FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Performance and Scalability of Smart Contracts in Private Ethereum Blockchains

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

Markus Schäffer, BSc (WU). BSc.
Matrikelnummer 01252935

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Ass.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn Monika di Angelo

Wien, 1. März 2019

Markus Schäffer	Monika di Angelo

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Performance and Scalability of Smart Contracts in Private Ethereum Blockchains

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Markus Schäffer, BSc (WU). BSc.
Registration Number 01252935

to the Faculty of Informatics

at the TU Wien

Advisor: Ass.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn Monika di Angelo

Vienna, 1st March, 2019

Markus Schäffer

Monika di Angelo

Erklärung zur Verfassung der Arbeit

Markus Schäffer, BSc (WU). BSc.
Kandlgasse 25/2/12, 1070 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. März 2019

Markus Schäffer

Kurzfassung

In letzter Zeit ist das Schlagwort Blockchain durch den Hype rund um die Kryptowährung Bitcoin in aller Munde. Bald wurde das Blockchain-Paradigma neben Kryptowährungen auch zur Inspiration für andere Anwendungen, wie beispielsweise Smart Contracts. Dies sind Programme welche in einem Blockchain-Netzwerk ausgeführt werden, um digitale Güter nach vordefinierten Regeln zu transferieren. Die Kombination aus den kryptographisch sicheren Verfahren einer Blockchain und der Möglichkeit Programme in solch einem Blockchain-Netzwerk laufen zu lassen resultiert in vielversprechenden Anwendungsfällen für den öffentlichen, sowie den privaten Sektor. Um Smart Contracts auszuführen ist eine Plattform wie Ethereum notwendig. Im Gegensatz zu öffentlichen Smart Contract Plattformen ist in privaten Versionen ein gewisser Spielraum gegeben, welcher beim Aufsetzen eines solchen Systems genutzt werden kann, um diverse Parameter zu konfigurieren. Beispiele für solche Konfigurationsparameter sind die Zeit welche zwischen zwei aufeinanderfolgenden Blöcken vergeht, die Größe von Blöcken, die Hardware der Nodes oder die Anzahl der Teilnehmer im Netzwerk. Wie jedoch all diese und weitere Parameter die Performanz von privaten Ethereum Smart Contract Plattformen beeinflussen ist derzeit nur bedingt bekannt. Außerdem ist nicht klar, in welchem Ausmaß diverse Parameter skalieren und wo sich der Flaschenhals in solch einem System befindet. Um dieses Problem zu lösen wird in dieser Diplomarbeit ein neuartiges Konzept vorgeschlagen, welches es erlaubt die Performanz und die Skalierbarkeit von privaten Ethereum-Netzwerken zu messen. Das Konzept wurde praktisch in ein Framework umgesetzt, welches es möglich macht automatisch verschiedenartig konfigurierte, private Ethereum-Netzwerke in der Cloud zu installieren, um anschließend Daten bezüglich der Performanz zu erfassen. Auf Grundlage der erfassten Daten konnten im nächsten Schritt verschiedene Grafiken erstellt werden, welche die Auswirkungen von Veränderungen eines Parameters auf die Performanz illustrieren. Die Ergebnisse zeigen, dass die Auswirkungen eines Parameters stark von der Konfiguration anderer Parameter abhängig sind. Dies trifft besonders zu, wenn das System an seine Grenzen gebracht wird. Nichtsdestotrotz konnte eine Struktur identifiziert werden, welche die Engpässe in privaten Ethereum-Netzwerken veranschaulicht. Weitere Forschung könnte an der Erweiterung des Frameworks ansetzen, um zusätzliche Clients und APIs zu unterstützen.

Abstract

Recently, the term blockchain has been on everyone's lips due to the media hype which has emerged around the cryptocurrency Bitcoin. Soon the blockchain paradigm has become an inspiration for additional applications next to cryptocurrencies. One type of such applications are smart contracts, i.e. programs which are executed on a blockchain network and move digital assets according to arbitrary pre-specified rules. Utilizing the combination of cryptographically secure mechanisms of a blockchain and the possibility to execute programs on a blockchain network results in promising use cases for the public and the private sector. In order to run smart contracts, a platform such as Ethereum can be used. In contrast to a public smart contract platform, a private version allows to configure some blockchain-specific parameters when setting up the system. Examples of these configuration parameters are the time passing between two consecutive blocks, the size of blocks, the hardware of the nodes running the blockchain software or simply the size of the network. However, how these and other parameters of a private Ethereum smart contract platform affect the performance of the system is still poorly understood. Moreover, it is not clear to which extent these parameters scale and which parameters represent the bottleneck of such systems. In order to tackle this problem, this thesis introduces a novel concept for measuring the performance and scalability of private Ethereum smart contract platforms. This concept is practically implemented in a framework which allows to automatically deploy differently configured private Ethereum smart contract platforms on the cloud for the purpose of gathering performance-related data. Based on the gathered data, various charts were created which visualize the effects on performance when changing a specific parameter. The results of the data analysis conducted show that the effect of variations in one parameter is highly dependent on the configuration of other parameters as well, especially when running the system near its limits. Nevertheless, a structure which depicts the bottlenecks of current private Ethereum smart contract platforms has been identified. Further research may be conducted in order to support additional Ethereum clients and APIs.

Contents

Kurzfassung vii

Abstract ix

Contents xi

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 3
- 1.3 Aim of the Work . . . 4
- 1.4 Methodology . . . 4
- 1.5 Structure of the Work . . . 6

2 Blockchains and Smart Contracts 7
- 2.1 Blockchain Basics . . . 7
- 2.2 Ethereum and Smart Contracts . . . 11
- 2.3 Consensus Algorithms . . . 14
- 2.4 Ethereum Transaction Life Cycle . . . 16
- 2.5 Scalability Trilemma and Possible Solutions . . . 19

3 Related Work 21
- 3.1 Performance and Scalability of Ethereum . . . 21
- 3.2 Performance and Scalability of Other Blockchain Platforms . . . 28
- 3.3 General Considerations on Performance and Scalability . . . 30

4 Results 31
- 4.1 A Concept for Measuring the Performance and Scalability of Private Ethereum Networks . . . 31
- 4.2 Analysis of Performance and Scalability . . . 53
- 4.3 Identified Bottlenecks . . . 80

5 Discussion 85

6 Limitations 91

7 Conclusion 93

List of Figures 95

List of Tables 97

List of Abbreviations 99

Bibliography 101

CHAPTER 1

Introduction

Soon after Satoshi Nakamoto had published the paper 'Bitcoin: A Peer-to-Peer Electronic Cash System' [40] in 2008, the idea of cryptocurrencies went viral. One of the key technologies behind Bitcoin is a public and distributed ledger which stores transactions. This distributed ledger is known as blockchain and is basically a growing list of immutable records stored in form of blocks, which are linked and secured using cryptographic methods. As a hype around Bitcoin emerged, the blockchain technology has become an inspiration for other applications apart from cryptocurrencies.

1.1 Motivation

One such application which has recently gained tremendous popularity in the blockchain community is the use of smart contracts. The term smart contract was coined by computer scientist Nick Szabo in 1996 [51]. Therefore, the concept of smart contracts had existed even before the blockchain technology became popular. Initially, when referring to smart contracts, Szabo described them as "a set of promises, specified in digital form, including protocols within which the parties perform on these promises" [51]. However, with the late development in the blockchain area which has facilitated the execution of smart contracts, they are not necessarily related to the classical concept of a contract any more. Today, smart contracts can be referred to as self-enforcing agreements expressed as computer programs which are enforced on a blockchain. Vitalik Buterin, a co-founder of the Ethereum project [6], describes smart contracts as "systems which automatically move digital assets according to arbitrary pre-specified rules" [15]. When imagining the blockchain as a decentralized state machine, a smart contract can be seen as a tool to encode state transition functions. As blockchains can be categorized as a special kind of distributed database, smart contracts may also be compared to stored procedures or triggers.

In the traditional concept of contractual agreements, often a trusted third party which oversees the agreement and execution is necessary in case the contract partners do not fully trust each other. For example, a lawyer or notary is sometimes needed in cases of transfer of ownership rights, e.g. for real estate deals. In the example of a crowd-funding project, a platform which acts as a trusted intermediary between stakeholders is needed for the management of finances. If the contract partners do not fully trust each other and do not want to rely on a trusted third party, smart contracts which are enforced on a blockchain are an opportunity. While smart contracts are a way to automate processes, the purpose of the blockchain is to ensure immutability and decentralization. Immutability means that a smart contract cannot be changed after creation and that no one will be able to tamper with the code of the contract. Decentralization implies that no single party is in control. Instead of trusting a single entity with the execution and validation of transactions, trust is put in a whole blockchain network.

Automating processes with smart contracts promises a faster execution time of agreements, less bureaucratic nuances and cost reductions, since there is no need to pay a fee to a bank, lawyer or notary. Hence, blockchain enthusiasts have soon tried to find areas of applications for smart contracts apart from the most popular use case - transferring digital money. Smart contracts can be especially useful in scenarios where an intermediary can be replaced but may also provide benefits where there is no intermediary to be replaced. One specific use case is funding, where smart contracts enable start-ups to cut out third parties, e.g. banks, via funding money with Initial Coin Offerings (ICO). As the blockchain is basically a distributed ledger, any use case which involves keeping track of assets is a possible application area for this new technology. For example, smart contracts and the Internet of Things (IoT) can be combined to provide transparency in the supply chain of products. Due to the blockchain, vendors can become more trustworthy and the risks of fraud can be reduced. Another area where smart contracts and the blockchain can be used to reduce fraud is electronic voting.

Currently, one of the biggest blockchains, based on market capitalization and amount of smart contracts deployed, is Ethereum. While Bitcoin's blockchain supports only a limited set of smart contracts, Ethereum has been the first blockchain to support arbitrary code execution on the blockchain, since it is specifically designed with an execution model for smart contracts. While the main focus of Bitcoin is on acting as a cryptocurrency, Ethereum's purpose is to provide a shared global infrastructure which acts as a platform for the execution of smart contracts.

Next to public, i.e. permissionless, blockchains and smart contract platforms such as Ethereum, there are also private, i.e. permissioned, versions. These private blockchains are sometimes more suitable for use cases within or among businesses. Instead of having a single trusted party maintaining a centralized record or multiple parties maintaining their own records independently, a private or federated blockchain can be used. Such a private blockchain enables a group to maintain a single view of the truth without ceding control to any individual participant. They can be seen as an addition to the toolkit for enterprise Information Technology (IT). As private smart contract platforms

allow to restrict the access to the network to specific pre-chosen entities, they enable businesses to hide sensitive information from the public. In addition, private smart contracts usually show lower transaction costs and can be tremendously faster than their public counterpart.

Indeed, one of the main concerns of public blockchains are performance and scalability [54], [32]. Vitalik Buterin, a co-founder of Ethereum, actually stated that "the Ethereum community, key developers and researchers and others have always recognized scalability as perhaps the single most important key technical challenge that needs to be solved in order for blockchain applications to reach mass adoption" [19]. Simply said, one of the main major problems is that current blockchains cannot keep up with centralized systems regarding performance and scalability. For example, Bitcoin's performance is roughly 1-3 transactions per second [32], while Ethereum's performance is currently capped at approximately 15 transactions per second, which can be observed on Etherscan.io [12]. In contrast, systems such as VISA can handle around 56,000 transactions per second [36]. With a higher amount of users or applications, this bad performance currently results in waiting times where transactions wait in a queue to be validated. That waiting time is a knock-out criterion for some applications and businesses. Nevertheless, in contrast to their public counterpart, private smart contract platforms offer some configuration possibilities. As some parameters can be adjusted, the opportunity to impact the performance arises.

1.2 Problem Statement

When a group of entities or a single business decides to establish a private smart contract platform, the system has to be set up at first. At this point, crucial decisions concerning the configuration of the platform have to be made. One might want to know how the size of blocks, the block interval or simply the power of a node running a blockchain client affects the throughput and latency of a smart contract platform. However, as most studies in the smart contract and blockchain context have primarily focused on security aspects and scalability issues in general, there has been little discussion about the effects of different configuration parameters on performance improvements so far. Although there have been discussions about the block frequency and block size, the effects of other parameters such as the network size or the type of smart contract is still not widely understood.

Thus, the *problem statement* for this master thesis is that it is not clear how specific configuration parameters of a private Ethereum smart contract platform affect its performance and scalability. Moreover, there is no precise concept for evaluating the performance of a private Ethereum smart contract platform. Additionally, it is unclear which parameters represent the performance and scalability bottlenecks of the system.

In order to contribute to the current state of knowledge the following three *research questions* were phrased:

- RQ1: What are means to measure the performance and scalability of a private Ethereum smart contract platform?
- RQ2: What are the effects of different parameter settings of a private Ethereum smart contract platform with respect to performance and scalability?
- RQ3: Which parameter of a private Ethereum smart contract platform represents the performance and scalability bottleneck?

1.3 Aim of the Work

The overall aim of the present work is to further extend the current understanding of the effects of different parameters in private Ethereum smart contract platforms with respect to performance and scalability. Therefore, the following sub-goals have been formulated in the context of private Ethereum smart contract platforms:

- Proposing a new concept for deploying differently configured Ethereum networks and measuring their performance.
- Practically implementing the concept to a framework which allows gather performance-related data in differently configured Ethereum networks.
- Contributing several charts which visualise the effect of identified parameters on performance and scalability.
- Re-examining findings reported in the related work.
- Identifying parameters which represent the performance and scalability bottlenecks.

1.4 Methodology

The methodological approach to fulfil the aim of the work and to answer the research questions specified in section 1.2 is listed below. In order to provide a better overview, the methodology is split and listed for each research question.

For research question one:

- Literature review on smart contract and blockchain performance and scalability

  First, a literature review is conducted to provide the theoretical basis for further steps. Moreover, general literature research regarding smart contracts and blockchains has to be conducted to acquire more knowledge about the rather new fields of smart contracts and blockchains.

- Development of a concept for deploying differently configured Ethereum networks and measuring their performance

  After having studied the state of the art in performance analysis of smart contract platforms and blockchains, a concept is formulated which depicts how differently configured Ethereum networks can be deployed and how their performance can be measured. Among others, the documentation of Ethereum and its available clients has to be studied to reveal different parameter settings which can be influenced.

For research question two and three:

- Practical implementation of the formulated concept to a framework

  Once the concept is formulated, the actual implementation of the concept has to be performed. This step involves the development of smart contracts as well as the whole infrastructure which enables to gather performance- and scalability-related data. For example, an application which handles requests and responses to/from the Ethereum network has to be implemented.

- Experiments on the effects of different parameters in various private Ethereum networks

  Next, data regarding performance and scalability is gathered. Performance measurements are performed in differently configured Ethereum networks. While the metrics reflecting the performance of the system represent the dependent variables, parameters which have been identified to impact the performance represent the independent variables. The parameters are varied in a range which also includes the minimum/maximum value in order to gather data which allows to identify bottlenecks in the system.

- Data analysis and interpretation

  The gathered data is analysed using exploratory, descriptive and inferential parts. The exploratory and descriptive parts to quantitatively describe and summarize are mainly covered graphically via box plots, bar charts and line graphs as well as measures of central tendency. For the inferential parts, statistical hypothesis testing (t-test) is used. Moreover, regression analyses are utilized to estimate the relationship between a specific parameter and the performance. In order to identify the bottlenecks of the system, theoretical information gained about the working of Ethereum and the results obtained from the data analysis are combined. Moreover, the yielded results are compared with the related work.

1.5 Structure of the Work

The rest of this thesis is organized as follows.

Chapter 2 provides more information on blockchains and smart contracts. Special emphasis is put on the smart contract platform under investigation in this thesis, Ethereum. Among others, Ethereum's consensus algorithms and the typical Ethereum transaction life cycle are discussed. In addition, methods which may help to improve the scalability of the Ethereum platform in the future are outlined.

Chapter 3 gives an overview on the related work. As some sources directly focus on Ethereum, others on different blockchain platforms and some on general aspects, the findings are categorized into these sections accordingly.

Chapter 4 presents the results of the work and comprises three sections. First, the novel concept for deploying and measuring the performance and scalability of private Ethereum networks is proposed. Next, the results of the data analysis for each parameter are presented. Third, the relevant bottlenecks are identified.

Next, chapter 5 discusses the findings and compares them with the results identified in the related work.

In chapter 6 the limitations of the present work are outlined.

Finally, chapter 7 concludes the work with a summary and an outlook on future work.

CHAPTER 2

Blockchains and Smart Contracts

After having provided a primer on blockchains, smart contracts and their performance limitations in the previous pages, this chapter aims at examining these technologies in more detail. Since the domain is rather new, knowledge about the underlying concepts and the function of such systems in general is crucial for the reader to be able to follow the rest of the work.

2.1 Blockchain Basics

A blockchain is basically a list of records which is distributed and replicated among the members of a Peer-to-Peer (P2P) network. Initially, the blockchain as a data structure was introduced with Bitcoin [40] where it was proposed as a solution for the double-spending problem [1] in combination with the Proof-of-Work (PoW) consensus algorithm. In Bitcoin, the blockchain data structure is used to store the validated and mutually agreed-upon transactions. In other words, the blockchain data structure serves as some sort of ledger which contains information about who owns what at a particular time. From a more technical perspective, a blockchain can be referred to as transactional singleton machine with shared-state [54]. Transactional singleton machine basically means that there should be a single global truth that everyone in the network believes in. The shared-state implies that the state stored on a machine is shared and open to other nodes in the network. In other words, any node with access to the blockchain network can read the current world state.

The structure of the blockchain is highly likely to be the reason for its name. One can think of this data structure as some sort of ledger or journal whose records are batched

[1] In a digital cash scheme, double-spending is a potential flaw which describes the successful use of the same funds more than once. This problem arises in the digital context as, unlike with physical cash, a digital token can be duplicated.

into timestamped blocks. A key feature of blockchains is the use of cryptographic hash functions to map the blocks' data to a smaller and fixed size value. Each block is identified through this cryptographic hash. Furthermore, this hash is also used to establish a link between blocks. If a new block is to be added, the hash calculation for the current block also takes the hash of the previous block as an input. Hence, a chain of blocks is created - the blockchain. For visual representation of the simplified structure of a blockchain figure 2.1 below is used.

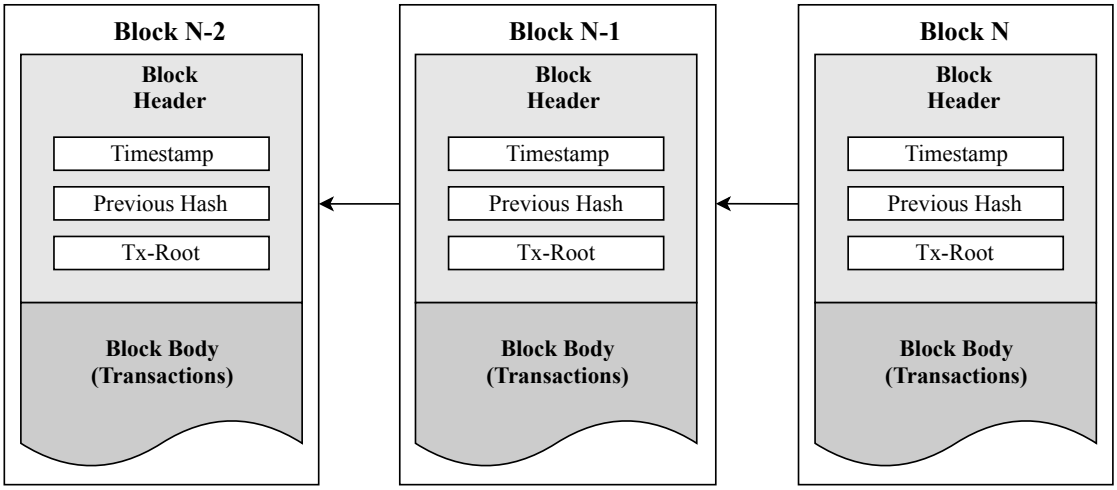

Figure 2.1: Simplified Blockchain Structure

One can see that a block consists of a header and a body. While the header holds some sort of meta- or additional data, the body comprises the payload, i.e. the transactions. Although the specific content of the block header depends on the actual implementation, a block header usually at least comprises information about the timestamp of the block, the hash of the previous block and a Merkle tree's root. This Merkle tree[2] contains the hashes of all transactions in the block's body. From this, one can derive that every transaction occurring on the blockchain network also has a hash associated with it. Since a block stores numerous transactions, all the transactions' hashes are also hashed, which results in a Merkle root. The advantage of this approach is that a single value can be used to prove the integrity of all of the transactions under it. It is easy to check whether transactions have been tampered with. Furthermore, organizing transactions in a Merkle tree structure uses fewer resources than hashing all the transactions in one batch and inserting that into the block header.

When examining figure 2.1 in more detail, one can come to the conclusion that a blockchain is basically the same as a linked list. Although there are many similarities

[2]In a Merkle tree or binary hash tree every leaf node is labelled with the cryptographic hash of a data block. Moreover, every non-leaf node is marked with the cryptographic hash of the hashes of its child nodes.

between those data structures, there is also an important distinction. In contrast to a linked list, the elements of a blockchain are linked with each other via their hashes. Thus, one can verify that the transactions stored in previous blocks were not tampered with. The use of hashes results in the following differences. First, while data stored in a linked list could be easily changed, altering a transaction in a blockchain context would require a recalculation of the hashes of all the following blocks in the chain. This is due to the reason that the hash of the previous block is included in the header of a block. Second, whereas an element of a linked list can be easily deleted, deleting a block in a blockchain would also require a hash-recalculation of other blocks. Third, while new data can be added at any place in a linked list, a new block can only be added at the end of the chain.

Other essential components of a blockchain are that they have a notion of history and some sort of a state transition rule [18]. Furthermore, blockchains usually heavily rely on asymmetric cryptography as public/private key pairs are usually used to ensure authenticity and integrity in the blockchain network [23]. The notion of history is basically a set of all previous transactions and blocks and the order in which they have been processed. This history is used as an input for a state transition function. Given the previous state and a specific transaction, a state transition rule calculates if the transaction is valid and what the state will be after the successful execution of the transaction. With regard to asymmetric cryptography, a public/private key pair in combination with the hashes is used to create and verify digital signatures. The sender of a transaction signs the transaction hash to produce a digital signature which is then added to the particular transaction. Afterwards, nodes in the network which receive the transaction can verify the transaction using the public key of the sender. This involves two steps. First, a hash of the received transaction is generated. Second, using the public key of the sender, the received digital signature is decrypted. If the decrypted hash matches the generated hash of the transaction, the verification has been successful. Hence, it has been ensured that the sender actually owns the corresponding private key (authentication) and that the data has not been tampered with since it has been signed (integrity). Thus, digital signatures give the data recorded on a blockchain its immutability. Furthermore, digital signatures ensure that assets on a blockchain can only be spent/transferred by their rightful owners.

As the blockchain is a distributed data structure, the question of how to reach a consensus is a challenging one. In a blockchain network, nodes in the network find consensus about the current state of the blockchain via consensus algorithms specifically designed for the use in the blockchain context. For finding consensus in a public blockchain network, the so called Proof-of-Work (PoW) consensus algorithm, which includes the process of mining, is currently used in the Bitcoin and Ethereum blockchains. However, there are also other consensus algorithms in the blockchain context such as Proof-of-Stake (PoS) or Proof-of-Authority (PoA). More specific information about the characteristics and mechanisms of these consensus algorithms can be found in section 2.3.

2.1.1 Types of Blockchains

Most of the time, when speaking about a blockchain, one refers to the public type of blockchain. This is due to the reason that the initial idea of a blockchain was that it should be public and accessible to anyone. Nevertheless, over time the concepts of private- and consortium blockchains have emerged. Currently, there are three different types of blockchains [17].

In a public blockchain anyone can read, send transactions and participate in the consensus process. Hence, anyone can take part in the process of determining the current state of the blockchain and may add new blocks. In this type of blockchain, the combination of economic incentives and cryptographic verification acts as a substitute for centralized or quasi-centralized trust. The general principle is that the degree to which someone can have influence in the consensus process is proportional to the quantity of economic resources that one brings into the network. However, some use cases may require to restrict the access to a blockchain network to a predefined set of participants. The most obvious reason for this is having information going over the network which is sensitive and should not be visible for everybody. This is where consortium and private blockchains come into play. Consortium blockchains are operated by a group of entities. The consensus process is controlled by a preselected set of nodes. For example, one can imagine a consortium of 15 financial institutions where each entity operates a node. A blockchain is fully private when write permissions are kept centralized in a single organization. In both the consortium and the private blockchain, the right to read the blockchain may be public or restricted to the participants.

Finally, the characteristics of the different blockchain types are visualized via figure 2.2 below. As can be seen, the characteristics of the consortium and private blockchain overlap to a great extent. As one can also find classification schemas which only distinguish between public and private blockchains [29, 46, 39] and due to simplicity reasons, the term private blockchain is used in this thesis to indicate permissioned, i.e. private and consortium blockchains.

	Public Blockchain	**Private/ Consortium Blockchain**
Access	Anyone	Single or Multiple Organization(s)
Participants	Permissionless, Anonymous	Permissioned, Known Identities
Security	Consensus Mechanism	Pre-approved Participants
Transaction Speed	Slower	Faster

Figure 2.2: Types of Blockchains and their Characteristics

2.2 Ethereum and Smart Contracts

As has been discussed so far, a blockchain essentially encodes state transitions. Also Ethereum is simply said a state machine. To be more precise, "Ethereum, taken as a whole, can be viewed as a transaction-based state machine: we begin with a genesis state and incrementally execute transactions to morph it into some final state" [54]. One can also say that Ethereum [15] is basically a blockchain with a built-in programming language which allows to perform any computation as part of a transaction. Ethereum allows users to create their own arbitrary rules, i.e. state transition functions. Compared to Bitcoin, Ethereum does not serve the predefined purpose of a cryptocurrency platform. The intention of Ethereum rather is to serve as a platform for many different types of decentralized blockchain applications. Applications may include cryptocurrencies, yet they are not limited to this one use-case. In the context of Ethereum, these decentralized blockchain applications are referred to as DApps. Ethereum DApps typically connect users via a web based application using a JavaScript API to communicate with the blockchain [24]. DApps typically utilize Ethereum scripts to encode business logic and to persist their state.

These Ethereum scripts are the key element in Ethereum and are known as smart contracts. They are a high-level programming abstraction and can encode any algorithm in them. The contracts are run by each node in the blockchain network, e.g. as part of the block creation process. They are written in specially built blockchain programming languages like Solidity, Serpent or Viper. After smart contracts have been defined with one of the mentioned high-level blockchain programming languages, they are compiled into a binary format (EVM bytecode) and deployed to the blockchain where they are executed in the Ethereum Virtual Machine (EVM). This virtual machine is the heart of Ethereum and acts as the runtime environment for smart contracts and transactions. It has a stack based architecture and it is Turing-complete. This basically means that EVM code can encode any computation that can be carried out. Like any blockchain, Ethereum also comprises a peer-to-peer network protocol. There is a blockchain database which is updated by the nodes connected to the network. Every node in the blockchain network runs the EVM and executes the same instructions. Therefore, Ethereum is sometimes described as a world computer. The EVM is completely isolated from the rest of the machine running the Ethereum software. That implies that the code running inside the EVM has no access to network, file system or other processes [24].

Similar to Bitcoin, Ethereum has a native value-token. Ethereum's token is called Ether. Ethers can be stored into account addresses and can be spent or received as part of transactions. Ether may also be earned via the generation of a block in the mining process. Nevertheless, in contrast to the Bitcoin, the main purpose of Ether on the Ethereum platform is not to act as a cryptocurrency. Ether rather serves as means to pay operators of Ethereum nodes in the Ethereum network to run transactions in the EVM. Furthermore, paying for transactions to be run is a solution to the possible problem of smart contracts running indefinitely, i.e. in a never-ending loop. Every time a smart contract is executed, a limit of Ether to be spent has to be stated. In Ethereum, this is

referred to as gas-limit (as in gasoline). Every operation executed on the EVM consumes a certain amount of gas. If the execution of a smart contract or a simple transaction consumes more gas than has been specified as gas-limit, the EVM stops executing and an error is returned.

2.2.1 Ethereum's Structure

The shared world state of Ethereum is a mapping of account addresses and account states. Transactions occurring between accounts alter the world state of Ethereum. There are two types of accounts in Ethereum: Externally Owned Accounts (EOA) and contract accounts. An EOA is controlled by a private key and has no smart contract associated with it. As it has an Ether balance and can send transactions, one can imagine it as a digital wallet. Accounts of the latter account type have code associated with them. It is important to distinguish between these two account types as only EOAs can initiate new transactions on their own. An EOA can send transactions to other EOAs or to contract accounts. Whereas a transaction between two EOAs is simply a value transfer, a transaction from an EOA to a contract account activates the contract account's code. The contract account's code, i.e. the smart contract, is executed as instructed by the input parameters sent as part of the transaction. Although account contracts can create internal transactions, which are also known as messages or function calls in Ethereum, to invoke other account contracts, any action on Ethereum is always set in motion by transactions fired from EOAs. An account state consists of the following components: `nonce`, `balance`, `storageRoot` and `codeHash`. The nonce represents the number of transactions sent or created from the account's address. The balance is simply the amount of Ether owned. StorageRoot is a hash of the storage contents of the account. The codeHash is the hashed EVM code.

In order to map account addresses and account states, the account states have to be serialised first. Then the mapping of addresses and states is stored in a data structure known as Merkle Patricia Tree in a database backend. This database is located on the machines running the Ethereum clients. A Merkle Patricia Tree is a modified Merkle Tree where nodes represent individual characters from hashes instead of each node representing an entire hash. This kind of tree needs a key for every value stored. The key tells which path to follow in order to get to the corresponding value stored in a leaf node. This means that Merkle Patricia Trees are tries[3]. The important property of Merkle Patricia Trees is that they are fully deterministic. This means that, with the same key-value-mapping, it is guaranteed that the resulting structure is exactly the same bit by bit whenever it is constructed. Furthermore, as the root hash of the structure depends on all the hashes of the sub-nodes, it is also identical any time the structure is built with the same key value binding. Any change to the tree results in a different root hash. Hence, the state trie's root node can be used as a secure and unique identifier for the state trie. The hash of the state tree is embedded in the header of the blocks in Ethereum.

[3] A trie is a tree structure which uses prefixes of the keys to decide where to put nodes.

With regard to figure 2.3, which displays the structure of Ethereum block headers and transactions, one can see that there are also root hashes of transactions and receipts. This is due to the reason that the same trie structure used for storing the global state is also used for storing transactions and receipts. Other important fields in the Ethereum block header, which have not been explained in the previous section, are the `gasLimit`, `difficulty` and `nonce`. The gasLimit is a value which describes the current limit of gas expenditure per block. Hence, it describes the size of blocks. The difficulty value specifies the time it takes to generate a new block when using the Proof-of-Work consensus algorithm (see Proof-of-Work in section 2.3). The nonce is a hash value which is used in the Proof-of-Work consensus variant to prove that a sufficient amount of computation has been carried out (see Proof-of-Work in section 2.3).

Transactions, the key element to alter the state of a blockchain, can be described as cryptographically signed instructions sent to the Ethereum network. There are two types of transactions: message calls and contract creations. The structure of transactions is as depicted in the right part of figure 2.1. The `nonce` is the number of transactions sent by the sender. `gasPrice` and `gasLimit` are values which describe the amount of Ether the sender is willing to pay per unit of gas and the maximum amount of gas the sender is willing to pay for the execution of the transaction. The `to` field is the address of the recipient and the `value` field is the amount of Ether to be transferred from the sender to the recipient. The values corresponding to the signature of the transaction which are also used to determine the sender of the transaction are in the `v, r and s` part of the transaction. The input data, i.e. the parameters, of a message call is in the `data` field.

More information about the internals of Ethereum can be found in the yellow-paper [54] and the beige-paper [26].

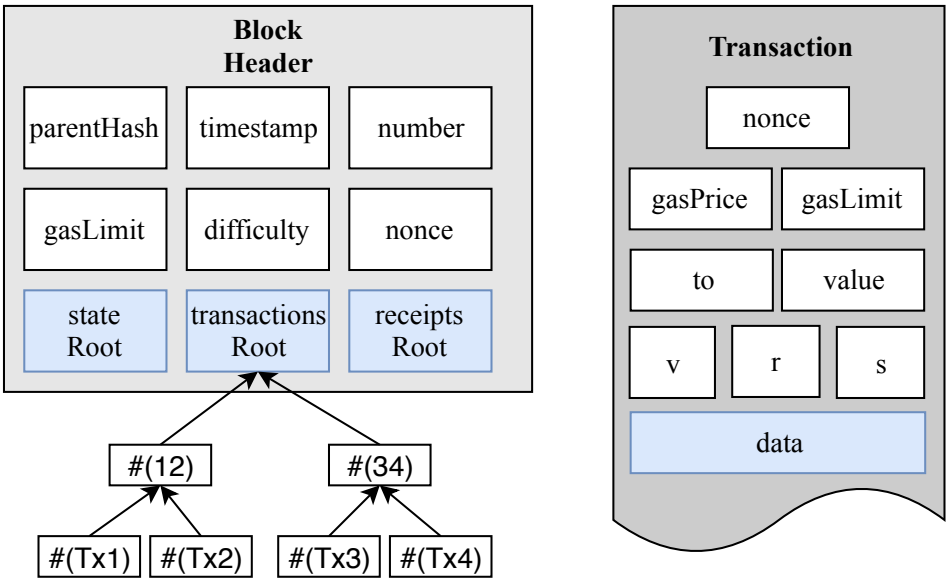

Figure 2.3: Ethereum Block Header and Transaction Structure

2.3 Consensus Algorithms

Building on the information provided in the last two sections, the different consensus algorithms can now be discussed. As blockchains are operated in a distributed environment, a protocol is needed which ensures that the nodes in the network reach an agreement upon the state of the chain. This involves verifying transactions and appending new blocks to the chain. Although various consensus variants have emerged over the last years, only the three protocols implemented in Ethereum are discussed in this section. These protocols are Proof-of-Work (PoW) [40], Proof-of-Stake (PoS) [38] and Proof-of-Authority (PoA). Whereas PoW and PoS, the most popular consensus algorithms [42], are typically used in public blockchains, the PoA variant is rather used in a private context.

The most utilized consensus protocol in today's blockchains is Proof-of-Work. The principle behind PoW is that a node will get the permission to append a new block to the blockchain if it has shown that a sufficient amount of work has been carried out. The work which has to be performed is solving a puzzle by randomly guessing a secret value. The node which is the first to find a secret value is granted the right to append its block to the global chain. Other nodes check the found block for validity. The puzzle to be solved is finding a hash value for a specific input which is below a certain threshold. Solving this puzzle also serves the purpose of increasing the supply of the blockchain's native token. As there is an analogy in finding gold in a mine and finding a block in a blockchain, the term mining got prominent with the PoW process. Before trying to solve the puzzle, a node includes its verified transactions into the block body and also fills the block header. The input for the hash calculation is the header of a block. As introduced in the last section, the block header includes fields such as the timestamp, the hash of the parent block and the root-hashes of different trees. Most important for the process of finding a hash value, which is below the threshold, are the `difficulty` and the `nonce` values. The nonce is the one value which is randomly altered until the hash of the block header is below the threshold. All nodes are constantly changing the value of the nonce value until any node in the network communicated that a new block has been found. Then the whole process is repeated. The difficulty value in the block header indirectly determines how much time it takes to find a new block. Given the hashing-power of a network, the difficulty value may be adjusted over time in order to get at a predefined time window between blocks. If the time window between two blocks is too small, a difficulty adjustment algorithm will increase the difficulty. In a practical context, the difficulty in this process can simply be imagined as the number of leading zeros of the nonce value. The specific implementation of the general PoW protocol in Ethereum is the ethash [4] consensus algorithm. In this algorithm, a seed is computed for each block on the basis of the block header. From the seed, a pseudo-random cache is computed. From this cache, a dataset is generated which has the property that each item in it depends only on a small number of items from the cache. This dataset is known as Directed Acyclic Graph (DAG). Mining nodes grab random slices of the DAG based on the nonce and the block header, hash them together and compare the result to the threshold.

Although the PoW algorithm is a method to prevent malicious entities to easily overpower the network and to take control over the blockchain, the algorithm also shows some weaknesses. The main issue with PoW is that it is highly energy inefficient. This is due to the nature of the PoW process where every node participating in the mining process is racing against each other to find a new block. However, only the effort of one single node of the massive amount of nodes mining is rewarded and the computations done by the rest are wasted. Thus, there is a huge amount of power and energy that is wasted with the PoW process. This and other issues have lead to alternative consensus algorithms such as Proof-of-Stake and Proof-of-Authority.

With PoS, the entire mining process is replaced with a set of validators. These validators stake their current economic wealth in the network to get a chance to propose and vote on the next block. The miner of a new block, in case of PoS the forger, is chosen in a semi-random, two-part process. First, a sufficient amount of the blockchain's cryptocurrency has to be locked up into a deposit. The deposit can be thought of as being in a virtual safe and is used as a collateral to vouch for the block. The weight of each validator's stake impacts the probability to get the right to propose the next block. Second, in order to avoid a scenario where only the wealthiest validators are selected to propose new blocks, a second element is added to the process which adds semi-randomness. For example, forgers may be selected by using a formula which considers a combination of the lowest hash value and the size of the stake. Due to some improvements when compared to a PoW system, there will be a transition in Ethereum from PoW to PoS consensus in the near future. The specific implementation of PoS in Ethereum is known as Casper [20, 7].

The Proof-of-Authority mechanism is a modified form of Proof-of-Stake. Similar to PoS, the assumption behind PoA is that those who hold a stake in a network are incentivized to act in the blockchains interest. In contrast to PoS, the stake in PoA is the reputation, i.e. the identity of a validator and not its wealth. Validators have to be approved beforehand in order to validate transactions and blocks and their identity is stored in a publicly available registry. Ethereum uses the PoA consensus algorithm in its Rinkeby and Kovan testnets. The specific implementation of PoA for Ethereum's Geth [13] client is known as Clique [5]. Clique requires a simple majority of signers to be honest in order for the network to work. New validators can be added by a voting mechanism where existing validators vote for or against a new validator. Each validator is only allowed to seal one out of a predefined limit of consecutive blocks. Although PoA has some advantages, e.g. the increased energy efficiency, there are also concerns about the lack of true decentralisation. In fact, the PoA model may be seen as a slightly more distributed version of a centralized system.

More information about different consensus algorithms can be found in "A survey about consensus algorithms used in Blockchain" [42] and "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks" [53]. Both review the state of the art of consensus protocols in the blockchain context.

2.4 Ethereum Transaction Life Cycle

When interacting with the Ethereum blockchain and updating its state, transactions have to be executed. The journey a transaction takes is usually from a browser or a console to the Ethereum network and back to its origin after it has been executed. The whole traversal of a transaction, i.e. its life cycle, is discussed in the list below. It is assumed that a local Ethereum client and the PoW consensus algorithm are used. Moreover, it is assumed that transactions are handled via the web3 [31] API.

1. Transaction is created
   Values such as the recipient, the gas-limit and the gas-price (see section 2.2.1) for the transaction have to be specified. If a smart contract function should be invoked, the recipient is simply the address of the contract and the data payload specifies the function to be invoked and its input parameters.

2. Transaction is signed
   The transaction is signed using the private key of the account, i.e. a digital signature is created for the transaction.

3. Transaction is pushed into the local transaction pool
   Each node has a pool or queue where transactions wait to be processed.

4. Transaction is validated locally
   The validation procedure [15] involves steps such as checking that:
   - the digital signature is consistent with the sender address.
   - the transaction is valid and well formed, i.e. has the right number of values.
   - the sender has enough Ether to fund the execution of the transaction.
   - the nonce matches the nonce in the sender's account.

5. Transaction is broadcast to the network
   If the validation is successful, the transaction will be propagated to all its directly connected peers. Each of these peers in turn also propagates the transaction to its peers. Hence, the transaction is spread across the whole network.

6. Transaction is queued in the transaction pool of nodes
   In the transaction pool all the transactions are usually ranked according to the gas-price.

7. Miners pick transaction from the pool
   Miners pick transaction from the pool to be processed in the next block. Most likely the transactions with the highest gas-price are preferred.

8. Transaction is validated on remote nodes
   The validation process described in step 4 is repeated on numerous nodes.

9. Transaction is executed and put in a pending block
   Once the miners select the transactions to be included in a block, the transactions are executed in the EVM and included in a pending block. Afterwards, the PoW process begins.

10. Miners try to solve the PoW puzzle for the pending block
    Given a block difficulty, miners try to find a solution to the PoW puzzle via changing the nonce of the block.

11. A miner has found a valid PoW
    A miner found a valid PoW, i.e. a hash below a given threshold.

12. A miner appends the newly-mined block to its local blockchain and broadcasts the block to its peers.
    The miner who has found a valid PoW for its local block appends the block to its local chain and propagates it to all its directly connected peers.

13. Nodes validate the received block
    Each node which receives the block will validate it, i.e. check that:
    - the reference to the previous block is valid.
    - the timestamp of the block is greater than the one of the previous block and less than 15 minutes into the future.
    - various low-level concepts are valid, e.g. block number, difficulty, transaction root, gas-limit.
    - the PoW on the new block is valid, i.e. if the hash input actually results in that signature and if the nonce is below the defined threshold.
    - there are no state transition errors after running all the transactions and that the state is exactly as stored in the received block.

14. Block is added to the blockchain of the node and further broadcast
    If the validation is successful, the node will add the block to its own blockchain and broadcasts the block to its peers.

15. The new block arrives at the node which initially created the transaction.
    Finally, the block containing the transaction created at the first step arrives at the node which initially created the transaction. A transaction receipt is created which contains certain information concerning its execution. Every time another block gets added on top of this block, it counts as a confirmation for the previous block.

For a visual representation of the transaction life cycle, figure 2.4 on the next page is used. The figure illustrates an overview of the transaction life cycle when using the web3 API. On a higher level, one may recognize that there is quite a similarity between sending a transaction to the blockchain and sending a HTTP request to a web server.

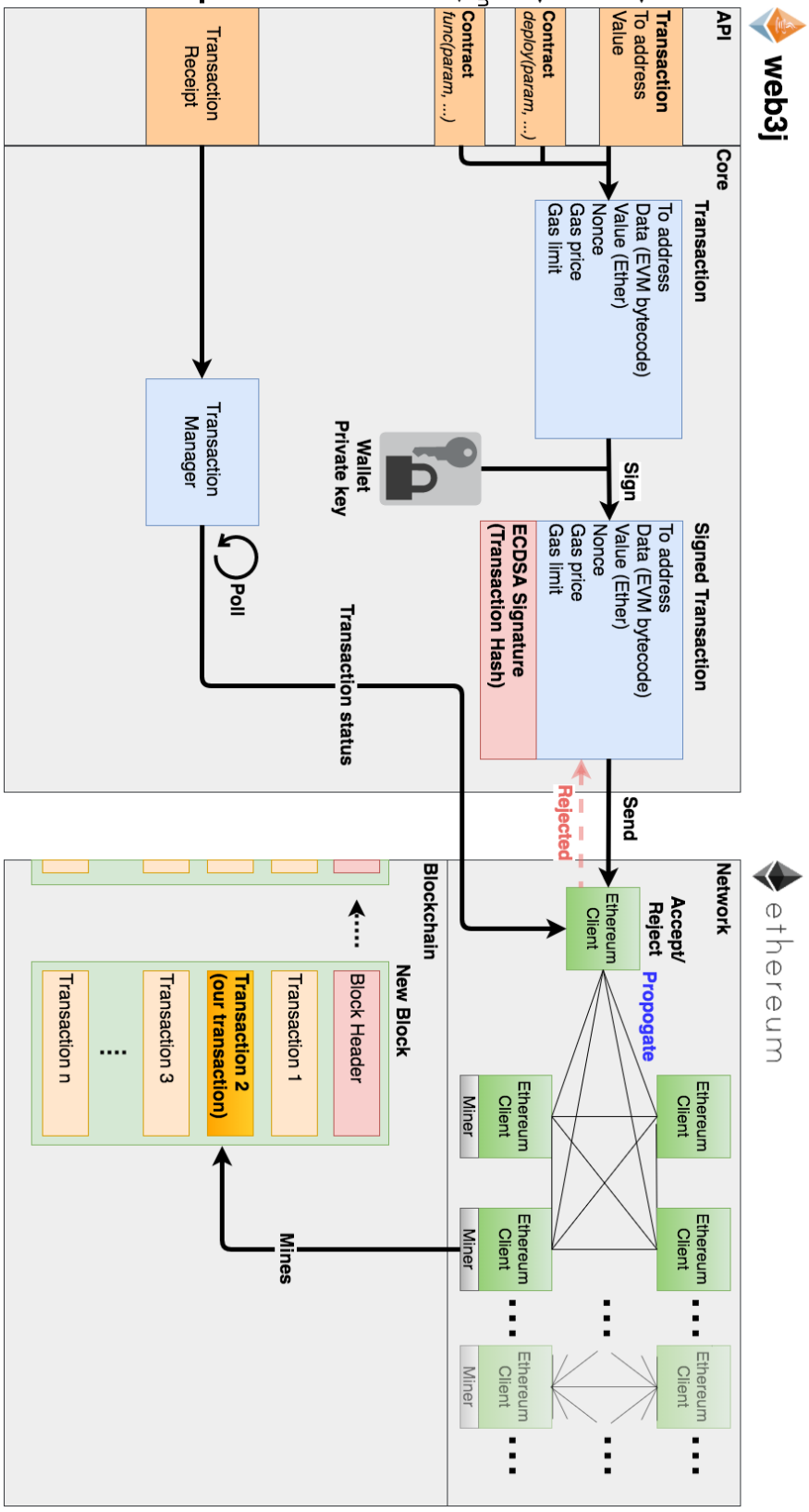

Figure 2.4: Overview of the Ethereum Transaction Life Cycle with the web3 API [30].

2.5 Scalability Trilemma and Possible Solutions

Currently, each node being part of an Ethereum blockchain stores the entire state, i.e. account balances, contract code and storage, etc. and also processes all transactions sequentially. This approach provides a high amount of security, yet greatly restricts scalability. This is due to the reason that the transaction processing capacity of the entire system is limited to the capacity of a single node. There is no parallel processing in the Ethereum network and all transactions have to go through one mining pipeline. In a large part, this is why the throughput of Ethereum is currently capped at roughly 15 transactions per second in the public network [9]. It seems that, for blockchain systems, there is currently a trade-off between decentralisation, security and scalability. This relationship is known as scalability trilemma [9] in the Ethereum community. It describes that a blockchain can only achieve two out of the three mentioned properties without compromising either one. Decentralization basically means that the blockchain software can be run on nodes with a very limited amount of resources, e.g. a regular laptop. Scalability is defined as being able to process more transactions than a single node is able to process on its own.

In order to drastically increase the scalability, the Ethereum key developers focus on a combination of two main approaches. The first approach involves requiring only a small percentage of nodes to see and process every transaction of the network, i.e. allowing transactions to be processed in parallel. The way to achieve this is by horizontal partitioning. This procedure is also known as sharding and has already been implemented in various distributed databases. The basic idea behind sharding is to group subsets of nodes into several partitions. Each shard only processes transactions specific to this shard. This implies that each shard has its own transaction history and state. A simple sharding schema would be to use shards which each process the transactions associated with one particular asset. However, in more advanced forms of sharding cross-shard communication capabilities are also imaginable. More information about Ethereum and sharding can be found at Ethereum's sharding FAQ [9]. The second approach involves creating additional chains which co-exist next to the main blockchain. Some of the transactions are taken away from the main chain and performed off-chain instead. The ideas behind this approach can be subsumed under the term side-chain or layer 2 protocol. Examples of layer 2 protocols in the context of Ethereum are Plasma [45] and Raiden [8]. Plasma allows to create child blockchains which can be attached to the main Ethereum blockchain. As these child blockchains can also spawn their own child-chains, one can think of Plasma as a hierarchical tree of side chains. All these side chains periodically transfer information back to the main chain. The Raiden Network is an off-chain scaling solution for transferring tokens between entities without the need for global consensus. The basic idea behind Raiden is to take transactions away from the main chain and build a payment channel between two parties. This is achieved using signed and hash-locked transfers which have been previously set up via on-chain deposits.

The implementation of these two main approaches and the switch from a PoW to a PoS consensus mechanism in Ethereum is tackled under the Ethereum 2.0 [2] initiative.

CHAPTER 3

Related Work

This chapter addresses related work in the context of performance and scalability measurements on blockchains and smart contract platforms. Since some identified sources directly focus on Ethereum, others on different blockchain platforms and some on general aspects, the findings where categorized into sections accordingly. For related work which directly addresses performance and scalability of Ethereum (see section 3.1), limitations compared to the present work are pointed out.

In summary, although some identified related work may partially overlap with this diploma thesis, to the best of the author's knowledge, none of the identified sources have fully covered the effects on performance and scalability of all parameters of the Ethereum platform listed in section 4.1.1 so far.

3.1 Performance and Scalability of Ethereum

The two papers by Dinh et al. **BLOCKBENCH: A Framework for Analyzing Private Blockchains** [29] and **Untangling Blockchain: A Data Processing View of Blockchain Systems** [28] are considered to be one of the most relevant papers for this thesis. The authors claim to describe the first evaluation framework for analysing private blockchains. It is stated that any private blockchain can be integrated to the described framework and benchmarked against pre-defined workloads. In order to compare different blockchains, four abstraction layers for the framework were identified. These abstraction layers are application, execution engine, data model and consensus. "The consensus layer contains protocols via which a block is considered appended to the blockchain. The data layer contains the structure, content and operations on the blockchain data. The execution layer includes details of the runtime environment support blockchain operations. Finally, the application layer includes classes of blockchain applications." [29] For benchmarking different blockchains and their layers, different workloads are used. These workloads are divided into "macro benchmark workloads" for performance benchmarking of the

application layer (e.g. key-value storage or a contract implementing a pyramid scheme) and "micro benchmark workloads" (e.g. a plain contract doing nothing or an input-output heavy workload) for benchmarking the lower layers. For the evaluation of the proposed framework, Ethereum, Parity and Hyperledger Fabric were used as different private blockchains. Throughput, latency and scalability were chosen as performance metrics to be investigated. For experiments, a 48-node commodity cluster was used. Each node in this cluster had an E5-1650 3.5GHz CPU, 32GB RAM, a 2TB hard drive, running Ubuntu 14.04 Trusty, and was connected to other nodes via a 1GB switch. The measured throughput and latency at the application layer level (macro workloads) are displayed in Figure 3.1 , while Figure 3.2 illustrates measured scalability. Peak performance was measured with 8 servers and 8 concurrent clients. Over a period of five minutes each client sent transactions to a server with a varying rate (8 tps to 1024 tps). The figures indicate that the measured peak throughput of Ethereum was approximately 255 to 284 tps, while the latency was roughly 92 to 114 seconds. Moreover, it is shown that the type of workload influences the throughput and latency. More precisely, there is a drop of approximately 10% in throughput and a 20% increase in latency when comparing different smart contracts. According to the authors this might be due to more writing to the blockchain's states. Regarding scalability, Figure 3.2 clearly illustrates, that with an increasing amount of nodes, the throughput decreases almost linearly, while the latency increases nearly exponentially. In order to measure the effect of different `block sizes`, the `gasLimit` parameter of the Geth client was tuned. Surprisingly, results show that an increase in block sizes does not improve the overall throughput. It is stated that this may be because an increase in block size leads to a proportional decrease in block generation rate. In addition to plots visualizing the performance of different blockchains, there are also statements such as "Hyperledger outperforms Ethereum and Parity across the benchmarks" [29] and "the consensus protocols are the main bottlenecks in Hyperledger and Ethereum" [29] among the results. After comparing the performance of the three mentioned blockchains to the performance of an H-Store database, it is also noted that "the results demonstrate that these systems are still far from displacing current database systems in traditional data processing workloads" [29].

Although the two mentioned papers show an overlap with this diploma thesis to some extent, the evaluation of Ethereum described in [29] and [28] was apparently conducted with only one consensus algorithm (PoW). Furthermore, the type of nodes, i.e. their configuration, was not varied and the Ethereum client used (Geth v1.4.18) is outdated as of today.

Another important related work is the study of Pongnumkul, Siripanpornchana and Thajchayapong **Performance Analysis of Private Blockchain Platforms in Varying Workloads** [44]. This work studied the performance and limitations of Hyperledger Fabric and Ethereum with varying numbers of transactions. As metrics the execution time, latency and throughput were measured. The results show that Hyperledger Fabric outperforms Ethereum at all metrics and that "both platforms are still not competitive with current database systems in terms of performances in high workload scenarios" [44].

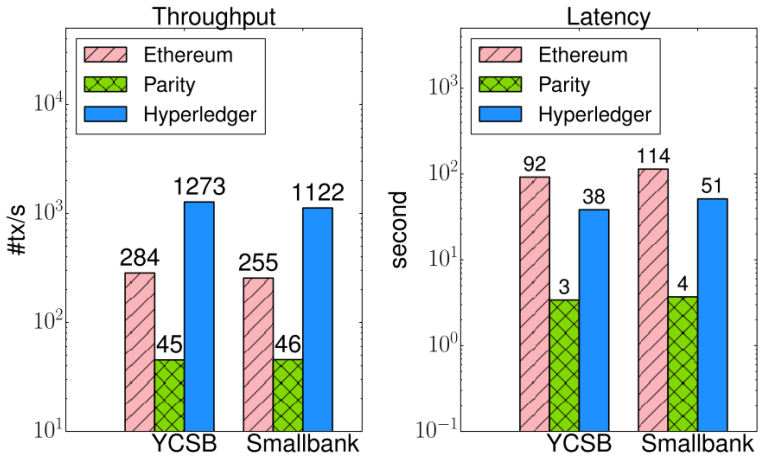

Figure 3.1: Blockbench Evaluation Results: Peak Performance for Different Workloads [29].

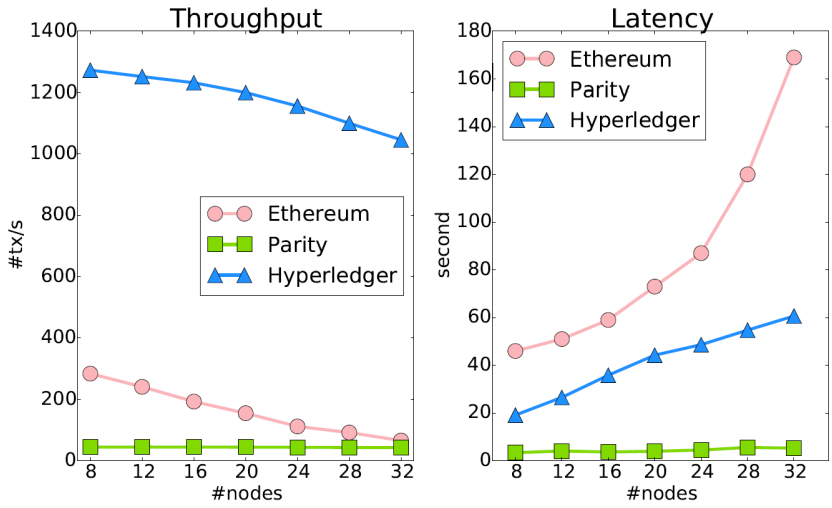

Figure 3.2: Blockbench Evaluation Results: Node Scalability [29].

Other findings reveal that the latency increases with a greater amount of transactions to be handled. The experiments were conducted on a single Amazon AWS EC2 c4.2xlarge instance with an Intel E5-1650 8 core CPU, 15GB RAM and a 128GB SSD hard drive running Ubuntu 16.04. The Geth client used was version 1.5.8 and the workload specified was an application for money transfer. For interacting with the Ethereum node, the Web3.js JavaScript API was used. The procedure for the experiments was that a client sent N requests to the blockchain platform in an asynchronous manner, i.e. requests were sent without waiting for a response. The number of requests N ranged from 1 to 10000. The measured throughput can be seen in Figure 3.3 and the average latency

measured is illustrated in Figure 3.4. For Ethereum, a throughput in the range of 4.69 to 38.93 transactions per second and a latency ranging between 0.21 and 484.78 seconds was measured.

In contrast to the present thesis, a key limitation of the described work is that the consensus mechanism was turned off. Furthermore, the analysis was only conducted on a blockchain network with only one single node. Moreover, the configuration of the node was not varied and other parameters such as the mining `difficulty` of Ethereum were not investigated either.

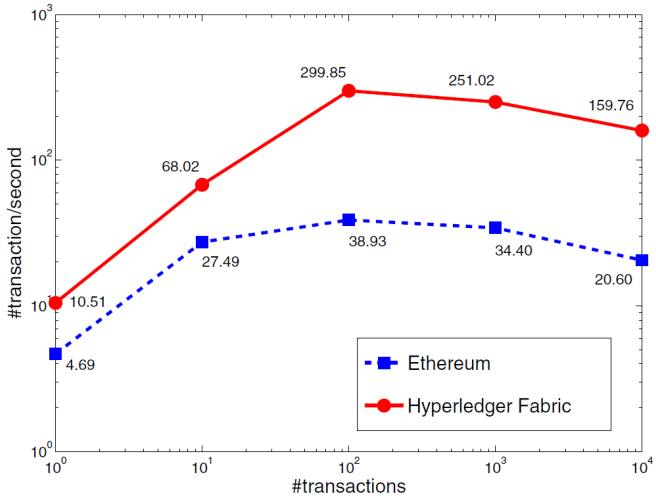

Figure 3.3: Average Throughput of Ethereum and Hyperledger with Varying Number of Transactions [44].

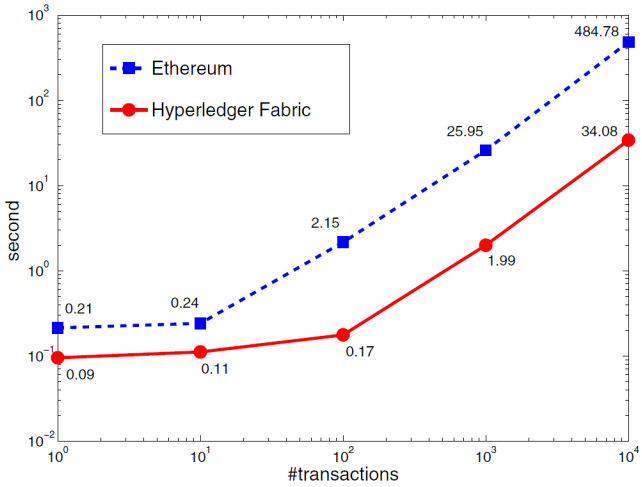

Figure 3.4: Average Latency of Ethereum and Hyperledger Fabric with Varying Number of Transactions [44].

The paper **Performance Analysis of Ethereum Transactions in Private Blockchain** [46] studies the effect of different Ethereum clients with respect to performance. In the study, the Ethereum clients Geth and Parity were compared. Parity, which has been written in the Rust programming language, is considered to emphasise efficiency due to its faster syncing process [46]. Indeed, the performance results show that Parity processes transactions significantly faster than Geth (see figure 3.5). Using the same system configuration on average, transactions are 89.8% faster with the Parity client when compared with the Geth client. Another important result is that the average time for processing transactions on a client decreases with an increasing amount of RAM. For processing 2000 transactions, the average time for each transaction using the Geth client with 4 GB of RAM was measured with 199.445 milliseconds. However, when using the same client with 24 GB of RAM 147.068 milliseconds on average were measured. It is also mentioned that an increasing amount of RAM would affect the Geth client more than the Parity client.

Figure 3.5: Performance Differences of the Parity and Geth Client[46].

Even though the analysis in [46] was conducted with different types of nodes (different amount of RAM is mentioned), there is no information about the consensus algorithms and its parameters (e.g. mining `difficulty` or `block frequency`) used. Apparently, also the amount of nodes were not varied during the experiments.

Chen, Zhang, Shi, Yan and Ke compared the performance of blockchains and relational databases in their work **A Comparative Testing on Performance of Blockchain and Relational Database: Foundation for Applying Smart Technology into Current Business Systems** [22]. For testing purposes, Ethereum was chosen as representative for the blockchain technology and MySQL was chosen as representative for relational databases. Concerning test results, it has been found that, in Ethereum, the amount of transactions processed per second was affected by the data volume and can be as low to 0.49 per second. In comparison, MySQL has a much higher average

number of processed transactions - 819.67 per second on average. The throughput in bytes per millisecond, was measured to be 1000 times higher in the MySQL database when compared to the private Ethereum network. While up to 3.2 bytes per millisecond were measured for the private blockchain, the relational database showed up to 5000 bytes per millisecond. As for the execution time of each transaction, the private blockchain needs more than 1660 times longer than the MySQL database. Although there is no apparent relation of data volume to time spent in the relational database, the private blockchain shows an increase in time spent with a growing data volume. In summary, the test results imply that private Ethereum blockchains may not yet be suitable for data intensive use cases.

When comparing the study in [22] with the present work, some differences become explicit. Although seven differently configured machines were used for the test, there is no information provided on how the machine's configuration has affected the measured metrics. Besides mentioning that the results varied among different machines, the results were only provided for a single configuration. In addition, the configuration of the machines specified only resemble consumer machines such as laptops. Moreover, the size of the private Ethereum blockchain was fixed to six nodes and not varied. The consensus algorithms and their parameters, e.g. mining `difficulty`, were not included in the study either.

Zheng et al. propose overall performance metrics and a performance monitoring framework with a log-based method in their paper **A Detailed and Real-time Performance Monitoring Framework for Blockchain Systems** [57]. The experimental results in their study show that the framework can be used for detailed and real-time performance monitoring of blockchain systems. Proposed metrics are divided into metrics for the users and metrics for developers. For the overall performance of different smart contracts on the Ethereum blockchain Zheng et al. propose Transaction Per Second, Average Response Delay, Transaction Per CPU, Transaction Per Memory Second, Transaction Per Disk I/0 and Transaction Per Network Data as metrics. Furthermore, via dividing the transaction processes into sub-processes, metrics especially relevant for developers are proposed. Examples of these metrics, which are more focused on the lower levels, are Peer Discovery Rate, RPC Response Rate, Transaction Propagating Rate, Contract Execution Time, State Updating Time and Consensus-Cost Time. Regarding performance measurements of Ethereum smart contracts, it was analysed that the throughput varies depending on the types of smart contracts. The fastest throughput (7.9 tps) was obtained with contracts mainly storing arrays. Contracts with many loop operations as well as contracts with many account-related operations showed a great consumption of computing resources and a moderate to low throughput (2.4 tps to 0.56 tps). The authors stated that, as for the account-related operations, this may be due to hash computing of the Ethereum "World State". After running the tests on the public Ethereum blockchain, experiments on a private Ethereum blockchain with different numbers of peers (1 to 4 peers) were conducted. Finally, the authors of [57] concluded that peer discovery, transaction propagation and the consensus-cost represent the bottlenecks in Ethereum.

Although reference [57] provides detailed metrics for measuring performance on the Ethereum blockchain, the performance analysis lacks some parameters. For example, only one consensus algorithm (PoW) is covered. Furthermore, parameters such as mining `difficulty` or the `block size` were not varied. In addition, other configuration possibilities such as the configuration of nodes were not addressed.

In the paper **On the Security and Performance of Proof of Work Blockchains** [33] Gervais et al. introduce a quantitative framework to analyse the security and performance tradeoffs of various consensus and network parameters of Proof-of-Work blockchains. In order to evaluate different blockchain instances from a performance and security perspective, a Bitcoin blockchain simulator was constructed. By using the simulator, different blockchain parameters such as the `block interval` and the `block size` were varied. While the block size was varied from 0.1 MB to 8 MB, the block interval was varied from 0.5 seconds to 25 minutes. The obtained results indicate that "different parameter configurations can yield the same throughput though with different security provisions (due to a different stale block rate)" [33]. For example, a block size of 0.25 MB and a block interval of 15 seconds, resulting in a throughput of 66.7 transactions per second, can lead to a lower security level than a block size of 0.5 MB and a block interval of 30 seconds, also resulting in a throughput of 66.7 transactions per second. Interestingly, it is stated that a throughput of 60 transactions per second can be reached with existing PoW blockchains without significantly affecting the security of the blockchain.

However, in contrast to this work, the paper neither includes other consensus variants (e.g. Proof-of-Authority) and its parameters, nor does it include different amounts and types of nodes in the blockchain network.

A prediction model which was derived from the core structure of Ethereum's "World State" is proposed in the paper **A Method to Predict the Performance and Storage of Executing Contract for Ethereum Consortium-Blockchain** [56]. The proposed model aims at predicting the performance and storage of executing contracts based on the transaction volume. The model itself consists of formulas which were derived via regression analysis. Test results show that the performance of Ethereum significantly reduces when the transaction volume reaches a certain scale. As an example, it is mentioned that with a block generation rate of one block per second, the number of transactions per second decreased from 200 to 100 as the transaction volume reached one million. Another result of this work is that the relationship of performance or storage increment and transaction volume is of factor log(n).

The major drawback of this approach is that it only depends on the amount of transactions. The derived formulas, e.g. for calculating the time cost, do not include other variables such as the consensus algorithm, the amount or the configuration of the nodes in the blockchain network.

3.2 Performance and Scalability of Other Blockchain Platforms

The work **Bitcoin-NG: A Scalable Blockchain Protocol** [32] deals with the issue of Bitcoin's scalability limits via proposing a new blockchain protocol which is designed to scale. Variables such as `block frequency` and `block size` were varied to make suggestions for a better blockchain protocol. Regarding experiments with different block sizes, it was observed that the transaction frequency increases with the block size. Thus, increasing the block size could result in improved throughput. However, increasing the block size at the same time decreases the level of security since large blocks take longer to verify and propagate and the chance for forks increases. Concerning the block frequency it could be shown that a higher value (achieved via a lower mining `difficulty`) reduces Bitcoin's consensus latency as transactions are recorded in the blockchain at a higher frequency. Overall, the results of the paper indicate that "it is possible to improve the scalability of blockchain protocols to the point where the consensus latency is limited solely by the network diameter and the throughput bottleneck lies only in node processing power" [32].

A blockchain benchmarking tool which is not finished yet and still in "incubation" status is **Hyperledger Caliper** [35]. As stated on the website of the project, the tool should allow measuring performance of various blockchains with a set of already predefined use cases. However, as stated on the Github page of the project [34], it is also possible to define test cases on one's own. As performance indicators currently Success Rate, Transaction/Read throughput, Transaction/Read latency (minimum, maximum, average, percentile)and Resource Consumption (CPU, Memory, Network IO etc.) are mentioned [34]. While the tool can already be used for some Hyperledger blockchains such as Fabric, Sawtooth or Iroha, Ethereum has not yet been supported.

One analysis that makes use of a modified version of the Caliper tool to analyse Hyperledger Fabric is the work **Performance Analysis of Hyperledger Fabric Platforms** [41]. The authors first remark that there is no clear methodology for evaluating and assessing different blockchain platforms and then propose their methodology for analysing the differences of Hyperledger Fabric v0.6 and v1.0. The evaluation of the two platforms includes metrics such as execution time, latency, and throughput. The workload was varied up to 10,000 transactions and also the scalability was investigated via varying the number of nodes up to 20. The results show that Hyperledger Fabric v1.0 consistently outperforms Hyperledger Fabric v0.6 across all metrics. However, the performance of Hyperledger Fabric v1.0 performance has still not reached the level of traditional databases. Furthermore, it has been found that while Fabric v0.6 is affected by the number of peers, Fabric v1.0 maintains the same performance if the number of nodes is increased. Via figures, one can observe that for a dataset of 1000 transactions the performance of Fabric v0.6 decreases from 155 transactions per second (tps) for 1 peer to 126.1 tps for 16 peers. However, the performance of Fabric v1.0 does not seem to follow a trend.

The paper **Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform** [52] includes an empirical study for characterizing the performance of Hyperledger Fabric. In addition, the authors emphasise finding potential performance bottlenecks of the system. Following a two-phased approach, first the impact of different configuration parameters such as `block size`, endorsement policy, channels, resource allocation, state database choice on the transaction throughput and latency were investigated. In the second phase, the authors focused on optimizing the performance of Hyperledger Fabric v1.0. The experimental results indicate that endorsement policy verification, sequential policy validation of transactions in a block, and state validation and commit (with CouchDB) were the three major bottlenecks of Hyperledger Fabric v1.0. An observation, which might also be relevant for the Ethereum blockchain, is the throughput versus latency trade-off when varying the `block size`. It was observed that, when increasing the transaction arrival rate, the throughput increased linearly and the latency also rose. When reaching a saturation point, the throughput flattened out and the latency increased significantly.

In the work **Performance Analysis and Application of Mobile Blockchain** [49] a new application which uses blockchain technology to secure transactions in the m-commerce domain is introduced. Via experiments, the authors evaluated the performance of their application. The results indicate that the mining process of a blockchain can be executed on mobile devices using their proposed Android core module. Furthermore, according to the authors, the results show that "blockchain can be an efficient security solution for future m-commerce" [49].

Spasovski and Eklund compared the performance and scalability of a web-based groupware communication application using both non-blockchain and blockchain technologies in their paper **Proof of Stake Blockchain: Performance and Scalability for Groupware Communications** [47]. For the blockchain technology, the private platform Tendermint, which has a Proof-of-Stake consensus algorithm, was used. The results show that when increasing the number of instances used, the blockchain implementation only scales linearly until a throughput threshold is reached.

3.3 General Considerations on Performance and Scalability

Cronman et al. address the bottlenecks in Bitcoin which are limiting higher throughput and lower latency in their paper **On Scaling Decentralized Blockchains** [25]. The results show that a re-parametrisation of `block size` and `block intervals` may have a positive effect on performance and scalability. Nevertheless, it is mentioned that a re-parametrisation can only be the first step towards better scalability as the network performance induced by Bitcoin's current peer-to-peer network architecture limits the effect of re-parametrisations. For better scalability the authors propose a rethinking of the design of Bitcoin and discuss topics such as sidechains and sharding (see section 2.5).

The authors of the paper **On Scaling and Accelerating Decentralized Private Blockchains** [55] address the issue of blockchain scalability via proposing an architecture which is more suitable for the requirements of a private blockchain. The three strategies proposed to improve scalability are: optimization of block construction, `block size` and time control optimization, and transaction security mechanism optimization. For the verification of their work, experiments were conducted which studied the relationship of block size and block construction. The results show that, when increasing the block size, the number of transactions processed per second increases at first, but decreases after a specific level is reached.

Li, Federov, Sforzin and Karame state that existing Byzantine tolerant permissioned blockchains only scale to a limited number of nodes. Furthermore, it is stated that blockchain technology is still not mature enough to satisfy industry standards. Thus, the authors propose a novel blockchain architecture in their work **Towards Scalable and Private Industrial Blockchains** [39]. The proposed architecture aims at boosting the scalability of private blockchains via the concept of satellite chains. In the suggested architecture, these satellite chains form interconnected sub-chains of a single blockchain system. Hence, blockchains could run with different consensus protocols in parallel via satellite chains.

A different design of blockchains is proposed in the paper **Improve Blockchain Performance using Graph Data Structure and Parallel Mining** [37]. The authors of the mentioned work argue that, for improving the performance and scalability of blockchains to a significant level, simply tweaking blockchain parameters such as block size is not enough. In order to overcome the current limitations, a new blockchain design/architecture is proposed. The key elements of this design are a change of the current chain data structure to a graph data structure, a change from the single miner to a multiple miner approach, and the introduction of data sharding. The data structure presented is named GraphChain and has been inspired by the concept of the Directed Acyclic Graph (DAG) which is used in the IoTA blockchain [37]. The authors state that the reason for changing the data structure was that a way for replacing the sequential operations with parallel operations was needed.

CHAPTER 4

Results

Based on the aim of the work (see section 1.3), this chapter is divided into three parts and comprises the results of the present work.

First, the developed concept for measuring performance and scalability of private Ethereum networks is presented. This concept comprises theoretical considerations as well as an experiment setup which allows to practically gather measurements regarding performance and scalability of differently configured private Ethereum networks. Using a dataset of roughly 4,000 measurements, an analysis of performance and scalability is introduced in the second part. This analysis includes several figures which illustrate effects of different parameters. Finally, building on the knowledge obtained from the analysis, a discussion of bottlenecks is addressed in the last section of this chapter.

4.1 A Concept for Measuring the Performance and Scalability of Private Ethereum Networks

Building on the information obtained from related work and several sources regarding Ethereum, e.g. the whitepaper [15] and the Ethereum Github repository [13], a concept for measuring the performance and scalability of private Ethereum networks has been developed.

The concept consists of several parts which consistently build on each other. First, a theoretical discussion of parameters which may affect the performance and scalability of Ethereum in the private setting, is presented. Next, metrics which are going to be used to analyse the subsequent experiments are defined. Third, the practical experiment setup is introduced. After an Ethereum client has been selected and further technology decisions have been made, identified parameters from the first step can be mapped onto the target architecture. Finally, the measurement process is presented.

4.1.1 Identified Parameters

The question of which parameters may affect the performance of private Ethereum blockchains was one of the first questions to be answered. In order to put the context of blockchain performance parameters to a real world scenario, a thought experiment including an assembly line is used. When imagining an assembly line where workers put printed messages from a stationary stack nearby into boxes which are located on the assembly line, understanding the effects of different parameters becomes more easily. Translating the assembly line thought experiment to the blockchain context, the messages represent transactions, the message stack is represented as a transaction queue and the boxes are the blocks of a blockchain.

One such parameter is the distance between two succeeding boxes. In the blockchain context this parameter can be interpreted as the `block frequency` or `block interval`. In theory, minimising the distance between two succeeding boxes allows the workers to put the messages into the boxes with a faster pace, i.e. the message stack may be minimised. Hence, the time a message waits until it is put into a box is smaller and overall the throughput of the system is higher. Thus, it is assumed that an increase in block frequency improves the performance of a blockchain system.

This assumption is also supported by related work. For example, the authors of the work "On Scaling Decentralized Blockchains" [25] mention that a re-parametrisation of block intervals/block frequency may have a positive effect on performance at the Bitcoin blockchain. However, the block propagation time may impose restrictions on how low the block frequency may be. This is due to the fact that the propagation of blocks in the network takes its time and is not instant. With a very high block frequency a potential risk is that a lot of uncle/stale/orphan-blocks[1] are created [16].

Another parameter which can be included in the assembly line analogy is the size of the boxes, i.e. the `block size` in the blockchain context. Theoretically, increasing the size of the boxes allows more messages from the message stack to be put in the boxes until the next box arrives. Hence, lower waiting times for messages in the message stack and an improved overall throughput are possible.

Turning to the blockchain world and to related work regarding effects of block size on performance, contradictory statements have been found. While there exists work that indicates a positive effect of block size increase on performance such as the work "On Scaling Decentralized Blockchains" [25], surprisingly one is also able to find a source which does not imply a positive effect of block size on performance. Results in the paper about the Blockbench framework [29] by Dinh et al. show that an increase in block size does not improve overall throughput. Dinh et al. state that this may be due to the block generation rate decreasing proportionally with bigger blocks.

[1] An uncle/stale/orphan-block is a block which was validly mined but was not appended to the main chain. They emerge from situations where different miners generated a block nearly simultaneously. As the miners propagate their different block to the network a dubious state is created where consensus about which one of the blocks should be rejected still has to be found.

The next parameter which may affect the performance of a blockchain system is the `workload type`. Turning to the assembly line thought experiment, one may assume that messages which are different in dimensions and which may need additional treatment (e.g. being put in an envelope) before being put into the boxes lead to different performances. In the context of Ethereum, smart contracts are programmed for a specific purpose and thus exhibit different content. Executing one smart contract might need a lot of Ethereum state changes, while others only comprise a few read statements. More precisely, the Ethereum Virtual Machine (EVM), which executes the smart contract instructions, may show a different runtime for different execution instructions. Therefore, it is assumed that the type of smart contract has an effect on the performance of a blockchain.

Indeed, sources such as the paper about the Blockbench framework [29] and a paper by Zheng et al. [57] mention that a different performance was measured for different types of workloads. For example, Dinh et al. observed a drop of approximately 10% in throughput and a 20% increase in latency when comparing different smart contracts.

When focusing on the workers at the assembly line, one can identify another parameter which may affect the performance. Using more skilled workers translates to more powerful nodes in the blockchain network. As the amount and type of CPU and RAM a node has is known as hardware configuration, the parameter describing the computational power of a node is called `node configuration` henceforth. In the context of Ethereum, the node configuration may be especially interesting when using the Proof-of-Work (PoW) consensus algorithm. This is due to the mechanics of Proof-of-Work. In a PoW system new blocks are created via the process of mining and the mining pace of a node basically translates to its hash rate. As a higher hash rate leads to a decreased time for finding a new block, the overall performance may be increased when using more powerful nodes, i.e. nodes with a higher hash rate.

Although one might want to know how much the computational power of a node affects the performance, only two papers to some extent include differently configured machines. The first paper [46], comprises the effect of different amounts of RAM of a machine on the average time needed to process a transaction. The authors of the second paper [22], which present testing results for nine different metric, mention that their results vary among different machines. However, the results are only displayed for one out of seven distinct machine configurations. Since the author of the present work strongly assumes a positive effect of more powerful nodes on performance, the question of scalability, i.e. how much a more powerful node improves the performance, is of special interest as regards this parameter.

Besides improving the performance on one single assembly line, i.e. vertical scaling, a straightforward approach may also be to increase the number of assembly lines, i.e. horizontal scaling. In the context of a blockchain system, this approach may be interpreted as increasing the amount of nodes in the network. When using a Proof-of-Work (PoW) consensus algorithm, one may argue that adding additional nodes boosts the overall mining power, i.e. hash rate, of the network. Thus, the performance of the system should be improved. However, when raising the number of nodes in the network the

communication overhead and the consensus cost also increase since new transactions and blocks have to be propagated to other nodes in the network and consensus has to be found. Additionally, the Ethereum client may increase the mining difficulty on its own when the network grows in size. This procedure is known from the public Ethereum network where Ethereum clients try to keep the block generation frequency within a target range (e.g. 10 to 19 seconds). When using a Proof-of-Authority (PoA) consensus algorithm, the block generation frequency is defined once and should not vary with an increasing or decreasing amount of nodes. Thus, theoretically the performance may only decrease marginally due to network overhead.

Turning to findings in related work, similar to the node configuration parameter, only a few sources [29, 57] could be identified which discuss the effect of network size on performance and scalability. Surprisingly, results obtained by Dinh et al. in [29] suggest that the overall performance of the system actually decreases when adding nodes to the network. This phenomenon is illustrated in figure 3.2. As the effect of the network size on the performance and scalability of a smart contract platform needs more clarification, the `network size` is added to the list of parameters for the subsequent analysis.

Next, one can find another parameter when considering the input size, i.e. the height of the message stack of the assembly line. With regard to the blockchain context, the amount of input is the amount of transactions to be processed, i.e. the `workload quantity`. Usually, the performance of a system is not optimal if the system does not utilize its full workload potential. Hence, in an optimal setting the workload quantity equals the workload potential of a system. In a non-optimal setting, e.g. with an extensive workload, negative effects may appear. For example, one can argue that, assuming that the workload quantity is higher than the amount of transactions fitting in a single block, the time for processing a single transaction increases with the amount of transactions to be processed. In addition, another negative effect of a high workload may be that more transaction propagation has to be handled. Thus, the performance could be negatively affected due to network communication overhead and consensus costs.

Negative effects of workload quantity on performance of Ethereum in a private setting were also reported in the paper "Performance Analysis of Private Blockchain Platforms in Varying Workloads" [44]. As figure 3.4 illustrates, the average latency of a single transaction increases with growing workload quantity. Considering the throughput of the whole system, the impact of a too low and too high workload quantity can clearly be seen in figure 3.3. Although the effect of workload quantity on performance of Ethereum seems to be present, the decision was made not to include the parameter in the experiments of this work. This is due to two reasons. First, an extensive evaluation solely on the effect of varying workload quantity has already been published in the work "Performance Analysis of Private Blockchain Platforms in Varying Workloads" [44]. Second, due to the limited amount of resources available and the amount of already identified parameters, another reason for not focusing on the respective parameter was apparent. Thus, based on workload sizes used in the related work, it has been determined to set the workload quantity for the experiments to a static number of 1000 transactions.

Another parameter identified is the `amount of miners` (when using PoW as consensus algorithm) or the `amount of sealers` (when using PoA as consensus algorithm) in the network. In the assembly line thought experiment, this parameter can be imagined as the number of workers actually working and not having a break. Since transactions and blocks have to be propagated in a blockchain network and consensus has to be found the network communication overhead and the consensus costs increase with an increasing amount of nodes. Therefore, the author of the present work assumes a marginal negative effect of an increasing amount of non-mining/sealing nodes in a network.

However, to the author's best knowledge, no studies that discuss the effect of additional non-mining/sealing nodes in a private blockchain network can be found. Thus, another possible parameter has been identified. Nevertheless, the parameter was also not included for the later experiments and analysis. Similar to the workload quantity parameter, this is due to limited resources available for this work and the amount of already identified parameters.

Finally, the last parameter identified is the environment where the process is executed, i.e. the working atmosphere. In the context of Ethereum this can be interpreted as the `blockchain client and API` used for interacting with the blockchain. There are a few Ethereum clients available which each may exhibit different performances for syncing the blockchain or processing the transactions. Examples for Ethereum clients are Geth (Go), Parity (Rust), cpp-ethereum (C++) or pyethapp (Python).

Indeed, performance differences between the most popular clients Geth and Parity [10] have been reported by Rouhani and Deters [46]. However, due to the massive extra effort and expenses needed to test parameters on different Ethereum clients the blockchain client parameter was not included for the later experiments either.

In summary, all identified parameters which may have an impact on performance and scalability in a private Ethereum blockchain network are listed in table 4.1 below. Parameters included in the experiments of this work (see 4.2) are in bold.

Table 4.1: Identified Performance and Scalability Parameters

Parameter	**Additional Description**
Block frequency	Time between two succeeding blocks
Block size	Amount of transactions fitting in a block
Workload type	Smart contract
Node configuration	CPU, RAM, Network of a node
Network size	Amount of nodes
Workload quantity	Amount of transactions to be processed
Amount of miners/sealers	Actively participating nodes
Blockchain client and API	e.g. Geth or Parity, web3.js or web3.py

4.1.2 Definition of Evaluation Metrics

After performing the literature study, it was surprisingly noticed that there exists no shared definition of evaluation metrics for performance and scalability in the blockchain area. Although nearly all studies identified and discussed in related work (see 3) share the concept of throughput and that it is measured in Transactions Per Second (TPS), there exists no universal understanding of additional metrics next to throughput. Sometimes metrics targeting the same purpose are denoted differently and sometimes they do not express the same underlying purpose despite being labelled equally. For example, there exists a study which defines latency as "the response time per transaction" [29], while another study defines it as "the difference between the completion time and the deployment time (t2-t1)" [44]. An example for using different labels is a study of Zheng et al. where the "gap between the time when transaction is firstly sent into the network and the time when it is confirmed" [57] defines the average response delay instead of the latency.

Thus, in order to avoid possible misunderstandings concerning the later experiments and the analysis it is vital to define included evaluation metrics and their meaning beforehand.

Focusing on the most overlapping metrics and definitions in related work the evaluation metrics to be used for this thesis were determined. Additionally, due to the high amount of parameters already identified (see 4.1.1) the decision was made to keep the amount of metrics low and to focus on simple and high level metrics. Hence, metrics focusing the hardware layer, e.g. degree of CPU or memory utilisation, were excluded. Moreover, metrics focusing on the security of the blockchain were excluded due to the limited resources and scope available for this work. The final evaluation metrics used in the experiments and the analysis of performance and scalability (see 4.2) are defined in table 4.2.

Table 4.2: Definition of Evaluation Metrics

Metric	**Description**
Throughput	The number of successful transactions per second (TPS)
Latency	The time difference in seconds between transaction submission and the time when a transaction is successfully included in a block
Scalability	The changes of throughput and latency when altering a parameter (e.g. the network size or the hardware configuration of a node)

4.1.3 Experiment Setup

After deciding on specific parameters for the experiments and the definition of evaluation metrics in the previous two sections, the question of how one can field-test the effect of the respective parameters on performance and scalability in a private Ethereum setting was tackled.

The aim for the experiment setup was to design and develop a system which allows to gather measured data in a fully automated way.

In order to develop such a system, the decision for an Ethereum client had to be made as a first step. As already briefly stated in the identified parameters section, Ethereum offers various different clients, e.g. Geth (Go), Parity (Rust), cpp-ethereum (C++) or pyethapp (Python). Although the paper "Performance Analysis of Ethereum Transactions in Private Blockchain" [46] states that the Parity client processes transactions significantly faster than the Geth client, the latter has been chosen as client to be used for the experiments. This is mainly due to the reason that the **Geth** client is Ethereum's default client (which can be observed on the Ethereum Github repository [3]). Moreover, Geth seems to be the most used Ethereum client [10] and, in contrast to the Parity client, it already has a built-in miner.

The decision for an Ethereum client has also affected the decision for specific consensus algorithms. In theory, Ethereum supported three different consensus algorithms when developing the present concept - Proof-of-Work (PoW), Proof-of-Authority (PoA) and Proof-of-Stake(PoS). However, not all clients supported each consensus variant. In particular, the Geth client only supported PoW and PoA and did not include PoS. In fact, the only client which supported PoS was the pyethereum (Python) client. In case one wanted to include all consensus variants and also the most used client, two different clients would have to be used. Due to extra effort and expenses needed to support different clients, the decision has been made to solely focus on the Geth client. Thus, although Ethereum would theoretically be able to run with the PoS consensus algorithm via its Casper protocol, defined in the paper "Casper the Friendly Finality Gadget" [20], only **PoW and PoA** have been chosen for the experiments.

Second, an interface for interacting with the Ethereum nodes had to be selected. Using a JavaScript runtime environment (JSRE), Geth allows interaction directly via an interactive JavaScript console or via script mode. In both cases, a JavaScript API is exposed which offers methods for Ethereum blockchain interaction. However, when one wants to include Geth in an environment of several applications and processes, using Geth's console or script mode might become cumbersome. For this purpose, Geth offers JSON-RPC [2] endpoints for its JavaScript APIs (e.g. web3, eth or admin). Additionally, the **Web3.js** JavaScript library is offered which communicates to nodes via RPC calls. Using this library, one can interact with the Ethereum network from inside another application.

[2] JSON-RPC is a stateless, light-weight Remote Procedure Call (RPC) protocol which uses JavaScript Object Notation (JSON) as data format.

Due to the reasons that the aim for the experiment setup is to achieve a high level of automation and that the **web3.js** API [31] is the most convenient and de facto standard interface for interacting with Ethereum, the decision has been made to use the library in a JavaScript application. As a runtime environment for the application **Node.js** was selected. Node.js has been chosen since it is the most popular server-side JavaScript runtime environment. In fact, according to the Stackoverflow Developer Survey 2018, it is the most commonly used framework of all technologies included in the survey [48].

Next, the computing environment had to be selected. Since an on-premise server cluster was not available, renting computing power from a cloud provider was the obvious choice. Due to personal experiences made in the past and a vast supply of differently configured computing instances, Amazon Web Services (AWS) has been chosen. Using the **Elastic Compute Cloud (EC2)** service, one can rent computing power for different purposes on demand. Since the experiments are carried out on an Ethereum network with more than one node (see network size parameter in 4.1.4), deploying all needed instances separately would take a tremendous amount of time and effort. After the AWS instance deployment has been performed, a blank computing environment is available. Hence, a state where the desired experiments can be started still needs to be established. In this case, the required software, e.g. the Geth client and the described JavaScript application, have to be installed and started on each instance. Thus, a service which allows to automatically provision a set of pre-specified EC2 instances and which is also able to automatically handle the software installation and start-up process is needed.

After having analysed available AWS products, the author of the present work came to the conclusion that **AWS Cloudformation** is the required service needed. Using a template, the service provisions a stack of defined AWS resources. Within the scope of the present work, such templates are used to describe AWS EC2 instances in various amounts and configurations. For illustration purposes, an excerpt of a developed Cloudformation template in JSON format is listed on the next page. It can be seen that JSON key-value pairs are used to describe the configuration of an EC2 instance. The *InstanceType*, *ImageId*, *AvailabilityZone* and *UserData* keys are especially important. *InstanceType* describes the configuration of the instance from a hardware perspective, i.e. the specification of CPU, RAM, storage and network available for the instance. Consequently, the instance type represents the node configuration parameter identified in section 4.1.1. Via a variable, the value for the InstanceType key is referenced and can therefore be specified when starting the Cloudformation stack, e.g. via the browser interface. However, the specific instance types used are not of interested for now as they are discussed in the next section of this concept. The *ImageId* descibes the Operating System (OS) used for the EC2 instance. For the experiments, an Ubuntu 16.04 image was selected because it reflects the same environment used on a local machine during the development process. The *AvailabilityZone* key specifies in which geographic region the EC2 instances are launched. In order to minimise the affect of network latency for nodes when communicating with each other, all of the EC2 instances were provisioned in the same region. In this particular case, all instances were located in the city of Frankfurt. Finally, *UserData* was used for

the installation and startup processes. Executing commands on a bash shell allowed to establish a state from which the experiments could be started.

```
"Node1" :
{
 "Type":"AWS::EC2::Instance",
 "Properties" :
 {
  "InstanceType":{"Ref":"EC2InstanceTypeGethNode"},
  "ImageId":"ami-027583e616ca104df",
  "AvailabilityZone":"eu-central-1a",
  "UserData" :
  {
   "Fn::Base64" : { "Fn::Join" : ["",[
   "#!/bin/bash \n",
   "sudo apt-get update \n",
   "sudo apt-get install -y docker.io \n",
   "sudo docker run --net=host ",{"Ref":"DockerImageName"},"
    /bin/bash -c \"cd ~/diploma-thesis; make node_start
    master_ip="{"Ref":"MasterIP"},"\" "]]}
  }
 }
}
```

Listing 4.1: AWS CloudFormation Template Excerpt

As already implied with the *InstanceType* key, Cloudformation templates may include references and variables. Using variables, the user can influence the setup of the stack. For example, one can allow the user to set the *InstanceType* for the instances being deployed. For the AWS Cloudformation interface of the experiment stack, as depicted in figure 4.1 on the next page, an IP for a **master node**, the instance type for the nodes running Geth, the instance type for the node running a **Bootnode** and **ETH-Netstats** service and the name of a **Docker image** have to be supplied by the user.

An IP for a master node has to be specified because of the communication model of the experiment setup. In the setup, a master node acts as some sort of command centre. The purpose of this node is to send assignments to the Ethereum network, e.g. deploy a specific contract, get the number of peers from a specific node or initiate a performance measurement. Furthermore, the master node's task is to persist measured data. Although an EC2 instance could be established as the master, a local machine has been chosen for the experiment setup. This is due to the reason that measured data needs to persist even after the EC2 instances have been shut down and costs for cloud resources need to be minimised.

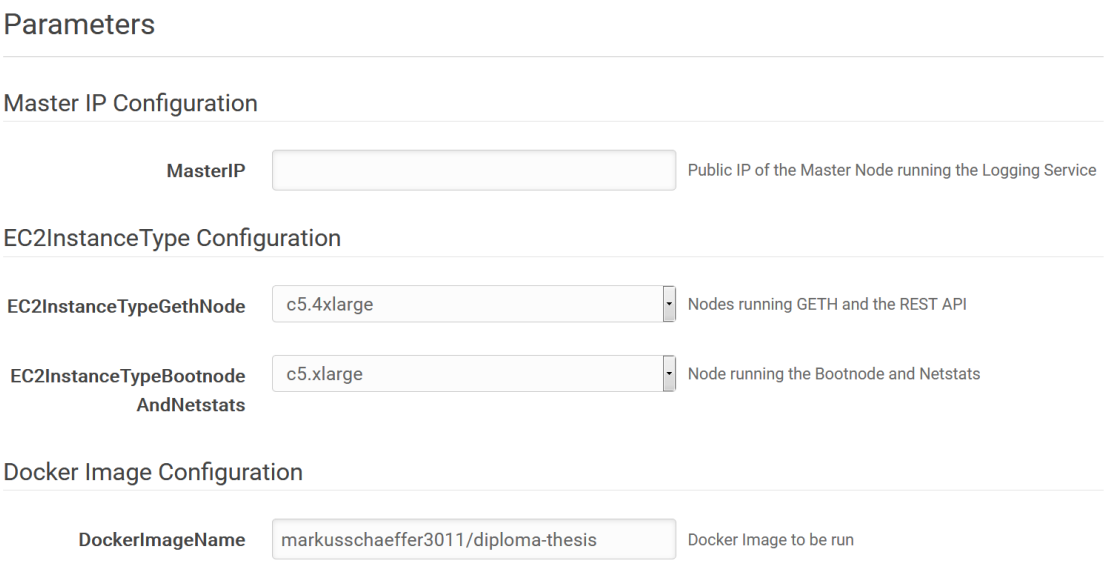

Figure 4.1: AWS Cloudformation Interface for the Experiment Stack

The Bootnode plays a vital part in Ethereum's discovery protocol. This discovery protocol describes how nodes running Geth can find other peers. In the discovery protocol, nodes talk with each other to find out about other nodes in the same network. In order to start this process initially, a lightweight bootstrap node is used - the Bootnode. Once a Bootnode is online it displays its address (enode URL). The first time a node running Geth connects to the network it uses the Bootnode's address to join the network and find other nodes. Although the Bootnode peer discovery service does not utilize much computing power, the Bootnode is placed on a separate node. This is mainly because a node running Geth should not share its computing power with an additional service.

Another service running on the same node as the Bootnode is ETH-Netstats[3]. ETH-Netstats is a web based application for monitoring an Ethereum network. The service monitors information on the network level and on the node level. Examples for network level data which can be monitored via a web frontend are the current block number, the time passed since the last block had been found and the current gas limit of a block. Data which can be monitored on node level includes the amount of peers, the amount of pending transactions and a particular node's current mining hash rate.

The Docker[4] image name in the AWS Cloudformation interface is the name of a public repository published on Docker Hub[5]. This Docker image serves three purposes.

[3]ETH-Netstats for the public Ethereum is available at https://ethstats.net/.
[4]Docker allows operating-system-level virtualization, which is also known as 'containerization'.
[5]Docker Hub is Dockers default registry and the place where Docker searches for images. It can be seen as the 'Github Variant' for Docker files as it allows to commit and publish Docker images to a cloud-based repository.

First, it decreases the startup time needed when provisioning the EC2 instances. This is due to the reason that all software and the self developed measurement application have already been installed on this image. Thus, no time is wasted for installing software. Second, since the required state for starting the experiments has already been established, the Docker image serves the purpose of reproducibility. With all software already installed, possible difficulties regarding different versions of software and their compatibility are eradicated. The third and most important purpose within the scope of the experiments is related to the functionality of Ethereum's Proof-of-Work algorithm - *Ethash*.

When using Geth with the PoW algorithm, a dataset known as the ethash Directed Acyclic Graph (DAG) has to be created before the actual mining can start. This dataset is approximately 1 GB in size and needs to be created for every Ethereum epoch (approximately every 30,000 blocks). The DAG itself is a two-dimensional array of uint32s (4-byte unsigned integers), with dimension (n*16) where n is a large number [4]. The dataset is needed in the mining process as mining involves grabbing random slices of the DAG and hashing them together.

The generation of the DAG for an epoch takes a while. During the practical development of the experiment setup, it was observed that the DAG generation is affected by the computing power of an instance. Especially when using EC2 instances with a lower computing power, the generation consumes a lot of time. Only after the DAG for the first epoch has been generated, the mining process and thus the processing of transactions can begin. As the DAG needs to be generated each time an EC2 instance is (re)started, using a Docker image where the DAG has already been created allows to save a lot of time. Right after the generation for the first epoch has been finished, the mining process and the generation for the second epoch are automatically started. As the generation of the DAG for an epoch consumes computing power, which is needed for the mining process, the DAG generation for the second epoch may influence the mining efforts. Due to this reason, the generation of the first and second DAG epoch has been included in the Docker image. In other words, two datasets consuming roughly 2.2 GB of disk space have been included in a Docker image. This approach saves a lot of time and avoids possible corruption of measurements due to computing resource scarcity during the DAG generation process.

As already briefly discussed, the local master node serves as a command centre in the experiment setup. For the communication between the master node and the nodes running the Ethereum blockchain as well as the Bootnode and the ETH-Netstats services, the Representational State Transfer (REST) architecture style has been chosen. Via the use of several **REST APIs** and clients, the measurement process is handled inside a JavaScript application. For decoupling major components, the Model-View-Controller (MVC) design pattern is used.

The most important instructions supported by the REST APIs are listed below.

- start Geth, Bootnode, ETH-Netstats
- store IP
- get peer-count
- deploy contract
- store contract address
- start measurement
- log measurement

Another important task which is handled on the master node is the persistence of data. Once a measurement has been performed, it needs to be stored for the later analysis. Due to the reason that the data exchange format used for the communication between the nodes is JSON, a **MongoDB**[6] database has been selected as persistence layer. In contrast to the approach of storing measurement data in a simple text file, the database allows to easily analyse the data.

Finally, figure 4.2 on the next page summarizes the experiment setup from an architectural perspective. In addition, the main technologies and tools with an description of usage are listed via table 4.3.

[6] MongoDB is classified as NoSQL database program and uses JSON-like documents with schemata.

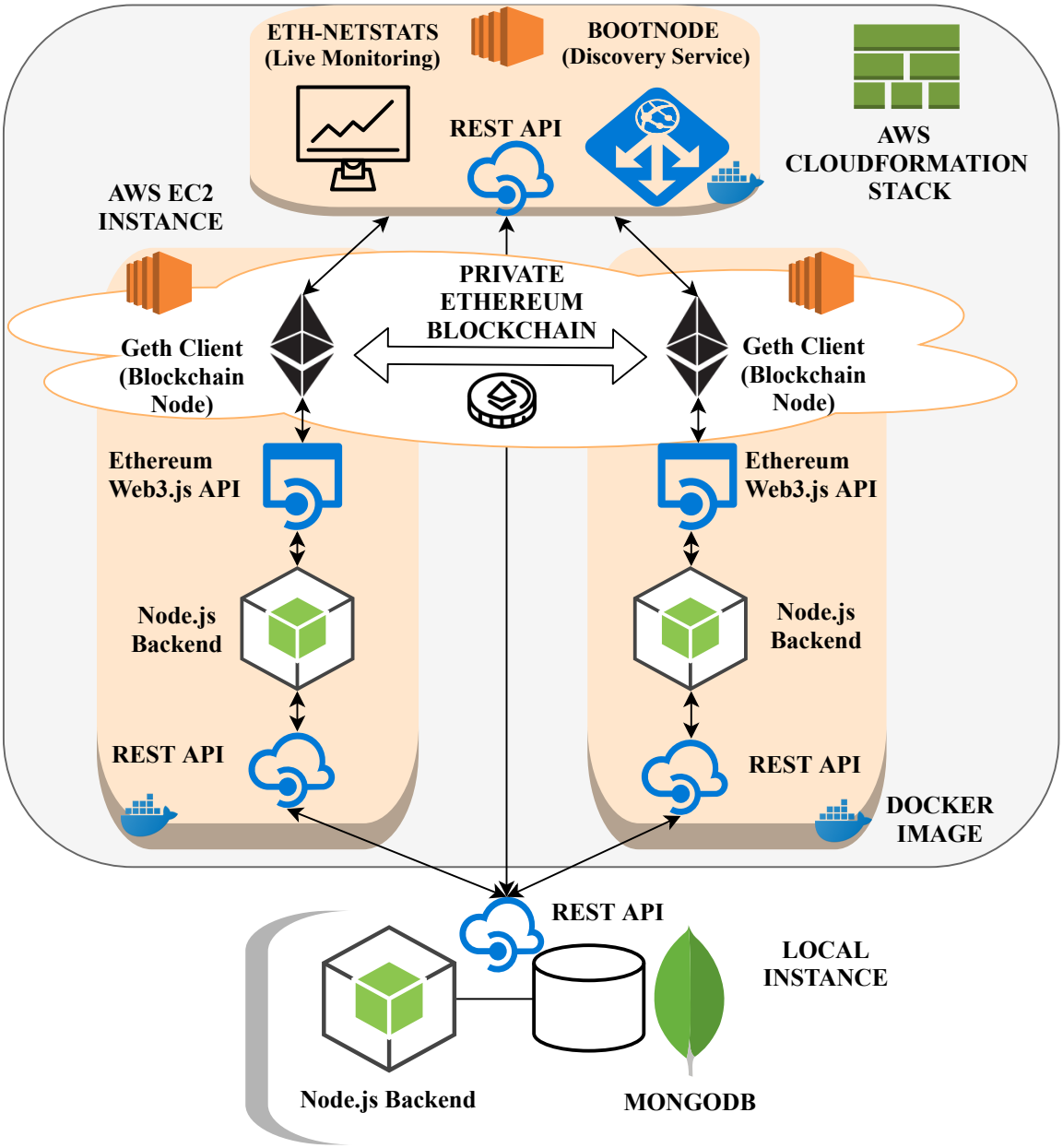

Figure 4.2: Experiment Setup

Table 4.3: Overview of Main Technologies and Tools

Name	**Version**	**Description**
Geth	v1.8.17-stable	Ethereum client
Bootnode		Peer discovery service for Ethereum nodes
ETH-Netstats		Blockchain network monitoring tool
Puppeth		CLI wizard which was used for generating genesis.json files
Web3.js	1.0.0-beta.36	API for interacting with Ethereum clients via JavaScript
Node.js, Express.js	v8.12.0, 4.16.3	Backend handling the deployment of smart contracts, communication between master and nodes, transactions for the measurements, and the persistence of measurements. The Express.js framework was used for creating the REST APIs
Shell Scripts		Automation of start-up processes for Geth
Solidity	0.4.0	Definition of Smart Contracts
MongoDB, Mongoose, PyMongo	3.2.20, 5.2.16, 3.7.2	Persistence Layer. Mongoose was used for interaction via JavaScript, PyMongo was used for interaction via Python
Jupyter Notebook, Matplotlib, Pandas	5.6.0, 2.2.3, 0.23.4	Jupyter Notebook served as data analysis environment. The Matplotlib library was used for the generation of figures
AWS Cloudformation Templates	2010-09-09	Automatic start-up of pre-configured AWS EC2 Instances
Docker	1.13.1	Operating-system-level virtualization

4.1.4 Parameter Mapping

After discussing the selected technologies for the experiment setup, the identified parameters in section 4.1.1 can be mapped onto the target architecture.

As the Geth client is used in the experiments, one has to know how the respective client can be configured before turning to individual parameters. Geth uses two ways of configuring the blockchain network. First, command line parameters can be stated when starting the Geth Client. For example, one can specify the id of the network and if the client should start to mine on startup. In addition to the command line parameters, a JSON file is used to describe the first block of the blockchain. This JSON file is known as *genesis.json* and plays a vital role in the blockchain configuration process as it contains all data needed to generate block zero. Amongst others, the genesis.json file is used to allocate the initial amount of Ether available in the blockchain to various addresses.

An excerpt only displaying the most important parts of a genesis.json file used in the experiments is depicted below.

```
{
 "config": {
  "chainId": 1515,
  "clique": {
        "period": 15,
        "epoch": 30000
  }
 },
 "gasLimit": "0x47b760",
 "difficulty": "0x80000",
 "alloc": {
        "5dfe021f45f00ae83b0aa963be44a1310a782fcc": {
        "balance": "1000000000000000000000000000000000000000"
    }
  },
}
```

Listing 4.2: Excerpt of a genesis.json File

As the configuration of Geth has been discussed, the first parameter, `block frequency`, is able to be mapped. As already mentioned in the previous section, Proof-of-Work and Proof-of-Authority are used as consensus algorithms. It has been found out that the configuration of the block frequency is dependent on the type of consensus algorithm used. As one can observe in the excerpt of a genesis.json file above, the consensus algorithm is specified in the *config* key of the JSON file. In case of Proof-of-Work, *ethash* has to be specified as value in the config key. When Proof-of-Authority is needed, `clique` is the value to be used.

When using ethash, i.e. PoW, the `difficulty` key determines the block frequency. This hexadecimal value can be interpreted by its reciprocal and determines how difficulty it is to mine a block. For example, when set to 0x400, there is a 1/1024 chance to successfully mine a block on the first attempt. In other words, on average one can expect a successful mining operation after 1024 hash computations.

During the development phase of the present work, it has been observed that the minimum difficulty, which can be set in the genesis.json file, is 131072. However, the actual difficulty of a single block is not only dependent on the difficulty value specified in genesis.json and it is not static. In fact, there is a difficulty adjustment algorithm in the Geth client. This difficulty adjustment algorithm is responsible for setting the block frequency in the public Ethereum to roughly 15 seconds. The difficulty of a new block is calculated via a complex algorithm[7] which takes the difficulty of the last block and the current time as input parameter. If one wanted to get a static block frequency, the method responsible for calculating the difficulty for the next block in the Geth client would have to be modified. Subsequently, this would result in a custom version of Geth. However, it has been decided to use the original version of Geth and thus to conduct the experiments knowing that the block difficulty may increase over time.

For determining a default value for the difficulty value in the genesis.json file, Ethereum's *puppeth*[8] tool was used. When generating genesis.json files puppeth also includes default values for various parameters, e.g. for the difficulty. This default value was set to be the default value for the experiments. As the effect of a varying difficulty on performance is of interest, varying the difficulty in a range is necessary. Hence, for the experiments, the approach of multiplying the default value with a factor was used. In particular, the default value for the difficulty (524288) in the genesis.json generated via puppeth was multiplied with the factors 0.25, 0.5, 1, 2 and 4. Thus, lower and higher values than the default value are included in the experiments.

When using clique, i.e. PoA, the `period` key in the genesis.json defines the block frequency. In contrast to the difficulty, the period is static and is the sole parameter influencing the block frequency. The period value represents the time between two blocks in seconds. Similar to the difficulty, the puppeth tool was used to get a default value for the experiments. However, in contrast to the difficulty, it has been decided to only decrease the default value. This is due to the reason that the default value (15 seconds) is perceived to be too high for the private Blockchain context. In fact, the 15 seconds block period is used in the public version of Ethereum. For that reason, the range of the period value was set between 2 and 15 seconds. More precisely, the values of 2, 4, 8, 12 and 15 seconds have been chosen.

[7] The actual method in the Geth client responsible for the difficulty adjustment is *CalcDifficulty*. It can be investigated via the go-ethereum/consensus/ethash/consensus.go path in the Ethereum Github repository [3].

[8] Puppeth is a command line interface program included in the Geth & Tools package. It helps at setting up a private Ethereum network.

The second identified parameter to be matched to the target architecture is the `block size`. As with the block frequency, the block size can be configured in the genesis.json file. The respective key of interest is `gasLimit`. The value describes the limit of gas expenditure per block. As transactions consume gas, the gasLimit value indirectly determines the amount of transactions fitting in a block. Since transactions may have a different payload, the amount of transactions fitting into a block may be different from one block to another. However, for the experiments only transactions consuming the same amount of gas have been used, i.e. a block only contained transactions of the same workload type. Regarding the default value for the gasLimit key, again the puppeth tool was used. Similar to the difficulty value, a range for the experiments was set by multiplying the default value with a factor. These factors are 0.5, 1, 2, 4 and 8. Initially, it was planned to use the factor 0.25 instead of 8 for an equal spread of lower and higher gasLimit values. Unfortunately, when the default value for the gasLimit per block is multiplied with the factor 0.25, a block is so small that the transaction for the smart contract deployment does no longer fit in a single block.

Similar to the difficulty, the gasLimit value in the genesis.json file only describes the block size of the blockchain's genesis block. Another parameter which affects the block size is the `targetgaslimit` option of the Geth client's miner. This option sets the target value for mined blocks. The target value thereby may be a lower or a higher value than the initial value stated in the genesis.json block. Hence, the block size of newer blocks may be lower or higher than the block size of the initial block. Consequently, more or less transactions may fit into a single block depending on the 'lifetime' of the blockchain. In order to mitigate this effect, the experiments were conducted in a way that the targetgaslimit value of the Geth client was always set to the same value defined in the genesis.json file.

The next parameter to be mapped is the `workload type`. The workload type simply translates to the `smart contract` used for the measurements. Regarding the smart contracts to be used for the experiments, it has been decided to focus on quite simple, but also realistic contracts. This is in contrast to the existing performance benchmarking framework Blockbench [29]. Although workloads used in Blockbench may serve the purpose of benchmarking blockchains in a very detailed way, some workloads, especially the ones for lower layers, may not be perceived as realistic scenarios of blockchain use cases. For example, one might argue that CPU intensive calculations, such as sorting arrays, are not considered to be a realistic use case for smart contracts and should better be performed in a more centralized system.

The actual smart contracts used for the experiments illustrate two simple use cases. The first contract is called Account and serves the purpose of simply transferring Ether between accounts. As transferring the native currency of a blockchain can be seen as the most simple and probably the most utilized use case, the Account contract serves as the default workload for the experiments. For illustration purposes, a code snippet of the Account contract which highlights the most important parts of the contract is printed on the next page.

Algorithm 4.1: Account Contract Code Snippet

```
1  contract Account {
2      function transferEther (address etherreceiver, uint256 amount) public
       onlyOwner {
3          require(amount <= getBalance());
4          etherreceiver.transfer(amount);
5      }
6
7      //fallback method to receive Ether
8      function() public payable{}
9
10     modifier onlyOwner() {
11         require(msg.sender == owner);
12         _;
13     }
14 }
```

It can be seen that the *transferEther* function is used to transfer a specified amount of Ether from the contract to a specified address. The transfer is only executed if the required amount of Ether is available and the transfer can only be initiated by the owner of the contract. The modifier *onlyOwner* thereby ensures that only the owner of the contract, i.e. the creator in this simplified case, can actually transfer Ether to a specified address.

The second smart contract developed implements another very popular smart contract use case - voting. Indeed, the use case of voting or executing a simple ballot is an often discussed use case for blockchains. Recently, there have been discussions about the use of blockchains for governmental election processes. For example, the U.S. state West Virginia tested a new method, which uses blockchain technology to store and secure digital votes for a federal election[50]. Another reason why voting is perceived as a standard use case for blockchains and especially Ethereum may be due to Ethereum's Remix[9] IDE. When opening Remix, the default contract, which serves as an example of a Solidity contract, is a ballot contract. This contract also served as the basis for the Ballot contract included in the experiments of this work. As with the Account contract, important code snippets of the Ballot contract are displayed on the next page.

The most important part of the Ballot contract is its *vote* function which increases the vote count for a specific proposal. In contrast to the Remix example contract, this modified implementation allows multiple votes from the same address. This approach has been chosen, as executing a ballot with only one vote per address would require a vast amount of distinct addresses in a private blockchain network.

[9] "Remix is a browser-based compiler and IDE that enables users to build Ethereum contracts with Solidity language and to debug transactions" [3].

Algorithm 4.2: Ballot Contract Code Snippet

```
1 contract Ballot {
2     function vote(uint8 toProposal) public {
3         //increase voteCount for specific proposal
4         proposals[toProposal].voteCount += 1;
5     }
6 }
```

The fourth parameter, `node configuration`, represents the `computational power` of a node. As already discussed in the last section, AWS EC2 instances serve as computing environment. For the selection of different instance types, the family of compute optimised EC2 instances has been chosen. Other requirement for selecting the instance types was that the network performance should be high and that the computational power should increase linearly. In addition, as AWS does not allow every user to occupy an endless amount of their resources, the limit of EC2 instances which can be run at the same time needed to be considered. With all these requirements in mind, instance types which are represented in table 4.4 and table 4.5 have been chosen. While the first table includes instance types which are relevant for measurements regarding the node configuration parameter, the second table includes the instance type which is used for measurements concerning the network size parameter. This split was necessary due to the maximum limit of EC2 instances available per instance type.

The c5.xlarge instance type has been chosen as default value for the node configuration. Hence, it was used with the block frequency, block size and workload type measurements.

Table 4.4: EC2 Instance Types used for Node Configuration Measurements

Instance Name	**vCPUs**	**CPU Speed**	**RAM**	**Network Performance**
c5.large	2	3 GHz	4 GB	10 Gigabit
c5.xlarge	4	3 GHz	8 GB	10 Gigabit
c5.2xlarge	8	3 GHz	16 GB	10 Gigabit
c5.4xlarge	16	3 GHz	32 GB	10 Gigabit

Table 4.5: EC2 Instance Types used for Network Size Measurements

Instance Name	**vCPUs**	**CPU Speed**	**RAM**	**Network Performance**
t2.xlarge	4	2.3 GHz	16 GB	Moderate

Regarding the `network size` parameter, a value range of 1-20 nodes in the blockchain has been selected. The basis for the particular range is the maximum amount of available instances which could be used at AWS. Another reason for the maximum amount of 20 nodes in the experiments was that it can be argued that a network with more than 20 nodes is highly unlikely in a private blockchain context. As the monetary resources and time needed for gathering data for each network size was perceived to be too high, five values have been selected for the experiments. In particular, the network sizes of 1, 5, 10, 15 and 20 nodes have been selected where 5 is the default value for the network size.

Turning to the `workload quantity` and the `amount of miners` parameters, the decision has been made to set the workload quantity to 1000 transactions and that all nodes in the network should mine/seal. The decision for a workload quantity of 100 transactions was influenced by the time spent on a single measurement and the workload quantity used in the related work.

Finally, all parameters identified and included in table 4.1 have been mapped onto the target architecture. Moreover, default values and value ranges have been defined.

4.1.5 Measurement Process

Now that all parameters have been mapped, the process of performing the measurements can be discussed. This process is illustrated via figure 4.3 on the next page.

Once the persistence layer and the REST service on the local master node have been started, the REST service listens for incoming messages. Meanwhile, the AWS Cloudformation stack has to be started via the provided AWS Cloudformation templates. These templates hold information about the node configuration and the network size parameters. The Cloudformation stack includes the EC2 instances running the Geth client and the JavaScript backend which handles the measurement. In addition, an instance running the Bootnode and ETH-Netstats services is provisioned. As soon as an EC2 instance is available, a specified Docker image is downloaded from Docker Hub and a Docker container is started. This container executes a REST service which is used to register the IP of an instance at the master node.

Next, the block frequency and block size have to be stated via a genesis.json file. After that, the Geth clients and the Bootnode as well as the ETH-Netstats services can be started via respective REST calls to each registered node.

Now that all services are online, the nodes running Geth find their peers via the Bootnode service and the ETH-Netstats service monitors the status of the private blockchain. When all nodes have found their peers, a *deployContract* message which includes the workload type as payload can be sent from the master node to a node running Geth. This initiates the deployment of the stated smart contract(s) to the blockchain. Afterwards, a REST client registers the smart contract addresses at the master node.

Finally, the measurement can be started via a REST call from the master node to a specified node running Geth and the JavaScript measurement application. This application

sends N requests to the blockchain platform in an asynchronous manner, i.e. requests are sent without waiting for a response. Once the specified amount of transaction has been successfully confirmed or the maximum runtime has been reached, the metrics for the measurement are calculated. In addition, information such as the current block difficulty, block size and hash-rate is queried from the blockchain and joined with the calculated metrics. As a last step, the gathered data is sent to the master node where it is stored in a database. An example of a single measurement which reveals all data gathered during the measurement process is displayed on the next page.

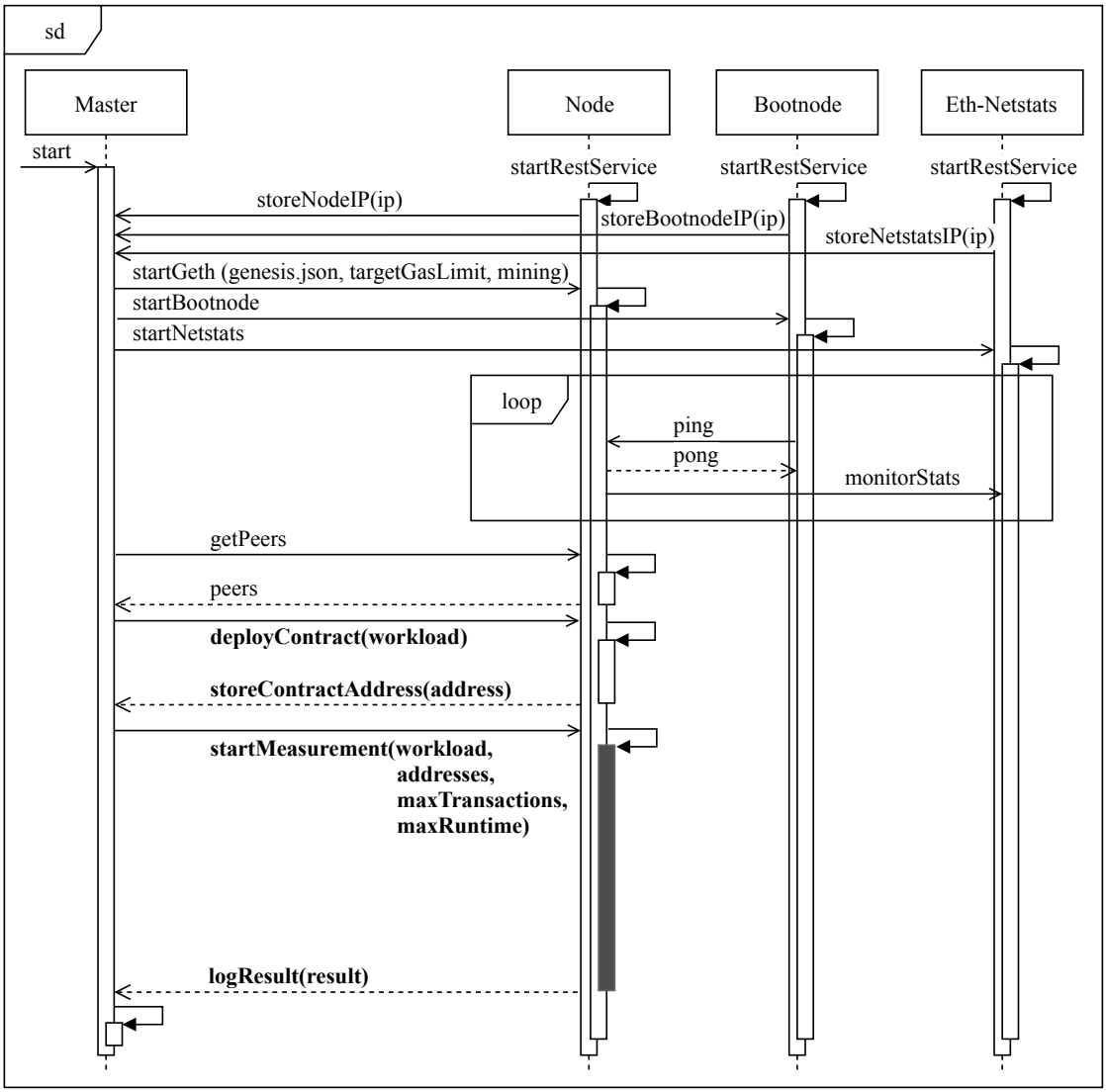

Figure 4.3: Sequence Diagram Illustrating the Measurement Process and the Message Flow Between Entities.

```
{
 "ip":"54.93.179.57",
 "measurementID":1307,
 "scenario":"account",
 "approach":2,
 "instanceType":"c5.xlarge",
 "docker":true,
 "nodes":5,
 "peerCount":4,
 "targetGasLimit":4700000,
 "gasLimit":4705290,
 "gasLimit_genesis_hex":"0x47b760",
 "gasLimit_genesis_dec":4700000,
 "usedGenesisJson":"genesis_pow_default.json",
 "clique":false,
 "clique_period":0,
 "ethash":true,
 "mining":true,
 "hashRate":93869,
 "miners":5,
 "difficulty":517762,
 "difficulty_genesis_hex":"0x80000",
 "difficulty_genesis_dec":524288,
 "startTime":2018-11-06 17:27:16.032,
 "maxRuntime":180,
 "runtime":14.87,
 "maxRuntimeReached":false,
 "maxTransactions":1000,
 "maxTransactionsReached":true,
 "successfulTransactions":1000,
 "txPerSecond":67.249,
 "averageTxLatency":9.259
}
```


Listing 4.3: Example of Gathered Data for the Analysis of Performance and Scalability

4.2 Analysis of Performance and Scalability

With the use of the concept presented in the last section, experiments on performance and scalability have been conducted. While the throughput, latency and scalability represent the dependent variables in the experiments, all parameters bold in table 4.1 represent the independent variables.

During the experiments around 4,000 data points, each representing a measurement result, were gathered. As the workload quantity was fixed to 1,000 transactions per measurement, roughly 4 million transactions were processed. When keeping in mind that, during the time of the experiments, the amount of transactions processed on the public Ethereum network was a little bit more than half a million transactions per day [12], the amount of data gathered for the analysis of performance and scalability of the present work can be compared to eight days of transactions on the public Ethereum blockchain.

According to a monthly EC2 running hours costs and usage report generated on AWS, the total EC2 instance usage in hours for the experiments was roughly 380. This number represents the aggregated usage time of all EC2 instances, i.e. in case 20 nodes were running for one single hour, 20 hours were added to the overall total usage. As expected, the two most used EC2 instance types were c5.xlarge (165 hours) and t2.xlarge (152 hours). This may be due to the fact that c5.xlarge served as the default instance type while t2.xlarge instances were used for measurements on the network size parameter.

Due to limited resources available and the amount of different parameters included in the experiments, each parameter was analysed on its own. Thus, default values for all independent variables were needed. These default values are listed in the table below. Unless noted otherwise, only one independent variable was varied while all other independent variables were set to their default value.

Table 4.6: Default Values for the Independent Variables

Difficulty	**Period**	**Gaslimit**	**Workload**	**Instance**	**Nodes**
524.288	15 sec	4.700.000	account	c5.xlarge	5

For the analysis of performance and scalability, several figures have been created which illustrate the effect of an independent variable on the specified dependent variables. As an analysis environment, several Jupyter Notebook[10] files were used. Besides the charts, two-sided t-tests were used to determine if two data sets are significantly different from each other.

With each parameter analysed, expected and measured results are discussed and compared. Furthermore, findings are set in relation to the related work. In addition, all findings are briefly presented and compared with the related work in chapter 5.

[10] Jupyter Notebook is an application which is commonly known for its use in data analysis projects. The notebook is a file which may combine code, equations and generated charts (see https://jupyter.org/).

4.2.1 Block Frequency

The first parameter to be discussed is the block frequency. This parameter describes the time difference between two succeeding blocks and is stated in seconds. In order to analyse the effect of the block frequency on throughput and latency for the two consensus variants PoW and PoA, the respective consensus parameters had to be varied. While the `difficulty` was changed with the first consensus variant, the block `period` parameter was altered at the second consensus algorithm.

Although a comparison of PoW and PoA on the same block frequency range was intended at first, experience has shown that, with PoW, pre-defining an exact range for the block frequency, e.g. 1 to 15 seconds, is nearly impossible. This is due to the reason that, in contrast to the PoA consensus algorithm, the block frequency in PoW is not static and is also influenced by many factors (see section 4.1.4).

With PoW, the block frequency is indirectly determined via the current hash-rate of the network and the current block difficulty. One may assume that the block frequency may be easily calculated via dividing the difficulty with the hash-rate. However, there are some difficulties with this approach. First, one does not know the average hash-rate of of a particular machine, nor the hash-rate of a network in advance. In addition, it has been observed that the actual hash-rate of a node exhibits a high variance. Indeed, the hash-rate may sometimes double or halve over time. As a consequence, each node in the blockchain network may show a different hash-rate besides having the same hardware configuration. Moreover, the full network hash-rate may not be effectively used as it is not guaranteed that a node is able to successfully propagate all transactions before a new block gets mined. Finally, the difficulty adjustment algorithm in ethash, which slightly increases the block difficulty over time, makes it quite impossible to set on a predefined and static block frequency of x seconds.

Therefore, the analysis of the block frequency with PoW was focused on higher block frequencies (i.e. a lower block period), while the block frequency range for the analysis with PoA was more spread out. More precisely, with PoW and the hash-rate provided via the default parameters (five c5.xlarge nodes), the block frequency ranged from 1 second (with a difficulty of 0.25 times the default difficulty) to 2.5 seconds (with a difficulty of 4 times the default difficulty). The block frequency at the default difficulty (factor 1) was roughly 1.1 seconds. These values have been measured via querying the timestamps of succeeding blocks. With the PoA algorithm, the block frequency ranged from 2 to 15 seconds where measurements were performed at 2, 4, 8, 12 and 15 seconds.

Based on the related work and the thought experiment conducted in section 4.1.1, the author of the present work assumes a decrease in throughput and an increase in latency when raising the mining difficulty.

With regard to figure 4.4, this is exactly what has been measured for the PoW variant. Using 350 measurements in total, i.e. roughly 70 for each data point, the throughput and latency is plotted against the mining difficulty factor.

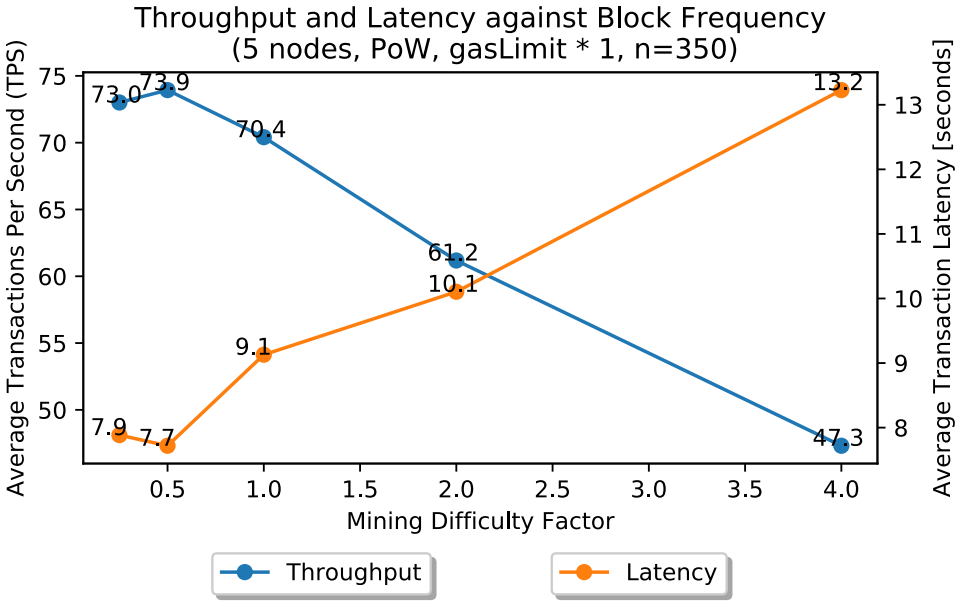

Figure 4.4: Throughput and Latency against Block Frequency (PoW)

One can see that the throughput coloured in blue was measured with the highest values using a low mining difficulty (factors 0.25 and 0.5) and with the lowest throughput using a high mining difficulty (factor 4). The slight increase in transactions per second from 73.0 to 73.9 between the mining difficulty factors 0.25 and 0.5 may be owed to the randomness factor included in the mining process. The author assumes that with a higher sample size, the throughput for the factor 0.25 would be higher than the one for the factor of 0.5.

The slight difference in throughput between the 0.25 (difficulty set to the minimum value of 131,072) and 0.5 (difficulty value of 262,144) factor suggests that a saturation point has been reached. As already indicated above, the five c5.xlarge machines used in the experiments have provided enough hashing power to get a block frequency of one second when using a difficulty factor of 0.25. Investigating the boundaries of the block frequency in more detail, it turned out that, in fact, the minimum block frequency is one second. As the `period` parameter cannot be set to a value lower than 1 second, this is not only true for the PoW variant, but also for the PoA consensus algorithm. With a block frequency of 1.1 seconds, the default mining difficulty is also quite close to the minimum. This may be the reason why there is only a small difference in throughput when comparing the default difficulty factor with the factor of 0.25.

As already mentioned, the difficulty adjustment algorithm increases the block difficulty over time. From this it follows that the difficulty set in genesis.json does not match the actual difficulty of a block during the measurements. Indeed, the actual measured median difficulty for the measurements ranged between 178,019 (factor 0.25) and 2,178,986 (factor 4). The median difficulty recorded for the default difficulty (factor 1) was 680,196. Since

differences between the difficulty value in the genesis.json file and the measured median difficulties are evident, the question of how much the increase of difficulty per block actually was, arises. A further analysis on this matter revealed that the difficulty increase per block was around 0.05% per block with the default settings. Thus, with a median runtime of 14 seconds for a single measurement (with difficulty factor 1), the median increase in difficulty over the timespan of the measurements amounts to approximately 25-30%. However, since all measurements have been affected by the difficulty increase over time, the gathered data for the individual mining difficulty factors can still be compared with each other.

The correlation between the difficulty increase and the throughput, another aspect to be analysed, is depicted in figure 4.5 below. The correlation between the succeeding measurements, i.e. the time passed, and the throughput is most striking. During the timespan of about 12 minutes, the difficulty rose from 508,265 to 821,003 (61.53% increase). Using a linear regression, it was calculated that, with each measurement, the average throughput declined with about 0.4 transactions per second. This is a crucial result and signifies that there could be extreme performance differences over time when operating a private PoW-based Ethereum smart contract platform.

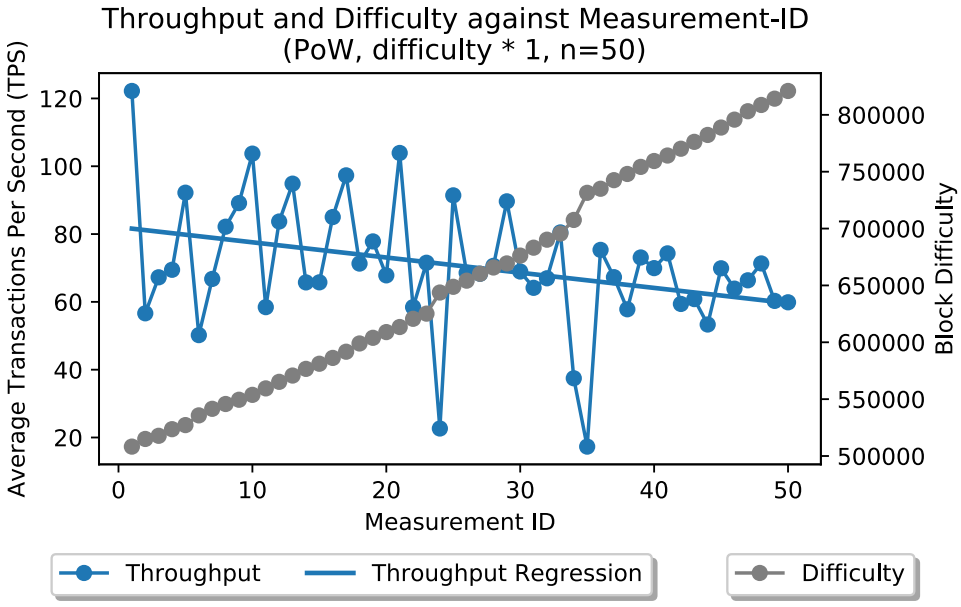

Figure 4.5: Throughput and Difficulty over Time (PoW, difficulty * 1)

Another major finding is that an increase in difficulty does not result in an increase of block frequency in the same proportion. For example, the default mining difficulty resulted in a block frequency of roughly 1 second, while the default difficulty multiplied by a factor of 4 resulted in a block frequency of approximately 2.5 seconds. This effect is also evident with the throughput and latency metrics. For example, the latency at the

default difficulty was measured with 9.1 seconds, while the latency at difficulty factor 4 was only measured with 13.2 and not with about 36 seconds.

Speaking of the latency, figure 4.4 clearly reveals that, as expected, with a higher mining difficulty, a single transaction has to wait longer to be successfully put in a block than at a lower mining difficulty. Similar to the throughput, the small differences between the factor of 0.25 and 0.5 may be due to the saturation point and the randomness factor included at the mining process.

Turning to the effect of the block frequency at the PoA consensus variant, the measured result also matches the expected result. With regard to figure 4.6, which illustrates throughput and latency against the `period`, one can see that, when doubling the block period, the throughput is roughly halved, while the latency is nearly doubled.

Initially, when analysing the throughput function, the author wondered why the throughput was not illustrated in a linear way. When keeping in mind that the values of the x axis are doubled, whereas the values of the y axis are roughly halved, it has gotten obvious that the representation of the throughput is actually as expected.

In summary, the effect of the block frequency on throughput and latency is as anticipated. However, a major finding concerning the block frequency and the PoW consensus variant is, that there exists a correlation between the performance and the time past. Furthermore it has been found that, an increase in difficulty at the PoW consensus variant does not result in a change of throughput and latency in the same proportions.

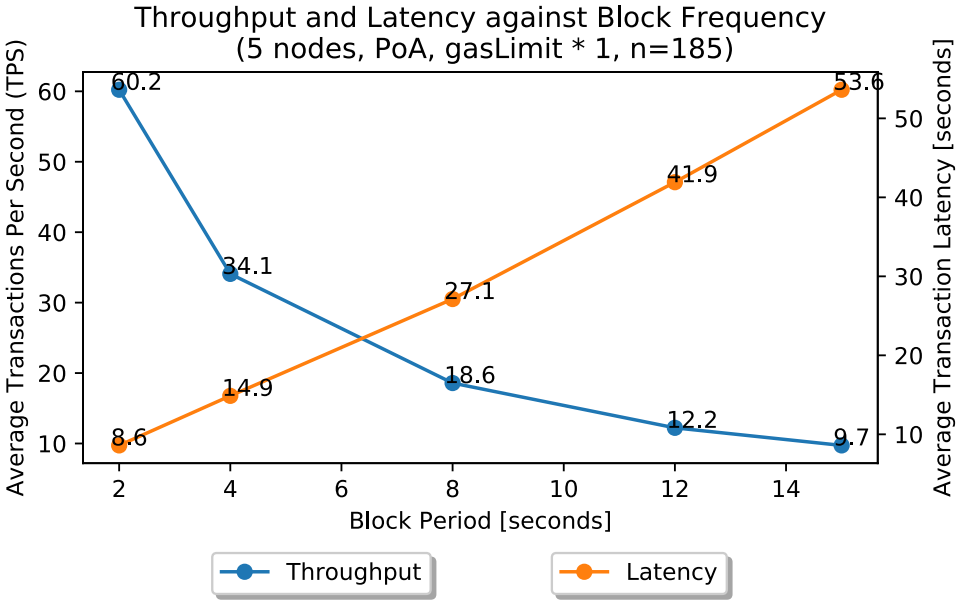

Figure 4.6: Throughput and Latency against Block Frequency (PoA)

4.2.2 Block Size

The next parameter to be analysed is the block size. In contrast to Bitcoin, where the block size is specified in megabytes per block, Ethereum uses the concept of gas to specify the size of a block. As there exists a limit for the amount of gas which can be included in a block and as each transaction consumes a specific amount of gas, the `gasLimit` parameter indirectly determines the amount of transactions fitting in a block.

At the time of writing the present work, the gas-limit of a block in the public Ethereum was roughly 8 million. This has been observed via the Ethereum exploration platform Etherscan.io [11]. However, as specified in section 4.1.4, the default gasLimit value for the experiments on block size in a private context was set to 4.7 million (factor 1).

Using the Geth JavaScript console, the amount of transactions fitting in a single block has been queried. It has been assessed that 146 transactions of the default workload (the account smart contract) fit in a single block when using the default gas-limit. Consequently, the range of transactions fitting in a single block varied from 74 (with a gasLimit factor of 0.5) to 1,168 (with a gasLimit factor of 8). Thereby, it is especially important that a gasLimit factor of the default value * 8 provides enough space for the whole workload quantity (1,000 transactions) to fit in a single block.

Before analysing the measured data, the affect of block size adjustments in the Bitcoin blockchain and the relevant related work is investigated in order to form the expected result.

Tuning the block size parameter is an idea that has been heavily discussed in the Bitcoin blockchain. The idea of increasing Bitcoin's block size for improved performance and scalability found proponents and opponents alike. Eventually, the Bitcoin community found itself in disagreement in the matter of block size adjustments and the future roadmap for better scalability. As a consequence, a hard fork[11] of the Bitcoin blockchain was realized which has resulted in the creation of a new cryptocurrency - Bitcoin Cash. Due to its greater maximum block size (currently 32 MB [1]), Bitcoin Cash is theoretically able to show a higher potential performance than Bitcoin (1 MB block size [14]). Thus, one might come to the conclusion that an increase in block size inevitably has to improve the throughput and decrease the latency in a blockchain system. The results presented in the paper "On Scaling Decentralized Blockchains" [25] also suggest that an increase in block size has positive effects on the Bitcoin blockchain. Nevertheless, it is also stated that there exist limits for block size increases. The authors of the mentioned paper argue that, due to the decentralized nature of Bitcoin, the higher the block size, the fewer nodes will be able to actually receive the blocks (since propagating bigger blocks takes more time).

With regard to the Ethereum blockchain and related work, the authors of the Blockbench framework [29] surprisingly imply that block frequency and block size affect each other.

[11] In the blockchain context, a hard fork is a rule change in the system which results in the need for a software change on the nodes. The software validating the blocks recognises the old blocks as invalid blocks, i.e. there is no backwards-compatibility.

Via measurements, the authors state that the block generation rate with the default block size was measured with 0.22 blocks/s, while block sizes of default * 0.5 and default * 2 resulted in block generation rates of 0.34 blocks/s and 0.12 blocks/s. Hence, the results demonstrate that "with bigger block sizes, the block generation rate decreases proportionally, thus the overall throughput does not improve" [29].

Combining all this information, one can produce the expected result. Assuming that blocks can be successfully propagated in the network within the timespan of the block frequency, the author of the present work assumes that, when increasing the block size by a certain amount, the throughout increases in the same proportion while the latency is expected to fall in the same proportion. For example, when doubling the block size, the throughput is expected to double, while the latency is expected to halve.

Turning to the measured result, figure 4.7 and figure 4.8 depict the effect of an increasing block size on throughput and latency. Unexpectedly, neither the throughput, nor the latency were measured as expected with the default PoW settings. Although the throughput and latency were not exactly measured as expected, the trend of a higher throughput for an increased block size is apparent. Equally, the latency apparently decreases with a rising block size. In contrast to results obtained with the default PoW settings, the throughput and the latency were measured as expected with the default PoA parameters.

Initially, it was thought that the differences between the two figures are due to the distinct consensus variants. However, the author of the present work came to the conclusion that the differences are due to different block frequencies. As already mentioned, the block frequency with the default PoW difficulty is approximately 1.1 seconds, while the default block period in the PoA setting is 15 seconds. Thus, figure 4.7 illustrates the effect of an increasing block size on throughput and latency at an extremely high block frequency, whereas figure 4.8 depicts the same at a relatively low block frequency (roughly the same as in the public Ethereum blockchain).

With the PoW variant and the high block frequency, a saturation is evident at the throughput function. The median throughput for the different gasLimit factors does not vary as expected and ranged from 53 tps for a factor of 0.5 to 74 with a gasLimit factor of 8. Hence, possible factors which may be responsible for the saturation in throughput needed to be systematically excluded.

As the authors of Blockbench [29] imply that the block generation rate is drastically reduced when increasing the size of blocks, this factor was investigated at first. The timestamps of blocks were queried via Geth's JavaScript console and the average block frequencies were calculated for each gasLimit factor. The calculations revealed that there was indeed a minor increase in block frequency when raising the block size. However, the differences were so small (around 0.3 seconds between gasLimit factor 0.5 and factor 8), that the differences may be due to the small sample size. A halving of the block frequency when doubling the block size, as reported by the authors of the Blockbench framework [29], was definitely not present.

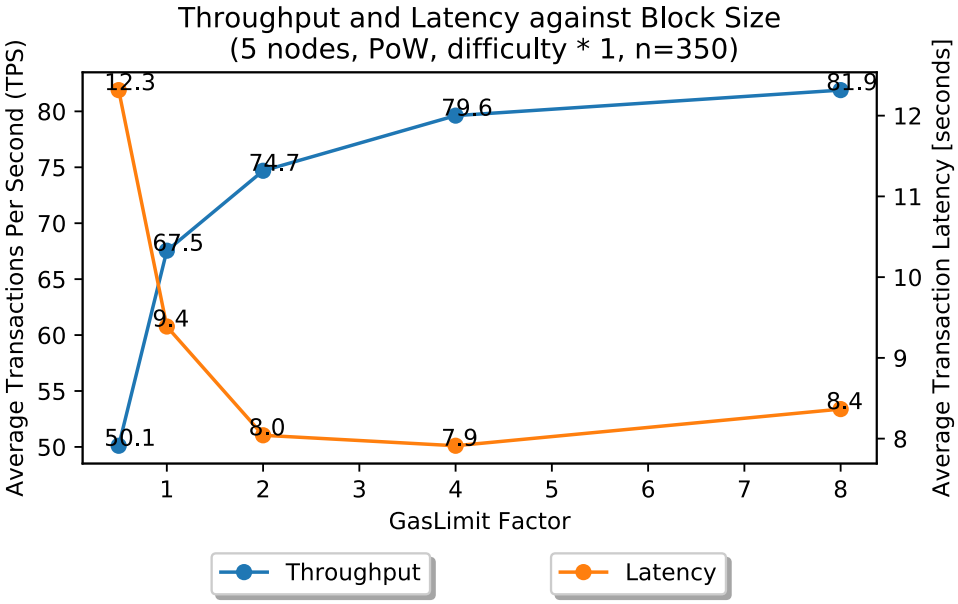

Figure 4.7: Throughput and Latency against Block Size (PoW)

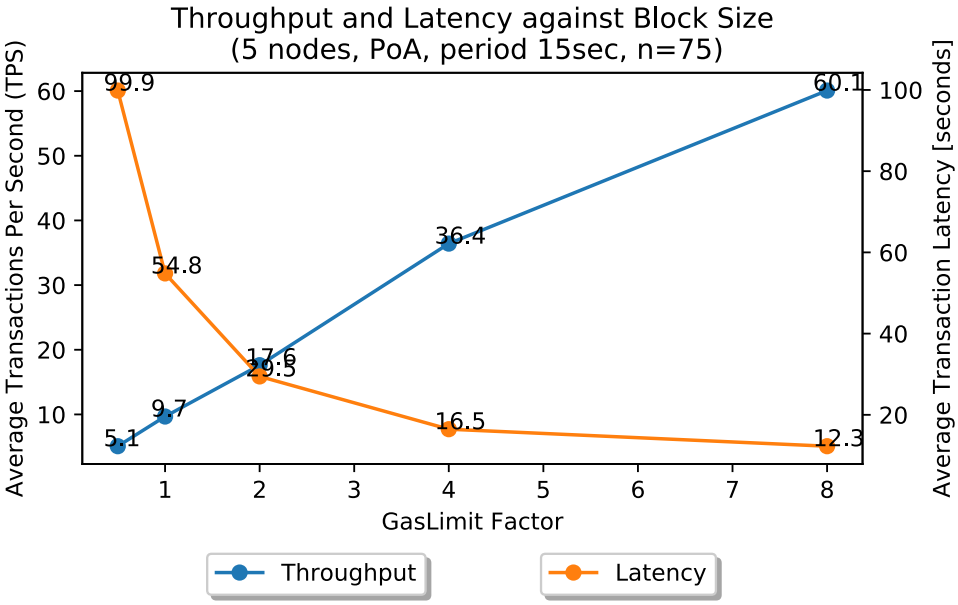

Figure 4.8: Throughput and Latency against Block Size (PoA)

Next, the information propagation overhead, the consensus costs and the low computational power of the machine were excluded. Regarding the consensus overhead, the assumption was that the time needed for propagating the transactions and blocks to the network and arriving at a consensus may be higher than the block frequency. As the time needed to propagate the blocks in the network and find consensus is affected by the size of the block and the number of nodes in the network, the network communication overhead and the consensus costs could be excluded via analysing the effect of the block size in a network with only one node. Regarding the computational power, the assumption was that the c5.xlarge instance-type did not provide enough computational power for creating, signing, propagating and executing the transactions during the small block period of a little bit more than 1 second. Thus, the computational power has been increased via changing the instance-type from c5.xlarge to c5.4xlarge.

The data gathered via the experiment on the block size with a single c5.4xlarge node is illustrated via figure 4.9. As can be seen, the throughput and latency functions are quite similar to the ones measured in the network of five nodes with the c5.xlarge instance-type. However, when compared to the results obtained in the five node network with the c5.xlarge nodes, the throughput is higher and the latency is lower in the one node c5.4xlarge network. Thus, the assumption that the network communication and consensus costs or the computational power of the node might impact the performance was strengthened (the computational power is analysed in more detail after the next section). Nevertheless, the saturation evident in figure 4.7 could not be explained using the single node with improved computational power.

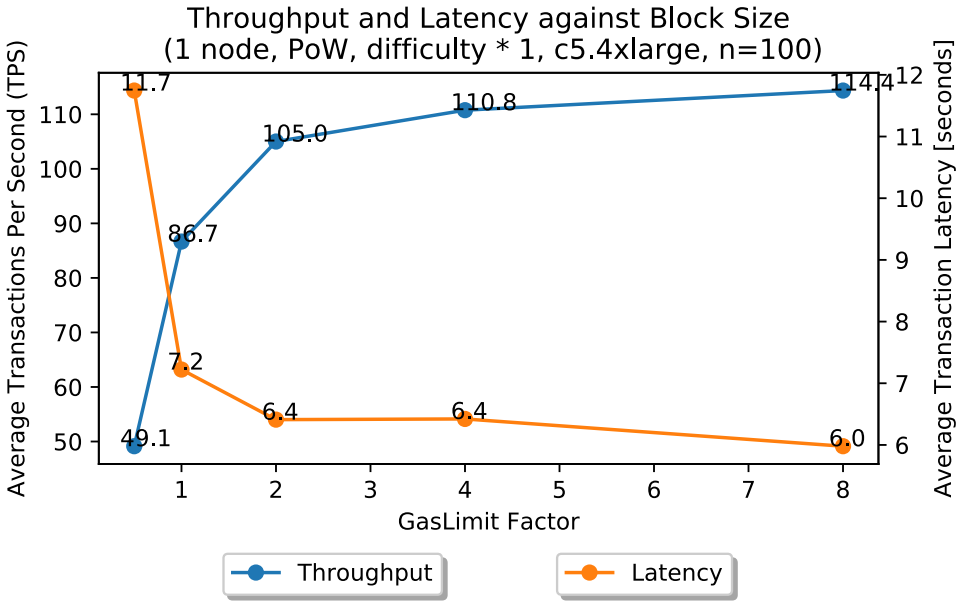

Figure 4.9: Throughput and Latency against Block Size (1 node, PoW)

Another analysis revealed that, for higher gasLimit factors, the maximum block size could not be fully utilized. For example, although a gasLimit factor of 8 provides enough space for 1,000 transactions to fit inside a single block, the transaction-count of a block varied around 10 to 400 (which was queried via Geth's JavaScript console).

Further considerations have led the author to assume that the reason for the saturation may be between the creation and execution of the transactions.

Therefore, the time for creating the transactions via the web3.js API, sending them to the local Geth client via JSON-RPC and signing them at the Geth client was measured. All these steps are covered when invoking the *send(options[, callback])* method of web3's *web3.eth.Contract* object. After the *send* method has been successfully performed, the transactions are put into the transaction-pool of the local Geth client. Surprisingly, the investigation has revealed that it takes roughly three to four seconds to invoke the *send* method in a loop for all 1,000 transactions with the c5.xlarge node configuration. As a result, the full workload cannot be fully generated during the low block period induced by the default PoW settings. The author therefore assumes that the slow transaction creation, sending to the local Geth client and signing processes are reasons for the throughput saturation illustrated in figure 4.7 and figure 4.9.

Nevertheless, it is argued that there have to be additional reasons for the throughput saturation due to the following considerations: Assuming that it takes roughly three seconds to create 1,000 transactions, a block frequency of around one second and that all transactions can fit inside a single block, the theoretical maximum throughput should be somewhere around 333 transactions per second. Yet, the average throughput with a gasLimit factor 8 on a single c5.4xlarge node was only measured with 114 transactions per second.

Indeed, analysing the transaction pool of the Geth node via the *txpool.status* command has revealed that usually the transactions stay in the transaction pool for longer than expected. For bigger block sizes, the client does not seem to be able to process as much transactions as can fit inside a single block during the timeframe of the block period. Therefore, the execution of transactions may also be partially responsible for the throughput saturation at a high block frequency.

Hence, the throughput saturation measured at the high block frequency can be explained via the combination of a too slow workload generation and a too slow transaction execution time.

In case the block frequency is high enough, the throughput behaves as expected. This can be observed via figure 4.8, which illustrates the effect of an increased block size on throughput and latency with a block frequency of 15 seconds. Comparing the theoretical maximum throughput for distinct gasLimit factors with the measured average values confirms that the expected and observed results match pretty well. For example, with a block frequency of 15 seconds and a gasLimit of factor 8, the maximum throughput was calculated with 66 tps and the measured average was 60.1 tps. For a gasLimit factor of 1, the calculated throughput of 9.7 tps was measured exactly with the same value. It

is assumed that the throughput function at the default PoA setting is linear because the block frequency of 15 seconds allows the JavaScript application to invoke the *send* method for all 1,000 transactions within the timeframe of the block period. Moreover, the Geth client has enough time left to execute all transactions before the next block is sealed.

As far as the latency is concerned, it is assumed that the floor measured in figure 4.7 and figure 4.9 is because of the same reasons as the throughput saturation. In figure 4.8, the latency function is exactly as expected. Similar to the throughput function in figure 4.6, the non-linear representation is due to the fact that the x values are doubled, while the y values are approximately halved.

In conclusion, provided that the block period is high enough, the effect of an increased block size is as expected. Gathered data from the measurements with the PoW consensus variant indicates that increasing the block size only effects the throughput and latency to a substantial level if the block period is higher than the transaction creation, signing and executing processes.

4.2.3 Workload Type

The workload type, i.e. the individual smart contract, is the third parameter to be analysed in the scope of the present work.

In order to form the expected result, the mechanics for smart contract executions are briefly repeated. Moreover, findings of sources which were identified in the related work section are briefly discussed.

In case the execution of a smart contract's method is to be initiated, a transaction which holds some input data as a payload needs to be sent to the smart contract's address. If the smart contract is a solidity program, the input data is interpreted as a method call and the Ethereum Virtual Machine (EVM) executes the respective method along with the input data. After the method has been executed, the smart contract's state might have changed. Consequently, the new state is updated in the blockchain.

Using the information provided in the paragraph above, it is assumed that the smart contract itself may influence the performance of an Ethereum blockchain due to the following two reasons.

First, the amount of gas needed for the smart contract to be executed affects the number of transactions fitting in a single block, which is limited via the gasLimit value. This is due to the reason that a fee for executing transactions has to be paid. More precisely, a fee for each instruction (e.g. *SSTORE*, *CALL* or *ADD*) which needs to be executed by the Ethereum Virtual Machine (EVM) has to be paid in gas. As executing a method may require more or less EVM transactions to be performed, the amount of gas needed for executing a smart contract method varies as well.

Second, the individual instructions to be executed by the EVM affect the time needed for processing a transaction. In case the current smart contract state is changed, the new state needs to be updated in the blockchain, which also takes its time. Moreover, as the runtime for each individual low level instruction also varies among the instructions, the amount and type of instructions needed may also impact the performance.

Findings reported in related work also indicate that there are considerable performance differences across smart contracts. For example, Dinh et al. report a drop of 10% in throughput and a 20% increase in latency when comparing different smart contracts in their Blockbench paper [29] (see figure 3.1). In addition, Zheng et al. report in a paper [57] that a throughput spread ranging from 0.56 tps to 7.93 tps with different smart contracts and the same blockchain parameters was measured.

Due to all these reasons stated above, the expected result for the effect of the workload type on the throughput and latency is that there are measurable differences between the two smart contracts. As the account contract, in contrast to the ballot contract, requires to update the states of two instead of one smart contract, it is expected that the account contract shows a slightly worse performance than the ballot contract.

Turning to the actual analysis of the effect of workload on throughput and latency, the

amount of transactions fitting inside a block was investigated at first. Using Geth's JavaScript console and the *eth.getBlockTransactionCount()* method, the amount of transactions included in various blocks was queried. It has been found that there are indeed different block-transaction-counts for different workloads. For the account contract, only 146 transactions could fit in a single block, whereas for the ballot contract, 170 transactions could be put in a block. When compared to the account smart contract, around 16.4% more transactions can fit inside a block when using the ballot workload. Hence, it was concluded that executing the *transferEther()* method of the account contract needs more gas than executing the *vote* method of the ballot contract. Dividing the gasLimit of a block with the block-transaction-count has revealed that around 32k gas is needed to execute the *transferEther* method, while only 27k gas is required for the *vote* method. To put these numbers in a context, the base fee of 21k gas [3], which is charged for any transaction on the Ethereum blockchain, can be used.

The effect of different workloads is visualised via figures 4.10 and 4.11. The results surprisingly show that there is no difference between the two workloads in the PoW variant, while there seems to be a difference with the default PoA settings. In order to statistically analyse if the average (expected) values differ significantly across the two workloads, the Welch's t-test was used. With a significance level alpha of 5% the results of the t-test showed that there is no significant difference of throughput nor latency in the PoW variant. In contrast, for the PoA variant, the null hypothesis (no significant difference in means for throughput and latency across the two workloads) could be strongly rejected. Thus, a significant difference in means of throughput and latency across the workloads could be statistically proven.

Similar to the analysis of block size in the previous section, it is argued that the inconsistent results between the PoW and PoA variant are due to the different default block frequencies. It is assumed that measurements with the PoW variant did not show any significant differences between the two workloads because the system has already been operating on its limits. The fact that the maximum block size could not always be fully utilized, i.e. the block-transaction-count was sometimes lower than 146 or 170, is assumed to be an underlying root cause for the absence of a significant difference between the workloads. Another possible root cause could be the randomness factor which is included in the mining process.

In summary, assuming that the block period is high enough, the results of the measurements with the default PoA settings suggest that the workload, i.e. the smart contract, indeed affects the performance of an Ethereum smart contract platform. The throughput difference between the workloads was measured with around 17%, whereas the latency difference was roughly 11%. The results obtained further suggest that the amount of gas needed and the amount of state changes are indicators for the time needed to execute the transaction and the further smart contract method call.

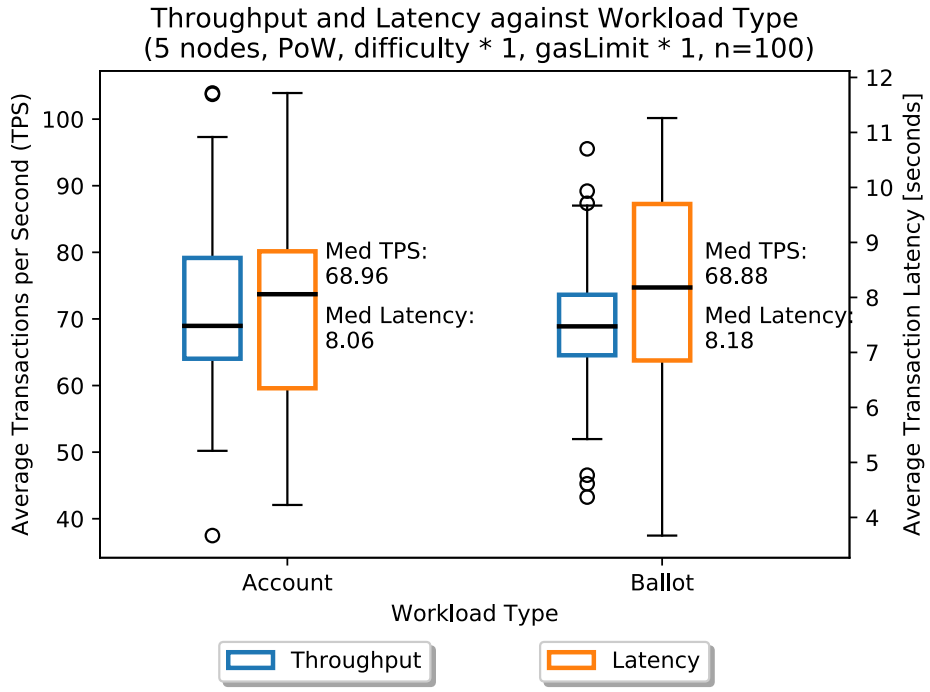

Figure 4.10: Throughput and Latency against Workload Type (PoW)

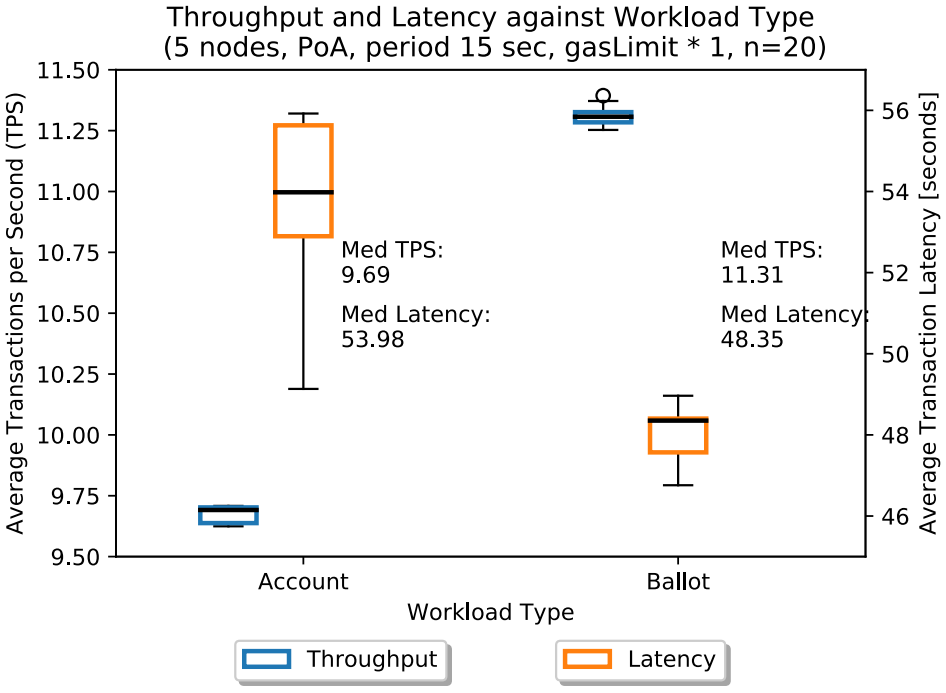

Figure 4.11: Throughput and Latency against Workload Type (PoA)

4.2.4 Node Configuration

A parameter where the scalability is of special interest is the node configuration. Yet, only one source identified as related work includes results on performance differences with different node configurations.

Findings of a paper [46] include that the average transaction processing time on Geth decreases with an increasing amount of RAM. In particular, when varying the amount of RAM of a node from 4 GB to 24 GB, the average time needed for processing a transaction decreased by 25%. As the amount of RAM is also varied between the machines used for the experiments on node configuration of the present work, it is assumed that the time needed to process the transactions decreases significantly when scaling up the RAM.

As mentioned in table 4.4, four different EC2 instance-types were selected for the examination of the node configuration parameter. Each instance-type includes twice as much computational power (CPU and RAM) as the previous instance-type. Next to the increased amount of RAM, the enhanced CPU power should be especially relevant for the PoW variant. This is due to the reason that the hash-rate of a node should rise with its CPU power. A higher hash-rate thereby theoretically decreases the time needed to find a new block in the PoW mining process. Thus, the block frequency might change with a different node configuration. However, due to the experiments' validity, no other independent variable should be altered. Aside from that, the block frequency has already been thoroughly analysed in section 4.2.1. Therefore, in order to mitigate the effects of block frequency changes, the `difficulty` parameter of the *ethash* algorithm was set to the minimum amount possible. On top of that, the `gasLimit` parameter was set to a value so that the whole workload quantity of 1,000 transactions can fit inside a single block (default * 8). This is due to the reason that the system should operate at its limits when measuring the effect of the node configuration. The consensus algorithm parameters should thereby not limit the performance of the node. Consequently, for the PoA variant the, gasLimit value was also set to the value of default * 8 and the `period` parameter was set to 1 second.

Regarding the expected result, it was assumed that there are significant, yet quite marginal improvements in the PoW and the PoA variant. Based on findings in related work, it is expected that the time needed to execute the workload decreases with rising computational power. From this it follows that, with a more powerful node configuration, the throughput should increase while the latency should diminish. Although the amount of RAM and CPU are doubled with each different node configuration, it is not expected that the throughput doubles, nor that the latency halves.

The results obtained via the measurements on the node configuration parameter are depicted via figures 4.12 and 4.13. Overall, the obtained result matches the expected result of increased throughput and decreased transaction latency for raised computational power. However, surprisingly the differences between the node configurations are not as minor as expected. Another remarkable observation is that there seems to be a performance difference between the PoW and the PoA variant.

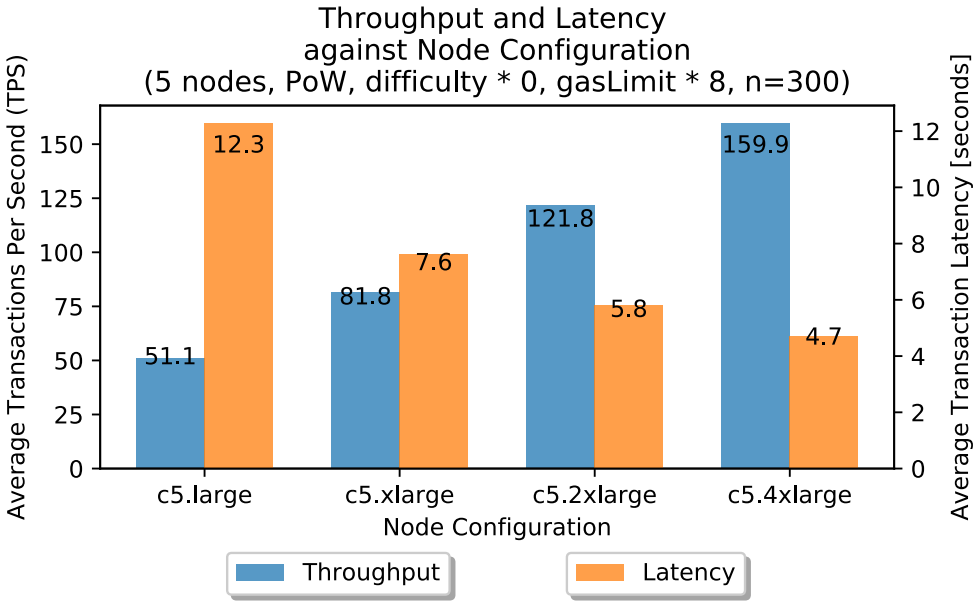

Figure 4.12: Throughput and Latency against Node Configuration (PoW)

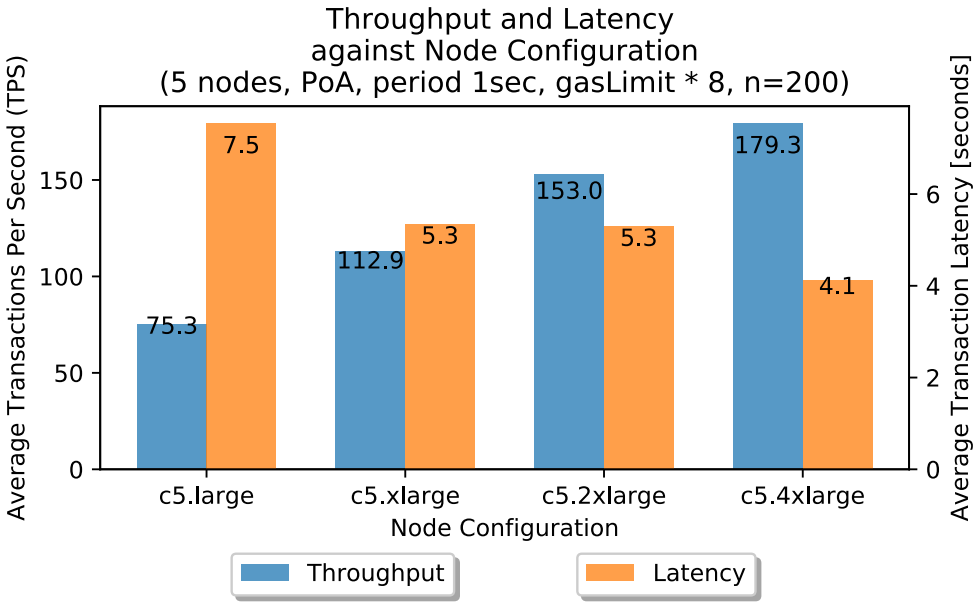

Figure 4.13: Throughput and Latency against Node Configuration (PoA)

Turning to the results obtained in more detail, one can clearly see that a significant and continual improvement in performance was observed when changing the node configuration parameter. The improvements listed in percent, i.e. the scalability, were calculated for each node configuration pair and are represented via table 4.7. Remarkably, a drop of scalability between the node configuration pairs is evident. For example, while the throughput scalability for a change from c5.large to the c5.xlarge nodes was measured with around 60% for the PoW variant, only an improvement of around 31% was measured when switching from c5.2xlarge to c5.4xlarge nodes.

Table 4.7: Mean Scalability of Node Configuration Pairs

Metric	**Consensus Variant**	**large to xlarge**	**xlarge to 2xlarge**	**2xlarge to 4xlarge**
Throughput	PoW	60.27%	48.9%	31.22%
	PoA	49.92%	35.55%	17.18%
Latency	PoW	37.82%	24.14%	18.89%
	PoA	29.03%	1.05%	22.05%

As the obtained performance differences were greater than expected, the author has tried to find the underlying reason(s). In addition, the root cause(s) for the declining scalability between the different node configurations was/were analysed.

At first, block frequency changes came into mind due to the improved hash-rate. As the block frequency with the default difficulty of 524,288 (factor 1) and five c5.xlarge nodes was already measured with around 1.1 seconds (see section 4.2.1), the author considered that even the weaker c5.large nodes should be easily able to provide enough power to arrive at a block frequency of 1 second with only the minimum mining difficulty of 131,072 (0x0). In addition, the block frequency could be excluded from the list of underlying reasons since vast performance differences were also obtained with the PoA variant (where the block frequency is static per design).

Next, the transaction and block propagation in the network was analysed. This is due to the reason that, during the experiments, it was observed that sometimes a node was not able to successfully propagate the transactions in its transaction pool. In particular, the c5.large and the c5.xlarge instance-types were occasionally not able to propagate the full workload to the network. Hence, with the PoW variant, the mining power of the remaining nodes in the network was wasted from time to time. As the stronger instance-types did not show any propagation difficulties, it is assumed that, for the PoW variant, this effect is partially responsible for the scalability decrease between the node configurations (as depicted in table 4.7). Nevertheless, with the PoA variant, the scalability decrease is also present although the performance should theoretically not change if the transactions cannot be propagated to the network. This is due to the reason that all nodes exhibit the same static block period of 1 second.

Despite that, one may want to know how much the communication overhead and the consensus costs actually affect the different node configurations.

In order to quantify the communication overhead for propagating transactions and blocks to the network and afterwards finding consensus, the results obtained with a node which had four peers were compared to the results measured on a node without any peers. The measured throughput and the latency of a node without any peers and therefore without any communication overhead nor consensus costs is illustrated via the next figure.

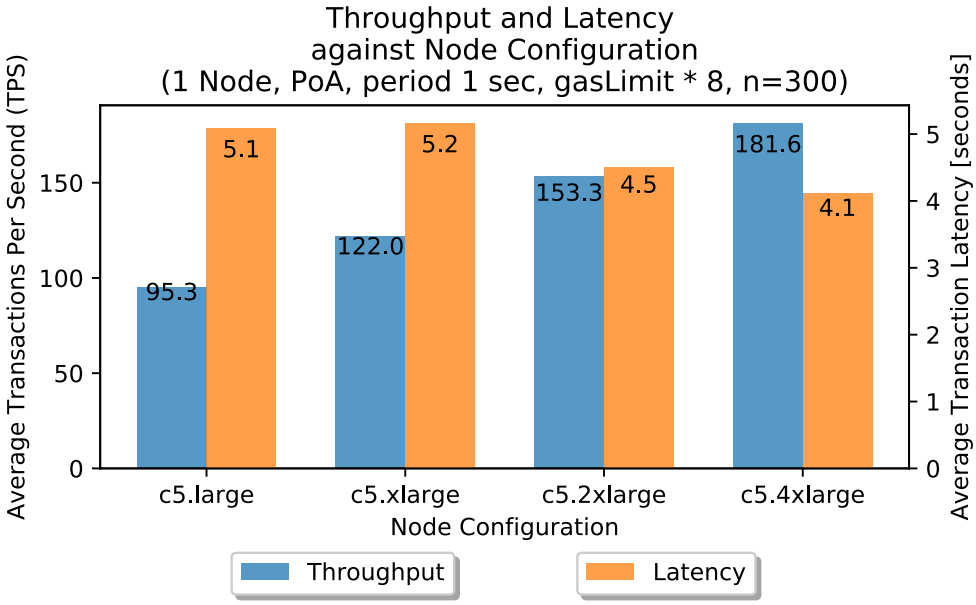

Figure 4.14: Throughput and Latency against Node Configuration (1 node, PoA)

From this figure it can be derived that, similar to the results obtained with the five node network, with improved node configuration, the throughput increases continuously whereas the average transaction latency declines. When comparing figures 4.13 and 4.14, one can clearly see that, for each node configuration, the throughput is higher and the latency is lower in the single node network than in the five node network. Interestingly, the difference between the single node and the five node network is higher for computationally weak node configurations. For example, the throughput deviation with the c5.large instance-type was measured with roughly 20 tps (95.3-75.3), whereas the deviation for the c5.4xlarge instance-type was infinitesimally small. Overall, throughput loss due to the network communication overhead and the consensus costs was measured with around 23%, 8%, 0% and 1% for the different node configurations.

Hence, it has been concluded that the different communication and consensus handling capabilities are partially responsible for the decreasing scalability difference depicted via table 4.7. Furthermore, the first out of a few reasons was identified which partly explains the differences between measured and the expected result.

Nonetheless, it was assumed that different network communication and consensus handling capabilities between the node configurations were not the only factor contributing to the huge performance differences. More underlying root causes still needed to be found.

As a more powerful machine theoretically allows the Geth client a faster transaction validation, quicker transaction and smart contract execution via the EVM and faster blockchain state changes, the time needed for these parts were analysed as a next step.

In order to measure the runtime needed for the mentioned steps, a new measurement approach had to be applied. Instead of starting the time measurement for the experiments immediately when a transaction is created, the time was only measured from the moment when the full workload had been placed in Geth's transaction pool. Moreover, processing the transactions in the process pool was prevented until the full workload of 1,000 transactions had been placed in the transaction pool. This is in contrast to the approach used for all other experiments where the generation of the workload and the processing of transactions typically overlap, i.e. processing starts at a time were the full workload typically has not yet been generated.

The measurement runtime for both, the full runtime and the runtime for only the transaction execution and the blockchain state changes are illustrated in figure 4.15 below. While the full measurement runtime is displayed in black, the runtime needed for executing the workload and changing the blockchain states is displayed in brown. In addition to the plot, table 4.8 on the next page comprises the measured values.

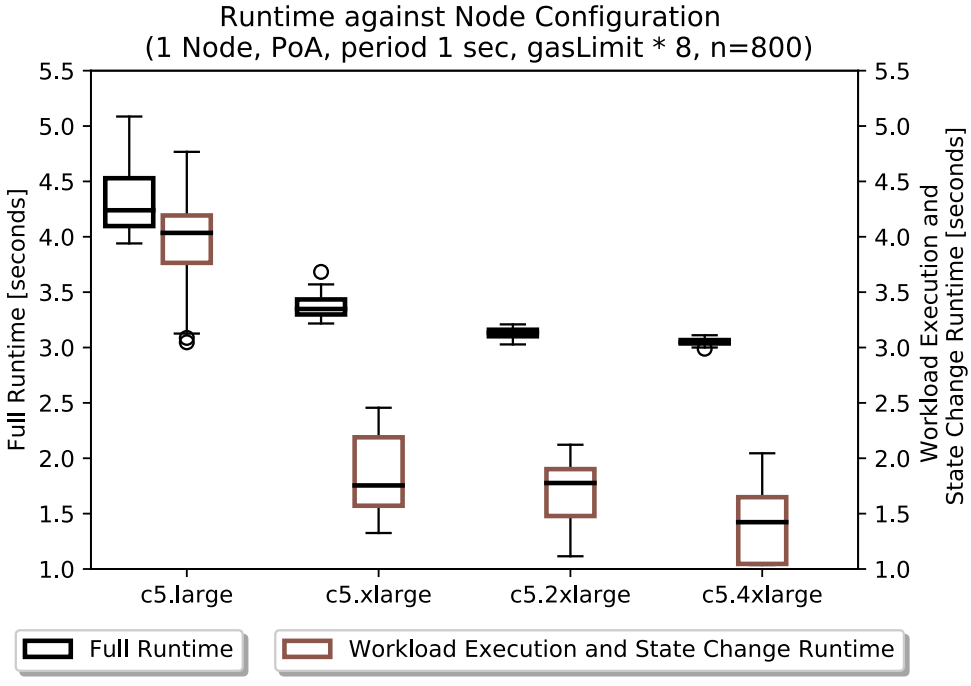

Figure 4.15: Runtime Analysis of Different Node Configurations

Table 4.8: Measurement Runtime for Different Node Configurations (1 Node, PoA, period 1 sec, gasLimit * 8)

Metric	**c5.large**	**c5.xlarge**	**c5.2xlarge**	**c5.4xlarge**
Full Runtime (Mean)	4.35s	3.37s	3.13s	3.05s
Workload Execution Runtime (Mean)	3.97s (91%)	1.87s (55%)	1.67s (53%)	1.38s (45%)

The plot and the table clearly indicate two phenomena. First, when focusing on the relative differences between the runtimes measured, one can see that, except for the c5.large instance-type, executing the transactions and changing the blockchain states requires approximately half the time of the full measurement runtime. Second, the time needed for executing the workload decreases with increasing computational power of a machine. For example, with the default node configuration (c5.xlarge), the runtime needed for executing the workloads and afterwards changing the blockchain state was measured with around 55 % of the total runtime, whereas only 45% of the total runtime were needed with the c5.4xlarge node configuration.

From that it was inferred that different workload execution and state changing mileages are another factor which contribute to the scalability decrease depicted in table 4.7. Moreover, next to the network communication and consensus handling capabilities, another underlying root cause could be identified which explains why the performance differences between the node configurations were greater than expected.

Next to the reduced workload execution and state changing runtime, the transactions are also generated and signed in a faster way when increasing the computational power of a node. Via querying the transaction-count per block, the author found that, with computationally weaker node configurations, although the block size is big enough, the nodes are not able to put the full workload in a single block. This is presumably due to a combination of a too slow workload generation, a too slow workload execution and a too fast block frequency. In case the full workload cannot be put inside a single block, the fact that at least another full block period has to pass until the rest of the workload can be put inside a block thereby amplifies the impact of a too slow workload generation and -execution.

Finally, the difference between the expected and measured result can be explained via a combination of the following factors: faster transaction generation and signing, different network communication and consensus handling capabilities, quicker execution of the workload and changing of the blockchain state.

Turning to the differences between the PoW and the PoA measurements (compare figure 4.12 and figure 4.13) it is assumed that the mining process simply consumes a lot of computational resources which are not available for generating the workload and executing it. Furthermore, the differences may also partially be due to the reason that, with the PoW variant, the default node configuration (c5.xlarge) was sometimes not able to propagate all its transactions to its peers, i.e. the full mining power was not utilized.

In conclusion, the effect of the node configuration on the performance is surprisingly large if the system is operating at its limits. In a five node network with a block frequency of around 1 second, the throughput almost tripled, while the latency was more than halved with a better node configuration. However, the author wants to mention that the findings on the node configuration parameter need to be interpreted with caution as the workload generation was also improved with a better node configuration. Concluding the analysis on the node configuration it is argued that the node configuration becomes more and more important for the performance of a blockchain network the higher the block frequency is. With a low block frequency, e.g. a block frequency similar to the one of the public Ethereum network, the impact of the node configuration in a private Ethereum blockchain might even be negligible in case the whole transaction life cycle (see section 2.4) can be handled in one block period.

4.2.5 Network Size

Sometimes when one wants to improve the performance of a system, horizontal scaling, i.e. increasing the number of machines, is a reasonable approach. In the blockchain context however, simply increasing the amount of nodes to improve the performance does not always result in the desired effect. Some authors argue that scaling is mainly an issue due to the design of the system where every node needs to process each transaction in the system [21, 25, 37]. Although the processing power of a network increases when a new node joins the network, other factors such as the network communication and consensus costs also play a vital role.

With respect to the scalability in the public Ethereum network, there are simply said no considerable performance improvements because of the time needed to propagate blocks and transactions in the network. The block frequency is artificially held somewhere around 15 seconds by a mining difficulty adjustment algorithm since propagating transactions and blocks simply takes its time in a network as large as the public Ethereum network. In case of Ethereum, one of the founders of the platform states in a blog article [16] that block frequency considerations were impacted by the paper "Information propagation in the Bitcoin network" [27]. One of the major findings of this paper is that it takes roughly 12 seconds to reach 95% of the nodes in the Bitcoin network when propagating a new block. The time needed to propagate blocks to the majority of the network simply cannot be reduced via adding new nodes to the network. On the contrary, it even makes the situation worse since more communication and consensus efforts have to be made. This is why the public Ethereum network is currently at a stage where scaling up to a bigger level is not possible without any design changes.

In a private setting, however, the information propagation is usually small due to the minor amount of nodes in the network. The small information propagation time theoretically allows to improve the performance when adding new nodes to the network. Surprisingly, the single source identified as related work mentioning effects of the network size on the performance of a private Ethereum network, the Blockbench framework [29, 28], shows different results. Instead of throughput improvements, a throughput degradation

was observed where the throughput decreased almost linearly while the latency rose nearly exponentially (for detailed results see figure 3.2 in the related work chapter). According to the authors of the Blockbench framework, the reasons for the throughput degradation were additional network communication costs, nodes not being able to broadcast transactions to their peers, and ethash's difficulty adjustment algorithm which increased the difficulty at a higher rate than the new number of nodes.

As always, the expected result is specified before the results of the analysis are presented. Prior to that, the author wants to recap that the default node configuration for the experiments on network size is the t2.xlarge AWS EC2 instance-type and that experiments were made with network sizes between 1 and 20 nodes. A change from the c5.xlarge to the t2.xlarge instance-type was necessary as, due to imposed AWS restrictions, it was not possible to provision 20 c5.xlarge instances at the same time. Regarding the expected result, the author anticipated that with an increasing amount of nodes in the network, the throughput improves at first until a peak is reached. Improvements are expected because of the increased hash-rate of the network which should result in a higher block frequency. Once a peak has been reached, it is expected that the throughput slightly decreases due to additional communication and consensus costs outweighing block frequency improvements. Vice versa, it is expected that the latency is reduced at first but then slightly increases once a floor has been reached.

The results of the experiments on the network size with the default PoW settings are illustrated via figure 4.16. The figure shows that, at first, the throughput rises and the latency diminishes drastically when adding more nodes to the network. The throughput increase and latency drop can be explained via an increased network hash-rate which reduces the time until a next block is mined. As anticipated, there exists a point where the additional hash-rate cannot be fully utilized anymore and the performance starts decreasing.

In order to analyse the underlying reasons for the performance degradation in more detail, the average hash-rate and the mining difficulty were plotted. Both are illustrated via figure 4.17. Interestingly the average hash-rate of a node is steadily shrinking with an increased network size. This suggests that the t2.xlarge node is not able to cope with the increasing network communication and consensus costs. The assumption that the network overhead and the consensus costs are responsible for the hash-rate decrease is also supported by observations made during the execution of the experiments. It has been observed that the node responsible for taking the measurements could sometimes not propagate the full workload to its peers. Hence, most of the mining power of the network was occasionally not utilized and the performance could not be improved. Next to the decreasing hash-rate, the plot illustrates that the mining difficulty slightly increased when new nodes were added to the network. This minor increase is most probably attributed to ethash's difficulty adjustment algorithm. In contrast to the findings reported by Dinh et al. in the Blockbench paper [29], the results obtained do not show that the difficulty increased at a higher rate than the number of nodes. Nevertheless, the increased difficulty may also be partially responsible for the performance degradation.

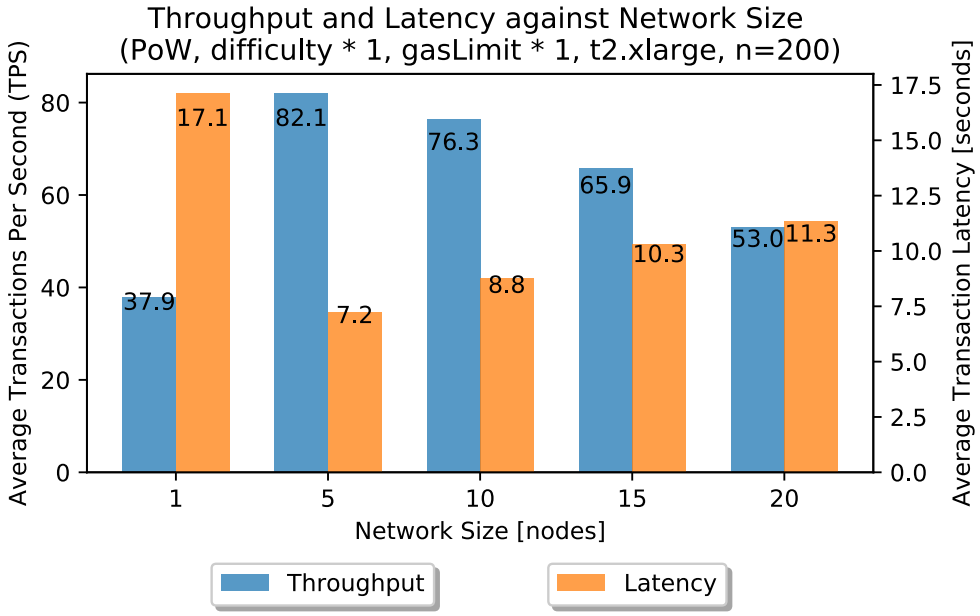

Figure 4.16: Throughput and Latency against Network Size (PoW)

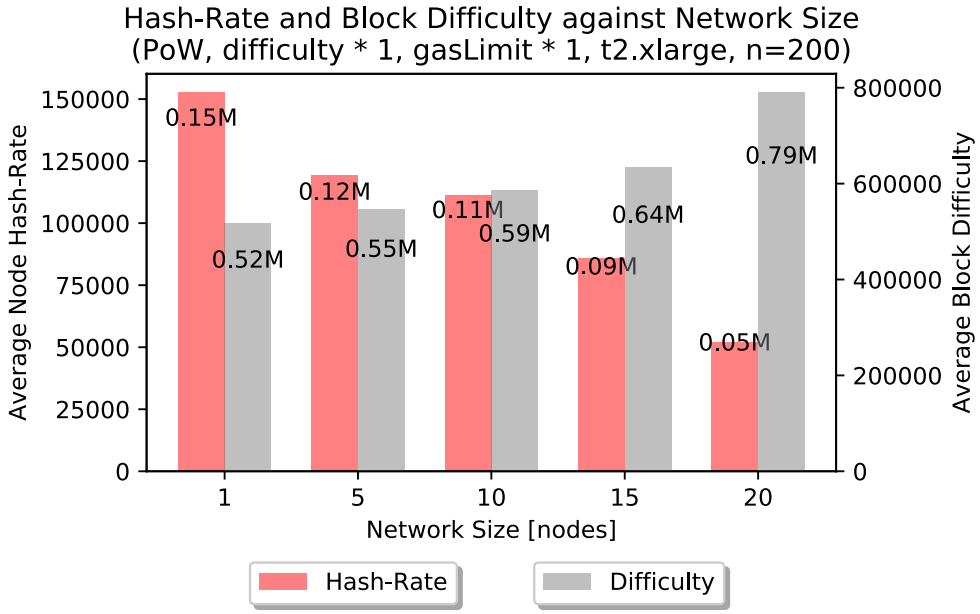

Figure 4.17: Hash-Rate and Difficulty against Network Size (PoW)

Another factor which was partially responsible for the performance degradation was that nodes sometimes got out of sync. In other words, nodes could sometimes not settle on a single canonical version of the blockchain which lead to the creation of *uncle blocks*. These uncle blocks generally have a negative impact on the performance since, with uncle blocks, some nodes are trying to mine a block for transactions which were already included in the blockchain by another node. Hence, nodes are not processing new transactions and mining power is wasted. The occurrence of uncle blocks is inherent in Ethereum and is basically due to the fact that spreading the news of a new block happens via a gossip protocol where not all nodes are going to be informed about the latest block at the exact same time. Although the network latency was minimised for the experiments via keeping all nodes inside the same AWS availability-zone (eu-central-1a), the network latency in combination with a high block frequency and the t2.xlarge nodes not being able to cope with the network communication and consensus overhead seem to be responsible for the creation of the uncle blocks. This assumption has been made since it could be observed that the node propagating all the messages to the network occasionally lagged a few blocks behind its peers. Sometimes the combination of handling the measurement, performing the mining process and handling the network communication unexpectedly resulted in the whole EC2 instance freezing.

Putting all the findings together, the author argues that the effect of the network size highly depends on other parameters as well. The two major important factors are the block frequency and the computational strength of a node. Generally speaking, if the nodes are not able to perform the propagation of blocks and transactions within a short period of time, other parameters such as the amount of nodes in the network or a high block frequency cannot unfold their true impact on the performance. A high block frequency does not have the desired effect if the transactions and blocks cannot be propagated within the network during the time of one block period at a max. In the worst case, when the transactions cannot be propagated at all, it would not matter if there were 1,000 or 1 peer which could contribute to the network's mining power.

As the computational power of the t2.xlarge nodes could not compensate for the network communication and consensus costs, additional measurements with computationally stronger nodes (c5.4xlarge) were made in order to analyse these overheads in more detail. Next to the node-configuration, the mining difficulty was also increased to counterbalance the increased hash-rate of the stronger nodes and to avoid a block frequency of 1 second with only a single node. In contrast to the measurements with the t2.xlarge nodes, it was only possible to study the effect of the network size up to only 10 nodes. This was due to AWS limits regarding the maximum allowed amount of instances which could be provisioned at the same time.

For visual representation of the data obtained with the stronger node-configuration it is referred to figure 4.18. As with the measurements conducted with the t2.xlarge nodes, an additional plot was created which depicts the hash-rate and the block difficulty against the network size (see figure 4.19).

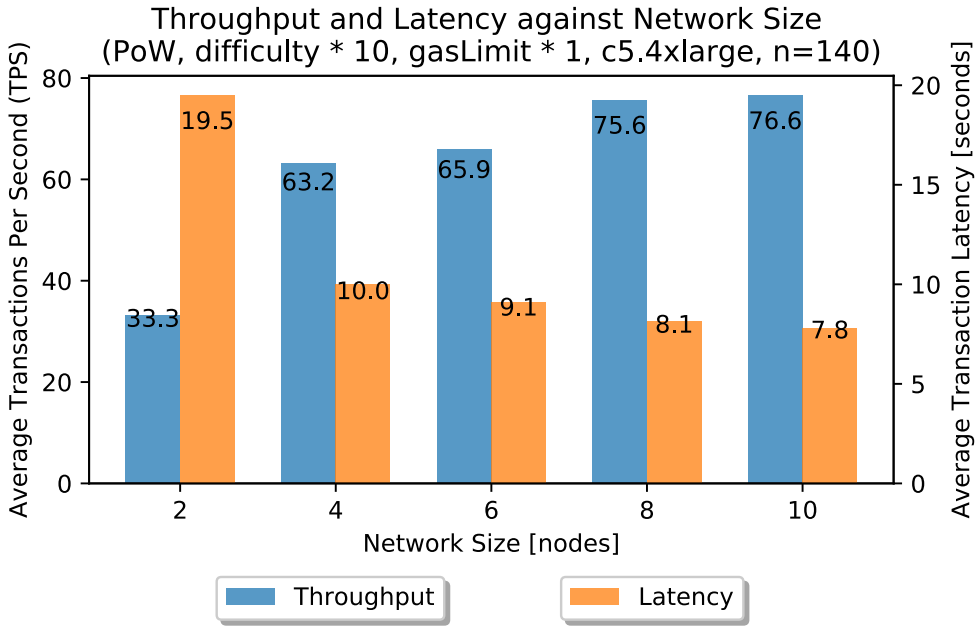

Figure 4.18: Throughput and Latency against Network Size (PoW, difficulty * 10, c5.4xlarge)

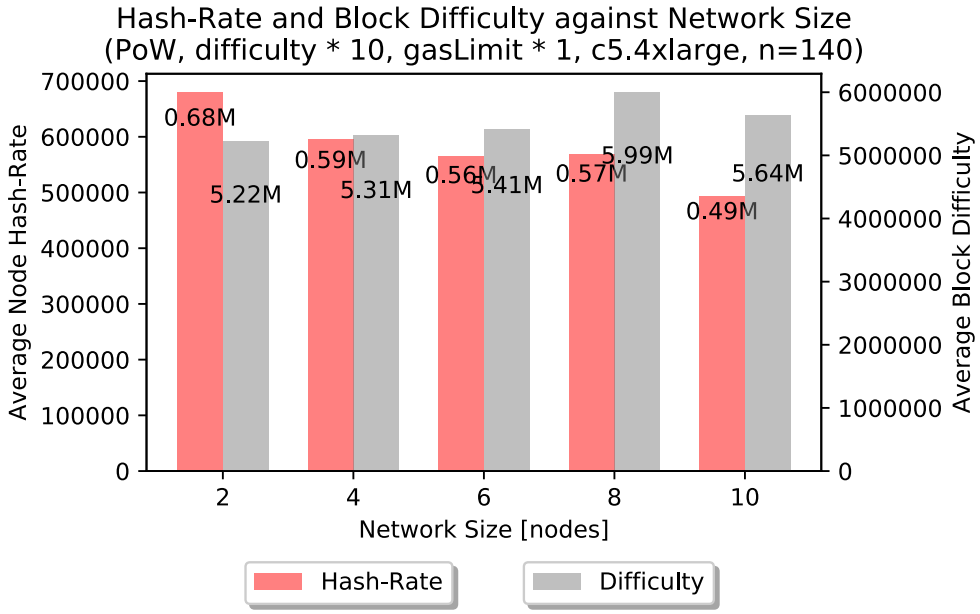

Figure 4.19: Hash-Rate and Difficulty against Network Size (PoW, difficulty * 10, c5.4xlarge)

As illustrated in figure 4.18, the results show that the performance increases until the 10^{th} node. This is in contrast to the measurements conducted with the t2.xlarge node where the performance peak had already been reached somewhere around the 5^{th} node. Considering figure 4.19, one can see that, similar to the results obtained with the weaker node configuration, even the stronger node shows a declining hash-rate for an increased network size. When comparing the average hash-rate of a node with 9 peers (10 node network), surprisingly both node configurations, the t2.xlarge (see figure 4.17) and the stronger c5.4xlarge (see figure 4.19), can only provide about 70% of the initial hash-rate. As the network communication and consensus costs increase with the network size while other costs should stay the same, this is an indicator for a hash-rate loss due to the increased network communication and consensus costs.

For the purpose of examining the impact of the information propagation and consensus costs, a comparison between the expected and the measured throughput has been made. This comparison is listed in table 4.9. Based on the average hash-rates and the block difficulty depicted in figure 4.19, the block frequency could be approximated as a first step. Using the information that the gasLimit provides enough space for 146 transactions to fit inside a single block and the fact that the full workload comprises 1,000 transactions, it was calculated that at least 7 blocks need to be mined for one measurement. Multiplying the block period with the quantity of blocks to be mined, the runtime was calculated as a next step. Finally, the expected throughput could be computed via the calculated runtime and the workload quantity.

Table 4.9: Calculated vs. Measured Throughput (PoW, difficulty * 10, c5.4xlarge)

Network Size [Nodes]	**Calculated Block Period [s]**	**Calculated Runtime [s]**	**Calculated Throughput [TPS]**	**Measured Throughput [TPS]**
2	3,8	26,9	37,2	33,3 (89.4%)
4	2,25	15,75	63,5	63,2 (99%)
6	1,61	11,27	88,72	65,9 (74.3%)
8	1,31	9,2	108,8	75,6 (69.5%)
10	1,15	8,06	124,1	76,6 (61.7%)

Considering the table above, one can clearly see that the expected/calculated throughput and the measured throughput drift apart with increasing network size. While there is a match between the expected and the measured throughput of around 90-100% with smaller network sizes, larger networks only show a match of around 70-60%. The discovery that networks with a higher quantity of nodes, in contrast to networks with a smaller amount of nodes, cannot fully utilize their available resources, is an indicator for the information propagation being the limiting factor.

Based on all the insights gained, the conclusion has been drawn that even with a stronger node, the network communication and consensus costs are still limiting the scalability of the network. Moreover, the time needed to propagate information and arrive at consensus

seems to prevent the system from achieving its full potential. Broadly speaking, a negative correlation between the network size and the performance gain for an additional node could be identified.

Turning to the results obtained with the default PoA settings (see figure 4.20), one can obviously see that there is a huge difference to the findings made with the PoW variant. With the standard PoA settings there is no significant performance difference for changing network sizes. It has been found that this is due to the following reasons. First, the static and low block frequency of 15 seconds allowed the node more than enough time for all transactions to be generated, propagated and processed during one single block period. Therefore, the increased network communication and consensus costs did not influence the performance. Second, even if the transactions could not be propagated at all, the performance of the system was also not affected. This is due to the reason that all the nodes were sealing blocks at a static pre-assigned time interval. In summary, with the PoA variant changes in the network size simply do not impact the performance substantially because of the consensus algorithm's design.

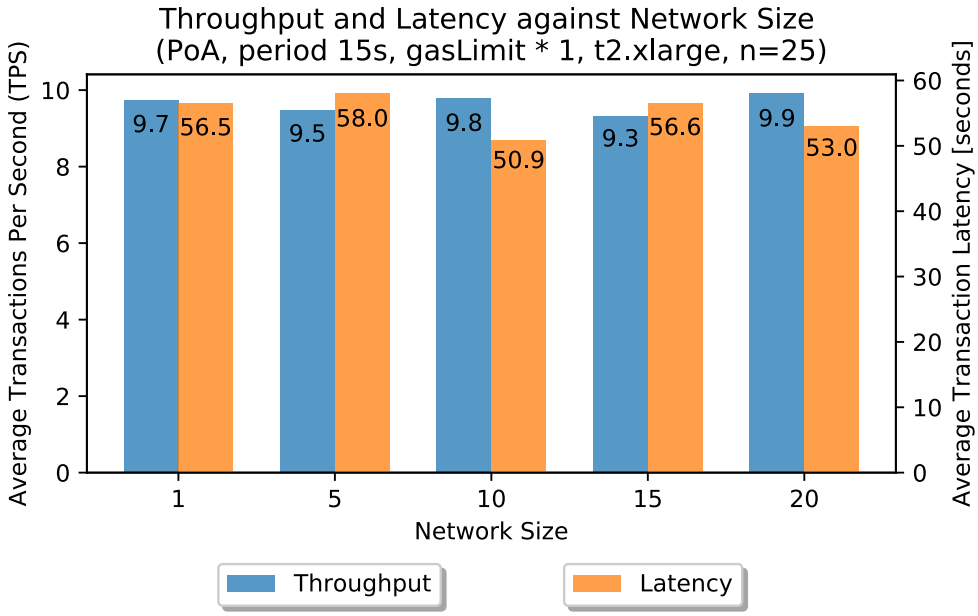

Figure 4.20: Throughput and Latency against Network Size (PoA)

In conclusion, concerning the network size parameter, the expected result matches the measured result quite well. The most striking result to emerge from the experiments on the network size is that, when using the PoW consensus algorithm, the network communication and consensus costs are the main limiting factors regarding the scalability of the platform. Nevertheless, when considering the effect of the network size on the performance, the author once again wants to highlight that other parameters such as the node configuration and the block frequency also have to be taken into account.

4.3 Identified Bottlenecks

After having analysed the effects of different parameters with respect to performance and scalability in the last section, the purpose of this section is to discuss the bottlenecks of a private Ethereum system. The author wants to emphasize that the mentioned bottlenecks are specifically relevant for private Ethereum blockchains. For the public version of Ethereum, different or additional bottlenecks may be in place. For example, in contrast to a private setting, the growing storage needed for a single node can be an issue in a public setting.

Before turning to the bottlenecks identified, a brief recap of the findings reported by other authors is provided at first. In the two papers by Dinh et al. about their Blockbench framework [29, 28], the authors state that, whereas the consensus protocol is the bottleneck for the Geth client, the bottleneck for the Parity client is caused by transaction signing. Zheng et al., on the other hand, report in their paper about a real-time performance monitoring framework for blockchain systems [57] that peer discovery, transaction propagation and the consensus-cost represent the bottlenecks in Ethereum.

Turning to the bottlenecks identified within the scope of the present work, the author wants to emphasize that one result of the analysis performed on performance and scalability is that the effects of one parameter are also highly dependent on the settings of other parameters. Hence, when altering one feature of the system, it is possible that another parameter becomes the bottleneck. In contrast to the available literature, it is therefore argued that only quoting some parameters is not enough. As the bottlenecks may shift from one parameter to another, additional information about the order of the bottlenecks should be added. Therefore, the author proposes a layered bottleneck structure consisting of different parameters where a higher layer gets relevant once the layer beneath does not longer represent the bottleneck anymore.

Based on information gained on the design of a blockchain system in general and the results presented in section 4.2, it is argued that the combination of the block frequency and the block size parameters represents the bottom layer of this structure. This is due to the reason that, by design, the performance of a blockchain system is mainly a function of the block frequency and the block size. In an ideal hypothetical setting, the block frequency would be 0 seconds and the amount of transactions which can be appended to the blockchain in one batch is infinite. However, there are artificial constraints in place which limit the size of a block and the minimum time passing between two blocks. Another reason why the combination of block frequency and block size represents the bottom layer in the bottleneck structure is that improvements in other parameters, such as the node configuration or the network size, are only enabled if the settings for the block frequency and block size allow it to. Due to the limited block frequency and block size, transactions will queue in the transaction pool if the workload is larger than the block size of a single block, even with the strongest node in the world. A stronger node is also limited by the block frequency if the block period is greater than the time-frame ranging from the creation of the transactions to the time where all nodes in the network

have successfully updated their blockchain status. For example, in a setting where a new block is available only every 15 seconds, but the timespan ranging from the creation of a transaction until all the nodes in the network have successfully updated to the new blockchain state takes only 1 second, the system is running in idle mode for about 14 seconds waiting for the next block. One could argue that, when using the PoW consensus algorithm, the block frequency can be improved with computationally stronger nodes. Nevertheless, the difficulty value set in the genesis.json file is still able to nullify the improvement gained from changing to a stronger machine. Despite the fact that there is a current minimum block period of 1 second in both consensus algorithms, PoW and PoA, the block frequency was not identified as the one and only bottleneck. This is due to the reason that findings made during the analysis on performance and scalability suggest that, with the minimum block frequency of 1 second, the bottleneck shifts to processes such as transaction signing, executing the transactions, changing the blockchain state and propagating information.

Provided that the block size is larger than the workload to be processed, the node configuration parameter represents the bottleneck if the nodes in the network cannot generate and execute the transactions during the timespan of one block period. With regard to figure 4.21 on the next page, this is exactly what has happened in a setup with only a single c5.4xlarge node, a block frequency of 1 second and a block size providing enough space for the whole workload to fit inside a single block. The values obtained for the full measurement process reveal that only about a third of the maximum performance could be achieved. Although the block frequency and the block size allow a throughput of 1,000 transactions per second, only about 328 tps were achieved on average. Generating a single transaction and confirming its successful execution took roughly 2.1 seconds on average instead of less than 1 second. In order to break the time needed for a full measurement down into two parts, additional measurements were conducted. Similar to the measurements carried out for figure 4.15, the time measurement was started from the moment where all transactions were pending in the node's transaction pool. Furthermore, sealing the blocks was only started at the moment the full workload was queuing in the transaction pool. Using this approach, the full measurement process could be separated into two independent parts. The first part reflects the efforts needed for creating, signing and pushing the transactions to the local transaction pool of a node, which is not separately illustrated in figure 4.21. The second part reflects the time needed for validating and executing the transactions in addition to an subsequent update of the node's blockchain state. Comparing the runtime of the two measurement approaches (see table 4.8) reveals that around 55% of the time was spent for the first part while 45% were utilized for the second part. As generating and signing the transactions takes a little bit more time than executing the transactions in the EVM and updating the blockchain state, the first part was declared as the bottleneck within the node configuration parameter. The finding that the transaction signing is in fact a bottleneck was also reported by other authors. For example, one of the co-founders of the Ethereum blockchain, Vitalik Buterin, states in the report 'Ethereum: Platform Review - Opportunities and Challenges for Private and Consortium Blockchains' [18] that "the time bottleneck is going to be

cryptographic verification, not processing a few if-then clauses" [18]. Furthermore, the authors of the paper 'Analysis of Blockchain technology: pros, cons and SWOT' [43] mention that the generation and verification of signatures is computationally complex and constitutes the primary bottleneck in products like theirs. Due to all these reasons stated, the node configuration constitutes the second layer of the proposed layered bottleneck structure.

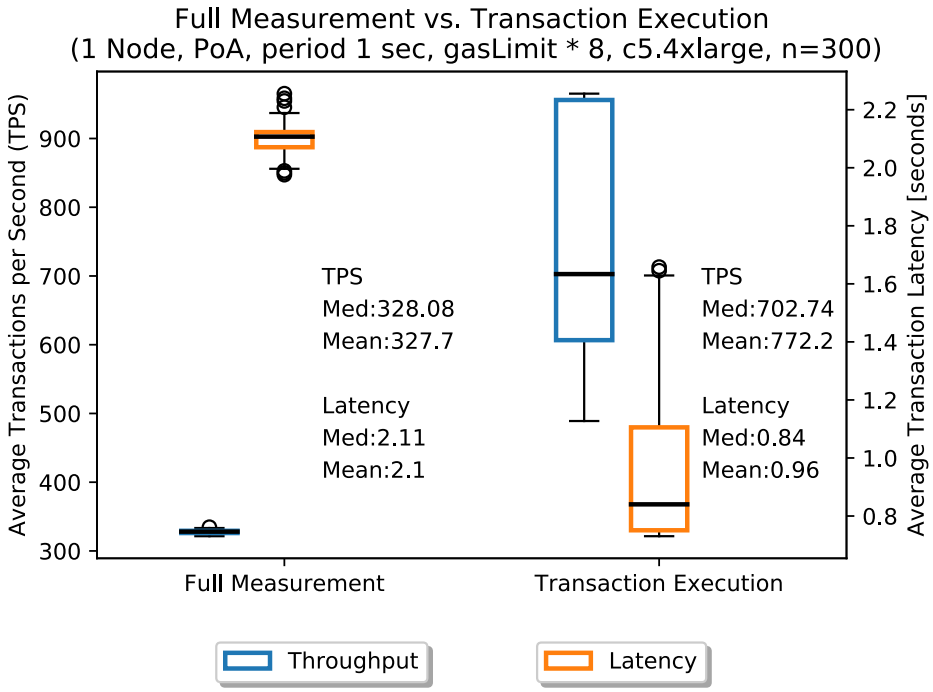

Figure 4.21: Full Measurement vs. Measurement on Transaction Execution

The third and final layer of the bottleneck structure is represented by the network size parameter. This is simply due to the increasing overhead induced to the network when adding a new node. This overhead limits the minimum block period of the system due to the following reasons. First, the information propagation throughout the system takes more time as transactions and blocks have to be propagated to more peers. Second, the consensus costs increase as each node only propagates received blocks to its peers if the prior block-validation has been successfully performed. In a nutshell, if the sum of the time needed for propagating the transactions and blocks to the network and the time needed for validating received blocks from peers is greater than the block frequency, the system cannot achieve the performance predefined via the block frequency and the block size. This is exactly what was observed in the experiments carried out for the analysis of performance and scalability (see figure 4.18). Additionally, the author wants to highlight that, in contrast to the public Ethereum network, the information propagation and consensus costs are not recognized as the leading factors limiting the scalability of a private Ethereum smart contract platform. This is due to the reason that the network

size is much smaller in a private context. Hence, when operating the network on its limits, parameters such as the block frequency, the block size and the node configuration rather restrict the performance before the performance is further diminished through an increased network size.

Finally, the identified bottlenecks for the private Ethereum networks using the current version of Geth are summarized in figure 4.22 below.

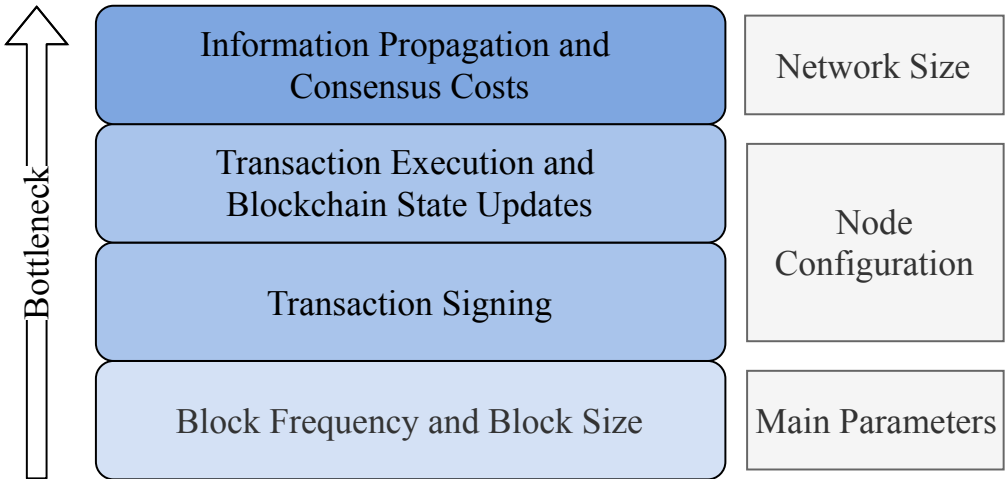

Figure 4.22: Identified Bottleneck Structure

In order to conclude the discussion on the bottlenecks of a private Ethereum smart contract platform, the author wants to emphasize that tuning some parameters can certainly increase the performance, yet the main bottleneck lies in the design of the system. Currently, the Ethereum blockchain simply does not scale that well because a transaction has to be processed by each node in the network, i.e. the blockchain algorithm is serial. These considerations are in line with the ones reported by several other authors [37, 25, 32]. Moreover, the Ethereum community acknowledges the need for an improved system design and is currently working to include concepts such as sharding (see section 2.5) to future versions of Ethereum.

CHAPTER 5

Discussion

In this chapter the detailed results are summarized and the important aspects are highlighted. Furthermore, the results obtained are compared with the ones reported in the related work and variations are clarified.

In order to develop a concept for measuring the performance and scalability of a private Ethereum network, a literature study was carried out as a first step. Based on information gained from this literature study, eight parameters were identified which impact the performance and scalability of a private Ethereum smart contract platform. Out of these parameters, five were chose to be included for a detailed analysis. The literature study has shown that, to the best of the authors knowledge, previous work did not address the effect of these five parameters on performance and scalability in a private Ethereum context to a sufficient extent. Previous work primarily focused on the block frequency and block size parameters for blockchain systems in general. However, other parameters such as the node configuration or the network size were heavily neglected. For Ethereum in a private setting, only very few researchers reported performance measurements with differently configured networks. Out of these identified sources, none included results with the PoA consensus variant, but only with the PoW variant. As the PoA consensus algorithm is especially tailored to private blockchain scenarios, an integral part has been missing in the existing literature.

Regarding the experiment setup, some decisions had to be made which may have influenced the obtained results. The author wants to highlight that the overall approach for the experiment setup was not to chose software, library and APIs which maximize the performance, but the ones which are used by most developers and users. For example, despite Parity showing a reportedly superior performance than the Geth client [46], the latter is currently the most used Ethereum client [10]. Therefore, it was chosen as the client to be used for the experiments. Moreover, the Web3.js JavaScript API was selected for interacting with the Ethereum node as it is Ethereum's default API. The author wants to emphasize that other forms of interaction may actually show a different performance.

Concerning the measurements and the transaction sending, an approach where a client sends the full workload to the network in an asynchronous manner was chosen. This approach is similar to the one reported in a paper about the effects of a varying workload size [44]. In contrast, the authors of the Blockbench framework [29] used an approach where various clients sent transactions to the network at a pre-defined rate (8 tps to 1024 tps). However, the author decided to use the first approach due to simplicity reasons. Furthermore, in contrast to the latter approach, the first approach allows to keep the workload quantity static. Concerning the peer discovery in the network the proposed concept included a dedicated Bootnode whose sole purpose is to connect the nodes with each other. However, after having performed the measurements in networks with up to 20 nodes, it can be concluded that the Bootnode is not recommended for networks with more than around 10 nodes. The experience was made that, with more than 10 nodes in the network, the peers did not find each other every time. In order to have all the nodes find each other in a network with more than 10 nodes, the Bootnode and the Geth clients had needed to be restarted several times until finally each node knew all his peers. As a consequence, the author now recommends to manually specify each node's peers, if the setup allows to.

Considering the maximum throughput reported in sources identified as related work and comparing them with the maximum throughput measured within the scope of this work, one can clearly reason that the parameter settings of a private Ethereum network have a tremendous impact on the performance. The range of reported values for the throughput ranged from a maximum of 284 tps in the Blockbench framework [29], to a minimum throughput of around 0.49 tps, as reported by Chen et al. [22]. The results of the experiments conducted within the scope of the present work have shown that, with a block frequency of 1 second, a block-size providing enough space for 1,000 transactions to fit inside a block, a c5.4xlarge AWS EC2 instance and a network size of only 1 node, the maximum throughput can be as high as to 328 tps on average (see figure 4.21).

Once the proposed concept for measuring the performance and scalability was practically implemented and all the data was gathered, an analysis on the effect of the five included parameters was carried out. In the following paragraphs the results for each parameter are discussed and variations with the related work are clarified.

For the first parameter, the block frequency, the results show that, when increasing the mining `difficulty`, the block frequency decreases and therefore the throughput diminishes and the latency rises accordingly. Although this is exactly as expected, the following findings have been made. First, there exists a saturation point as the minimum block frequency was identified with 1 second. Second, the difficulty adjustment algorithm implemented in ethash may lead to drastic performance changes due to an increased mining difficulty over time. For example, during roughly 12 minutes, the block difficulty value was increased by around 61.5%. In addition, a negative correlation of the throughput and the past time was identified. The throughput diminished by around 0.4 transactions per second for each subsequent measurement. Third, an increase in difficulty does not result in block frequency changes in the same proportion. For example, the

default mining difficulty resulted in a block frequency of roughly 1 to 1.1 seconds but the default difficulty multiplied with a factor of 4 resulted only in a block frequency of around 2.5 seconds. For the PoA variant, the results obtained match the expected result. When doubling the block `period`, the throughput is roughly halved and the latency is nearly doubled. Although not a single source could be identified which has already discussed the effect of a varying block frequency in a private Ethereum network, the results are consistent with the more general findings already reported. For example, the results obtained within the scope of the present work can confirm the work done by Cronman et al. [25] which shows that a re-parametrisation of the block interval may have a positive effect on performance and scalability. Furthermore, the results are in line with the considerations of various blockchain communities which discussed an increase in block frequency in order to improve the performance of a blockchain. Nevertheless, the results obtained also confirm the argument provided by Buterin in a blog post [16] stating that the block propagation time in the network imposes restrictions on how low the block frequency can actually be. Finally, the author of the present work wants to highlight that an increase in performance due to a raised block frequency may negatively impact one of the most vital arguments for a blockchain, the security.

The analysis of the second parameter, the block size, indicates that the combination of block frequency and block size is of utmost importance. With the PoA variant, as expected, a performance boost in the same proportion of the block size increase was measured. For example, when doubling the block size, the throughput nearly doubled and the latency was nearly halved. Unexpectedly, the results obtained with the PoW consensus algorithm show a saturation point. An in-depth analysis has revealed that the saturation could be explained via the combination of a too slow workload generation and a not fast enough transaction execution. The most important finding made is that an increment in block size only affects the performance to a substantial level if the block period is higher than the time needed for creating, signing, propagating and executing the transactions as well as finding consensus. Unlike the authors of the Blockbench framework [29], the author of the present work argues that, if the above-mentioned prerequisites are fulfilled, an increase in block size has the potential to boost the performance of a private Ethereum network. In contrast to the results obtained by the authors of the Blockbench framework, it could not be observed that an increase in block size results in a decrease of block frequency in the same proportion. However, the obtained results are in general agreement with the work of Xin et al. [55] which highlights block size optimizations as one strategy for improving the scalability of blockchains. The effect reported by Xin et al. where, when increasing the block size, the performance increases at first but diminishes once a specific level is reached, could not be observed. Nevertheless, it is not clear if the block size was just not high enough in the experiments of the present work to detect that effect. Finally, the author wants to emphasize that, similar to the block frequency, the block size may be changed over time if the `targetGasLimit` value of the Geth client and the `gasLimit` value in the genesis.json file are not in line. Although this implies that the performance may potentially vary over time, it is not clear if other authors haven taken this effect into account.

In order to analyse the impact of the workload parameter, two different smart contracts were developed. Whereas the first smart contract, account, depicts the use case of a simple transfer of Ether between two accounts, the second smart contract, ballot, implements the popular use case of voting on a blockchain. The outcome of the analysis on performance and scalability was that there were inconsistencies between the PoW and the PoA variant. Similar to the results obtained from the block size parameter, it was found that this is due to the different block frequencies used. Whereas a block frequency close to 1 second was used with the PoW variant, the PoA variant was run with a block period of 15 seconds. Using a Welch's t-test, it could be statistically proven that there is indeed a statistical significant difference in performance between the account and the ballot contract. The throughput difference measured between the two different workloads was around 17% and the latency difference roughly 11%. Therefore, it has been concluded that the type of workload can impact the performance of a private Ethereum blockchain. The amount of gas needed and the amount of blockchain state changes are assumed to be indicators for the time needed to execute the transactions. The results obtained substantiate previous findings of Dinh et al. [29] and Zheng et al. [57], which both reported differences in performance depending on the type of smart contract used.

The results obtained regarding the node configuration parameter surprisingly show that, when operating a private Ethereum blockchain on its limits, i.e. with a high block frequency and block size, the effect of the node configuration on the performance is quite large. The data gathered showed that the throughput almost tripled and the latency halved with four times computationally stronger machines in a network with five nodes. The unexpectedly high differences between the four node configurations could be explained via the combination of a faster transaction generation and signing, different network communication and consensus handling capabilities, quicker execution of the transactions in the EVM and faster blockchain state changes. An analysis of the time needed solely for executing the transactions and changing a node's blockchain state confirmed the previous work done by Rouhani and Deters [46], which reported a decrease of 25% for processing transactions when changing the amount of RAM of a machine from 4GB to 24GB. Indeed, when changing the node configuration from c5.4xlarge (8GB RAM) to c5.4xlarge (32GB RAM) an improvement of roughly 26% was measured. Nevertheless, in contrast to the present work, previous work has failed to address the aggregated improvements due to computationally stronger nodes. Improvements in processes such as transaction signing, network communication and consensus handling due to nodes with a superior hardware specification were not discussed in the related work. Another finding is that computationally weaker nodes may have problems propagating transactions to the network. Therefore, the performance may be limited. Moreover, performance differences between the PoW and the PoA consensus variant were measured. It is assumed that this is due to the mining overhead imposed by the PoW variant. Hence, the author recommends the use of the PoA algorithm in private Ethereum smart contract platforms. Finally, the author wants to emphasize that the experiments on the node configuration were conducted with a very high block frequency. Moreover, it has to be highlighted that the effect of the node configuration diminishes with an increasing block frequency.

For the last parameter analysed, the network size, it was found that the effect on performance and scalability highly depends on other parameters as well. With the standard PoA settings, no significant performance difference could be observed when changing the network size. This is due to experiment setup where all nodes were sealing blocks in a pre-assigned and static time interval. On the other hand, with the PoW consensus algorithm, the results obtained show that, when adding additional nodes, the performance rises at first but starts to decrease once a peak has been hit. Although it may seem ironical that adding more power to the network can actually reduce its performance, these results are in line with previous work reported by Dinh et al. in their paper about the Blockbench framework [29]. Similar to the authors of the Blockbench framework, the author of the present work found that additional network communication and consensus costs are the main reason limiting the scalability of the network. It could also be observed that, when adding the nodes to the network, the block difficulty was increased by ethash's difficulty adjustment algorithm. However, the data gathered shows that, in contrast to the findings reported in the Blockbench paper, the difficulty increase by the difficulty adjustment algorithm was only marginal and did not nullify the additional mining power provided through the new nodes. Another interesting finding made is that the average hash-rate of a node is steadily shrinking with an increased network size. For example, a node with 10 peers could only provide around 70% of the initial hash-rate. Ultimately, the author also wants to highlight that the effect of an additional node in the network also highly depends on the block frequency, node configuration and the current size of the network. Nodes can get out of sync and uncle blocks may be created if the time needed for propagating information in the network is higher than the block period.

Regarding the bottlenecks of the system, the results obtained confirm the findings made by Dinh et al. [29] and Zheng et al. [57]. Similar to the related work, it was found that the information propagation and consensus-costs are the ultimate factors limiting the scalability of private Ethereum networks. However, utilizing the information gathered via the analysis of five different parameters, the author came to the conclusion that the bottleneck may actually shift from parameter to parameter. Indeed, the combination of the block frequency and block size as well as the node configuration may restrict the scalability of the system before information propagation and consensus-costs even get relevant.

CHAPTER 6

Limitations

Regarding the limitations of the present work, it is plausible that a number of factors could have influenced the results obtained. The major limitations may be the experiment setup and the level of generalisation of the results.

The proposed experiment setup exhibits several sources for possible errors. First, as with nearly any measurement performed, the process of measuring itself might have influenced the system under observation. In the case of this work, a Node.js application had to be executed on each node. Therefore, the Ethereum client had to share the machine's resources with another application. Although this application was necessary as it handles the communication with the Geth client and the master node, it could have reduced the resources available for the Geth client. Next, the additional services used for the peer discovery (Bootnode) and the live monitoring (ETH-Netstats) of the system represent potential sources of unwanted influence. Moreover, the decision to use operating-system-level virtualization (Docker) may have influenced the obtained results to an unknown extend. Finally, the chosen transaction generation and sending approach could be seen as a limitation as there exists uncertainty whether the results obtained can be validated with another approach. For example, an approach could be used where several clients send the workload to the network at a specified rate.

Another major limitation of the research conducted is the level of generalisation. As a major part of the data analysis carried out is only of exploratory or descriptive kind, more inferential approaches may be needed to confirm the findings. In addition, sometimes investigations may have been carried out on a too small scale. Although the aggregated amount of data gathered is equal to around eight days of transactions of the public Ethereum network, the small data basis used for individual plots and findings entails a certain amount of uncertainty. With the vast amount of factors included, more data is needed to further strengthen the validity of the obtained results. Unfortunately, it was not possible to increase the amount of data gathered due to monetary and timely limited resources.

CHAPTER 7

Conclusion

This work has investigated the effects of different parameters in a private Ethereum smart contract platform with respect to performance and scalability. It has been demonstrated that the novel concept proposed for deploying differently configured Ethereum networks and measuring their performance can be successfully used in the field. An analysis based on around 4,000 measurements was conducted and several charts visualizing the impact of parameter changes were created. Summing up the results, it can be concluded that the impact of variations in one parameter is highly dependent on the settings of other parameters as well. However, the results obtained indicate that scaling is only possible to a limited extend due to the current system design. Moreover, this work has highlighted the importance of the computational power of the machine running the Ethereum client and has shown that the type of smart contract is able to impact the performance of the system. The results obtained also indicate that, from a performance point of view, the Proof-of-Authority consensus algorithm is preferable to the Proof-of-Work variant in a private context. Finally, a layered structure depicting the identified bottlenecks and their order of occurrence has been contributed to the current state of knowledge.

Although this study has provided insights into the effects of different parameters on performance and scalability in private Ethereum networks, further research is needed. As security considerations were beyond the scope of the present work, one aspect to consider in the future is to combine the obtained results with implications on security. For example, the tradeoff between security and performance could be analysed. Furthermore, the proposed concept could be readily used to gather more data and focus on the combination of various parameters. Instead of bivariate analyses performed in the present work, multivariate analyses could be applied in the future to enhance the understanding of Ethereum's limitations in the private context. A possible use case is to use 3D scatter plots for visualization purposes. In order to further understand the effects of specific parameters in more detail, additional metrics next to throughput and latency could be measured in the future. For example, the blockchain-specific metrics proposed by

Zheng et al. [57] could be included in the proposed concept. As the current setup is tailored to the Geth client and the web3.js API, abstracting from these technologies and developing a layer which allows using other clients and APIs is a further desirable endeavour. Thus, the blockchain client and the used API could be included as another parameter to be analysed in the future. In addition, a layer could be introduced which allows to easily include new workload types to the framework. Another idea for future work is to automate the analysis of the gathered data. For example, automatically generated reports could be generated which summarize the results of measurements. Moreover, once newer Ethereum clients will be released and the Proof-of-Stake consensus algorithm as well as the concept of sharding will finally be supported, additional analyses focusing on the improvements with these newer clients could be performed. Finally, a comparison between the performance of future Ethereum clients and the performance of different (distributed) databases is proposed.

List of Figures

2.1 Simplified Blockchain Structure 8
2.2 Types of Blockchains and their Characteristics 10
2.3 Ethereum Block Header and Transaction Structure 13
2.4 Overview of the Ethereum Transaction Life Cycle with the web3 API [30]. 18

3.1 Blockbench Evaluation Results: Peak Performance for Different Workloads [29]. 23
3.2 Blockbench Evaluation Results: Node Scalability [29]. 23
3.3 Average Throughput of Ethereum and Hyperledger with Varying Number of Transactions [44]. 24
3.4 Average Latency of Ethereum and Hyperledger Fabric with Varying Number of Transactions [44]. 24
3.5 Performance Differences of the Parity and Geth Client[46]. 25

4.1 AWS Cloudformation Interface for the Experiment Stack 40
4.2 Experiment Setup 43
4.3 Sequence Diagram Illustrating the Measurement Process and the Message Flow Between Entities. 51
4.4 Throughput and Latency against Block Frequency (PoW) 55
4.5 Throughput and Difficulty over Time (PoW, difficulty * 1) 56
4.6 Throughput and Latency against Block Frequency (PoA) 57
4.7 Throughput and Latency against Block Size (PoW) 60
4.8 Throughput and Latency against Block Size (PoA) 60
4.9 Throughput and Latency against Block Size (1 node, PoW) 61
4.10 Throughput and Latency against Workload Type (PoW) 66
4.11 Throughput and Latency against Workload Type (PoA) 66
4.12 Throughput and Latency against Node Configuration (PoW) 68
4.13 Throughput and Latency against Node Configuration (PoA) 68
4.14 Throughput and Latency against Node Configuration (1 node, PoA) . . . 70
4.15 Runtime Analysis of Different Node Configurations 71
4.16 Throughput and Latency against Network Size (PoW) 75
4.17 Hash-Rate and Difficulty against Network Size (PoW) 75
4.18 Throughput and Latency against Network Size (PoW, difficulty * 10, c5.4xlarge) 77

4.19 Hash-Rate and Difficulty against Network Size (PoW, difficulty * 10, c5.4xlarge) 77
4.20 Throughput and Latency against Network Size (PoA) 79
4.21 Full Measurement vs. Measurement on Transaction Execution 82
4.22 Identified Bottleneck Structure 83

List of Tables

4.1 Identified Performance and Scalability Parameters 35
4.2 Definition of Evaluation Metrics 36
4.3 Overview of Main Technologies and Tools 44
4.4 EC2 Instance Types used for Node Configuration Measurements 49
4.5 EC2 Instance Types used for Network Size Measurements 49
4.6 Default Values for the Independent Variables 53
4.7 Mean Scalability of Node Configuration Pairs 69
4.8 Measurement Runtime for Different Node Configurations (1 Node, PoA, period 1 sec, gasLimit * 8) 72
4.9 Calculated vs. Measured Throughput (PoW, difficulty * 10, c5.4xlarge) . 78

List of Abbreviations

AWS Amazon Web Services.

DAG Directed Acyclic Graph.

DApp Decentralized Application.

EC2 Elastic Compute Cloud.

EOA Externally Owned Account.

EVM Ethereum Virtual Machine.

ICO Initial Coin Offering.

IoT Internet of Things.

IT Information Technology.

JSON JavaScript Object Notation.

OS Operating System.

P2P Peer-to-Peer.

PoA Proof-of-Authority.

PoS Proof-of-Stake.

PoW Proof-of-Work.

RPC Remote Procedure Call.

TPS Transactions Per Second.

Bibliography

[1] Bitcoin Cash Specification. https://github.com/bitcoincashorg/bitcoincash.org/blob/master/spec/block.md. Accessed: 16.12.2018.

[2] Ethereum 2.0 Specifications. https://github.com/ethereum/eth2.0-specs. Accessed: 20.02.2019.

[3] Ethereum Github Repository. https://github.com/ethereum. Accessed: 19.11.2018.

[4] Ethereum Github Repository Wiki: Ethash. https://github.com/ethereum/wiki/wiki/Ethash. Accessed: 03.12.2018.

[5] Ethereum Improvement Proposal Repository: Clique Proof-of-Authority Consensus Protocol. https://github.com/ethereum/EIPs/issues/225. Accessed: 14.02.2019.

[6] Ethereum Project Homepage. https://www.ethereum.org/. Accessed: 14.10.2018.

[7] Ethereum Proof-of-Stake FAQ. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs. Accessed: 20.02.2019.

[8] Ethereum Raiden Network. https://raiden.network/. Accessed: 20.02.2019.

[9] Ethereum Sharding FAQ. https://github.com/ethereum/wiki/wiki/Sharding-FAQs. Accessed: 20.02.2019.

[10] Ethernodes.org. https://www.ethernodes.org/network/1. Accessed: 19.11.2018.

[11] Etherscan.io: Ethereum average gaslimit chart. https://etherscan.io/chart/gasLimit. Accessed: 13.12.2018.

[12] Etherscan.io: Ethereum transaction history. https://etherscan.io/chart/tx. Accessed: 14.10.2018.

[13] Go-Ethereum Github Repository. https://github.com/ethereum/go-ethereum. Accessed: 19.11.2018.

[14] Blockchain.com. Bitcoin Blockchain Charts. https://www.blockchain.com/en/charts. Accessed: 16.12.2018.

[15] V. Buterin. A Next-Generation Smart Contract and Decentralized Application Platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2014. Accessed: 14.10.2018.

[16] V. Buterin. Toward a 12-second Block Time. https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/, July 11 2014. Accessed: 19.11.2018.

[17] V. Buterin. On Public and Private Blockchains. https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/, August 6 2015. Accessed: 19.01.2019.

[18] V. Buterin. Ethereum: Platform Review - Opportunities and Challenges for Private and Consortium Blockchains. Technical report, 2016.

[19] V. Buterin. Ethereum scalability research and development subsidy programs. https://blog.ethereum.org/2018/01/02/ethereum-scalability-research-development-subsidy-programs/, 2018. Accessed: 14.10.2018.

[20] V. Buterin and V. Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.

[21] A. Chauhan, O. P. Malviya, M. Verma, and T. S. Mor. Blockchain and scalability. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 122–128, July 2018.

[22] S. Chen, J. Zhang, R. Shi, J. Yan, and Q. Ke. A comparative testing on performance of blockchain and relational database: Foundation for applying smart technology into current business systems. In N. Streitz and S. Konomi, editors, *Distributed, Ambient and Pervasive Interactions: Understanding Humans*, pages 21–34, Cham, 2018. Springer International Publishing.

[23] K. Christidis and M. Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.

[24] E. community. Ethereum Homestead Documentation. http://ethdocs.org/en/latest/index.html. Accessed: 12.02.2019.

[25] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains. In J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, editors, *Financial Cryptography and Data Security*, pages 106–125, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[26] M. Dameron. Beigepaper: An Ethereum Technical Specification. https://github.com/chronaeon/beigepaper/blob/master/beigepaper.pdf, 2018. Accessed: 29.11.2018.

[27] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sep. 2013.

[28] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang. Untangling blockchain: A data processing view of blockchain systems. *CoRR*, abs/1708.05665, 2017.

[29] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1085–1100, New York, NY, USA, 2017. ACM.

[30] Ethereum. web3j - Ethereum Java API Documentation. https://web3j.readthedocs.io/en/stable/transactions.html. Accessed: 21.12.2018.

[31] Ethereum. web3.js - Ethereum JavaScript API Documentation. https://web3js.readthedocs.io/en/1.0/. Accessed: 21.12.2018.

[32] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse. Bitcoin-ng: A scalable blockchain protocol. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 45–59, Berkeley, CA, USA, 2016. USENIX Association.

[33] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 3–16, New York, NY, USA, 2016. ACM.

[34] Hyperledger. Hyperledger Caliper Github Repository. https://github.com/hyperledger/caliper. Accessed: 14.10.2018.

[35] Hyperledger. Hyperledger Caliper Website. https://www.hyperledger.org/projects/caliper. Accessed: 14.10.2018.

[36] V. Inc. Visa Inc. at a Glance. https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf, 2015. Accessed: 14.10.2018.

[37] J. Kan, S. Chen, and X. Huang. Improve Blockchain Performance using Graph Data Structure and Parallel Mining. *ArXiv e-prints*, Aug. 2018.

[38] S. King and S. Nadal. PPCoin : Peer-to-Peer Crypto-Currency with Proof-of-Stake. 2012.

[39] W. Li, A. Sforzin, S. Fedorov, and G. O. Karame. Towards scalable and private industrial blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, BCC '17, pages 9–14, New York, NY, USA, 2017. ACM.

[40] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008. Accessed: 01.06.2018.

[41] Q. Nasir, I. A. Qasse, M. Abu Talib, and A. B. Nassif. Performance Analysis of Hyperledger Fabric Platforms. *Security and Communication Networks*, pages 1–14, sep 2018.

[42] G. T. Nguyen and K. Kim. A survey about consensus algorithms used in Blockchain. *Journal of Information Processing Systems*, 14(1):101–128, 2018.

[43] M. Niranjanamurthy, B. N. Nithya, and S. Jagannatha. Analysis of Blockchain technology: pros, cons and SWOT. *Cluster Computing*, 5(2):1–15, 2018.

[44] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong. Performance analysis of private blockchain platforms in varying workloads. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6, July 2017.

[45] J. Poon and V. Buterin. Plasma : Scalable Autonomous Smart Contracts Scalable Multi-Party Computation. Accessed: 14.10.2018, 2017.

[46] S. Rouhani and R. Deters. Performance analysis of ethereum transactions in private blockchain. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 70–74, Nov 2017.

[47] J. Spasovski and P. Eklund. Proof of stake blockchain: Performance and scalability for groupware communications. In *Proceedings of the 9th International Conference on Management of Digital EcoSystems*, MEDES '17, pages 251–258, New York, NY, USA, 2017. ACM.

[48] Stackoverflow. Stackoverflow Developer Survey Results 2018. https://insights.stackoverflow.com/survey/2018#technology, 2018. Accessed: 19.11.2018.

[49] K. Suankaewmanee, D. T. Hoang, D. Niyato, S. Sawadsitang, P. Wang, and Z. Han. Performance analysis and application of mobile blockchain. In *2018 International Conference on Computing, Networking and Communications (ICNC)*, pages 642–646, March 2018.

[50] N. Syeed. Is Blockchain Technology the Future of Voting? https://www.bloomberg.com/news/articles/2018-08-10/is-blockchain-technology-the-future-of-voting, 08 2018. Accessed: 05.12.2018.

[51] N. Szabo. Smart Contracts: Building Blocks for Digital Markets. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html, 1996. Accessed: 14.10.2018.

[52] P. Thakkar, S. Nathan, and B. Vishwanathan. Performance benchmarking and optimizing hyperledger fabric blockchain platform. *CoRR*, abs/1805.11390, 2018.

[53] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim. A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access*, pages 1–1, 2019.

[54] G. Wood. Ethereum: a secure decentralised generalised transaction ledger. https://gavwood.com/paper.pdf, 2014. Accessed: 14.10.2018.

[55] W. Xin, T. Zhang, C. Hu, C. Tang, C. Liu, and Z. Chen. On scaling and accelerating decentralized private blockchains. In *2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids)*, pages 267–271, May 2017.

[56] H. Zhang, C. Jin, and H. Cui. A method to predict the performance and storage of executing contract for ethereum consortium-blockchain. In S. Chen, H. Wang, and L.-J. Zhang, editors, *Blockchain – ICBC 2018*, pages 63–74, Cham, 2018. Springer International Publishing.

[57] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu. A detailed and real-time performance monitoring framework for blockchain systems. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '18, pages 134–143, New York, NY, USA, 2018. ACM.