing the centralisation, and theoretically throughput as well. When we compare these blockchain solutions to more standardised cloud technologies such as Apache Kafka, the difference is enormous. Apache Kafka is on average more than 298 times faster than Quorum, 437 times faster than Hyperledger Fabric and 654 times faster than Tendermint.

The purpose of the Apache Kafka test is to create a baseline for comparison on the same use-case, and with the same in-the-wild distributed test approach, and draw some conclusions about the efficiency and use of Apache Kafka as an ordering service in Hyper-

ledger Fabric. The core difference between Hyperledger Fabric and Apache Kafka is the inbuilt components as smart contracts and identity management that Hyperledger Fabric provides. It is evident that Apache Kafka is not only capable of satisfying the national B2B/B2G use-case for a country the size of Denmark, but is also capable of satisfying the requirement for the entire EU-zone using standardised cloud infrastructure, under the assumption that crash fault tolerance suffices as system trust the e-business use-case.

5. Key Findings

The key performance findings are summarized in Table 6. Based on the performance claims made in the framework survey in Table 1, we assumed from the outset that Tendermint would be faster than Quorum with the reasoning that the smart contract layer is implemented in GOLANG compared to the implementation of smart contracts in Quorum running on the distributed Etherum Virtual Machine. This proved not to be the case using our testing regime, however Tendermint might still outperform Quorum in a computational intensive use-case—one relying on the presence and interaction of many smart contracts. Although this is not proven, it underscores an important learning: tuning the platform choice to the application use-case is important and will influence design choices made in future systems. System developers need to ask themselves, how much smart contract activity is anticipated and how important are smart contract transactions to the orderly operation of the e-business ecosystem?

Table 6. Throughput results for all Blockchain frameworks, averages and maximums for 2 KB and 10 KB transaction sizes, Quorum, Tendermind and Hyperledger Fabric. The best 2 KB throughput is highlighted in red, The best 10 KB throughput is highlighted in orange. Unexpectedly, the average tps is faster for Quorum than for Hyperledge Fabric.

Quorum	**Type**	**Max (tps)**	**Average (tps)**
2 KB	Calldata	622	360
	Memory	620	323
	State	205	182
10 KB	Calldata	139	125
	Memory	139	120
	State	43	39
Tendermint	**Block size**	**Max (tps)**	**Average (tps)**
2 KB	2 MB	337	164
	4 MB	472	120
10 KB	2 MB	76	31
	4 MB	58	29
Hyperledger Fabric	**#ch**	**Max (tps)**	**Average (tps)**
2 KB	1	763	245
	4	723	142
10 KB	1	714	137
	4	678	130

Hyperledger Fabric was used as a baseline since it is one of the most popular enterprise blockchain frameworks, and while it is not Byzantine fault tolerent (BFT), it is crash fault tolerant (CFT), and would therefore be expected to be faster than its BFT alternatives. This proved to be the case, but only for maximum throughput, not for average throughput. Apache Kafka, uncoupled from Hyperledger Fabric, was tested in order to have a distributed CFT alternative that runs on the same infrastructure. Uncoupling was performed to understand the performance gap between Hyperledger Fabric using Apache Kafka and plan Apache Kafka, under identical network, hardware and transaction loads.

From Table 5, the findings are that Hyperledger Fabric handles large messages well in comparison to its overall performance. Furthermore, Hyperledger Fabric incurs a performance penalty when using multiple channels with 28 participants on small machines. Other work [36] has shown that endorsement policies are highly influential on Hyperledger Fabric performance, our results show this not to be the case for smaller machines, as the ordering service is computational bounded.

Apache Kakfa's performance advantage is achieved because the system can write data in parallel, it does not have to achieve consensus before it adds data to the ledger. This observation implies that systems designers need to consider the cost and need for distributed trust: especially if the proposed system has the nature of being permissioned and private. The question needs to be asked, is Byzantine fault tolerance necessary when participants are known to each other and have a vested interest in the e-business system being trustworthy?

However, in large dynamic e-business ecosystems, where bad actors could be present, the extra level of distributed trust that BFT delivers may be worthwhile. This is one of the reasons that blockchains are so attractive in applications for supply chain logistics [21], because supply chains are dynamic, multi-agent e-business ecosystems where there is an ever-present incentive from bad actors to disrupt authenticity. System designers need to offset the costs of managing trust, e.g., reconciling and monitoring the authenticity of data is labor intensive. Therefore, a significant performance overhead in system performance might be an acceptable for immutability with decentralised and inbuilt trust.

6. Future Work

The testing methodology we employ, using real-world scenarios, serves as a framework for evaluating emerging blockchain systems. It is important to emphasize that our intention is not to claim that blockchain technology will never be able to scale to a transnational e-business platform for the EU. Instead, our objective is to identify the necessary requirements and conditions for such a framework to operate successfully on such a large scale.

We acknowledge that advances in consensus protocols and other innovations—such as compression algorithms, caching, and parallel processing—may lead to enhanced scalability and efficiency in blockchain-based e-business solutions. Additionally, we recognize that commodity cloud-based hardware is continually improving. Hence, there are good prospects that future developments will enable blockchain to meet the demands of transnational e-business. Through our testing methodology, we aim to provide insights and considerations that can guide the evolution and optimization of blockchain frameworks for potential transnational applications.

One final comment, the in-the-wild methodology we pursue may be an inspiration for blockchain testing in other large-scale applications, such as IoT, Internet of Vehicles (IoV), and supply chain management, where performance at scale, and trust, may be limiting factors.

7. Conclusions

The methodology of this paper is driven by the use-case of operating a (trans)national e-business platform in Europe and deploying an in-the-wild test method using four distributed data replication platforms. The use-case provides the benchmark for system performance and the topology of the network and infrastructure. Two readily available Byzantine fault tolerant blockchain platforms, Quorum and Tendermint, one crash fault tolerant blockchain, Hyperledger Fabric and one crash fault tolerant data streaming platform, Apache Kafka, are tested and compared with identical transaction loads, network and hardware conditions. The overall finding, that the crash fault tolerant data streaming is significantly faster than the Byzantine fault tolerant platforms, is unsurprising. A surprising finding however is that the crash fault tolerant blockchain, Hyperledger Fabric, which uses Appache Kafka as a ordering service, performs very differently compared to a stand-alone

implementation of Apache Kafka. Moreover, this paper's main contribution is to quantify how much faster crash fault tolerant platforms are compared to Byzantine fault tolerant ones, for the same use-case running on the same hardware and network infrastructure. Measuring the difference is a way of putting a performance cost on decentralised trust as a system feature, and provides incentives to carefully consider whether decentralised trust is an essential system property. With respect to the e-business use-case, it is clear that Byzantine fault tolerant protocols, running on standardised cloud infrastructure, with realistic transaction volumes and sizes, are currently insufficient to work on a EU scale, and only adequate to work on a national scale.

Author Contributions: J.S.S.: Conceptualization, Methodology, Project Administration, supervision, Writing—original draft. P.W.E.: Conceptualization, Methodology, supervision, Writing—review & editing. L.H.: Software, investigation, data curation. J.S.: Conceptualization, Methodology, investigation, Software. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. Diedrich, H. *Ethereum: Blockchains, Digital Assets, Smart Contracts, Decentralized Autonomous Organizations*; Wildfire Publishing: Sydney, Australia, 2016.
2. Stockton, N. China takes blockchain national: The state-sponsored platform will launch in 100 cities. *IEEE Spectr.* **2020**, *57*, 11–12. [CrossRef]
3. Lamport, L.; Shostak, R.; Pease, M. The Byzantine generals problem. *ACM* **1982**, *4*, 382–401. [CrossRef]
4. consensys.net. A Permissioned Implementation of Ethereum Supporting Data Privacy. 2021. Available online: https://github.com/ConsenSys/quorum (accessed on 22 May 2023).
5. Buchman, E. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains. 2016. Available online: http://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf (accessed on 22 May 2023).
6. Vukolić, M. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, Co-Located with ASIA CCS 2017*; Association for Computing Machinery: New York, NY, USA, 2017; pp. 3–7. [CrossRef]
7. Kreps, J.; Narkhede, N.; Rao, J. Kafka: A Distributed Messaging System for Log Processing. In Proceedings of the NetDB, Association for Computing Machinery, Athens, Greece, 12–16 June 2011; pp. 1–7. [CrossRef]
8. Deloitte. Analyse Af Erhvervsøkonomiske Konsekvenser Af Udvalgte E-Handelsinitiativer. 2017. Available online: https://www2.deloitte.com/dk/da/pages/finance/articles/deloitte-economics.html (accessed on 16 April 2023).
9. Danmarks Nationalbank. Available online: https://www.nationalbanken.dk/da/bankogbetalinger/betalingsraad/Documents/BR_Rapport_om_virksomhedsbetalinger_2015.pdf (accessed on 22 May 2023).
10. Baily, P. Purchase Orders and Contracts. In *Purchasing and Supply Management*; Springer: Cham, Switzerland, 1987; pp. 46–59.
11. Danish Business Authority. Submission of Annual Report. 2019. Available online: https://indberet.virk.dk/myndigheder/stat/ERST/Regnskab_20.pdf (accessed on 16 April 2023).
12. OpenPEPPOL. PEPPOL Invoice Example. 2019. Available online: https://raw.githubusercontent.com/OpenPEPPOL/peppol-bis-invoice-3/master/rules/examples/base-example.xml (accessed on 16 April 2023).
13. de Best, R. Available online: https://www.statista.com/statistics/647523/worldwide-bitcoin-blockchain-size/ (accessed on 22 May 2023).
14. Zhuang, Y.; Tredger, S.; Matthews, C.; McGeer, R.; Coady, Y. Distributed Systems in the Wild: The Theoretical Foundations and Experimental Perspectives. In Proceedings of the 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Victoria, BC, Canada, 12–14 November 2012; pp. 87–94. [CrossRef]
15. Spasovski, J.; Eklund, P. Proof of Stake Blockchain: Performance and Scalability for Groupware Communications. In Proceedings of the 9th International Conference on Management of Digital EcoSystems, Bangkok Thailand, 7–10 November 2017; pp. 251–258. [CrossRef]
16. Hao, Y.; Li, Y.; Dong, X.; Fang, L.; Chen, P. Performance analysis of consensus algorithm in private blockchain. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018; pp. 280–285.
17. Wang, Z.; Dong, X.; Li, Y.; Fang, L.; Chen, P. Iot security model and performance evaluation: A blockchain approach. In Proceedings of the 2018 International Conference on Network Infrastructure and Digital Content (IC-NIDC), Guiyang, China, 22–24 August 2018; pp. 260–264.
18. Pongnumkul, S.; Siripanpornchana, C.; Thajchayapong, S. Performance analysis of private blockchain platforms in varying workloads. In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017; pp. 1–6.

19. Dinh, T.; Wang, J.; Chen, G.; Liu, R.; Ooi, B.; Tan, K. BLOCKBENCH: A Framework for Analyzing Private Blockchains. *arXiv* **2017**, arXiv:1703.04057v1.
20. Kruger, A. Chainhammer. 2019. Available online: https://github.com/drandreaskrueger/chainhammer (accessed on 22 May 2023).
21. Sund, T.; Lööf, C.; Nadjm-Tehrani, S.; Asplund, M. Blockchain-based event processing in supply chains—A case study at IKEA. *Robot. Comput.-Integr. Manuf.* **2020**, *65*, 101971. [CrossRef]
22. Cachin, C.; Vukolic, M. Blockchain Consensus Protocols in the Wild. *CoRR* **2017**. [CrossRef]
23. Beck, R.; Eklund, P. Factors that Impact Blockchain Scalability. In Proceedings of the 11th International Conference on Management of Digital EcoSystems (MEDES 2019), Limassol, Cyprus, 12–14 November 2019.
24. Ongaro, D.; Ousterhout, J. In Search of an Understandable Consensus Algorithm. In Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14), Philadelphia, PA, USA, 19–20 June 2014; pp. 305–319.
25. Moniz, H. The Istanbul BFT Consensus Algorithm. *arXiv*, **2020**, arXiv:2002.03613.
26. Buterin, V. The Etheruem Whitepaper. 2014. Available online: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf (accessed on 6 June 2023).
27. Baliga, A.; Subhod, I.; Kamat, P.; Chatterjee, S. Performance Evaluation of the Quorum Blockchain Platform. *arXiv* **2018**, arXiv:1809.03421.
28. Castro, M.; Liskov, B. Practical Byzantine fault tolerance. In Proceedings of the OSDI, New Orleans, LA, USA, 22–25 February 1999; Volume 99, pp. 173–186.
29. Hyperledger. Introduction to Hyperledger Fabric. 2020. Available online: https://hyperledger-fabric.readthedocs.io/en/release-2.2/whatis.html (accessed on 22 May 2023).
30. Kafka, A. Apache Kafka Documentation. 2020. Available online: https://kafka.apache.org/ (accessed on 22 May 2023).
31. Magnoni, L. Modern Messaging for Distributed Sytems. *J. Phys. Conf. Ser.* **2015**, *608*, 012038. [CrossRef]
32. Microsoft. Azure Locations. 2019. Available online: https://azure.microsoft.com/en-us/global-infrastructure/locations/ (accessed on 16 April 2023).
33. Campean, D. How to Build a Distributed Load Testing Infrastructure with AWS, Docker, and JMeter. 2019. Available online: https://dragoscampean.medium.com/how-to-build-a-distributed-load-testing-infrastructure-with-aws-docker-and-jmeter-accf3c2aa3a3 (accessed on 16 April 2023).
34. Microsoft. Available online: https://learn.microsoft.com/en-us/answers/questions/31361/how-can-i-use-ethereum-through-azure (accessed on 22 May 2023).
35. Available online: https://github.com/tendermint/tendermint/blob/master/docs/nodes/running-in-production.md (accessed on 22 May 2023).
36. Thakkar, P.; Nathan, S.; Viswanathan, B. Performance benchmarking and optimizing Hyperledger Fabric blockchain platform. In Proceedings of the 26th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Milwaukee, WI, USA, 25–28 September 2018; pp. 264–276. [CrossRef]
37. Gorenflo, C.; Lee, S.; Golab, L.; Keshav, S. FastFabric: Scaling hyperledger fabric to 20,000 transactions per second. *Int. J. Netw. Manag.* **2020**, *30*, e2099. [CrossRef]
38. Sukhwani, H.; Martínez, J.M.; Chang, X.; Trivedi, K.S.; Rindos, A. Performance modeling of PBFT consensus process for permissioned blockchain network (Hyperledger Fabric). In Proceedings of the IEEE Symposium on Reliable Distributed Systems, Hong Kong, China, 26–29 September 2017; pp. 253–255. [CrossRef]
39. Sukhwani, H.; Wang, N.; Trivedi, K.S.; Rindos, A. Performance modeling of hyperledger fabric (permissioned blockchain network). In Proceedings of the NCA 2018—2018 IEEE 17th International Symposium on Network Computing and Applications, Cambridge, MA, USA, 1–3 November 2018. [CrossRef]
40. Kuzlu, M.; Pipattanasomporn, M.; Gurses, L.; Rahman, S. Performance analysis of a hyperledger fabric blockchain framework: Throughput, latency and scalability. In Proceedings of the 2019 2nd IEEE International Conference on Blockchain, Blockchain 2019, Atlanta, GA, USA, 14–17 July 2019; pp. 536–540. [CrossRef]
41. GitHub [Source Code]. Available online: https://github.com/hyperledger/caliper-benchmarks/ (accessed on 22 May 2023).
42. Geneiatakis, D.; Soupionis, Y.; Steri, G.; Kounelis, I.; Neisse, R.; Nai-Fovino, I. Blockchain Performance Analysis for Supporting Cross-Border E-Government Services. *IEEE Trans. Eng. Manag.* **2020**, *67*, 1310–1322. [CrossRef]
43. Kreps, J. Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines). 2014. Available online: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines (accessed on 22 May 2023).
44. GitHub [Source Code]. Available online: https://github.com/tendermint/tendermint/issues/1835#issuecomment-402054099 (accessed on 22 May 2023).
45. GitHub [Source Code]. Available online: https://hyperledger.github.io/caliper-benchmarks/fabric/performance/2.1.0/goContract/nodeSDK/submit/create-asset/ (accessed on 22 May 2023).
46. Le Noac'h, P.; Costan, A.; Bougé, L. A performance evaluation of Apache Kafka in support of big data streaming applications. In Proceedings of the 2017 IEEE International Conference on Big Data, Big Data 2017, Boston, MA, USA, 11–14 December 2017; pp. 4803–4806. [CrossRef]
47. Du, Y.; Chowdhury, M.; Rahman, M.; Dey, K.; Apon, A.; Luckow, A.; Ngo, L.B. A Distributed Message Delivery Infrastructure for Connected Vehicle Technology Applications. *IEEE Trans. Intell. Transp. Syst.* **2018**, *19*, 787–801. [CrossRef]

48. Nguyen, D.; Luckow, A.; Duffy, E.; Kennedy, K.; Apon, A. Evaluation of highly available cloud streaming systems for performance and price. In Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Washington, DC, USA, 1–4 May 2018; pp. 360–363. [CrossRef]
49. IBM. Available online: https://www.ibm.com/blogs/blockchain/2019/04/does-hyperledger-fabric-perform-at-scale/ (accessed on 22 May 2023).