Performance Evaluation of a Blockchain-based Traceability System
- A Case Study at IKEA

Prestandautvärdering av ett blockkedje-baserat spårbarhetssystem

Claes Lööf
Tobias Sund

Supervisor : Simin Nadjm-Tehrani
Examiner : Mikael Asplund
Upphovsrätt


Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page [http://www.ep.liu.se/](http://www.ep.liu.se/)

© Claes Lööf
Tobias Sund
Abstract

Establishing traceability in global supply chains is a complicated problem. Current solutions for achieving traceability are expensive or imperfect, and give rise to organizational and trust-related issues. Blockchain could present itself as a solution to many of these issues. This thesis aims to build a blockchain-based traceability system. Based on the event characteristics in IKEA Supply Chain, our analysis show that, for timely processing, the capacity of a traceability system should be 10,593 events per second. Additionally, 14 requirements were identified and included in the system design. A system was designed that consists of six components, a client application, a controller, a smart contract pool, IPFS and Quorum. In order to reduce the potential load on the system, certain optimization measures were taken. The system design resulted in a load requirement of 14,975 transactions for a delay bound of one minute. The resulting performance of the developed system revealed itself to be a throughput of 159 transactions per second and a convergence time of 4.71 seconds, which is not enough to reach the requirement. However, a solution is proposed to divide the network into many smaller networks that together can produce the necessary throughput.
Acknowledgments

We would like to extend an appreciation and thanks to our tutors at IKEA, Olof Orstadius and Claudio Marconi. Without them, this thesis would not have been possible. They helped guide us throughout this work, were always there to help us, aided us in understanding IKEA and connected us with personnel at IKEA that was of interest for the interviews.

The personnel at IKEA that we interviewed were also invaluable. We had many great discussions about IKEA, blockchain, traceability and which problems exist in their supply chain. Everyone embraced us with open arms and were willing to go the extra mile to help us.

Finally, we want to thank our examiner and tutor at Linköping university, Mikael Asplund and Simin Nadjm Tehrani. Their guidance, knowledge and understanding has been incredibly valuable. This work proved much harder and took longer time than anticipated. Without them, this thesis would never have been completed.
Contents

Abstract iii
Acknowledgments iv
Contents v
List of Figures vii
List of Tables viii

1 Introduction 1
  1.1 Motivation 1
  1.2 Aim 3
  1.3 Research questions 3
  1.4 Overview and Methodology 3
  1.5 Delimitations 3

2 Background and related work 5
  2.1 Blockchain 5
  2.2 Data storage 7
  2.3 Privacy and security 8
  2.4 Related work 8

3 Supply chain data extraction and modelling 11
  3.1 IKEA - Case company overview 11
  3.2 Traceability-related events 13
  3.3 Supply chain modelling 14

4 Requirements 18
  4.1 Method for identifying requirements 18
  4.2 Identified requirements 19

5 System design 21
  5.1 Blockchain choice 21
  5.2 System architecture 23
  5.3 Validator 24
  5.4 Smart contract and off-chain storage 25
  5.5 Client application 28
  5.6 Controller 29
  5.7 Smart contract pool 29
  5.8 Transaction capacity requirement 30

6 System performance evaluation 32
  6.1 Method 32
**List of Figures**

1.1 Work flow diagram representing the thesis methodology. The numbers represent the ordering and the arrow represents association .................................................. 4

3.1 Illustration of the three main distribution flows in IKEA Supply Chain .................. 12

3.2 Relationship structure for the information objects .......................................... 12

3.3 A sum of the following events: shipment creation, book transport, transport confirmation, shipment start, shipment end, close delivery, consignment dispatch, consignment creation, and close receipt over each hour between the 14th and the 22nd of December 2017 (the weekdays have been relocated to illustrate an ordinary week, Monday to Sunday) ................................................................. 14

5.1 Overview of the system architecture. The arrows depict how requests can be sent to the smart contract ......................................................... 23

5.2 Sequence diagram for the system when an information object is created and an event is registered to the smart contract ........................................... 25

5.3 A representation of the smart contract and IPFS, and their relationship .................. 26

5.4 Relationship between a product smart contract and the ownership contract .......... 27

6.1 Example of a validator’s CPU load during test with high tps load which leads to failure ................................................................. 34

6.2 Example of a validator’s CPU load during test with restricted tps load which results in a stable run .......................................................... 35

6.3 Impact of choice of client hardware in a setting with one client (varying instance type), one controller (t2.xlarge) and one validator (t2.xlarge) ............. 38

6.4 Impact of choice of controller hardware in a setting with one client (t2.xlarge), one controller (varying instance type) and one validator (t2.xlarge) ....... 39

6.5 The performance impact of the change of hardware on the validators ability to process transactions. Setup: 1 client (t2.xlarge), 1 controller (t2.xlarge), 1 validator (variable) ................................................................. 40

6.6 Impact of choice of block time in a setting with one client (t2.small), one controller (t2.medium) and one validator (t2.xlarge) ......................................... 41

6.7 Impact of the number of clients (t2.small) per controller in a setting of one controller (t2.medium) and one validator (t2.xlarge) ................................. 42

6.8 Impact of the number of controllers (t2.medium) per validator in a setting of three clients (t2.small) per controller (t2.medium) and one validator (t2.xlarge) .... 42

6.9 Impact of the number of validators (t2.xlarge) in the network in a setting of ten clients (t2.small) and five controllers (t2.medium) .................................. 43

6.10 Impact of network latency in a setting of ten clients (t2.small), five controllers (t2.medium) and ten validators (t2.xlarge) ............................................. 44

6.11 Performance of IPFS when uploading files with increasing file size .................. 45

6.12 Distribution of the average ttpt for all transactions sent during the simulation ........ 46

6.13 RAM usage during the simulation ................................................................ 46
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>The identified events</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>Ratio between <em>shipment create</em> (SHFC) and all other shipment events (Evt)</td>
<td>16</td>
</tr>
<tr>
<td>3.3</td>
<td>Calculated number of events per second that needs to be handled in order to guarantee different delay bounds</td>
<td>17</td>
</tr>
<tr>
<td>5.1</td>
<td>Estimated number of transactions per second that the system must be able to handle in order to guarantee different delay bounds</td>
<td>31</td>
</tr>
<tr>
<td>6.1</td>
<td>The used AWS EC2 instance types. vCPU stands for virtual CPU and is equal to one thread. * [Wittig2018]</td>
<td>36</td>
</tr>
<tr>
<td>6.2</td>
<td>System configuration</td>
<td>37</td>
</tr>
<tr>
<td>A.1</td>
<td>Software versions</td>
<td>58</td>
</tr>
</tbody>
</table>
1 Introduction

In this chapter, a motivation for blockchain-based traceability systems is presented in section 1.1. Sections 1.2 and 1.3 specify the aim and research questions of this thesis. Finally, delimitations and the thesis outline is given in sections 1.5 and 1.4 respectively.

1.1 Motivation

Blockchain technology is believed to be a disruptive technology that has the power to transform how internet applications and services are used [4]. Blockchain technology first came to life in 2008, when a pseudonym named Satoshi Nakamoto released the paper *Bitcoin: A peer-to-peer electronic cash system* [40]. Since then, the most established use case for blockchain technology has been cryptocurrencies. However, in recent years, a magnitude of other use cases for blockchain technology have emerged. When Ethereum — a blockchain with built-in functionality for smart contracts — went live, developers were enabled to explore use cases beyond cryptocurrencies. A smart contract is, simply put, self-executing code stored on a blockchain. With the innovation of smart contracts, supply chains quickly became one of the most promising areas which could be a showcase for blockchain technology. Utilizing blockchain technology in supply chains is expected to increase transparency and improve traceability [56, 36].

Why is traceability important for supply chains? Research shows that it improves supply chain management and product quality, increases the ability to retain customers and their willingness to pay a premium price for the information provision, reduces customer complaints, liability claims and lawsuits, and reduces production costs and inventory [35, 2, 1]. On the other hand, a lack of traceability can have severe consequences. In April, E. coli contaminated romaine lettuce caused five deaths and 96 hospitalizations in the USA. After several weeks of searching, due to incomplete or unavailable shipping records and virtually no product labeling, the Centers for Disease Control (CDC) believe they found the source [9]. Another consequence is the trade in illegally harvested timber [44]. Interpol estimates the illegal trade is worth between $51 and $152 billion annually, suppressing world timber prices by between 7% and 16% depending on the product [21]. A transparent supply chain with verifiable traceability could help consumers make more informed purchase decisions, ultimately giving consumers the power to ensure sustainable supply chains [2].
1.1. Motivation

In practice, there are three ways traceability is achieved. (1) In each step of the chain, information is stored locally and selective information is sent forward along with the product. (2) Information follows the product through all steps of the chain, e.g. on a radio-frequency identification (RFID) transponder. (3) Information is stored in a database that is either managed centrally by an information provider or the dominant enterprise of the supply chain, or distributed among the supply chain partners. [38] [37]

There are several problems with the three ways traceability is achieved today. In (1), information regarding a product can only be traced backwards one step at a time. It requires information to be shared either through manual processes, which is time-consuming and error prone, or by integrating internal traceability systems. The diversity of internal traceability systems makes integration costly and difficult [47]. Additionally, inter-organizational trust, defined as “the extent of trust placed in a partner organization by the members of a focal organization”, must be factored in (e.g. can we trust this information and can we trust them with this information?). Information integrity and privacy are two vital parts of inter-organizational trust [50]. In (2), the problem comes down to cost. The cost of RFID transponders could be reduced through the restriction of storage capacity, disincentivizing enterprises from utilizing storage in RFID transponders until the cost is deemed profitable [37]. In (3), many problems are organizational [2], who should own and administer the database, who owns the data produced, how can the data integrity be trusted etc. There are also issues with standards and integration with local systems. For example, if a supplier produces goods for multiple retailers, they might need to integrate their systems with multiple other centralized information systems or use expensive electronic data exchange (EDI) solutions. Additionally, a single point of failure is introduced; if there is a failure at the system’s central point, the whole supply chain is affected. No system is a magic bullet that can solve the problems listed above, and a blockchain is not an exception. However, a blockchain could potentially solve organizational and trust issues due to its immutable and decentralized nature [29].

What defines a blockchain varies greatly, from a distributed database utilizing a consensus algorithm to public blockchains such as Bitcoin and Ethereum. One definition is that a blockchain is a decentralized ledger that keeps a record of digital transactions and is distributed among a network of participants, a ledger that no single entity single-handedly owns nor controls. A basic principle for a blockchain is that once data is uploaded to a blockchain, it becomes immutable, ensuring data integrity [22]. Immutability is an incredibly important attribute for a transparent supply chain, especially for preventing forged documents and digital records about product and raw material origins. Additionally, since the ledger is shared between all participants in the network, anyone can verify themselves if information is authentic. For a blockchain to conform to this definition, it is important that the blockchain utilizes a consensus algorithm that is fault-tolerant and can achieve reliability in a network involving multiple unreliable nodes. [61].

While blockchain technology has a promising future, the technology itself is still in its infancy. Ethereum’s permissioned blockchain had a throughput of 15.64 transactions per second at its peak [15], which is unlikely to be sufficient for large-scale usage in global supply chains. Quroum — an enterprise-focused permissioned version of Ethereum [39] — can handle a considerably higher amount of transactions with lower latency [3]. While there have been previous studies evaluating the performance of different blockchains [32] [3] [42] [12], there are few — if any — studies that explore the performance that would be required for a large retailer and evaluate which performance a fully designed and implemented system would have.
1.2 Aim

This thesis aims to explore how a traceability system could be designed and implemented utilizing the blockchain at its core, as well as what performance aspects and bottlenecks exist for such a system. The case company used in this thesis — to identify the requirements and the performance aspects — is IKEA, the world’s largest furniture retailer.

1.3 Research questions

The following research questions will be used to guide the work in this thesis:

1. What requirements exist for a blockchain-based traceability system in the IKEA Supply Chain?
2. How can a traceability system be designed and implemented utilizing the blockchain?
3. What are the performance aspects and bottlenecks for such a system?

1.4 Overview and Methodology

Figure 1.1 presents the work flow comprising the methodology in this thesis. A literature study (1) was conducted to gain an understanding of supply chains, previous work exploring both blockchain technology and traceability, as well as general requirements for a blockchain-based traceability system. Informal discussions (2) with 20 employees at IKEA took place in order to identify traceability-related events (3) and possible requirements (6) for the traceability-system. After identifying the events, personnel at IKEA aided in extracting real-time data (4) from their databases for some of the events. A supply chain model was then constructed (5) based on the information gathered, resulting in various performance requirements depending on the delay bound within which transaction finality time is guaranteed. A prototype was then designed (7), implemented (8). The prototype was then evaluated by setting up a test environment and conducting a series of performance experiments (9).

1.5 Delimitations

This thesis will not explore the feasibility of integrating a blockchain-based traceability system with existing system infrastructure. Neither will it explore the costs associated with implementing such a system.

Traceability events for a product’s life-cycle before the product has been manufactured will be disregarded.
1.5. Delimitations

Figure 1.1: Work flow diagram representing the thesis methodology. The numbers represent the ordering and the arrow represents association.
This chapter provides a theoretical background for the results presented in this thesis. Section 2.1 presents theory on underlying concepts of consensus algorithm and smart contracts. Section 2.2 review theory on on-chain and off-chain storage. Section 2.3 presents theory on privacy. Finally, section 2.4 provides an overview of related work in this area.

2.1 Blockchain

A variety of new blockchains have been created since the creation of the first blockchain in 2009, the Bitcoin blockchain. The common denominator in all these blockchain solutions is that they have an underlying data structure of linked blocks and are distributed to a varying degree. Bitcoin’s original vision was to be a decentralized, distributed system that was permissionless (i.e. anyone is welcome take part in the system) and secure enough to be used as a payment network [40]. This required trade-offs in terms of processing throughput, processing latency, and guarantees that state changes are final. In many use cases, these trade-offs are not acceptable and therefore other solutions contending with these shortcomings have been created, although not without making their own trade-offs. These trade-offs mostly revolve around the decentralized and permissionless nature of blockchain technology, effectively making them more similar to established distributed systems than the original Bitcoin blockchain [52].

All of these attributes and trade-offs originate in the choice of consensus algorithm. What an consensus algorithm is and what these different attributes are will be explained in more detail in the following subsection. Additionally the concept of a smart contract will be introduced.

2.1.1 Underlying concepts of consensus algorithms

Consensus algorithms are a central part of blockchain technology. There are several different kinds of consensus algorithms. To understand the differences, a couple of concepts are introduced and explained. Consensus algorithms are used in distributed architectures where interconnected computers want to communicate and collaborate for a common goal. This is often called Distributed Computing. In the context of blockchains the goal is generally to
maintain and update a shared immutable state, mediating exchanges, and providing a secure computing engine [6].

**State-machine replication (SMR).** The process of reaching and maintaining consensus among distributed entities, or nodes, can be divided in two parts. One part where replicas of a state machine runs on all the nodes. It drives the logic of the service aimed to be provided. The second part consisting of the actual consensus algorithm which is tasked with propagating the requests among the nodes so that each node executes them in the same order [46].

**Network model.** In order for a consensus algorithm to be deemed feasible for a SMR system, it must guarantee safety and liveness [6]. Safety means that the algorithm should always be correct during normal execution and in presence of a given fault model, and liveness means that every execution should be completed correctly at some point in time. However, as discovered by Fischer et al. [16], this is impossible to guarantee in an asynchronous network model, where messages may be delayed indefinately. It would be possible to guarantee in a synchronous network model, where messages always arrive when in a correct order. For blockchains, it is not possible to guarantee a synchronous network. Due to the geographical distance between the distributed nodes, one power outage may cause a partitioning of the network, which effectively results in messages becoming unable to be transferred between the partitions. To combat this problem, Dwork et al. [14] introduced the eventual-synchrony network model. The model assumes that asynchronous networks will eventually become synchronous within some time bound. This model is widely accepted for designing resilient distributed systems today [6].

**The CAP theorem.** The CAP theorem [17] states that in distributed system, only two of the three following properties can be fulfilled: Consistency (C), Availability (A), and Partition tolerance (P). Consistency is obtained if a system can guarantee that any read receives the most recent state or an error, e.g. two clients are not to receive two different values for the same request if done at the same time. Availability requires that every request receives a non-error response at all times. Partition tolerance states that a system must be able to continue to operate even when messages are unable to be distributed to all nodes in the network. Since the theorem states that only two out of three properties is possible to obtain at the same time, a system can either be CA, CP or AP. However, as Brewer explains [5], it is actually more nuanced than this and most of the time it is not possible to escape the need for Partition tolerance, effectively making a CA system unfeasible. This is especially true for blockchains, since one fundamental idea is to geographically spread the nodes to increase decentralization [6]. The same example with a power outage at the wrong place, as explained before, can be used here as well for illustrating the need for partition tolerance.

**Failure models.** When developing consensus algorithms it is vital to understand under which circumstances it is meant to work. Every distributed system can be subjected to different kinds of issues such as communication delays, system failures or even malicious actors that want the system to fail or behave in their favor. These different failures have been classified by Cristian et al. [11] in a series of nested classes. For example, one of the inner classes is crash faults, which represent a failure model where systems can crash. Consensus algorithms that are modeled based on this failure model, are referred as crash-tolerant. However, a system can be subjected to other types of failures which is harder to handle than a node that crashes. The failure model that encompasses all of these failures is the byzantine failure model, which was first introduced by Lamport et al. [30]. In a byzantine failure model, a system needs to handle malicious actors that actively works against the common goal of reaching agreement. It does not have to be done consciously by an actor, but can arise from software bugs or other malfunctioning in the physical infrastructure [13]. Consensus algorithms that can tolerate these faults are called Byzantine fault-tolerant, or BFT for short.
2.2. Data storage

One aspect important to understand is that these failures can only be tolerated to a certain extent, even if a consensus algorithm is stated to tolerate the failure. Generally, in the case for crash-tolerant consensus algorithms, the system continues to work as long as more than half of all nodes still are working. However, in the case of BFT-consensus algorithms, the number of malicious, or byzantine, nodes must be less than 1/3 of the total number of nodes [30].

**Permissioned versus permissionless, private versus public.** In the context of blockchains, one of the first factors used to differentiate blockchains is if they are permissioned or permissionless. A permissioned blockchain only allows a certain set of actors to take part in the consensus process, while a permissionless blockchain is open for everyone [6]. This is more of a choice than a specific attribute of a consensus algorithm, since most consensus algorithms can be run in a permissioned or permissionless manner with some minor tweaks. However, in most cases, consensus algorithms are developed with one of the two in mind. Furthermore, the labels private and public are also a choice even if certain consensus algorithms cater more towards one or the other. A private blockchain is only accessible for actors with access, while a public blockchain is accessible for everyone.

2.1.2 Smart contracts

Smart contracts were introduced in 1994 by Nick Szabo [51], who defined smart contracts as “a computerized transaction protocol that executes the terms of a contract”. Szabo suggested that smart contracts could be used to translate contractual clauses into code and embed them into hardware or software that could self-enforce the code, minimizing the need for trusted intermediaries [51].

In the context of blockchain technology, smart contracts are scripts that are stored on the blockchain and are identified by a unique address [52]. Ethereum was the first blockchain that was designed as a smart contract platform, and its origin is linked to critique made by Vitalik Buterin on the limitations of Bitcoin’s scripting language. In Ethereum [58], smart contracts are executed in the *Ethereum Virtual Machine* (EVM), which provides a Turing complete programming language. Since every single operation is executed by every node in the network, a mechanism is applied to limit the resources used by each contract. Each operation therefore has a cost measured in gas, and each gas unit consumed by a transaction is paid in Ether, a cryptocurrency. Gas price refers to how many Ether each gas unit costs. Since each block that is included on the blockchain has a gas limit, transactions that use a lot of gas limit the amount of transactions that can be included in a block.

2.2 Data storage

Decentralized applications running on blockchain platforms might need to store and retrieve data, just as conventional applications do using MySQL, MongoDB etc. EVM allows storing data permanently, which is referred to as storing data on-chain. Storing data on-chain can be expensive and inefficient since it is replicated across either all nodes or a select number of nodes, depending on whether the contract is public or private. An alternative is to store data outside the EVM, which is referred to as storing data off-chain.

*InterPlanetary File System* (IPFS) is a distributed file system that works well with blockchains [61]. It is free, but requires an IPFS server to host the data. Data stored on IPFS can be replicated automatically or depending on user behavior, e.g. data is only replicated once a user accesses it. While storing data in an off-chain system lacks the fundamental properties of a blockchain, it is more cost-efficient, has better performance and is more flexible than storing data in smart contracts [61]. The downside of using off-chain storage is that the data is not directly accessible by the smart contracts, making it more difficult to automate processes on the blockchain.
2.3 Privacy and security

Cryptography is a fundamental operation in computer security that converts information from a readable form to an unreadable form. There are two main subcategories to cryptography, symmetric and asymmetric encryption. Symmetric encryption is also known as single key cryptography, as the same secret (key) is used for both encryption and decryption. The three most common symmetric key cryptography algorithms are DES, AES and Blowfish [53]. Among these, Blowfish has best performance and does not have any known security flaws.

Asymmetric encryption is known as public key cryptography and uses two keys instead of one, a public key and a private key. The public key is used for encryption and the private key is used for decryption [49]. The two largest blockchains, Bitcoin and Ethereum, use public key cryptography for creating wallets and signing transactions [58, 33]. When a transaction is signed with a private key, it produces a unique signature. Once the transaction is signed by the owner, it is sent to the memory pool where it stays until it is processed by miners or validators. The miner or validator then uses the sender’s public key to ensure that the signature is authentic and the transaction is included in the next block. Given the importance of a private key, it must be stored and managed securely.

While privacy is impacted on many blockchains because the information is available to each participant in the network, Quorum provides functionality for private transactions and contracts [39]. In blockchains that do not offer this functionality, sensitive data can be encrypted and stored either in smart contracts or on off-chain storage. While jurisdictional boundaries could increase risk when immutably storing data in smart contracts, storing data off-chain provides better control access and does not risk undermining regulatory controls (e.g. GDPR, the General Data Protection Regulation in EU law) when a network spans multiple jurisdictional boundaries.

2.4 Related work

The interest in blockchain and its potential applications have increased significantly the last couple of years, not the least in the context of supply chains. In this section, a brief rundown of some of the relevant previous work is presented.

2.4.1 Blockchain-based traceability systems

There are several academic reports and papers that explore the concept of a blockchain-based traceability system [45, 7, 27, 55, 41, 48, 60, 59, 54]. Many of these works explores traceability within the context of food and agriculture. The depth of the work is varying. In most cases a simple proof of concept is conceived and implemented, but not extensively tested, if tested at all. From an architectural aspect most of the systems store all of the traceability data on-chain. However, in the work done by Ramachandran et al. [45] and Shafagh et al. [48] they store the actual traceability data off-chain, linking it to the blockchain via a hash of that data. They do this mainly due to the high on-chain storage costs.

2.4.2 Performance evaluations

Through the literature review a couple of blockchain performance evaluations [32, 3, 42, 12] were found. Sadly Lin and Baliga et al. [32, 3] is not peer-reviewed. Furthermore there does not seem to be any standard surrounding performance evaluations of blockchains. This is unfortunate especially when talking about smart contract transactions since the computational complexity of each transaction affects the performance. Dinh et al. [12] have tried to solve this issue through their blockchain benchmarking framework Blockbench. However, the tool
2.4. Related work

is still early in development thus has not seen wider adoption as a standard at the time of writing.

Despite that some of the works have not been peer-reviewed and their lack of standardization they might provide insights into general aspects that affect the performance of a blockchain system.

In the non-peer-reviewed work carried out by Baliga et al. [3], they explore how different parameters affect the transaction throughput and latency. They used the Ethereum-based Quorum platform [39] running with the Istanbul Byzantine Fault-tolerant (IBFT) consensus algorithm. In their tests they run a couple of clients who together send between 150 and 1650 transactions per second (tps) for about 30 seconds, amounting to a total of max 49500 transactions. Each of the four blockchain nodes in the network ran on machines equipped with 8 vCPUs at 3.6 GHz and 16 GB of RAM. Their tests consisted of, among other things, updating a value for a randomly selected key in the key-value store on a simple smart contract. Their system was able to achieve a throughput of upwards 1900 tps with a latency of around 3.5 seconds.

The team behind the IBFT consensus algorithm mentioned above have also done some evaluations (which are not peer-reviewed) of their consensus algorithm together with the native Ethereum-node-software Geth implementation [32]. In their tests they used a Standard D2 V2 Azure VM instance running a 2 core CPU with 7 GB of RAM. Their network consisted of 4 validators and ordinary transactions (no smart contracts were used) were sent from 100 accounts in total, 25 for each validator. They ran a test for 10 minutes and achieved an average tps of 831 and saw peak values of 1207 tps. Their conclusion was that Geth together with IBFT can reach 1000+ tps, and there are multiple factors in the Geth implementation that limits the throughput. These limiting factors are (1) that there seems to be a transaction generation overhead, (2) that Geth seems to have issues when processing large blocks.

Another performance evaluation that is of interest is the work done by Pongnumkul et al. [42]. They make an analysis of the execution layer of Geth v1.5.8 and Hyperledger Fabric [32] i.e. they only look at the process of updating the state excluding all consensus related processes. By doing this they hope to find a theoretical max performance. In their test, they used a AWS EC2 c4.2xlarge instance equipped with 8 cores 15 GB of RAM. They tested the performance by sending between 1 and 10000 transactions invoking a value transferring function in a smart contract. Relevant measurements was the average throughput (tps) and the average convergence time for the transactions. The average tps varied based on the amount of transactions that was sent. The maximum tps of 38.93 was reached when 100 transactions were sent. The average convergence time did also vary with the amount of transactions sent, increasing with the number of transactions sent. When 100 transactions was sent the average convergence time was recorded at 2.15 seconds. An interesting result from their test was that it seemed to be the CPU that was the limiting factor when load was increased to its maximum.

One of the more interesting evaluations is the work carried out by Dinh et al. [12]. They set out to create the blockchain benchmarking framework Blockbench, and in doing so they also tested it on Ethereum (Geth v1.4.18 and Parity) and Hyperledger Fabric. In their tests of Geth, they use Proof-of-Work (PoW) as the consensus algorithm. In their results they present that Geth seems to be limited by the CPU (8 cores at 3.5GHz with 32GB RAM were used). Furthermore they saw non-negligible negative performance impact when the complexity of the smart contract transactions were increased, indicating that a substantial part of the CPU utilization is used to process the transactions.

---

1https://geth.ethereum.org
3https://www.parity.io/
Confais et al. [10] evaluated the performance of object store systems in fog/edge computing infrastructures, with one of the object store systems being IPFS. They tested both the mean writing and reading time on 1, 7, and 11 sites with a latency of 50 ms and a 10 Gpbs throughput of the network links. Uploading 100 objects á 1 MB to IPFS, on one site, resulted in a mean writing time of 2.77 seconds and a mean reading time of 2.02 seconds. Uploading 10 objects á 10 MB to IPFS, on one site, resulted in a mean writing time of 2.80 seconds and a mean reading time of 2.04 seconds. They found that with IPFS, neither writing nor reading time is significantly affected by additional sites.
One of the research questions in this thesis included identifying the performance aspects of the traceability system that was designed and implemented. Throughput is one of these performance aspects — how many transactions per second would such a system be required to handle? This chapter provides a background about IKEA and an analysis of IKEA Supply Chain aimed at estimating the magnitude and frequency of the information the system would need to process. A simplified model of IKEA Supply Chain was created, and based on this model, traceability-related events used in the estimation process were identified. After extracting data related to the events previously identified, a model was created aimed at estimating the magnitude and frequency of the events.

3.1 IKEA - Case company overview

IKEA is the world’s largest furniture retailer and was founded by Ingvar Kamprad in Sweden, 1943. IKEA is an acronym that consists of the initials of the founder’s name, Ingvar Kamprad, the farm where he grew up, Elmtaryd, and his hometown in Småland, Agunnaryd. In the fiscal year ending in August 2017, there were 403 IKEA stores across 49 markets and nearly four billion products were sold, amounting to a volume of 36 622 756 m³.

3.1.1 Distribution network

Understandably, IKEA Supply Chain and its logistics network is very complex. IKEA Supply Chain includes roughly 1 000 home furnishing suppliers across 51 countries, each of whom have their own suppliers of e.g. product parts or materials. In order to transport products from suppliers and ultimately to consumers, a distribution network is utilized. Based on the discussions with personnel at IKEA, an overview of the logistics network was obtained. There are three main distribution channels, illustrated in figure 3.2, through which roughly 90% of the products flow. 39% of the products flow through distribution centers. Products are stored in distribution centers until there is a demand from the stores or customer distribution centers (CDC), which is a logistic unit that is the first choice to execute picking and dispatch to a customer’s requested address or a pick up point. 38% flow directly to store or CDC and 23% flow through consolidation points, which are units where a consolidation or deconsolidation process occurs. In the calculations for this thesis, a simplification was made to disregard
3.1. IKEA - Case company overview

Figure 3.1: Illustration of the three main distribution flows in IKEA Supply Chain

all distribution channels except for these three. A deeper analysis would provide a fairer representation of the product’s life-cycle, but it was outside the scope of this thesis.

3.1.2 Information objects in a product’s life-cycle

During a product’s life-cycle, various information objects are used to represent either a single product or a group of products. These information objects are listed below:

- **Product** - An article that is manufactured and refined for sale.
- **Order** - A request for one or several products to be manufactured.
- **Shipment** - Transportation perspective of shipping products. When and where does which transport unit load the products, and when and where is the transport ended.
- **Consignment** - Product perspective of shipping products. Which products need to be shipped, from which sender and to which receiver.
- **Pallet** - A flat structure on which several products are transported.

Figure 3.2: Relationship structure for the information objects
These information objects have various relations to each other, as illustrated in figure 3.2. An order has a *one-to-many* relationship to products. Shipments have a *many-to-many* relationship to consignments, one consignment can consist of multiple shipments before arriving at the receiver and one shipment can contain multiple consignments. Shipments have a many-to-many relationship to pallets, one shipment contains many pallets and one pallet may require multiple shipments to arrive at its end point. A pallet has a one-to-many relationship to products in most cases, exceptions may occur in cases where a pallet needs to be repackaged.

While a shipment represents the transportation perspective of shipping products from A to B, a shipment may contain multiple shipment legs. A shipment may for example involve one shipment leg by train, another shipment leg by a cargo ship and one or more shipment legs by truck.

### 3.2 Traceability-related events

Relevant events that could be of interest in a traceability system had to be identified. This was achieved by interviewing key personnel at IKEA with knowledge about IKEA Supply Chain and how information regarding a product’s life-cycle is produced and used. Most of the events were extracted from various work-flow processes within IKEA. When the events were identified, raw data was requested from IKEA for further analysis.

Some of the events that would be registered in the traceability system are already registered in various systems in IKEA today, making it easier to estimate the magnitude of them since raw data exists. Other events were new and needed deeper analysis to arrive at a reasonable estimation. All the data that was collected was then used to estimate how many events occurred per second. This information was later used when designing the system.

The events that were identified during the discussions with IKEA personnel were events that affected information regarding a product, from a traceability perspective. The most important events were, in the interviews, identified as events that have legal and/or economic value. For example, ensuring that raw materials originate from certified sources and not through illegal trade, or who has had custody of a product during each step of its life-cycle. However, the events that exist within the scope of this thesis are mainly related to tracking and tracing of a product, such as 4, 12 and 15. The 28 identified events are listed in table 3.1.

<table>
<thead>
<tr>
<th>1. Product creation</th>
<th>15. Shipment end</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Order creation</td>
<td>16. Check logistic unit and seal</td>
</tr>
<tr>
<td>3. Consignment creation</td>
<td>17. Check delivery documents</td>
</tr>
<tr>
<td>4. Shipment creation</td>
<td>18. Close delivery</td>
</tr>
<tr>
<td>5. Pallet creation</td>
<td>19. Unload unit load</td>
</tr>
<tr>
<td>6. Schedule shipment</td>
<td>20. Close receipt*</td>
</tr>
<tr>
<td>7. Book transport</td>
<td>21. Scan of unit load at gate</td>
</tr>
<tr>
<td>8. Transport confirmation</td>
<td>22. Put and verify</td>
</tr>
<tr>
<td>9. Arrival for loading</td>
<td>23. Receive assignment</td>
</tr>
<tr>
<td>10. Register delivery documents</td>
<td>24. Verify article</td>
</tr>
<tr>
<td>11. Consignment dispatch</td>
<td>25. Verify destination</td>
</tr>
<tr>
<td>13. Register metadata</td>
<td>27. Sell product</td>
</tr>
</tbody>
</table>

*Table 3.1: The identified events*
3.3 Supply chain modelling

The identified events can be attributed directly or indirectly to the information objects that exist in IKEA Supply Chain. These objects relate to each other in a hierarchical structure (see figure 3.2). This relationship structure has a large impact on the number of transactions that have to be made, and thereby an impact on the final performance requirements. Many of the events identified impact multiple information objects. A consignment can include many orders, and an order can include many products. At the same time, a consignment can include multiple shipments and a shipment can include multiple consignments. Furthermore, a shipment consists of several pallets. Pallets that in turn consists of the actual products. An information update for a shipment, consignment, order, etc. affects all the products that it encompasses. This is important to keep in mind when estimating the performance requirements and later on when designing the system.

Most of the events identified are events that are already defined and used in systems at IKEA today, and it was therefore possible to extract raw data for event occurrences. However, IKEA does not track individual products today and since the goal of our system is to track every individual product, product specific events needed to be derived from its relation to its parent objects.

Event data was then extracted from a database at IKEA for one week, for the events: shipment creation, book transport, transport confirmation, shipment start, shipment end, close delivery, consignment dispatch, consignment creation, and close receipt. The raw data for these specific events were extracted due to availability, the remaining raw data was not available for extraction. The raw data was presented on a minute by minute basis for a period of seven days. To illustrate the distribution of the event occurrences, all events from the raw data were summed over each hour during the week and illustrated in figure 3.3.

In the figure, a recurring pattern can be identified that shows that the sum of event registrations culminates between 09:00 and 10:00 GMT every day. The same pattern can also be seen when looking at the events individually. By calculating the Bhattacharyya coefficient [25], which refers to the degree of similarity between distributions, the similarity between these individual events can be measured. The distributions of the events result in a coefficient of 0.923, inferring that there is a high degree of similarity.
3.3. Supply chain modelling

Given that all events in figure 3.3 are related to the transportation of the same products and there is a high degree of similarity between the events, an assumption can be made about the distribution for events that lack raw data. In other words, the event occurrence distribution for the events that lack raw data is assumed to be similar to the distributions of the events depicted in the figure 3.3. This assumption is important, because the distribution of the total number of event registrations affects the performance requirements, depending on the delay bound.

Delay bound refers to the maximum acceptable delay for an event or transaction to be processed and finalized by the system. With a low delay bound, the amount of events the system must be able to handle each second would increase due to the peaks in event registrations. With a flat distribution (event registrations over time is constant), changes in the delay bound would not affect the amount of events per second the system would need to be able to handle.

3.3.1 Load calculation

This subsection explains the method used to determine the potential load on a blockchain-based traceability system. Due to a lack of raw data, it was first necessary to derive the relationship between the events. Since the frequency of the events varies throughout the day, it was then necessary to determine how many events per second the system must be able to handle in order to guarantee that an event transaction is processed and finalized within a specified time unit.

A base event was chosen to determine how the events relate to each other. Shipment create was chosen as a base event. Then, additional information was needed to determine the ratio between the base event and the remaining 27 events. There were some difficulties obtaining numbers for the same fiscal year. However, this problem was omitted since the magnitude is more important than details, and the numbers between adjacent fiscal years generally differ by less than 10%.

The key figures that were used in the calculation was provided by IKEA personnel. The figures are:

- 3 833 357 700 - The total number of products sold during the fiscal year of 2016.
- 50 000 000 - Number of pallets transported during the fiscal year of 2017 (a rough estimate).
- 35 784 666 - Total m³ output from suppliers during the fiscal year of 2016.

The number of orders, consignments, and shipments that were created during 2017 was also needed in the calculation. These figures were derived from the raw data. The corresponding created events were summed up over one week and an average per day was calculated. This average was then multiplied with the number of days in one year, which might be risky since the data is not guaranteed to be representative for the rest of the year.

The ratio was then calculated between the base event shipment create and all other events. One important consideration was to represent the same relational structure between information object as they are structured within IKEA, as figure 3.2 shows. Even though all events are related to a physical product, it would be inefficient to update the same information regarding shipments, consignments etc. to each product affected. By instead using a hierarchical structure to represent the information objects, it is possible to link information objects in the system, and a single update would affect up to thousands of products.

Another aspect that affects the load is using single or double linking. In a single link strategy, a single update to an information object representing a pallet would be linked to the products
the pallet contains. The downside with this strategy is that it is not possible to see which pallet a product belongs to by accessing that product’s information object. It would require searching through each pallet to find which pallet contains the product. This problem could be solved by using double linking. However, using double linking would increase the load since the representation of the linkage between the same pallet and its products would require one transaction for the pallet and one transaction for every product. Given the performance results of previous work, reducing the amount of load was considered more important than searchability, thus single linking was used.

When calculating the amount of events that get produced per shipment create, events are divided into two groups: shipment events and product events. Product events include the events product creation, sell product and transfer ownership. Shipment events include all of the remaining events and transfer ownership. Both groups contain the transfer ownership event but in varying frequencies, since the event happens at multiple stages in the supply chain. The ratio between shipment events and shipment creation is correlated and these events are not affected by the structure of the logistics network. The ratio for these events (number of events per shipment) is presented in table 3.2. One notable thing here is that the event transfer ownership on average occurs 3036.67 times per shipment, as a result of having to transfer ownership for each product that is shipped and delivered.

<table>
<thead>
<tr>
<th>Event</th>
<th>Evt/SHP.C</th>
<th>Event</th>
<th>Evt/SHP.C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order creation</td>
<td>8.70</td>
<td>Check LU and seal</td>
<td>1.00</td>
</tr>
<tr>
<td>Consignment creation</td>
<td>1.59</td>
<td>Check delivery documents</td>
<td>1.00</td>
</tr>
<tr>
<td>Pallet creation</td>
<td>24.45</td>
<td>Close delivery</td>
<td>1.00</td>
</tr>
<tr>
<td>Schedule shipment</td>
<td>1.00</td>
<td>Unload unit load</td>
<td>24.45</td>
</tr>
<tr>
<td>Book transport</td>
<td>1.00</td>
<td>Close receipt</td>
<td>1.00</td>
</tr>
<tr>
<td>Transport confirmation</td>
<td>1.00</td>
<td>Scan of unit load at gate</td>
<td>24.45</td>
</tr>
<tr>
<td>Arrival for loading</td>
<td>1.00</td>
<td>Put and verify</td>
<td>24.45</td>
</tr>
<tr>
<td>Register delivery documents</td>
<td>1.00</td>
<td>Put and verify</td>
<td>24.45</td>
</tr>
<tr>
<td>Consignment dispatch</td>
<td>2.59</td>
<td>Receive assignment</td>
<td>24.45</td>
</tr>
<tr>
<td>Shipment start</td>
<td>1</td>
<td>Verify article</td>
<td>24.45</td>
</tr>
<tr>
<td>Register metadata</td>
<td>1.00</td>
<td>Verify destination</td>
<td>24.45</td>
</tr>
<tr>
<td>Consignment arrival</td>
<td>2.59</td>
<td>Close consignment</td>
<td>1.59</td>
</tr>
<tr>
<td>Shipment end</td>
<td>1</td>
<td>Transfer ownership</td>
<td>3036.67</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>3261.33</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Ratio between shipment create (SHP.C) and all other shipment events (Evt).

Product events are linked to the structure of the logistics network. This is mainly due to the fact that a product gets physically created only once regardless of the route it travels. Since a route can involve several shipments, it is not possible to directly link a product creation event with a shipment creation event. However, it is possible to calculate an average of how many products and product creation events exist per shipment, since it is known how many products and shipments are created during one year.

Based on the data used in this thesis, 2045 017 shipments are produced during the period of one year. This number was derived from the daily average number of shipment creation events based on the raw data. Given the number of shipments and products sold per year, the average number of product creation events per shipment creation event is 1874. The event product sold only happens at the IKEA store when a customer buys the product and consequently, a transfer ownership event also occur at this stage. Therefore, the same reasoning as with the product creation event can be applied for this event. This results in an average of 1874 for the product sold and transfer ownership events respectively. By summing the average number of product creation, product sold and transfer ownership events the total number of events for the product events can be calculated to 5622 events (1874 + 1874 + 1874). By adding the events
from the two groups, the total number of events per shipment creation event is obtained, which results in 8883 events (5622 + 3261).

With the average number of events per shipment known, it is possible to calculate the number of events per time unit by deriving the number of shipments per time unit. However, since the distribution of events is volatile, as illustrated in figure 3.3, it is not as simple as dividing the total number of shipments with the total number of time units during one year. When defining performance requirements, one aspect is guaranteeing that a transaction will not take longer than an arbitrary but given time unit. Due to the volatility in frequency of events, an average would not provide sufficient information about the peak values. Therefore, delay bounds are used instead of time units. A delay bound is a constraint that specifies the maximum allowed delay for an event transaction to be processed and finalized by the system.

The delay bound interval with the highest total number of event occurrences was identified in order to calculate which load (measured in events per second) will guarantee each delay bound. Since the raw data is presented on a minute by minute basis, the delay bound interval is iterated over the whole data set with one minute-increments to capture all possible intervals for a given delay bound. The average number of events is then calculated to make it easier to compare the load requirements of the different delay bounds.

Table 3.3 presents the load requirements, in events per second, the system must be able to process based on different delay bounds. When allowing a longer time for a transaction to be completed in the worst case, the load requirement decreases. In other words, the system must be able to process an increasing amount of transactions as the delay bound decreases. A delay bound interval from one minute to 1440 minutes (one day) was examined. Table 3.3 presents five representative delay bounds and their resulting load requirements.

<table>
<thead>
<tr>
<th>Delay bound [min]</th>
<th>Load requirement [events/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10 593</td>
</tr>
<tr>
<td>5</td>
<td>5 546</td>
</tr>
<tr>
<td>10</td>
<td>4 860</td>
</tr>
<tr>
<td>60</td>
<td>3 383</td>
</tr>
<tr>
<td>1 440 (24h)</td>
<td>921</td>
</tr>
</tbody>
</table>

Table 3.3: Calculated number of events per second that needs to be handled in order to guarantee different delay bounds.

In table 3.3 above, it is possible to see the impact the delay bound has on the required performance. A delay bound of 1 minute requires the system to handle 10 593 events per second. If the delay bound is increased to 5 minutes, the load requirement is reduced to 5 546 events per second. As seen in the graph the requirement decreases rapidly with an increased delay bound between 1 minute to 1 hour. Thereafter, increasing the delay bound does not affect the load requirement to an equal degree. This is because the load greatly differs throughout the day.
4 Requirements

Software requirements are a good way of understanding the fundamental problem that needs to be solved and the goals that need to be reached. Without properly formulated requirements, the risk increases that the end product does not meet the needs, does not function as expected, or is difficult for the users to understand and use [43]. In this chapter, the identified requirements will be presented as well as the method used to derive them.

4.1 Method for identifying requirements

One of the research questions for this thesis is: “What requirements exist for a blockchain-based traceability system in the IKEA Supply Chain”. An answer to this question includes formulating performance requirements so that the system performance can be evaluated in relation to the goal. However, other types of requirements are also of interest, since they define the functionality and capabilities of the system in a clear way so that the performance requirements can be seen in the right context. The research question also provides context for answering the second research question: “How can a traceability system be designed and implemented on Quorum?”.

Requirements are definitions of what behavior is desired, but they do not answer how that behavior should be obtained [43]. The process of defining these requirements involved interviewing key personnel at IKEA and a literature study on the topic. Since this thesis aims at creating a prototype, the focus has been on the fundamental requirements and the detailed requirements have been omitted.

The interviews were carried out in an informal manner and consisted of discussions about how the system would be used. It often produced a more detailed discussion of how blockchain specific features affect the use and desired functionality.

The literature study mostly revolved around identifying similar blockchain projects [55, 34, 24, 28] as well as projects [26, 20] that aimed to solve similar problems but with conventional technologies. In the case of blockchain projects, the requirements that relate to blockchain specific features were of interest.
4.2 Identified requirements

Based on the discussions with personnel at IKEA, 14 requirements were identified. These requirements are presented and motivated in this section.

1. The user shall be able to create information objects that represent real world assets.
2. The user shall be able to register arbitrary information to an information object in the system.
3. The user shall be able to control which other users have read access to information registered by the user.
4. Each information object shall have an owner.
5. The owner of each information object shall be able to add and/or remove users that have write access.
6. The owner of an information object shall be able to transfer the ownership of that object.
7. The information object shall be verifiable with respect to its creator.
8. Information that has been registered in the past shall be removable.
9. The user shall be able to change which information that is relevant for an event.
10. All users shall be able to see if a user changes some information relevant to an event.
11. The system should provide access control on each state altering action performed.
12. Confidential data stored on the system shall be encrypted.
13. The users private key shall only be used on the client-side.
14. The system should be able to handle all continuously generated events in a timely manner.

Since the aim is to build a traceability system, most of the requirements revolve around the notion of a system that should be able to register information about entities. Thus, many of the listed requirements are more or less self-evident in the way they have been formulated. Requirement 1 and 2 represent functionality that is required for the system to be useful for the intended users. Requirements 3, 4, 5, 6, and 7 are all defined with a legal utility in mind. As expressed by Gipp et al., blockchain allows tamper-proof and trustless timestamps [18]. This is one of the main reasons to why blockchain enables these new traceability systems. This was something that was apparent from the discussions with IKEA personnel, as well as from a review of earlier work [55, 26, 20]. The ability to verifiably determine who has been, and who is the current owner of a certain entity is vital in legal situations. In the same way it is equally important that changes to the entity should be verifiable and traceable back to the person responsible for the change. To create this capability in a decentralized manner is seen as one of the main benefits of using the blockchain for this kind of system.

Requirements 8 and 9 are a bit controversial in the context of blockchains since these are functionality that cannot be offered directly on the blockchain since it is immutable. However, these are still necessary in certain circumstances, and therefore cannot be left out. In regards to 8, GDPR and situations where e.g. compromised private keys were used to encrypt the information require the ability to remove information. Requirement 9 is apparent since mistakes, human errors, and bugs still are a reality. One should be able to correct faulty information, something that is expressed in the work of Lützenburg [34] as well. How each functionality is achieved is presented in chapter 5.
4.2. Identified requirements

In regards to requirement 11, the interviewees expressed that access control is of great importance. The system is intended to register information that have legal significance. It could be devastating to let anybody be able to make changes to information, which could end in significant economic damage or worse.

Requirement 12 was included since the system is more or less open to actors that have privilege to certain processes, but not others. Requirement 13 is connected to 12 since it makes the system less vulnerable to man-in-the-middle attacks. It also makes the platform more dynamic to different key-management schemas.

The formulation of requirement 14 can be seen as ambiguous, however, that is intended. In our discussions with IKEA personnel, it became clear that all events do not have to be processed within the same time span. Different events would therefore have different delay bounds. Some events, such as creation events do not have to be immediately processed. A delay bound of a couple of minutes is acceptable. On the other hand, some events (such as event #18, close delivery) need to be processed as fast as possible so the truck driver can continue with his next assignment. All this means is that "in a timely manner" is unique to the specific event and that no general delay bound can be given without making the requirement unnecessarily demanding or too weak.
This chapter presents an overview of the system and its components, an introduction of each component and how they interact with each other. A detailed description of each component is then presented and the chapter is concluded with a description of the performance requirements the system design results in.

5.1 Blockchain choice

A complete traceability system consists of several parts. The novelty in this thesis is the use of a blockchain as the base layer. Thus the choice of which blockchain technology to utilize and how to use it, becomes central for the whole system.

5.1.1 A permissionless or permissioned blockchain

The traceability system developed in this thesis should be able to be public at some point, since one of the perceived use-cases is to allow the public to verify a product’s origin and production process. Furthermore since the system is supposed to process critical information with significant economic and legal value, the consensus process becomes critical. The use of a permissionless blockchain technology is possible. While the public Ethereum blockchain is more decentralized, the performance is no way near the figures needed and additionally every transaction has to be paid for, which would be very costly. To start a new permissionless blockchain cannot effectively be argued for either. Depending on the consensus algorithm itself, the open nature of a permissionless blockchain would make it prone to attacks since malicious actors might be able to obtain a majority relatively easy. The use of a permissioned blockchain on the other hand would result in a secure blockchain with better prerequisite for higher performance since fewer validators can result in higher performance [12].

One feature is somewhat lost with a permissioned blockchain, namely the ability for the public to validate and trust the information, which is one of the main reasons for using the blockchain in the first place. If nobody outside the group of permissioned validators records all activity on the blockchain throughout time, no one would be able to prove that something in that blockchain’s history has been mutated by the permissioned group. For example, the group realizes that a product contains toxic material, and the proof of that fact is present
on the blockchain. The permissioned group decides to remove that information from the blockchain and rebuild a new valid chain, effectively removing the proof. If outside actors store the history of the blockchain the malicious act can be detected. However, the public still need to trust that outside actor is not malicious in of itself. Fortunately, this problem can easily be avoided by utilizing an established public blockchain such as Ethereum or Bitcoin as an anchor. Every hour or so the current blockhash (a cryptographic link between the blocks) of the permissioned blockchain can be recorded in the public blockchain with a simple transaction, effectively anchoring it in the public chain.

5.1.2 Desired capabilities of the consensus algorithm

The next choice to make is which blockchain platform and consensus algorithm to use. Since we want a permissioned blockchain that is performant, cheap to run, and most of all secure, a BFT-capable consensus algorithm that support as many validators as possible to maintain the decentralized nature of the blockchain, seems to be the best fit.

Before we can decide upon which BFT-algorithm to use we need to choose our preference in terms of its capabilities. Based on the CAP theorem, which two properties do we want of consistency, availability and partition tolerance. However, as becomes clear, partition tolerance is almost unavoidable, thus the phrase "two out three" becomes misleading. In a blockchain design the different nodes are located on different geographical locations often with a significant distance between them, simply a power outage at the wrong spot could partition the network. With partition tolerance given we have consistency or availability to choose from. Although the choice between them is more nuanced than simply giving up one for the other a preference has to be made. The choice landed on consistency for this system since it seems logical that inconsistencies would be detrimental to a traceability system that aims to provide information that should be final.

5.1.3 Final choice

Based on these preferences (a public, permissioned, BFT capable blockchain that is partition tolerant and provides consistency) the choice landed on the Istanbul Byzantine Fault-Tolerant (IBFT) algorithm. With the choices made the IBFT algorithm was the only implemented algorithm that was available on a smart contract enabled platform to our knowledge. It is important to note that the IBFT algorithm and implementation have not been peer-reviewed and could thus contain weaknesses and errors. However, the algorithm is included in the Quorum-platform, one of the most popular platforms for permissioned blockchains. IBFT is also offered as an option by Kaleido.io, who sells enterprise blockchains as a service.

IBFT is an active replication, multi-primary, state machine replication algorithm that is inspired by the Practical Byzantine Fault Tolerant (PBFT) consensus algorithm, presented in the paper by Castro and Liskov. IBFT inherits the 3-phase consensus process from PBFT, and works under the eventual synchrony assumption which guarantees safety and liveness, as long as two thirds of the network are available. The system can at most tolerate $f$ faulty nodes in a network of $n$ validator nodes, where $n = 3f + 1$. Blocks are final in the IBFT protocol, which means that when a block is processed and accepted it cannot be changed, which is one part to why IBFT delivers the consistency attribute.

As mentioned earlier, IBFT is included in Quorum, which is a permissioned and enterprise-focused version of the Ethereum blockchain. They use the official Ethereum node software Geth as their basis for their software. By doing this they maintain compatibility with tools and other helpful software developed in the Ethereum community.
5.2 System architecture

Designing a blockchain-based traceability system requires careful consideration about the design decisions being made. It was concluded that with a delay bound of one to five minutes, on average 5 546 - 10 593 events per second in IKEA Supply Chain need to be handled. While events per second does not correlate exactly to transactions per second (tps), a comparison can still be made to the performance evaluation by blockchain developer group AMIS [31]. They got an average of 821 tps when they tested the capacity of IBFT, the consensus algorithm used in Quorum for this thesis. Although one important difference is that they used normal transactions in their tests, and this system only utilizes smart contract transactions, which are more computationally expensive. The highest throughput achieved, based on previous work, was 1900 tps [3]. That would infer a delay bound of roughly 10 hours if events per second would directly be translated to tps. Based on those results, it was apparent that a system based on Quorum and IBFT would not be able to handle the performance requirements identified earlier and throughput had to be carefully considered throughout the design process. Since it is apparent that performance will be an issue for the system, and some events require a lower delay bound than others, being able to prioritize transactions would prove useful.

The system designed for this thesis consists of six components, which are illustrated in figure 5.1. The client application, controller, smart contract pool and smart contracts were designed for this thesis. It was implemented with JavaScript and Node.js as the runtime environment.

![System Architecture Diagram](image)

**Figure 5.1:** Overview of the system architecture. The arrows depict how requests can be sent

- **Client application:** This is the application the users interact with. The client is responsible for hashing and encrypting event data, and uploading it to IPFS. It is also responsible for creating and signing transactions.
• **Controller**: A service layered between the client and Quorum. The controller is used to schedule and order transactions, balance load and provide error handling.

• **Smart contract pool**: A service that deploys smart contracts to the blockchain and stores the addresses of these contracts.

• **IPFS**: Off-chain storage, a distributed file system that stores data. Is connected to a network of IPFS nodes.

• **Validator**: The validator runs the blockchain node software, in this case Quorum. The validator interacts with the smart contracts used in the system. The validator is connected to several other validators to create a blockchain network. The validator validates incoming transactions and commits them to the blockchain network for processing. The validator takes part in the consensus process in the blockchain network.

• **Smart contracts**: There are two types of smart contracts: a smart contract representing information objects and a smart contract representing ownership and contributors of information objects. The first smart contract is used for registering events and providing read access control. The second is used to validate and/or transfer ownership of products and providing write access control.

The network can include thousands of client applications since each user would have the client application on their smartphone or computer. The client application would then send transactions to a controller. In order to increase decentralization, each company participating in the network would be able to have their own controller, IPFS and Quorum node. Figure 5.2 illustrates how the components interact with each other.

A smart contract can represent an information object (e.g. order, shipment or product). When one or more information objects are created, the client requests empty contracts from the smart contract pool, which satisfies requirement 1. The smart contract pool, which is the current owner of the smart contracts, transfers ownership to the user that requested it and returns the contract addresses to the client. When an event occurs in IKEA Supply Chain and the client receives or creates it, the event data is hashed and encrypted with a unique symmetric key. The symmetric key is then encrypted one or more times using the public keys of each user with read-access to the event data. The client application creates two data objects, a sensitive data object and a public consumer data object, and uploads them to IPFS.

The client requests a status update from the controller to determine what nonce to use before sending a new transaction. A nonce is a number that defines the ordering of transactions. When the controller receives one or more transactions, it places them in a scheduler. Later on, depending on the transactions priority and current blockchain load, the transaction is removed from the scheduler and sent to Quorum. Once the transaction has been included in a block, it is considered finalized and the response is returned to the client.

### 5.3 Validator

Transactions sent to the validator are stored in a transaction pool, until they can be included in the next block. A transaction pool is a set of data structures inside the Quorum software that stores pending transactions until they are validated and included in a block. Transactions in the pool are prioritized based on gas price. Since Quorum is a permissioned blockchain, economic incentives are not necessary and gas price could be omitted if desired. However, in this thesis gas price is not omitted, since it is desirable to prioritize transactions when they have been sent to the validator. Transactions from a single sender are released from the pool and finalized in sequential order based on the nonce, which implies that a sender’s transaction with nonce \( n + 1 \) cannot be finalized before the transaction with nonce \( n \).
5.4. Smart contract and off-chain storage

The pool can only hold a finite number of transactions. If higher gas price transactions fill up the pool, lower gas price transactions will be discarded. To mitigate this problem, the controller observes the pool on the local Quorum node and stops sending transactions when the pool is close to being full.

Quorum has built-in functionality for private transactions. However, this functionality is not utilized in the system described in this thesis since all data is stored off-chain. This infers that every Quorum node in the network has a copy of the same state and any changes made will be visible to each participant in the network, fulfilling requirement 10.

5.4 Smart contract and off-chain storage

Since all public data stored on Quorum is replicated across every node in the network, the smart contract was designed to store as little data as possible. The smart contract was also designed to be as generic as possible in order to fulfill requirement 2, since it may not be known which events exist for specific information objects or what data is required. In addition, storing less data in the smart contract provides better performance, since transactions require less data.
5.4. Smart contract and off-chain storage

5.4.1 The chosen design

Figure 5.3 provides an illustration of the data stored in the smart contract and IPFS respectively, as well as the relationship between them. The client uploads two objects to IPFS — a data object that contains sensitive event-related data and a consumer data object that contains unencrypted non-sensitive data that consumers can access. In more detail, the sensitive data object contains:

- **Encrypted data**: The event data, which has been encrypted with a unique symmetric key.
- **Keys**: A list of blockchain addresses associated with users that have read-access to the event data as well as a corresponding encryption of the symmetric key, which can be decrypted by the private key of the user associated with it.

The smart contract contains a list of events, where each event contains:

- **Sensitive data**:
  - *Data hash*: A SHA-256 hash of the sensitive event data.
  - *IPFS hash*: An IPFS hash referencing an IPFS object that contains encrypted data and address-key pairs.

- **Consumer data**:
  - *Data hash*: A SHA-256 hash of the consumer event data.
  - *IPFS hash*: An IPFS hash referencing an IPFS object that contains unencrypted data.

Two hashes, for the sensitive data and consumer data respectively, is immutably stored in the smart contract in order to fulfill requirement 7, *information shall be verifiable*. Once the hash is stored in the smart contract, all actors in the network have a copy of the hash and it cannot be corrupted, modified or removed. When a user downloads and decrypts the event data at a later stage, the data can be hashed and the resulting hash can be compared to the hash stored in the smart contract — validating the data integrity.

A file stored on IPFS is referenced by a prefix, indicating which algorithm used, followed by its hash [23]. This implies that if changes were to be made to that file, the IPFS hash, and hence the file reference, would change. Since it may not be known upon event creation which
users require read-access to the sensitive data, it may be necessary to update the data object containing the encrypted data and symmetric keys. When a user requests read-access to an existing event, the user providing read-access decrypts the symmetric key and re-encrypts it with the public key of the user that made the request. The new address-key pair is added to the data object and uploaded to IPFS. The IPFS hash in the smart contract is then updated to correctly reference the updated data object. When removing read-access for a user, a new unique symmetric key is generated and the event data is re-encrypted. The symmetric key is then encrypted one or more times with the public keys of the users that still have read-access, and the sensitive data object is updated and its IPFS hash is added to the smart contract. This process ensures that requirement 3, distributed access control, is fulfilled. Storing data on IPFS, which is not an immutable storage, fulfills requirement 8. However, since IPFS is a distributed file system, data may be redundantly stored across multiple IPFS nodes and internal processes may be necessary to make sure that the data is removed from all nodes that hold a copy of it.

Figure 5.4: Relationship between a product smart contract and the ownership contract

As can be seen in figure 5.4, each smart contract has an owner who is the only one with the privilege to transfer ownership and remove contributors. This fulfills requirements 4, 5 and 6. Since each product is represented by a smart contract and there are roughly four billion products sold per year, ownership transfers for products would amount to billions of transactions per year if done once or more per smart contract. To mitigate this, ownership transfers are done in batches. An owner of one or more products can send a list of addresses to an ownership and contributors smart contract, and a user address they wish to transfer ownership to. After the smart contract validates that the sender is the current owner of all addresses included in the transaction, the smart contract is updated.

The contract also contains a list of contributors to provide access control to the information object smart contracts (e.g. product smart contract), which fulfills requirement 11, state change access control. Contributors are users that have write-access to certain functions in the smart contract, e.g. register event. Contributors can also add other users as contributors, but cannot remove any. When a user registers an event in a smart contract, a validation is performed by calling a view function in the ownership and contributor smart contract to ensure that the user is a contributor.

5.4.1.1 Alternative design

In comparison, owner and contributors could be stored in the smart contract representing each information object. However, this has proven to be more inefficient than the current solution. Comparing the gas usage, which is an abstraction of how computationally expensive
5.5 Client application

When the client receives or creates an event, the event data is hashed. The event data may contain sensitive information and since IPFS is a distributed file system, anyone that has an IPFS address of a file can retrieve it. The sensitive event data is therefore encrypted with a unique symmetric key before being uploaded to IPFS, fulfilling requirement 12. Symmetric key cryptography is used to store as little data as possible since the event data only needs to be encrypted and stored once, as opposed to encrypting the event data and storing it once for each public key. The symmetric key is then encrypted one or more times with the public key of each actor that has read-access to the event data. A user with read-access can use their private key to decrypt the symmetric key, which can be used to decrypt the event data. If a user wishes to alter or update information for an event, as required by requirement 9, it must be registered as a new event in the smart contract.

To call a function in a smart contract, a transaction is created. For example, the register event function requires a data hash, a consumer data hash and IPFS addresses to where the data objects are stored. The transaction is then signed by the users private key. In order to fulfill requirement 13, the user’s private key shall only be used on client-side, transaction creation and signing is done by the client. The IPFS addresses of the data objects are calculated before being uploaded to IPFS. The client therefore does not need to wait until the data objects have been uploaded to IPFS before sending the transaction to the controller.

Before sending a transaction \( t \), the client checks active transactions (transactions that have been scheduled by the controller, but not yet finalized) to determine if the gas price of transaction \( t \) is higher than any active transactions. If any active transactions have a lower gas price than \( t \), the client requests a transaction update from the controller. The nonces of all transactions \( t_i \) with gas price \( g(t_i) < g(t) \) are incremented and then the nonce of transaction \( t \) is set to \( \min(t_i) - 1 \). This is done to mitigate the problem that even though transactions should be prioritized based on gas price, Quorum processes transactions from a single sender in the order of the nonce. The client then signs all transactions and sends them to the controller for
5.6 Controller

Based on the requirements, sensitive data has to be hashed and encrypted, and transactions have to be signed on the client-side. Since there are limitations to Geth (and thus Quorum) when handling large numbers of transactions, a service, layered between the clients and Quorum, provides useful functionality. This service, a controller, queues and prioritizes transactions and provides functionality to mitigate the limitations of Quorum. The controller also provides functionality for prioritizing transactions, which is necessary when the load on the system is high and certain transactions require a lower delay bound than others.

The controller’s task is to schedule and differentiate treatment of transactions, and to ensure that transactions are finalized on the blockchain. Performing health checks for transactions on the controller also has the benefit that it enables clients to be offline without risking transactions to fail and not be resubmitted. Each transaction received by the controller is placed into a scheduler object where each key references a client’s id and the corresponding value contains an array of transactions sent by that client. Each transaction contains four values that are of importance to the scheduler:

- **Nonce**: A sequential number that specifies how many transactions the sender account has made on the network.
- **Gas price**: Gas price of the transaction.
- **Timestamp**: A timestamp of when the transaction was scheduled.
- **Transaction signature**: The signed transaction that will be sent to the blockchain.

The array is sorted by nonce in ascending order first and then by gas price in descending order.

To be able to prioritize transactions, the client can request a transaction update from the controller before sending a transaction. When the controller receives the request from the client along with a gas price $g$, the array containing transactions is iterated. All transactions with gas price $g_i < g$ are removed from the array and sent back to the client so their nonces can be incremented and the transactions can be re-signed. The new transaction, which has a higher gas price and therefore priority, is given the lowest nonce. When the controller receives the transactions again, along with the new transaction, the array is sorted by both nonce and gas price.

Transactions are sent from the scheduler based on gas price first and timestamp second, provided that nonces from a single client are in sequential order. How often the scheduler sends transactions depends on how large the transaction pool is, i.e. how many transactions are pending on the local blockchain node. When a transaction is removed from the scheduler, the signed transaction is sent to the blockchain in an asynchronous call. If the controller receives a successful response from Quorum, the response is forwarded to the client. If a failed response is received (e.g. the client does not have access rights to register an event in the smart contract), an error message is returned to the client.

5.7 Smart contract pool

Since each information object requires a smart contract to register the events in, there is a large number of contracts that needs to be deployed on the blockchain. These deployments transactions are in addition to the event registrations. Therefore, the number of transactions that
would be needed to be processed each second is higher than the figures presented in chapter 3. More specifically, it would increase the number of transactions by 32.6% (on average 2060 contracts are needed per shipment). Additionally, the increased gas usage per second would be larger since a contract deployment transaction is significantly more expensive than registering an event.

Our solution to this problem is the introduction of the smart contract pool. By continuously deploying contracts during low load hours, a buffer of empty contracts can be produced. This means that the added load created by the contract deployment can be avoided. One important thing to mention is that this solution is only viable because of the change in load throughout the day. If the load would have been constant, the smart contract pool would have no value.

The distribution of events over a week, described in section 3.3, shows that the load is very high during specific times of the weekdays. Deploying smart contracts on the go might not be viable, since deploying contracts is an expensive process (as it consumes more gas) and it is not possible to guarantee the delay bound of a contract deployment. The smart contract pool was introduced to mitigate this problem. Smart contracts are continuously deployed to Quorum using low gas price transactions. Deployment transactions are created and signed in the pool, and sent to the controller. Once a smart contract has been deployed, the smart contract address is returned and stored in the pool. If the pool is running low on smart contracts, the gas price could be temporarily increased, giving deployment transactions a higher priority.

A user can request an arbitrary number $N$ empty contracts from the pool. When this request is received by the pool, a transaction to update ownership of the contracts is created and sent to the controller. The design and use of an ownership contract implies that a user request for $N$ contracts only results in one transaction.

5.8 Transaction capacity requirement

There are several aspects of the system design that affect the total performance of the system. One significant part is the choice to use and enforce ownership directly in the contracts. This means that with every legal ownership transfer in the real world, the transfer ownership function needs to be invoked on the blockchain. These legal ownership transfers are not generally seen as explicit events all the time. For example when the event sell product takes place, the transfer ownership function is invoked in a separate transaction at the same time. If one would have chosen to represent legal ownership in another way, it might have been possible to reduce the need for the transfer ownership function call. However, as just explained, it is necessary to represent ownership directly on the blockchain if one wants to have access control on the blockchain.

This results in a drastic increase of gas usage. On average 8 660 transfers of ownership takes place per shipment: once at the producer of the product, once when it arrives at a distribution center or consolidation point, once when it arrives at the store, and at last once when the customer buys the product (see figure 3.1). The gas cost of a batched transfer of ownership is around 80 000. However, with only one ownership transfer per transaction the gas cost is 101 000. A register event transaction costs around 203 000 gas. To remain conservative in regards to the system’s load requirement, an average of two ownership transfers per transaction was decided upon. This means that the average transfer ownership transaction and register event transaction is more or less equivalent in terms of gas cost.

Next we have the contract deployments. However, its impact on the needed performance during peak load can be ignored.
5.8. Transaction capacity requirement

<table>
<thead>
<tr>
<th>Delay bound [min]</th>
<th>Load [tps]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14975</td>
</tr>
<tr>
<td>5</td>
<td>7840</td>
</tr>
<tr>
<td>10</td>
<td>6871</td>
</tr>
<tr>
<td>60</td>
<td>4782</td>
</tr>
<tr>
<td>1440 (24h)</td>
<td>1302</td>
</tr>
</tbody>
</table>

Table 5.1: Estimated number of transactions per second that the system must be able to handle in order to guarantee different delay bounds.

Based on these arrangements the resulting transaction capacity requirement (see figure 5.1) for the system, in terms of transactions per second, becomes a quite bit higher than the information flow estimation presented in chapter 3. It also becomes significantly higher than the performance figure from earlier evaluations. Not even with the most optimistic view of a throughput of 1 900 tps [3] would the system be able to guarantee a delay bound of 24 hours for all transactions. However, these performance evaluations have been conducted on systems and setups that differ from this system. Exactly how the system developed here performs, will be evaluated in the next chapter.
This chapter presents an evaluation of the system that was designed. First, the method used for evaluating the system is presented, followed by a presentation of the selected parameters and measurements. The testing environment is then described, and finally the test results are presented.

6.1 Method

In order to decide to which degree the system reaches the identified performance requirement, the system needed to be evaluated and reasoned about. The evaluation was decided to be carried out through extensive testing. As formulated in chapter 4, the performance requirement evaluated is the following:

14. The system should be able to handle all continuously generated events in a timely manner.

By solely testing for the performance requirement above, the evaluation only touches upon the commit process of creating and processing a transaction and not the process of retrieving information registered in the system. The information retrieval process is not as interesting as the commit process since the unknown performance factors are fewer and rely on more conventional solutions. For example: there are no consensus process needed when retrieving information.

The best possible test to be carried out would be an exact simulation of the live environment the system is planned to be used in. However, in the context of this master’s thesis this would be too time consuming and expensive. Just the processes of identifying what a live environment would constitute in detail is outside the scope of this thesis. Thus, the tests need to be simplified, which means that testing results can only be indicative of how the system would perform in a live environment. With that said, the test method and environment has been carefully thought through to produce results that are as indicative as possible given the confinements of this thesis.

The chosen approach was to create a test environment where parameters, that were thought to impact the performance, could be controlled and where the environment itself was as close
as reasonably possible to a live environment. The environment was used to find the maximum performance and how the different parameters affect the performance of the complete system. When the effect of the different parameters were better understood a potential live environment setup was created. This potential setup was then tested, and that result is what has been used to answer the answer the question if the system reaches the performance requirement presented above. The complete suit of tests carried out is used to answer the third research question from chapter 1: “What are the performance aspects and bottlenecks for such a system?”.

6.1.1 Performance metrics

The performance metrics used are: average throughput, measured in transactions per second (tps), and average total transaction processing time (ttpt), measured in seconds. Throughput refers to how many transactions the system can handle per second. Total transaction processing time refers to the time it takes from the point when a transaction is created on the client until it is included and finalized on the blockchain. It is worth noting that transaction processing time is not the same as the latency presented in most performance evaluations [32, 3, 7, 42, 12]. In these works the latency refers to the time it takes for the blockchain network to process the transaction. The total transaction processing time includes the latency, but also the time it takes for the client to create the transaction and how long it takes for the controller to process it.

The average throughput is calculated by summing all the transactions that get included in each block and dividing it with the time from the start of the test until the last transaction is processed. The average total transaction processing time is obtained by adding each individual transaction and total transaction processing time together, and then divide it with all transactions sent during the test (failed transactions are excluded).

The metrics measured, average tps and average ttpt, are inversely related to a high degree, but not completely inversely correlated. The system could potentially be slow to process transactions but still be able to process multiple transactions in parallel, resulting in a high total transaction processing time and high tps.

6.1.2 Measurement approach

Since the system was built in such a way that the storage is separated from the blockchain, the different processes can be run in parallel on different machines. In order to decide what the actual total transaction processing time is, it is necessary to compare the time for the parallel processes. The total transaction processing time was calculated as $ttpt = tct + tqt + tpt$:

- **tct**: transaction creation time on the client-side
- **tqt**: transaction queuing time, waiting for the scheduler (controller)
- **tpt**: transaction proposal time, sending transactions to Quorum and waiting for processing and response

Additionally, the processing time on IPFS is not as critical as the processing time on Quorum, since it is the registration of an event in time on the blockchain that is the central factor in giving the blockchain-based traceability system its value. With the time and cost constraints in mind, only simpler tests were carried out for the storage module, e.g. IPFS.

Since there are multiple parameters that could be of interest in the tests, the potential number of tests that could be run to exhaust all possible combinations was insurmountable. Thus, a carefully selected set of tests was initially produced based on theoretical reasoning. Some
of the tests were carried out in an attempt to confirm or refute the theoretical hypothesis. The tests were then updated based on those results. Thereafter, the complete set of tests was carried out followed by an analysis in order to understand and present the results in the best way.

A test is defined as a round of 10 minutes where the system is exposed to a constant load generated by the client. The time period was decided upon after some initial tests revealed that the longer the system was exposed to the load, the higher would be the chance of the blockchain network stalling. The same problem was presented in the presentation of Istanbul BFT produced by AMIS [32]. At the same time, the test could not be allowed to be run too long due to cost and time constraints. A period of 10 minutes was decided to be a good compromise, it was also the chosen test length made by AMIS in their performance evaluation [32]. Why exactly the system stalls is unknown and not analyzed in any depth in this thesis work. However, it seems to be linked with the saturation of processed transactions in the blockchain networks nodes. Nevertheless, the probability of failure and performance degradation decreases with time. Thus, a performance figure resulting from a 10 minute test is probably not the exact figure one would obtain in a system running 24/7. However, based on our initial tests it is believed to be much closer than a figure produced by a 1 minute long test.

![Figure 6.1: Example of a validator’s CPU load during test with high tps load which leads to failure](image)

### 6.1.3 Choosing the load in tests

Due to the stability issues experienced when high load was produced, the test process was not as simple as running one round for every test setup. Since Quroum would stall and cease to process new transactions on higher loads, several rounds with different loads were needed to be carried out in order to arrive at a maximum load for the setup in question. Additionally, it would reveal itself that a completed round where all transactions were processed without errors did not necessarily mean that the system could be deemed stable. As illustrated in figure 6.1, unstable behavior could be seen in the CPU utilization which would start to become volatile, as opposed to a slightly lower load that resulted in a more stable run with lower volatility, as seen in figure 6.2. If the unstable test would have been run a couple of minutes longer it would very likely stall at some point.

A search method for finding the maximum stable load within 10 tps from the actual maximum load was introduced. Because of the search algorithm, it was possible to decide the maximum loads within an average of 5 test rounds a 10 minutes. In total, nearly 400 test
rounds were conducted with the method presented above. However several more test rounds preceded these in order to arrive at a satisfactory test method.

6.2 Parameters

The objective of the evaluation was to test the performance of the complete system. The parameters used in this thesis were identified by going through earlier work and reasoning which parameters would be necessary to reach the goal of identifying performance aspects and bottlenecks. The chosen parameters are listed and motivated below:

1. The number of validator nodes
2. The number of client nodes
3. The number of controller nodes
4. Node hardware
5. Network latency
6. Block time

As can be seen in earlier work [32, 12] the number of validator nodes that take part in producing new blocks - the consensus process - affect the performance of the blockchain network. This is perfectly reasonable since increasing the number of nodes also increases the amount of data that needs to be sent around and processed before consensus can be reached. To which degree this affects the performance is also dependent the consensus protocol itself. The expectation was that the performance will decline as the number of validator nodes increase. According to the developers of IBFT, a clear performance decline will appear between 20 to 30 nodes [32]. The rate of decline after that size was not further elaborated on in their results.

The number of client and controller nodes were chosen as a parameter with the goal to understand how the controller reacts to different configurations, in terms of throughput and tptt. The interesting aspect to explore with regards to clients and controllers, is the increased overhead when more clients are connected to a controller and when more controllers are
connected to a validator node. If the controller is able to handle a high degree of clients simultaneously and the validator node can handle several controllers, the system would be more cost effective. The expectation here is that there will be some decrease in performance as the number of nodes increases due to the increased overhead to handle every unique node.

The node hardware parameter is applicable to all different node types. It is believed to have an especially significant effect on the validator node performance based on other tests and studies \[12, 42, 7\]. With hardware, aspects such as CPU-, RAM-, and disk performance are of interest and are believed to impact the performance to a varying degree. In previous studies, CPU seems to be the most important factor \[12, 42, 7\].

Network latency and network bandwidth affect the rate with which inter-node processes can be carried out. It was expected that the performance would decrease with added latency and reduced bandwidth. The interesting part was with which rate the performance changes in relation to changes of the parameters.

Lastly, block time was chosen as a parameter to get a better understanding of the impact it has on the performance. Earlier evaluations \[3\] have shown that a higher block time results in an increased latency for the transactions, but no significant change in throughput. The default block time used in the tests was 1 second, since it is the lowest possible setting when using IBFT \[31\] and a low convergence time is desirable.

### 6.3 Environment

We created a test environment that was as close to a live environment as possible within the scope of this thesis, while at the same time providing the tools to alter the given parameters. Amazon Web Services (AWS) was chosen and more specifically their Elastic Compute Cloud (EC2) service. EC2 provides the ability to spin up virtual machines (VMs) in a simple and efficient manner through their API. There are several different instance types, in terms of hardware configuration, to choose from. The instance types that were used in this thesis are listed in table 6.1. In the table each instance types hardware specification is listed.

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>vCPU</th>
<th>Clock Speed (GHz)</th>
<th>RAM (GB)</th>
<th>Network Performance (Mbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2.nano</td>
<td>1</td>
<td>2.3-2.4</td>
<td>0.5</td>
<td>Low (30*)</td>
</tr>
<tr>
<td>t2.small</td>
<td>1</td>
<td>2.3-2.4</td>
<td>2</td>
<td>Low to Moderate (130*)</td>
</tr>
<tr>
<td>t2.medium</td>
<td>2</td>
<td>2.3-2.4</td>
<td>4</td>
<td>Low to Moderate (250*)</td>
</tr>
<tr>
<td>c5.large</td>
<td>2</td>
<td>3.0-3.5</td>
<td>4</td>
<td>Up to 10 Gbps (740*)</td>
</tr>
<tr>
<td>t2.large</td>
<td>2</td>
<td>2.3-2.4</td>
<td>8</td>
<td>Low to Moderate (510*)</td>
</tr>
<tr>
<td>c5.xlarge</td>
<td>4</td>
<td>3.0-3.5</td>
<td>8</td>
<td>Up to 10 Gbps (1240*)</td>
</tr>
<tr>
<td>t2.xlarge</td>
<td>4</td>
<td>2.3-2.4</td>
<td>16</td>
<td>Moderate (740*)</td>
</tr>
<tr>
<td>c5.2xlarge</td>
<td>8</td>
<td>3.0-3.5</td>
<td>16</td>
<td>Up to 10 Gbps (2490*)</td>
</tr>
<tr>
<td>t2.2xlarge</td>
<td>8</td>
<td>2.3-2.4</td>
<td>32</td>
<td>Moderate (990*)</td>
</tr>
</tbody>
</table>

Table 6.1: The used AWS EC2 instance types. vCPU stands for virtual CPU and is equal to one thread. *\[57\]
same time providing more choices when it comes to altering network parameters and hardware configuration.

An automated deployment system was developed in order to streamline the testing process. Manual deployment and configuration would be very time consuming and in turn very costly, since AWS EC2 is a pay-to-use service. Terraform[1] together with custom made bash scripts, was used to create and deploy the instances and install the necessary software on EC2. Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. The type of instance, e.g. hardware, and the number of nodes that would be used for the different node types was decided in this infrastructure deployment phase. Table 6.2 shows the software components used and their version numbers.

<table>
<thead>
<tr>
<th>Component</th>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Ubuntu</td>
<td>16.04.2 LTS</td>
</tr>
<tr>
<td>Client and controller</td>
<td>Node.js</td>
<td>8.13.0</td>
</tr>
<tr>
<td>Blockchain</td>
<td>Quorum</td>
<td>2.0.2</td>
</tr>
<tr>
<td>Blockchain interface</td>
<td>Geth</td>
<td>1.7.2</td>
</tr>
<tr>
<td>Distributed file system</td>
<td>IPFS</td>
<td>0.4.17</td>
</tr>
</tbody>
</table>

The next phase consisted of configuring and launching the system. In this phase, the topology of the network was created, that is, which clients would connect to which controllers and which controller would be connected to which validator node. In all the tests, every validator was connected to every other validator. This is not necessary when using the IBFT consensus algorithm [31], but with the size of the network tested here, it was deemed very plausible that one would do that even in a live environment, since all actors would be known. Start-up parameters for Quorum and IPFS were also set during this phase (see appendix A for details). This configuration and launch was achieved through custom made scripts.

Next, network latency was fixed. The latency was set to be consistent across all instances. The latency was changed using `netem`[2], a package in Bash used for network emulations. Finally, the last phase was to initiate and execute the tests. During this phase the rate at which the clients would send transactions was set. Which clients would connect to which controllers was also set by using bash scripts. Since the test initiation takes some time, the user was given the option to set a time when the actual test should start so that all client nodes start to send transactions at the same time. The test duration was also set during this phase.

6.4 Results

There were many tests to run in order to achieve an understanding of how the different parameters impact the performance metrics. Some tests were dependent on other tests when determining the “optimal” configuration while varying a certain parameter. The general approach was to run the system under conditions that were believed to be the best for the given scenario and then change the parameter that was being studied. Doing that makes it possible to evaluate how the parameter affects the whole system and reduces the risk of interference by other potentially limiting factors. The resulting average throughput for all the results below are denoted in terms of transactions per second.

---

[1]https://www.terraform.io/docs/providers/aws/
6.4. Results

6.4.1 Hardware

This section explores how performance is impacted when varying the hardware for the client, controller and validator respectively. The aim of this section is to get an understanding of which hardware aspects affect the performance.

6.4.1.1 Impact of choice of client hardware

The first experiment conducted evaluated how performance changes with a one-client network given different client hardware. The goal was to understand how many tps the client could create and send to the controller, which in turn sent the transactions to the validators. The varying parameter was the client’s hardware, and the metrics were average tps and average total transaction processing time. Due to the complexity of the test environment and time limitations, only the instance type on which the client software was run on varied. With each instance type, the processing power (clock speed and/or number of vCPUs) and RAM were changed (see Table 6.1). During the test, the CPU utilization and RAM usage was monitored to make it easier to determine to which degree the different aspects of the hardware affected the performance. A block time of 1 second was used in all of the hardware tests. Lastly, in all of the tests conducted the gas limit of each block was set to a number that would not restrict the amount of transactions that could be processed. The reasoning behind this decision is simply that the goal is to find the highest possible load a client should generate for the system to manage at all.

![Table 6.1: Impact of choice of client hardware in a setting with one client (varying instance type), one controller (t2.2xlarge) and one validator (t2.2xlarge)](image)

In this experiment a setting of one client, one controller and one validator was used. The controller and validator hardware were set to the best possible instance type, t2.2xlarge. The client hardware was incrementally upgraded from t2.nano to t2.2xlarge. As seen in Figure 6.3, the performance reached 157 tps with the least performant instance type t2.nano. When looking at the load of CPU and the RAM usage on the instance it becomes apparent that RAM was the limiting factor. The RAM usage reached close to 100% within seconds after the test was started. The CPU load on the other hand was stable around 35% throughout the test.

When the instance was upgraded to t2.small, the average throughput increased to 291. RAM usage reached 70% and CPU load increased to 55%. Upgrading the instance specifications further did not increase the performance. A slight increase can be recorded for the c5 instance types, which likely was the results from higher clock speeds for those instance types. RAM usage, percentage wise, decreased since the amount of available RAM increased with better
instances while the absolute use remained the same. However, the additional vCPUs did not bring any additional performance, the CPU load stayed at 55%. This is logical since Node.js is single-threaded. These results give indication that the controller or the validator are likely factors that may impact the system’s total performance.

Varying the instance type of the client did not significantly affect the average ttpt. The processing speed of each individual transaction depends mostly on the clock speed of the CPU and the connection between the nodes in the network. In this experiment, the clock speed remains constant for all instance types, and only the number of vCPUs varies.

It is important to note that in a live environment, there are probably very few clients that would need to send this many transactions. In the normal case clients would probably send less than 1 tps, since all users would run their own client on their computer or smartphone. This experiment was important for setting the load limits in the rest of the experiments. Since the performance did not increase significantly with better instance types than t2.small, this instance type was chosen as the default for the other tests.

6.4.1.2 Impact of choice of controller hardware

To evaluate the performance changes given different controller hardware, one client and one validator were used, both run on t2.2xlarge instances. The instance type was varied for the controller as can be seen in figure 6.4.

The hardware setup did not impact the performance in any meaningful way, at least not in a configuration with one client and one validator connected to it. The CPU load was consistently 50% for each instance type, which again, can be attributed to the single-threaded nature of Node.js. RAM usage behaved similarly and did not approach 100% in any of the tests for this configuration, indicating that the amount of RAM was not the problem. Given that neither CPU nor RAM was fully utilized by the controller, it is likely that the controller is not a bottleneck for our system in this setup.

Average ttpt performance did not change in this test either, for the same reasons as presented for the client hardware profiling.

Based on these results, t2.nano would be sufficient for running the controller with one client, but because a live environment reasonably consists of multiple clients, t2.medium was chosen as a likely default type.

![Figure 6.4: Impact of choice of controller hardware in a setting with one client (t2.2xlarge), one controller (varying instance type) and one validator (t2.2xlarge)]
6.4.1.3 Impact of choice of validator hardware

A similar test was carried out to profile the choice of hardware for the validator, one client and one controller was used, both run on a t2.2xlarge instance. The validator instance type was varied as can be seen in figure 6.5.

![Figure 6.5: The performance impact of the change of hardware on the validators ability to process transactions. Setup: 1 client (t2.2xlarge), 1 controller (t2.2xlarge), 1 validator (variable)](image)

Validator node hardware

The validator was significantly affected by the number of vCPUs at disposal. CPU usage was between 50-80% for all vCPUs up to t2.xlarge, which has 4 vCPUs. From there, CPU usage stabilized at 20-40%. An interesting question is why CPU usage never hit 100%, as seen in earlier performance evaluations [12, 42, 7]. One potential reason is that it does not receive enough transactions. Since the automated test script successively increased the offered load until the error rate started increasing, it is possible that a higher average throughput was possible, but not while guaranteeing an error rate of 0%. The error rate was calculated by comparing the number of transaction produced by each client with the number of transactions finalized on Quorum. It is not surprising that the average throughput increases when increasing the computational capacity. The three instance types t2.medium, c5.large and t2.large all have 2 vCPUs. Between t2.medium and c5.large, there is no difference in neither the number of vCPUs nor RAM, and as shown in figure 6.5 there is no performance difference between these two instance types. There is a performance difference between t2.medium/c5.large and t2.large, which is likely explained by t2.large having 4 GB more RAM than the other two. Why RAM affects the average throughput when it is not fully utilized remains unexplored.

Since it still is uncertain what exactly is limiting the performance it was decided to use the t2.2xlarge as the default instance type the validator in the upcoming tests.

In our three profiling tests, we gained some understanding of whether (and when) CPU and RAM were fully utilized in a one-client setting when the load is increased. Based on the results from these tests, it is likely that the CPU impacts performance more than RAM.

6.4.2 Block time

Before continuing with the performance analysis with respect to network size, a test was made to explore the impact of the block time in regards to the performance metrics. In the test, a setting was used that consisted of one client, one controller, and one validator. They
were run on the default settings: t2.small-, t2.medium-, and a t2.2xlarge-instance respectively.

![Graph showing impact of block time on throughput and ttpt](image)

**Figure 6.6:** Impact of choice of block time in a setting with one client (t2.small), one controller (t2.medium) and one validator (t2.2xlarge)

As can be seen in figure 6.6, performance is negatively affected by increasing the block time. Average throughput decreases and average ttpt increases. It was expected that the average ttpt would increase. It takes a longer time to process every transaction because of the increased time period with which a block gets created. However, the decrease in average throughput is not as simple to explain. Given an increased block time, it was expected that the validators would be able to process more transactions for each block.

Since one of the goal is to minimize the average ttpt and maximize the average throughput, the block time was kept at 1 second for the remaining tests.

### 6.4.3 Network size

The aim of the next three experiments was to better understand the overhead of the client, controller and validator, i.e. how they scale when increasing the network size. In the first experiment, the number of clients varied in a setting of one controller and one validator. This is interesting as it gives an indication of how the system scales when increasing the number of nodes for the different types.

#### 6.4.3.1 Number of clients per controller

In this experiment, the varying parameter was the number of clients per controller, while the number of controllers and validators remained fixed at one of each.

As shown in figure 6.7, an increase from one to two clients significantly impacted the average throughput. For two clients, the average throughput increased to 378 tps, while four clients yielded a result of 431 tps. The average throughput stabilized thereafter at around 400 tps. Given that the performance is not affected by increasing the number of clients to more than two to four clients, it is likely that the limiting factor is either the controller or the validator.

The average ttpt differed marginally between the tests. This is expected since all the results are conceived under stable conditions, and no parameter that is known to affect the average ttpt was changed.

On the clients, CPU and RAM usage decreased with each added client, from 50-80% down to 10%. As more clients shared the load, it is logical that CPU and RAM usage decreased.
However, on the controller and validator, both CPU and RAM usage increased slightly. Since having more than two to four clients did not affect the performance, three clients was chosen as the default for the remaining tests.

The results from this experiment is mostly interesting in the context of further testing. In a live environment, the number of clients per controller would likely be much higher, because of all the people involved in the production and logistics process. The system is built in such a way that every user would run their own client and sign their own transactions. This would mean that a client would normally only send a couple of transactions per day and idle most of the time. Thus the performance limit of the client is not seen as a problem in a live environment.

### 6.4.3.2 Number of controllers per validator

In the next experiment, the performance impact of the number of controllers per validator was explored in a setting with three clients per controller and one validator.

**Figure 6.8:** Impact of the number of controllers (t2.medium) per validator in a setting of three clients (t2.small) per controller (t2.medium) and one validator (t2.2xlarge)

Figure 6.8 shows that average throughput is marginally decreased when increasing the number of controllers per validator. One controller per validator yielded an average throughput
of 412 tps, and with ten controllers the average throughput dropped to 387 tps, which indicates that there is a small overhead in handling a controller on the validator. Given the marginal difference in performance in this experiment, the number of controllers used per validator in the following experiments was of lesser significance.

In regards to average ttpt, the result was nearly constant between the tests. Given that no parameter known to affect average ttpt was varied, this was expected.

6.4.3.3 Number of validators in the network

The third and last experiment, in regards to network size, aimed to explore how performance is affected by adding more validators to the network. In this experiment, a setting of ten clients and five controllers was used while the number validators was increased from 5 up to 50. The use of two clients per controller was decided upon based on the cost of performing this experiment. Using two clients instead of three is not believed to have impacted the performance.

As seen in figure 6.9, the decrease in average throughput accelerated after 30 validators. This result is in line with the results presented by AMIS [32]. In terms of CPU load and RAM usage, a slight increase could be observed between the tests up too around 40% per vCPU, while RAM usage was roughly constant between them. This was expected, since increasing the number of validators also increase the number of messages that needs to be processed, ultimately increasing CPU load.

The average ttpt was constant between the test, for the same reason as before - no parameter that affects average ttpt was changed. However, it might have been expected that average ttpt would marginally increase at some point due to the increased number of consensus messages that is needed when the number of validators gets higher [31]. At which point this takes place is more or less of no concern since the average throughput deteriorates well before that could happen.

6.4.4 Network latency and bandwidth

In this experiment, the impact latency has on performance is tested in a setting with ten clients, five controllers and ten validators with the default instance types: t2.small, t2.medium, and t2.2xlarge respectively. The latency in this experiment is added in addition to the already existing latency in the network. However, the native latency is so small (<5 ms)
that it can be omitted. It is therefore worth emphasizing that it is the added latency and not the actual latency that is showed in figure 6.10.

![Figure 6.10: Impact of network latency in a setting of ten clients (t2.small), five controllers (t2.medium) and ten validators (t2.2xlarge)](image)

The network latency affected the average tpt as expected. Since each data transfers and requests take longer, it becomes logical that the whole process takes longer. The expected behavior is linear since the number of messages involved in each transaction process should be the same. As explained in section 5.1, the IBFT consensus algorithm has three phases, resulting in several messages needed to be sent. Therefore, the effect should be the number of messages times the latency, which is also the behavior seen in figure 6.10.

In the regards to the average throughput, it is not obvious why it decreases and is left unexplored in this thesis.

The network bandwidths affect on the performance metrics was not tested separately due to time constraints. Instead a general analysis was made on all the tests. Since the rate of which data was transmitted and received was recorded, the general bandwidth requirements could be presented. However, with this approach nothing more than that the average throughput would decrease can be said. Exactly with which rate and how the average tpt would be affected is unknown at this point.

As expected, both the rate with which the data is transmitted and received increases with higher load and more validators. However the increase is only seen on the validator nodes. Which make sense since clients and controllers only have to send (and receive in the case of the controller) a transaction once. Thus an increase in the number of validators does not affect the controller or client. The validator on the other hand have more transaction data to send and receive with increased load and number of validators. The number of messages also increases with more validators. However the size of the consensus messages are minuscule in comparison to the blocks that needs to be sent and received.

The client and controller only require a connection capable of speeds less than 1 Mbit/s up and down. On the other hand the validators needs a connection speed of around 24 Mbit/s down and 32 Mbit/s up in the most demanding case.

### 6.4.5 Performance of the distributed file system IPFS

The aim of this experiment was to explore the performance of the distributed file system IPFS and determine if it could be a bottleneck for the system. IPFS was tested on a Macbook Pro with a 3,1 GHz Intel Core i5 and 8 GB RAM, with a single IPFS node. Since IPFS has
been configured to only replicate data when accessed by another user, a single node was deemed sufficient for testing even though IPFS is a distributed file system. The test involved uploading files with varying file sizes, ranging from 10 KB to 1000 KB. The aim of this test was to give an indication of how many files IPFS can upload and to see if IPFS was a potential bottleneck. The test results are presented in figure 6.11.

As expected, uploads per second decreases and upload time increases as the file size increases. As shown in figure 6.11, IPFS uploaded an average of 135 files per second with a file size of 10 KB and 68 files per second with a file size of 1000 KB. What is more important is the upload time, which is 7.4 ms and 14.8 ms, for 10 KB and 1000 KB files respectively. Since the upload time for IPFS is a magnitude lower than for the complete system, IPFS is not a bottle-neck for the system. Neither will uploads per seconds, which is lower than the average throughput of the complete system, be a bottle-neck for the system, since the number of IPFS servers can be scaled up.

6.4.6 Simulation of a near-live system

When all the parameters of interest had been examined, a plausible live setting was configured and tested in order to observe the system’s stability over a longer time period, in this case 60 minutes. An even longer time period would have been preferable, but because of time and experiment cost constraints, as well as the earlier indications that performance decreases with longer tests, a time period of 60 minutes was deemed appropriate.

In this experiment, the focus was on the behavior during the run time instead of the behavior between different runs, as has been the case in earlier experiments. A setting of 30 clients, ten controllers, and 30 validators was used with the default instance types: t2.small, t2.medium, and t2.2xlarge respectively. One important aspect of the configuration was the number of validators since it affects the degree to which the system is decentralized. Since the one of the goals with the system is to make it as decentralized as possible, in order to take advantage of the blockchain features, it is desirable for the number of validators to be as high as possible. At the same time, it is necessary to maintain an average throughput high enough to make the system feasible. Based on the tests above and with these aspects in mind, the most optimal amount of validators are around 30.

The number of clients and controllers were not as important as the number of validators. Because of time and cost constraints, 30 clients and 10 controllers are used. The number of clients does not seem to present a problem as long as there are more than three in total. As explained earlier, a live environment would consist of possibly thousands of clients. The controller does not affect the performance in any meaningful way either.
Finally, a block time of 1 second was used and a latency of 200 ms was introduced. The block time was chosen since it gave the most optimal results based on the earlier tests. The latency was decided upon since it was deemed reasonable based on average figures for latency between different locations in the world [19].

Apart from the resulting performance metrics, the CPU load, the RAM usage and the used network bandwidth were looked closely at.

The test resulted in an average throughput of 159 tps and an average ttpt of 4.71 seconds. The latency remained near-constant at the given value of 200 ms, and the network bandwidth usage stayed at around 24 Mbit/s for both upstream and downstream transmissions. The behavior during the test period is accounted for and analyzed below.

![Figure 6.12: Distribution of the average ttpt for all transactions sent during the simulation](image)

Figure 6.12 shows the distribution of the average ttpt for all transactions. Once again, the average ttpt is not just the time it takes for a transaction to be processed by the blockchain network. It is the time it takes for the whole system to process it. From the time it gets created on the client until it is finalized on the blockchain.

The load is kept between 10 and 40% per vCPU on average per minute. Additionally, a slight increase in the load over time can be seen. If this is constantly increasing, or if it levels off down the line is unexplored in this thesis. Although the behavior is interesting and it is possible that the increased load sooner or later would decrease the stability.

![Figure 6.13: RAM usage during the simulation](image)
The memory, or RAM usage is constantly increasing as showed in figure 6.13. This is expected since as much that is allowed of the growing blockchain state is kept in memory for easy access. The question of what happens when the limit is reached is left unexplored in this thesis. Although it might result in reduced performance and stability.

6.4.7 Summary of performance aspects and bottlenecks

In the following subsection, the results from our experiments are summarized and compared to the requirements calculated in chapter 3. A commentary is provided surrounding the behavior of the system and what the potential bottlenecks could be.

The system implemented had an average throughput of 159 tps and an average tpt of 4.71, during a 60 minutes simulation that had a setting of 30 clients, ten controllers and 30 validators. It becomes apparent that a single blockchain network does not reach the performance requirement, even with a delay bound of 24 hours. The requirement for one hour is 2 277 tps. The system’s load requirement for a delay bound of one minute is 26 181 tps, two orders of magnitude away from the reached throughput of 159 tps. Transactions do get processed fast when they are let through the system since the average tpt is relatively low. However, with the demanded load, transactions would be stuck in a queue for an increasing amount of time, since the system would not even be able to handle the load during low load hours as shown in 3.3.

Another potential weakness is that the system only seem to be able to maintain a network size of around 30 validators before the throughput declines with a increasing rate. If IKEA and all possible tier one distributors would be included in the network, each with its own validator node, the system would need to accommodate thousands of nodes.

The most limiting factor is likely the IBFT consensus algorithm. Since it is BFT, time consuming message phases and processing is needed. However, the BFT property is fundamental for the value proposition of the system.

The second largest limiting factor seems to be the processing of transactions in the Quorum software. The CPU does seem to be used in such an extent that it becomes a bottleneck. If this is due to the IBFT implementation or the EVM remains unexplored, but there are indications that the EVM is quite CPU demanding [42].

An added latency also impacts the performance. The fact that the average tpt would be affected is logical. However, the impact on the throughput was not expected. This means that the latency can become a bottleneck in environments where it is high.
In this chapter, we will discuss our analysis of the information flow in IKEA Supply Chain and which flaws that exist. We will then discuss our chosen system design, potential weaknesses with the design and the system’s load requirements that were identified. Finally, we will discuss the results from evaluating our system, which aspects impacted the performance and to what extent the system fulfilled the load requirements previously identified.

7.1 Supply chain data extraction and modelling

Analyzing the information flow in IKEA Supply Chain was a challenging task, greatly due to a lack of data as well as the complexity of their supply chain. For the scope of this thesis, it was necessary to simplify IKEA Supply Chain, providing a basis for creating a model that could provide a rough estimate of the magnitude of events the system needed to handle. Since roughly 90% of the products in IKEA Supply Chain flow through three distribution channels, simplifying the distribution network should not have had a large impact on the final results. However, the distribution channels that were omitted are more complex and could possibly require more events for the entirety of the channel to be recorded on a blockchain. The load requirements may be lower or larger than what is expected in real life due to omitting distribution channels, lack of data and making certain assumptions. Even so, they should give a good representation of the magnitude in question and while the load requirements were based on data and numbers provided by IKEA for fiscal years 2016 and 2017, the supply chain model can be updated to conform to future data.

7.2 System design

A large part in designing the system consisted of minimizing potential load on Quorum since earlier results indicated that the capacity of a Quorum based blockchain network is lower than the estimated information flow. Storing data off-chain and immutably storing the hash in the smart contract was one key aspect in achieving this. The ownership and contributors contract, which was evaluated in section 5.4.1.1, is another important part of limiting the load requirements on the system.
One area of potential discussion is the choice of chasing smaller gas usage in the pursuit of performance optimization. Gas represents different kinds of opcodes in the EVM and it is natural that they demand processing from different hardware parts, e.g., some opcodes are more CPU intensive, while others demand more of RAM and disk IO. In this aspect, not all gas can be regarded as equal.

Another potential weakness is that neither user interface nor user experience were regarded when designing the system; it was designed to allow companies themselves to build custom applications that make use of the system. However, due some of the requirements regarding security, certain functionality must be implemented in the applications built on the client-side. More specifically, client applications must securely handle private keys, encrypt data, and build and sign the transactions that are sent to the controller.

While the implementation of the system contains functionality for registering and fetching events, it does not contain the functionality for searching for events. The system lacks searchability since data is encrypted and stored off-chain. The single-binding used for registering information objects, which intends to minimize load on Quorum, reduces the searchability further. To mitigate this problem, a separate traditional database could be used. Traditional databases offer increased performance, but lack the verifiability and data integrity that a BFT blockchain provides. A combination would prove useful, but it could increase the costs for the system infrastructure and integration with internal systems.

As identified in chapter 3, 10,593 events per second would need to be processed in order to guarantee a delay bound of one minute. The chosen system design handles events in such a way that 26,181 transactions per second is needed to guarantee the same delay bound. The difference between actual event occurrences and transactions is mainly due to having to deploy contracts and transfer ownership of products. Managing ownership directly within the contract proved expensive, but provided vital functionality that could not be omitted in this thesis. However, there might be areas where the smart contract code for transferring ownership can be optimized.

7.3 System evaluation

As shown in chapter 6, sections 6.4.1.3 and 6.4.3.3, it is evident that the validator is the factor limiting the performance. In the system evaluation, CPU-speed is seen as a key aspect affecting the performance of the validator. RAM does not seem to be a bottleneck. However, the RAM utilization is constantly increasing. Exactly what happens when it approaches 100% is unknown. It is a possible that the performance degrades. Additionally, it is important to note that only one function call, register event, was used when testing the system. Other function calls such as "transfer ownership" might have another impact on the performance.

In the way Quorum is implemented today, hardware is the factor limiting performance. However, the biggest bottlenecks seem to be its implementation and consensus algorithm (IBFT in the case of this thesis). First, when the load is very high, major problems occur that render the system unusable. These problems seem to occur more or less at random, but the probability increases as Quorum reaches its limit. Secondly, the consensus algorithm and its BFT attribute place a hard limit on what performance is possible as the algorithm decides how much information needs to be sent across the network. As the implementation becomes more efficient and hardware improves, these bottlenecks might disappear. Instead, latency and bandwidth of the networks that the validators need to communicate with become the limiting factors. A more efficient consensus algorithm that does not require to send as much data or as many messages, would increase the performance. Since the BFT attribute is critical, the question is how much more efficient the algorithm can be made without any compromise to security. Performance can be improved by choosing a consensus algorithm that is not BFT, but that introduces other problems that affects the trustlessness and decentralization of the system.
The components implemented for this thesis, i.e. the client, controller, smart contract and smart contract pool, do not seem to limit the performance of the system. Lack of performance can be improved by increasing the number of instances of each component. However, increasing the number of controllers connected to each validator marginally degrades the performance. The same goes for an increasing number of clients per controller. It is therefore reasonable to assume that the implementation of the client and controller is not a limiting factor of the system if scaled properly.

In the end the resulting performance did not match the required performance. As presented in section 6.4.7, the convergence time in the system is acceptable, while the throughput is two orders of magnitude away from the requirement in some cases. As detailed in section 5.8, a delay bound of one minute would require a throughput of 26 181 tps. In comparison, our system generated an average throughput of 159 tps, 165 times smaller than the requirement. However, this might not be a factor that renders our system unfeasible. A solution to this problem is the concept of using multiple smaller blockchain networks that could be linked together. There are no technical restrictions that prevent splitting the tier 1 suppliers and IKEA into smaller blockchain networks. Given that our system generated a throughput that was 165 times smaller than the system’s load requirement, at least 165 blockchain networks would be required. With 1000 tier 1 suppliers in IKEA Supply Chain, this implies that every network would have at least 7 validators (every blockchain network needs an IKEA validator) and every supplier only has to be connected to one blockchain network. However, when reducing the number of validators in a network, the rate of decentralization is reduced, which is one of the properties that is desired. In order to increase the size of the networks to include up to 30 validators, which would not negatively affect performance (see section 6.4.3.3), every supplier would need to take part in five or more networks on average. If this is desirable or not becomes a question of decentralization versus economic cost.

Whether a delay bound of one minute is necessary or not was not explored in this thesis. If the delay bound could be relaxed the throughput requirement would decrease, resulting in fewer blockchain networks. What the optimal delay bound is, could be explored in future works.

The constellation presented above would result in a system that would reach the throughput requirement. Additionally, because of the priority functionality in the controller, events that need to be processed as fast as possible get a preferential treatment even at high load.
Conclusion

In this chapter, we revisit the research questions and aim stated in chapter 1 and describe our insights from analyzing IKEA Supply Chain, designing a blockchain-based traceability system and evaluating our system’s performance.

In order to answer our first research question, "What requirements exist for a blockchain-based traceability system in the IKEA Supply Chain?", we had informal discussions with personnel at IKEA and an made an estimation of the information flow at IKEA, in regards to the events that would be registered in our system. This estimation concluded that 10 593 events per second would need to be processed to guarantee a delay bound of one minute. 14 requirements were identified in the discussions, including the importance of being able to verify information and custody, and a time-stamped history of who has done what, and when.

Our second research question, how a traceability system can be designed and implemented utilizing the blockchain, resulted in six components: client application, controller, smart contract pool, IPFS, Quorum and smart contracts. The system was designed with decentralization in mind. Data encryption and transaction signing is done on the client application. The controller schedules and orders transactions based on priority, balances load and provides error handling. The smart contract pool continuously deploys smart contracts with a varying frequency, increasing the frequency when the blockchain load is low. IPFS was used to store data off-chain and minimize the blockchain load. Since data is stored off-chain, a hash is stored in the smart contract to provide verifiability. The system was designed with optimization measures in mind, but resulted in a load requirement of 26 181 transactions per second when guaranteeing a delay bound of one minute.

In regards to our third research question, "What are the performance aspects and bottlenecks for such a system?", the system evaluation revealed that 30 validators is the largest network size before the performance starts to decrease rapidly. It is the validator that is the limiting factor in regards to performance. Performance could probably be increased with better hardware, especially better CPUs, but only to a certain extent. The greater limiting factor is the IBFT consensus algorithm and probably the EVM implementation.

The resulting performance of the system is a throughput of 159 tps and a convergence time of 4.71 seconds. The convergence time is acceptable while the throughput is far from the
needed 26 181 tps needed if a delay bound of one minute are to be guaranteed. However, the
network can be divided into 165 smaller networks where they together would produce the
needed throughput. One question to be asked is, however, if networks with 30 validators still
can be seen as sufficiently decentralized.

8.1 Future work

While this thesis tries to do an extensive analysis of the feasibility of a blockchain-based trace-
ability system, there is still much to be done. Future work includes a more extensive explo-
ration of the hardware effects on the performance. A deeper analysis of Quorum’s software
and the reasons for observed instabilities, would aid enterprises and developers to use Quo-
rum. Furthermore, the relation between the gas usage and the throughput could be explored
to conclude to which degree gas-usage-optimization is worth pursuing during development.
A blockchain-based traceability system also needs to be tested in a live environment to evalu-
ate usability and the costs of implementing and maintaining such a system, and if, and how, a
network of participants could be split into multiple blockchains in order to lower the perfor-
mance requirements. Finally, since IPFS is a vital part of the system, the performance needs
to be tested further to see which hardware aspects affect the performance of IPFS.


[20] GS1. GS1 Global Traceability Standard GS1’s framework for the design of interoperable traceability systems for supply chains. Tech. rep. 2017, pp. 1–58. URL: [https://www.gs1.org/sites/default/files/docs/traceability/GS1%7B%5C_%7DGlobal%7B%5C_%7DTraceability%7B%5C_%7DStandard%7B%5C_%7D.pdf](https://www.gs1.org/sites/default/files/docs/traceability/GS1%7B%5C_%7DGlobal%7B%5C_%7DTraceability%7B%5C_%7DStandard%7B%5C_%7D.pdf).


[23] IPFS. URL: https://docs.ipfs.io/guides/concepts hashes (visited on 08/14/2018).


[40] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: ()


Appendix: Test environment and software parameters

A.1 Software

The version of the software used.

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu (OS)</td>
<td>16.04.2 LTS (aws-image: ami-a8d2d7ce)</td>
</tr>
<tr>
<td>Quorum (Geth)</td>
<td>2.0.2 (1.7.2)</td>
</tr>
<tr>
<td>IBFT-tool</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Node.js</td>
<td>8.13.0</td>
</tr>
<tr>
<td>Terraform</td>
<td>0.11.7</td>
</tr>
</tbody>
</table>

Table A.1: Software versions

A.2 Software parameters

The software parameters that were used that is not auto-generated by the test scripts.

A.2.1 Client

No custom parameters

A.2.2 IPFS-node

No custom parameters

A.2.3 Controller

Geth:

    --nodiscover --rpc --rpcapi='db, eth, net, web3, personal, admin'
    --rpcport 8545 --maxpeers '75'
A.3. The Genesis block

The Genesis block is the first block in the blockchain. A number of settings are made in the genesis block. The genesis block used in this thesis is presented below:

```json
{
  "config": {
    "chainId": 2017,
    "homesteadBlock": 1,
    "eip150Block": 2,
    "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "eip155Block": 3,
    "eip158Block": 3,
    "istanbul": {
      "epoch": 30000,
      "policy": 0
    },
    "isQuorum": true
  },
  "nonce": "0x0",
  "timestamp": "0x5bfad0ff",
  "extraData": "generated during setup",
  "gasLimit": "0xffffffff",
  "difficulty": "0x1",
  "mixHash": "0x63746963616c2062797a616e74696e6574616e6365",
  "coinbase": "0x0000000000000000000000000000000000000000",
  "alloc": {
    "05884347abd9fa562f9686c59afeefa268e5a050": {
      "balance": "0x446c3b15f9926687d2c40534fdba5640000000000"
    },
    "number": "0x0",
    "gasUsed": "0x0",
    "parentHash": "0x0000000000000000000000000000000000000000"
  }
}
```

X = request timeout (ms): Variable, default=10000
Y = blocktime (s): Variable, default=1