university of groningen

faculty of science and engineering

Proof of Actual Work

Bachelor's thesis

July 26, 2018

Student: Barnabas Busa (s2922673)

Primary supervisor: Vasilios Andrikopoulos

Secondary supervisor: Jorge A. Perez

Contents

Abstract

The ability to execute tasks in a distributed manner is a valuable resource in today's world, and one realized by many different open source and commercial solutions. However, the ability to commercialize distributed work in a domain-agnostic, secure manner, is not. One approach that might offer such a solution involves combining peer-to-peer supercomputing networks with blockchain technology. Existing peer to peer networks are generally unfeasible for executing distributed tasks. This is because the task owners cannot make sure their task is actually being executed, and workers cannot ensure they will be compensated for their work. A peer-to-peer supercomputing network harnessing blockchain technology may solve this problem by providing a transparent mechanism for distributing work while keeping private information encrypted. In this research paper, I will conduct a feasibility study on whether a solution that can ensure general task execution can still be secure and reliable for task owners and workers.

Introduction

We are living in a world where the number of internet users amounts to half of our global population. The Internet creates a gateway for humans and computers to communicate with each other. With a predicted increase in both the global population and internet users, comes an increasing demand for computational tasks. Hence there will be a greater need for computers to solve different computational problems such as video rendering or storage management. There are two main ways to solve such tasks, either with a centralized system or a decentralized system.

In this thesis, the focus will be on how a decentralized computer system could solve different types of computational obstacles. A decentralized way of solving tasks offers many advantages compared to centralized execution. The execution of large, computationally heavy applications can consume lots of time and resources for a task owner, if he/she runs it on their own machine. A task owner might like to run a task a limited number of times, in which case purchasing a new computer, or renting computational time at a supercomputer would not be economically efficient. Task owners could be researchers running simulations, media companies running a rendering task, or small companies expecting increased traffic on their website (such as selling tickets for an event).

This project aims to find a solution for a system that keeps track of all tasks, task owners, and workers on the network, while ensuring that a given task will be executed and rewarded. To achieve the desired solution, a version of blockchain technology will be used.

The structure of this thesis will cover related work, define limits of the problem and to find a conceptual solution to such problems. This will be followed by a feasibility study where the two main types of blockchains will be considered. Overall, the goal is to show that any computer could potentially become part of a supercomputer system. Each of the nodes of such a system could solve real world problems, and get compensated for task execution. The system that could potentially be implemented will be referred to as Decentralized General Computer System (DGCS).

Preliminary and Related Work

This chapter discusses relevant background information and related work to develop such a DGCS. In Section 2.1, basic background information that is needed to fully understand the mechanism of this project will be included. In Section 2.5, related work relevant to this project will be discussed. The projects mentioned in Section 2.2 have been selected for the reason that they are in some capacity trying to solve a similar or same problem as that proposed by this project. Some examples will reference Bitcoin[7] due to the fact that this is the most known application of blockchain as of today.

2.1 Preliminary work

2.1.1 Centralized versus Decentralized

Businesses and corporations use a centralized (see Figure 2.1) computer topology, because these systems allow fast transaction throughput and ensures privacy. One of its downsides is that it requires trust from all parties. A decentralized (see Figure 2.2) solution may be attractive to parties that require a system where there is a lack of trust. Another advantage is that it provides multiple levels of failure, meaning that there is no central point of failure for such a system. However, a decentralized system will lack of speed compared to its centralized variant.

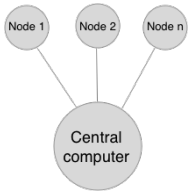


Figure 2.1: A centralized system

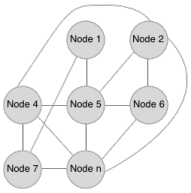


Figure 2.2: A decentralized system

Computers have been improved drastically regarding their processing speeds. Nowadays their performance exceeds the requirements for most smaller tasks. While some computers might run under load most of the time, most computers do not run at their maximum capacity. Idle CPU cycles could be considered as

wasted energy and time. These wasted cycles could be utilized for solving other people's tasks with a decentralized computer system.

High performance computer systems are much more expensive to build and maintain. Therefore accessing them can be difficult process for small business owners or individuals.

2.1.2 Basics of Blockchain

Although many definitions exist for the concept of the blockchain, the most fitting description is that of Dr. Julian Hosp, *"A blockchain is a decentralized community's complete and unchangeable transaction history that everyone who is part of the community agrees on. This ledger automatically gets updated in regular time frames, is accepted by the community as a fact, and gets stored on every participant's computer. This way no central party has to govern the community, since no one can double-spend. That would create an immediate conflict in every participant's transaction history."* (Cryptocurrencies, 2017, p. 37) [22]

Simply put, a blockchain is a linked list that holds information and each time a new so called block is created an extra element is added to an existing list. By creating multiple copies of this list, multiple users can ensure that they have data redundancy. Due to a consensus algorithm, the users can ensure that the integrity of the list is consistent over the whole network. In most cases a block has the following properties:

- An index: used for identifying a certain block in the blockchain
- The previous block's hash value: ensuring that the integrity of the previous block cannot be modified
- The time stamp of the block: this is the time when a block was created
- Data: this can be any sort of data, in case of bitcoin, data consists of transactions
- Hash of the current block: the current block's hash value is calculated by the combination of all parts of the block hashed with a certain hashing algorithm such as SHA256[57].

A blockchain is an immutable ledger for recording transactions. Such ledger is maintained by untrusting peers within a single network. All nodes of this network store a copy of the whole ledger. As this ledger grows, more transactions are placed in them. The peers follow the same consensus algorithm to validate transactions and group them into a block. Then this block will be hashed. The first and largest blockchain is Bitcoin [7].

Up until now, one of the biggest problems was that there was no way of tracking who did what task and how the volunteers would get compensated for that. However, in 2009 Satoshi Nakamoto introduced a revolutionary new distributed database system [7].

This new distributed data base is a so called public blockchain. It enables anyone to see all transactions that have been done and will happen.

2.1.3 Smart Contract

Smart contract is a protocol proposed by Nick Szabo in 1994. Some blockchain technologies support these while others don't [1]. Smart contracts are providing a platform for users to exchange money, property, shares, or anything of value. Thanks to smart contracts, the exchange of such things are done in a conflict-free and transparent manner, removing the need for third party. A smart contract is a protocol intended to verify and enforce a digital contract. Smart contracts allow creditable and irreversible transactions without interference of third parties.

Using smart contracts can ensure that each executed task will not go unrewarded. Each task should have a smart contract regarding to that task and each task should also have a pre-defined value. This value should be defined by the task owner in advance.

2.1.4 Consensus protocol

The consensus protocol behind different blockchain technologies are a way to reach an agreement in a group of nodes. The consensus makes sure that the agreement is reached that would benefit the entire group as a whole. The method that is used for consensus decision making is called "consensus machanism" [32]. This protocol is created in order to ensure that the integrity of a blockchain stays the same over all the different nodes. This enables users to trust a set of nodes, but trust for each individual node independently is not required.

2.1.4.1 Proof of Work (PoW)

The first and most common example of a consensus algorithm is a proof of work algorithm. Most large cryptocurrencies use the proof of work algorithm, because it has been successfully tested over the past years. The first example of using proof of work together with blockchain was in 2009. This is when the Bitcoin network went online [7]. During this time the Bitcoin network PoW provided protection against 51% attacks that could enable malicious participants to cheat the system. PoW works in a way that computers need to solve a cryptographic puzzle. Usually the more computers there are to solve the puzzle, the harder it will be to solve it. Difficulty represents how difficult it is to "solve" the puzzle. In Bitcoin's case, the cryptographic puzzle is a hashing algorithm (SHA-256). Here the computers need to find a nonce (number once), which combined with the block's content (transactions), will give a hashed value with x number of leading zeroes. The difficulty is periodically adjusted to keep the block time around the target time. For solving such tasks the so called miners will get rewarded from the network.

51% attack

51% attack refers to an attack on a blockchain which uses PoW consensus algorithm. It occurs when a group of miners controlling more than 50% of the network's net hashrate deliberately create a fork of the main chain, which allows them to reverse transactions (for cryptocurrency coins). By reversing transactions, the attackers who took over control over the network at this point are able to double-spend coins.

2.1.4.2 Proof of Stake (PoS)

Proof of stake is an algorithm where money creates voting power[8]. If you can stake more of your money, you will be able to be more creditable to the network. You as a stake holder can ensure that the transactions on the network are confirmed, and are indeed legit. In case you try to be a fraudulent node, your stake can be lost. This ensures that nodes stay honest.

2.1.4.3 Other consensus algorithms

Another example of a consensus algorithm would be proof of importance (PoI) [53] and proof of burn (PoB) [59]. PoI is a consensus algorithm where the most important members of a network decide what transaction did happen and which did not. There are multiple ways of gaining importance in a network, this can be done by how long you been part of the network, number of miners trusting you by opting in to receive information, or by the number of coins you have staked in a certain network. PoB is an algorithm where you can create value by "burning" away some other cryptocurrency coins. Burning means that a coin gets sent to an address which has no owner. Hence creating a greater scarcity for that certain coin, thus gaining value on another chain.

2.2 Related Work

2.2.1 The History of Grid Computing

The term grid computing originated in the early 1990s. The idea behind grid computing was that computational power should be accessible as easily as an electric power grid. This power grid metaphor for accessing large computer networks became accepted when Ian Foster and Carl Kesselman published their seminar work [3].

In 2014 Muhammad Nouman Durrani and Jawwad A.Shamsi proposed a solution to this problem by adding volunteers' computers to the network. However, the progress of simultaneous projects has started to decline due to lack of volunteers. People still wanted to run high computational programs, but there was just not enough volunteers who would lend their computer for this task for free. There was no economical motivation behind being a volunteer. The only reason why someone would participate in a volunteer network was ethical desire to help a certain research. Because of high computational needs and low participation rate of volunteers, attracting more volunteers and using their resources more efficiently have become extremely important [10].

2.2.2 Current State of Grid Computing

There are still plenty of currently running projects that are still trying to solve the same issue what Ian Foster and Carl Kesselman tried to solve back in 1999. However it is clear, that without any financial motivation, people will not likely put their computer on the "grid", most likely because heavy computation will consume more electricity at their own home. Therefore there is a need to find a way to compensate those that decide to join to the network and try to keep them dedicated by rewarding them.

2.2.3 Currently running projects

2.2.3.1 Project Golem

Project Golem [17] is a global supercomputer that anyone can get access to. Anyone can participate in their system, from small PC's to large data centers. Golem will be able to process a wide variety of tasks, but currently the only task that can be run is computer-generated imagery (CGI) rendering using Blender[41] and LuxRenderer[49] scenes. The results for this rendering task should arrive faster than if it would be executed on the requestor (task owner)'s computer. Also the task owner can define the price that will need to be accepted by the supplier of computing power (farmers). Due to this feature, the task owner will know in advance, how much he/she will need to pay for a certain task, and also the farmers will know how much they will get paid for executing that certain task. This will create a healthy competition in the marketplace created by Golem. Another feature is that a task owner could join in as a farmer with its own computer to offset the cost, when they are not working on their own projects.

The future expansion options include machine learning and a viable alternative to existing cloud providers. Their application will be running based on the Ethereum blockchain, that provides a wide variety of applications such as the use of smart contracts (see more about smart contracts at subsection 2.1.3) [17].

2.2.3.2 Berkeley Open Infrastructure for Network Computing

Berkeley Open Infrastructure for Network Computing (BOINC) is an open-source middleware system. *"It became generalized as a platform for other distributed applications in areas as diverse as mathematics, linguistics, medicine, molecular biology, climatology, environmental science, and astrophysics, among others."*[15] This project is running on a huge scale, it brings together about 157,224 volunteers active participants and 828,724 active computers worldwide processing creating a 24-hour average: 20.581 PetaFLOPS as of 14^{th}th June 2018 [34].

2.2.3.3 IOTA Qubic

A very recent publication of IOTA Qubic project involves a transaction free machine to machine option. It states that using a protocol called Qubic, "it provides a general-purpose cloud-, or fog-based, permissionless, multiprocessing capabilities on the Tangle."[43] Qubic project's goal is to offer a platform for world-wide unused computing capacity for all computational needs, while creating an even more secure tangle [42]. They envision an IOTA-based world supercomputer.

2.2.3.4 Decentralized Data Storage Solution Storj and Sia

With the drastic growth in the cryptocurrency market, and blockchain technology deployment, there were a couple of companies who tried to solve the problem for decentralized storage. One of them is called Storj [19] and the

other is called Sia [12]. They both argue against storing everything at a central data center (such as amazon web services [28]).

Storj and Sia are two quite familiar projects that are still ongoing and gain lot of attraction recently. These two projects involve a distributed data storage solution for individuals and for corporations. They ensure data integrity by submitting challenges to so called farmers, and they offer data redundancy by ensuring a copy of your data exists on multiple nodes of the network.

Problem Definition

3.1 Cast of Characters

Following information security tradition, Alice and Bob represent the good guys. In this case Alice will play the task owner, while Bob will be the task farmer (or the computer that executes Alice's task). In the current security description, the bad guy will be played by Trudy, who is trying to attack the system in some way. In this case, Trudy could be a task owner as well as a task farmer. It will always be specified what part Trudy plays in the specific examples described. Note that Alice, Bob, and Trudy need not be humans. For example, one scenario could be Alice plays a server, while Trudy plays a personal computer. Also Bob could represent more than a singular system/human.

3.1.1 Motivation for creating Decentralized General Computer System (DGCS)

The main aspect of this thesis is to create a decentralized computer system to distribute any task using blockchain. With the growth in interest in blockchain, we can clearly see a corresponding increase in the number to decentralized projects, compared of the earlier years [30]. Such a growth can lead to an increase in the amount of developers in the field, that switch to decentralized project development. This thesis explores possible options to create a decentralized general computer for both small and large user bases. Due to the nature of this project, participants of this network cannot be limited to their geographic location. Whoever has access to a computer with internet connection should be able to participate in this system. The participants of DGCS are referred to as "task owner", "nodes", and "masternodes". A task owner acts as a user that has tasks that needs to be executed. Nodes refer to workers of the system. Such workers are the ones that solve tasks submitted by the task owners. These nodes work on their own or some other node's low level blockchain. The masternodes are the nodes that keep a copy of all past/present/future tasks that task owners submit. Such masternodes provide the backbone of the entire DGCS. A participant in a DGCS could be a single computer or a set of computers.

There is an anticipated demand for decentralized computer system. One of the main reason why one would choose a decentralized solution over a centralized one is to eliminate centralized competition who control the market using predatory pricing strategies. By decreasing number of competitors, these large corporations can dictate the price for their products [11]. *"...For the past two years, Gartner estimated that Amazon ran more than ten times the computing*

capacity as the next 14 cloud providers combined. ..." [21] Such a centralization may be a great threat to our open and competitive market [25].

The idea of a free market has been appealing to many, but due to technical and financial limitations, none of the decentralized solutions seemed feasible.

3.2 Problem

Based on the thesis proposal, a more refined question is: *"What do we put in the blockchain to prove that a task defined by Computer A, was executed by Computer B in the time that was reported, without any need for trust between the two parties?"* The problem consists of a task owner (Alice), who would like to get *any* of her computational tasks solved by a third party. Computer tasks such as large amount of data storage or video rendering of large files can require large and expensive computers. It is not feasible for most task owners to purchase a brand new machine, in case they only need to run the task a few times.

3.3 Desired properties of solution

The following properties of the solution are desired:

1. The solution must respect the 5 pillars of **security** (see below)
2. The solution must respect the **privacy** of the user
3. The solution must be **general** such that *any* task can be executed
4. The solution must be **easy to use** for the end user so that users' with any level of skills can join the community
5. The solution must be **efficient** in order to solve as many problems as possible with as little overhead as possible.

The properties above are the basic requirements that an end user might have in order to be able to complete the task mentioned in the problem statement. These properties will be discussed further in Sections 3.4 - 3.7 as well as touched in Chapter 5.

3.4 Security

Note: Most specific information in this section will be cited from Mark Stamp's book about Information Security: Principles and Practice (2005) [5].

3.4.1 5 Dimensions of Security

When we are talking about information security, we are usually talking about the 5 general dimensions:

1. **Confidentiality**
2. **Integrity**
3. **Availability**
4. **Non-repudiation**
5. **Access control**

It is very important for this application to ensure that all of these 5 security properties hold, to ensure that user's data is kept privately and that the whole system works securely. Even if one of these 5 properties do not hold, there could be a potential attack from a fraudulent party.

3.4.1.1 Confidentiality

Confidentiality tries to prevent an unauthorized party from reading information. We need to ensure, that when Alice gives out a sub-task (See more in Section 5.7) to Bob, he will not have access to the task details. For example, lets assume that the task is the following: Alice would like to store a video file in a distributed manner, however Alice's video is a private video, and obviously, she would not want Bob to be able to watch that video, even though the whole video file might be stored on Bob's computer.

To ensure confidentiality, we will need to make sure that each sub-task will have client side encryption[60], which allows the sub-task to be encrypted before it gets distributed to the network of workers. Note that using client side encryption allows all sort of sub-tasks to stay private, and we will not need to use different ways to encrypting files or web hosting code. Therefore using the same encryption method could allow confidentiality to *all* sub-tasks.

By ensuring confidentiality we can make sure that even if Trudy will get access to a sub-task, she will not be able to see personal information of Alice.

Privacy is a fundamental human right recognized in the UN Declaration of Human Rights. Hence data security and privacy of the users is utmost importance [56].

3.4.1.2 Integrity

Integrity tries to prevent, or at least detect, unauthorized changes to data. We need to ensure, that when Alice gives out a sub-task to Trudy, she will not be able to modify that sub-task by any means. For example, lets assume again that the sub-task is to store Alice's private video. If Trudy could modify the video to make it reduced in size, while still getting paid for storing the whole video, then Alice should be notified that the content has been tampered with.

To ensure about integrity of the file, we will need to make sure that Alice will have a way of checking whether the sub-task has been tampered with or not. To check this property, Alice could generate a Merkle proof [55] using a Merkle tree.

Sharding [48] would be a much more complicated exercise when it comes to rendering task or to web hosting. The integrity check for web hosting would require the user to send out challenges checking for keywords in a website. However, when it comes to rendering, ensuring data integrity can only be enhanced by the consensus of the network.

3.4.1.3 Availability

Availability tries to ensure that a network working on a particular sub-task is always online. Centralized computer systems could have a potential down time at times when power outage occurs, or if the servers are overwhelmed by a denial of service [6] (DoS) attack. Availability is an issue for both Alice and Bob. If Alice goes offline before the all sub-tasks could be distributed, then the sub-tasks cannot be executed. If Bob goes offline, then there will be no-one to distribute the sub-tasks to. By creating a peer-to-peer network, we are removing a central point of attack, and also we can ensure (by financial motivation) that the nodes will stay online. An example could be, when Alice distributes the sub-tasks to Bob, she would need to stay online as long as Bob has a copy of each sub-task, which then could be distributed to other nodes. This methodology was first used by the BitTorrent[31] protocol.

Using this methodology, we can ensure for maximum uptime for the network, and also ensure of maximum availability for the sub-tasks.

3.4.1.4 Non-repudiation

Non-repudiation ensures one's intention to fulfill their promise to a certain contract. It also implies that the other party of that transaction cannot deny having received that transaction, nor the other party deny having sent this transaction. We will need to make sure that Alice will only pay Bob **if** Bob actually executes the sub-task. Also it is very important to ensure that Bob **will** get paid for his effort. To solve this problem, smart contractscan be used.

There needs to be a reward system, where a task owner can offer certain amount of money. This reward would go to the workers who complete a task. The different conditions for different tasks could would need to be placed into a smart contract. This contract would enforce these rules. There are different ways of handling payments of workers and masternodes. Currently there are 2 main pool types that PoW pools operate. Workers get rewarded for their computation in cryptocurrency using two types of mining reward structures:

1. Pay Per Share (PPS)
2. Pay Per Last N Share (PPLNS)

One of these two reward systems could be introduced for problem solving also where creating a block would be equivalent of submitting a solution for a sub-task. Solving larger problems could be split between a group of people who create a pool. Submission of shares could be possible for solving parts of a subTask.

PPS

Pay Per Share pays a miner on the average number of shares that you contributed to your pool in finding a block. PPS eliminates luck factor. This makes it suitable for people who prefer standard payout system.

PPLNS

Pay Per Last N Shares calculates payments based on the number of shares submitted between finding two blocks. PPLNS involves the luck factor, hence there will be less consistency in the payouts after the submitted shares.

Payout to the workers of the network

Payouts and reward system will need to be thought of more thoroughly after implementing the marketplace functionality in DGCS. The payout will refer to the workers and masternodes who are placing their computing power for the purpose of DGCS.

3.4.1.5 Access control

In the world of computers, gaining **trust** from others is very difficult due to the lack of human contact. Restricting access to certain users is a fundamental need of this project. Access control is a way of restricting sub-task execution rights to a subset of people who are *authorized* to possess them. By restricting user access, we can ensure that *all* farmers of the network are trustworthy. To enable a trustworthy circle of workers while also ensuring we have as many workers as possible, we will need to allow anyone to join, and later restrict certain (fraudulent) users who are trying to cheat the system.

3.5 Trustless consensus

In the previous chapter I made an introduction to Proof of Work (PoW) and Proof of Stake (PoS). For this project a selection of a trustless consensus is required. This will be further discussed in Section 4.2.

3.6 Generality

One of the main parts where the project splits from Golem's (see more in Chapter Sub-Section 2.2.3.1) is that Alice should be able to run *any* sort of task (Project Golem focuses on a few particular tasks only). To ensure that this is possible, she will need to be able to modify the properties of the blockchain(s) that the project will be using. As was mentioned earlier, the properties of the blockchain can be modified. By modifying the block's size (how much data can go into a single block) and the block time (how often each block will be *created*[1], Alice could create a suitable blockchain for *any* type of task.

[1] note, that these blocks are generated (and not created out of thin air) by verification of the sub result that other farmers create

3.7 Efficiency

It is necessary that this project will only run when the owner of the computer does not use his/her computer. When a computer is idle, a software should start up, and run the tasks automatically, ensuring that the computer's idle power is not wasted.

When creating a distributed network, we know that connecting each peer to every other peer is not a scalable option. However, the workers of the network needs to be able to reach any other node within a few hops. An example of communication protocol in a peer to peer system is Kademlia [4].

This project aims for a solution where communication overhead is not an issue. Task execution should take place on a local machines, and only the necessary information (such as sub solutions and solutions) should be sent back and forth between the nodes.

Proposal of Solution

4.1 Problem Analysis

The essence of the problem involves the process of determining whether a task is being executed in a distributed network of nodes. This problem can easily be solved when the task execution happens on a centralized system. When you send your data to Amazon or upload it to Dropbox you placing trust that those companies will hold on to your data, and handle it confidentially. The involvement of a distributed network would require the task owner to place trust into all nodes of the distributed network. This is an unrealistic request to be made for the task owner.

Most Tasks can be split up into subTasks. Such tasks can be distributed on this distributed network, and task solutions should be proposed by the nodes of this network. Such proposals should be compared and triaged to determine which ones are legitimate and which ones are fraudulent. Once all subTasks are completed, the true solutions should be combined and returned to the task creator.

Solving the problem posed a lot of questions:

- How is it possible to split up tasks into subTasks?
- How is it possible to ensure confidentiality while keeping integrity of task?
- How can one create a network of nodes where individual need for trust is not required?
- Why would anyone join this network as a worker?

However, these questions could only come after we tackle the initial problem, determining task execution in real time.

A proposed solution to each of the questions above will be addressed in Section 4.2 -Section 4.6

4.2 Solution Design

The desired properties of the solution listed in Section 3.3 were security, privacy, generality, ease of use, and efficiency.

There needs to be a controlling mechanism in place in order to create a generally safe environment for most users. To identify a user, their computer's MAC/IP address could be used. That way, if a person performs fraud on within the network once, his/her MAC/IP address could be put on a blacklist, and they will not be able to join from that computer again. (Unless a user decides to spoof their addresses)

For this project, the most straightforward solution should be used, which is proof of work consensus. Each time a worker submits a solution to a subTask the solution should be compared to other workers completing the same subTask. Once the majority of all nodes get the same solution, the solution can be placed into the solution queue. This is somewhat similar how PoW works for Bitcoin, but in this case each worker would need to compute the solution for each subTask.

These 5 properties are selected due to the fact that they are some of the most important properties that reflect the goals and expectations of DGCS.

The provided solution will address the above mentioned properties in the feasibility study part (Section 5.4.3).

The design section is divided up to 4 parts; the conceptual model, a use case diagram, an activity diagram, and a set of sequence diagrams.

4.2.1 Conceptual model

In order to get a better idea of the overall system, a conceptual model was created. The conceptual model is a representation of a general task execution.

Figure 4.1 shows how Alice generates a new Task by selecting a file. This Task then is sharded into subTasks which are sent to the masternodes. The masternodes then process the subTasks, and place it into a queue of Tasks. Once Alice's Task is in the front of the queue, the Task will be popped from the queue, and sent off to the workers, where the first worker will need to create a new blockchain for that specific Task. This is necessary in order to ensure that each part of the task will be executed in an honest way. After workers have worked on a subTask, the solution of that subTask will be compared with other workers' solution. If a solution appears on 50% or more on different workers' computer then due to consensus it will be placed on the subSolution queue. They will then move on to the next subTask until they finish the task. Once a task is finished, they will return the whole blockchain (now consisting of an empty taskQueue and a completed task) to the masternodes, who will report to Alice that her task is completed. When the solution is returned to Alice, the funds from the escrow gets released to the masternodes and to the workers. (Note that Alice could have checked during execution of her task, whether the subSolutions are correct or not, by running a challenge by the workers) The figure below represents a high level representation of how a task will be executed in this decentralized network.

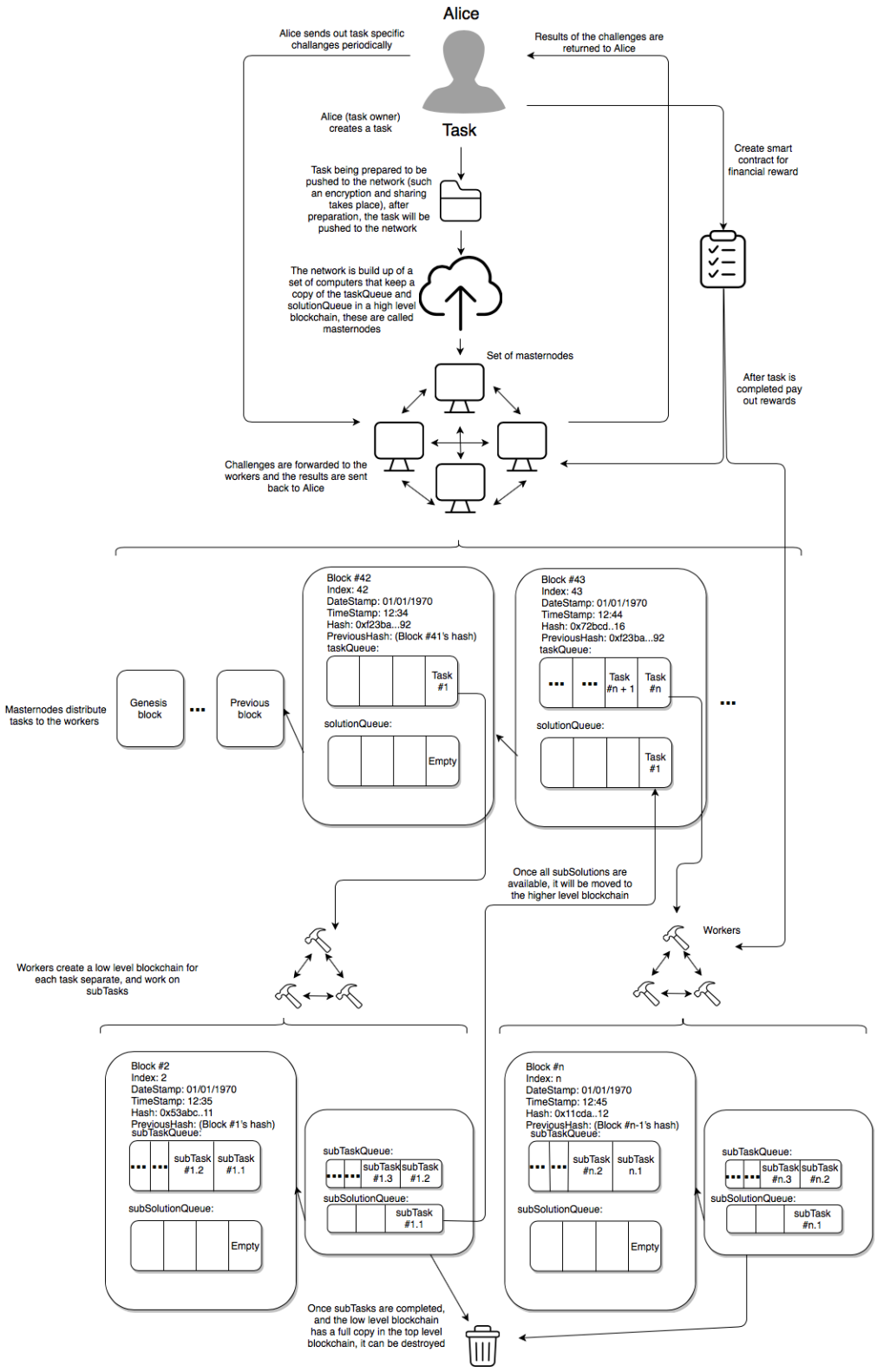


Figure 4.3: High level structure diagram

4.2.2 Use case

In the following diagram you can see the simplest representation of a user's interaction with the system, which shows the relationship between the different users. This use case diagram identifies three different actors (task owner, masternode, and worker) of the system and shows what is going on in the system at it's highest level. This diagram shows how a task owner sending a task to the masternode while sending rewards for it to an escrow. This operation has to happen at the same time, so the task owner could not get out of paying for the task. Masternodes then distribute the task to workers, and once the workers complete the task they return the solution to the masternode, which later returns the solution to the task owner. After task is completed the masternode and worker will get paid by the escrow.

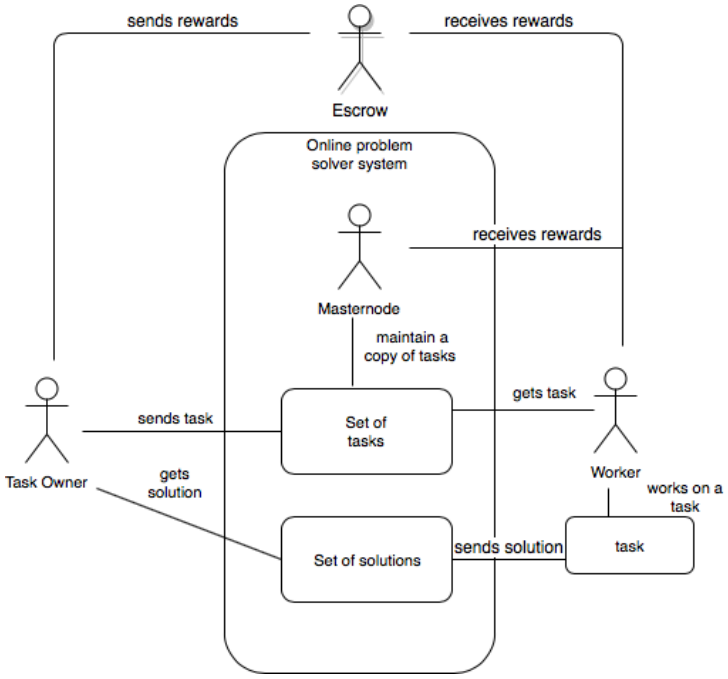
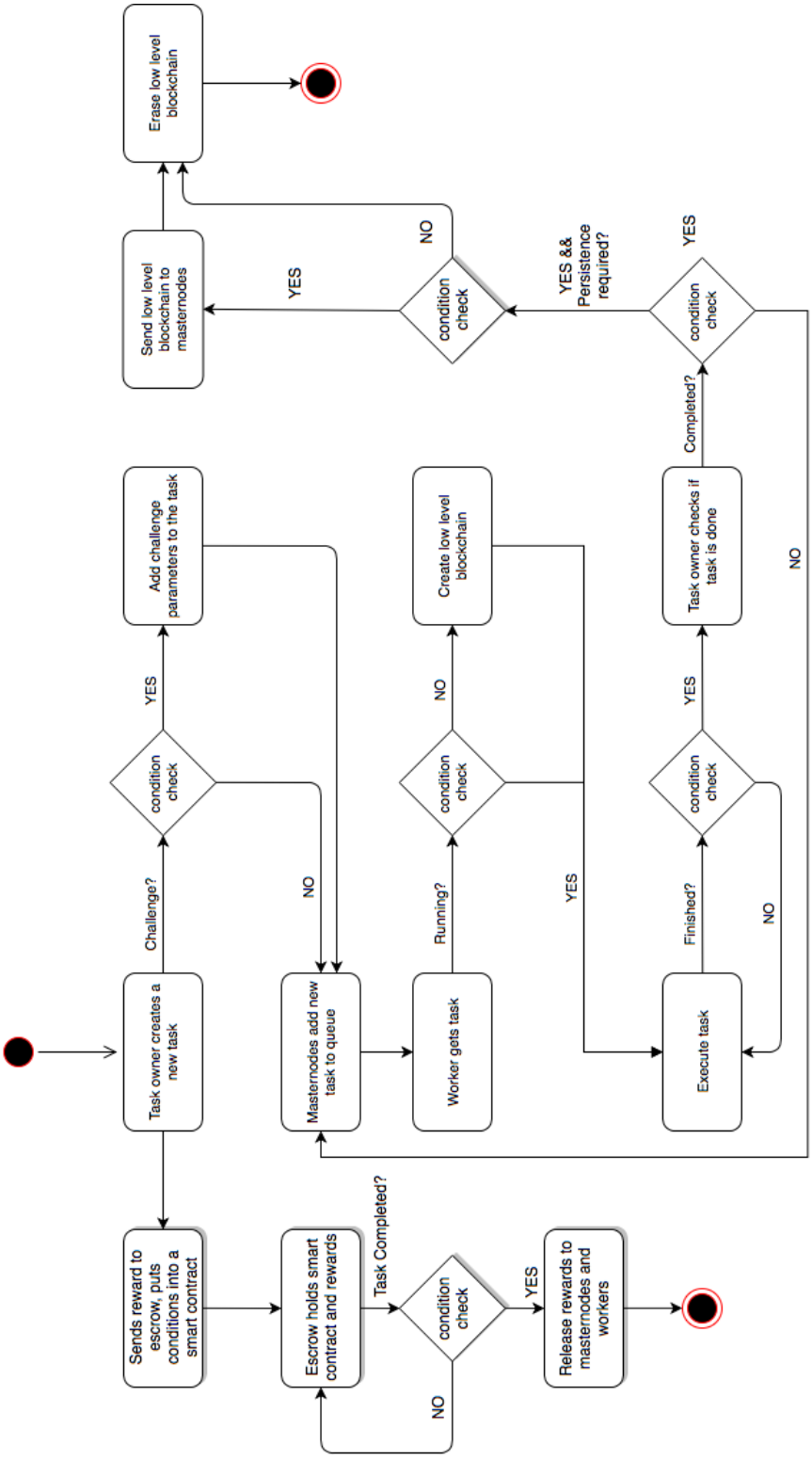


Figure 4.4: Use Case Diagram

4.2.3 Activity Diagram

The following figure represents the dynamic aspects of the system. This figure shows choices that the users will need to take into consideration while running the DGCS.

Figure 4.5: Activity Diagram



4.2.4 Sequence Diagrams

4.2.4.1 Adding a task

Sequence Diagram for adding a task: The following figure represents the events that are involved when adding a task to the network. Here you can see how a file gets selected, then later encrypted. Then the sharding process takes place, followed by the optional challenge that generates the Merkle Proof by building up the Merkle Tree with the hashed value of each leaf. After that the shards get distributed to the network, and the network (masternodes) add the task to the queue. In the end, the masternodes this process will repeat for each shard. After the whole task has been sent to the masternodes, they will need to send a verification of it back to the task owner.

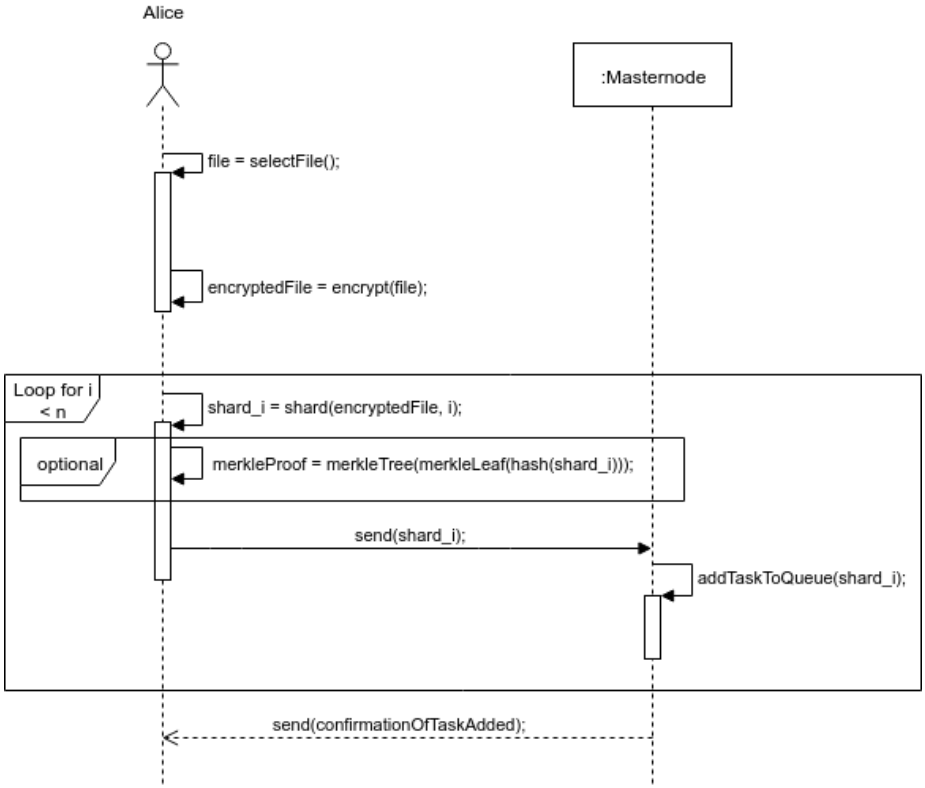


Figure 4.6: Sequence Diagram for adding a task

4.2.4.2 Getting a task

When getting a task, a worker (Bob, this could be plural), requests a new task from the masternodes. A masternode will get the next task from the queue. That task will be then passed back to the worker. Bob at this point will check whether the task is being executed or not; if not he will create a new blockchain for that task, else he will start executing the subTasks. Optionally it will also need to solve the challenges that are coming from the masternode (that gets it from task owner). Once all subTasks are solved, the solutions will be pushed

back to the masternode, and the local copy of that low level blockchain will be erased.

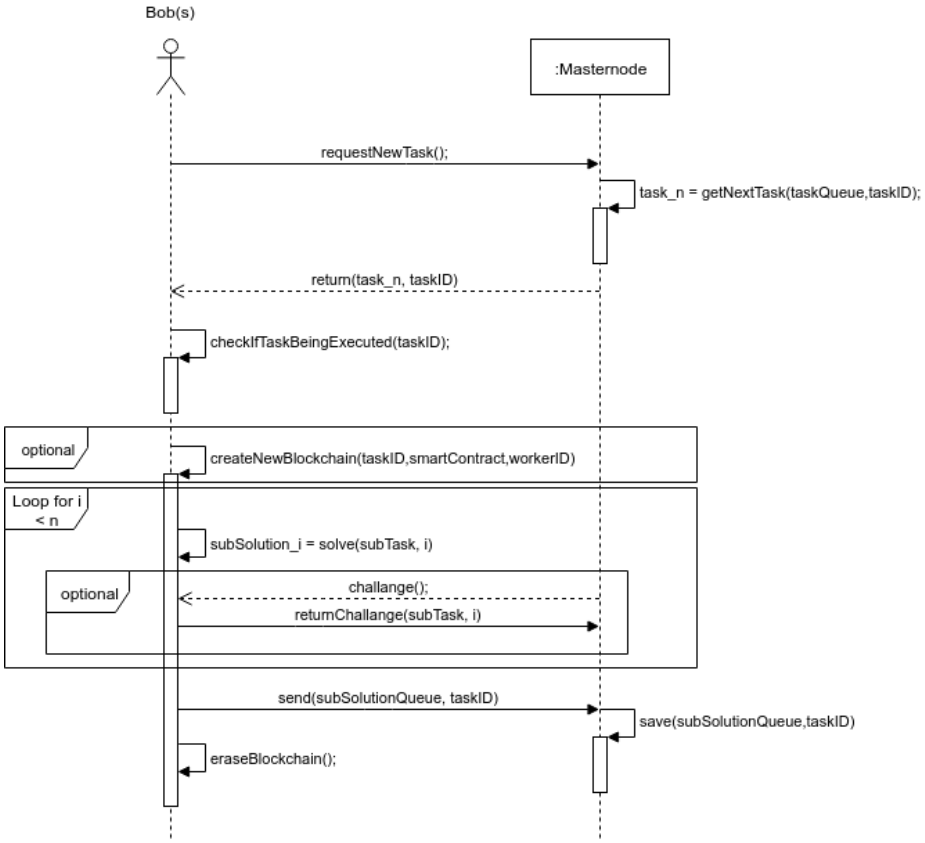


Figure 4.7: Sequence Diagram for getting a task

4.2.4.3 Executing a task

Executing a certain task (Figure 4.7) involves just submitting results and comparing that result with the other results of the network. The comparison is based on a consensus algorithm (specifics can be decided later). Once a result is accepted, the worker should be notified, so he can get on to the next subTask.

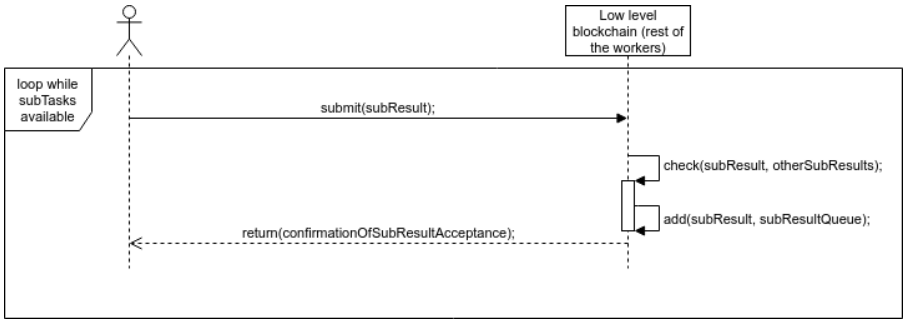


Figure 4.8: Sequence Diagram for executing a certain Task

4.2.4.4 Finalizing a task

Finalizing a certain Task (Figure 4.8) means that the worker notifies the masternodes that the Task is completed. Then they send the solutions to the masternodes. After that the masternodes notify the task owner (Alice). Once she receives the results, the funds are released to the masternodes and to the workers. (funds/rewards will be kept at that escrow until that time, ensuring that the task owner cannot run away with the solutions without paying for it, and vice versa)

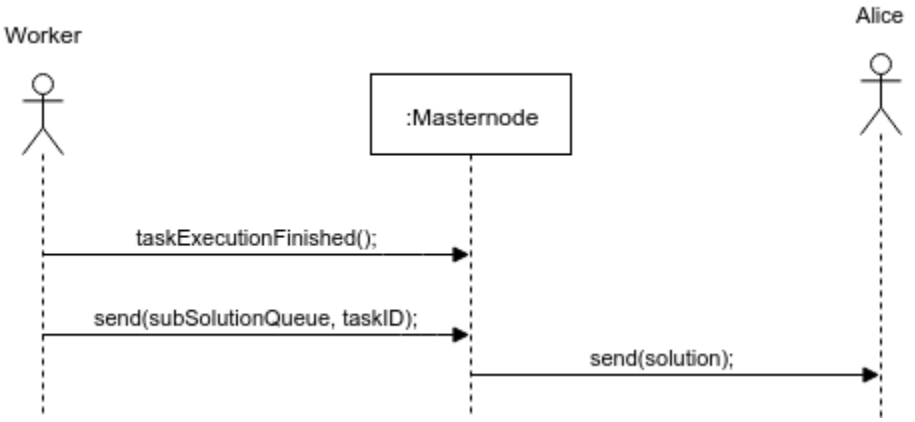


Figure 4.9: Sequence Diagram for finalizing a certain Task

4.3 Task sharding process

Each Task should be divided up into smaller parts. This is to ensure that each subTask can be checked manually or automatically. This is also provides possibilities to distribute the Task much easier. Note that this sharding process is similar to what you might find for project Storj. The sharding process goes as follows:

1. There is one data owner: Alice
2. There are multiple masternodes, forming a network of masternodes.

3. There can be one or more data workers (Bob_1, Bob_2 ... Bob_n)
4. Data is split into shards (negotiable contract parameter) → usually standardized (8-32 MB) → smaller files filled with zeroes
5. Sharding large files (videos) → end user takes advantage of parallel transfer (BitTorrent like)
6. Redundant mirrors of shards are created and sent to the masternodes.
7. Availability is proportional to the number of nodes storing the data.

4.3.1 Sharding process step by step

1. Files are encrypted
2. Encrypted files are split into shards, or multiple files are combined to form a shard
3. Partial audit pre-processing (special way of calculating the Merkle tree to avoid significant computational overhead for the data owner and for the workers) is performed for each shard
4. Shards are transmitted to the network

4.3.2 Partial audit process

1. This extension relies on two additional selectable parameters
2. Data owner stores a set of 3 tuples (s,x,b) → (salt, byte indices within the shard, set of section lengths)
3. To generate pre-leaf i, the data owner prepends s_i to the b_i bytes found at x_i.
4. During the audit process the verifier transmits $(s, x, b)_i$ which can be used by the worker to generate a pre-leaf.
5. The Merkle proof is generated and verified as normal afterwards.

4.4 Challenges

Challenges that Alice submits to the network can differ per Task. You can see an example for a challenge for data storage:

1. Alice stores a set of challenges, the Merkle root, and the depth of the Merkle tree. These challenges can differ per Task.
2. Alice transmits the tree's leafs to Bob(s)
3. **Periodically**, the data owner selects a challenge and transmits it to Bob(s).
4. Bob uses the challenge and the data to generate a pre-leaf.

5. This pre-leaf, together with the Merkle proof, will be sent back to the data owner.

6. Using this information, the data owner can check whether the Bob still has the shards that he is supposed to have.

4.5 Use cases for three specific task types

These three specific Tasks were created, because they represent three vastly different types of Tasks. These three Task types should be feasible to implement in the future. For other Tasks, where latency is crucial, blockchain solution would not be a reasonable choice. An example of a Task with low latency requirements would be hosting a gaming platform.

4.5.1 Data storage

A use case for data storage is rather similar to what Storj implements. **Store a 1GB file for 1 week**

1. Task owner encrypts this file.

2. Task owner divides up the 1GB file into smaller chunks (shards) lets say 10MB each. Creating about 1000 different encrypted shards.

3. Task owner hashes these shards, and stores the Merkle tree and the Merkle Proof.

4. Task owner uploads encrypted chunks to the network with the Merkle Proof.

5. Masternodes receive the 1000 chunks.

6. Masternodes will place 1000 pieces into the queue as new Tasks to be solved (this queue will be placed into each block of the top level blockchain).

7. Masternodes will search for workers who are up for this Tasks.

8. Masternodes send the chunks off to workers who have been selected (more on how to select suitable workers later).

9. Task owner can send challenges to Masternodes, that will forward the challenge to the selected workers.

10. Workers compare their challenges with others and verify it (trying to get 50+% of the whole network).

11. Workers return the challenge and place it into a private blockchain.

12. Task owner can always check the content of that private blockchain and see if the majority of the network is still results positive on her challenges.

4.5.2 Computational task

For sequential Tasks we will need to consider different approaches for each case. The following computationally heavy applications should be studied, and to split up into subTasks if possible. An example of a computational Task would be video rendering with Blender, as discussed by Project Golem. Blender is a 3D visualizing software for simulations and rendering. Blender cycles allows you to subdivide a single Task into parts, so different computers could work on the same Task but on a different part. Diffuse, glossy, transmission etc. are available in "Cycles", along with the ability to split these up and recombine them. The hashed cycle could be placed in the private blockchain, and those results will be compared to the rest of the network's solutions. Another larger block size and less frequent block should hold the rendered cycle. The final Task would be to collect these cycles and combine it to a single footage, which then could be distributed back to the data owner.

There are plenty of other sequential or parallel computational Task out there like protein folding, or machine learning.

Render a 1 minute video file

1. Task owner sends the 3D scene file to the network.
2. Masternodes receive the file.
3. Masternodes specifies ranges of time / timeline segments to be rendered.
4. Masternodes select suitable workers for the subTasks.
5. Masternodes send of each subTasks to multiple suitable workers.
6. Each worker will render their frames.
7. Each worker will calculate a hashed value for the subSolution of all frames.
8. For each Task there will be a new personalized private blockchain created by the masternodes.
9. Each selected worker will compare their subSolution hashed value with other workers.
10. Once 50+% of the workers have the same subSolution hashed value, one of them return the sub solution to the master node.
11. Masternode will place the subSolution into a new queue called subSolution-Queue and this will be placed to the private blockchain.
12. Repeat until all sub solutions have a solution.
13. Once all sub solutions are produced masternodes will combine it into one solution by popping elements from the subSolutionQueue (and the private blockchain can be destroyed).
14. Use all sub solutions to create a solution, then place the solution file into solutionQueue.
15. Return the solution to the Task owner by popping the element from the solutionQueue.

16. If task owner decides to keep a copy of the solution on the network, it will create a new Task for data storage, else task owner can download the solution from the masternodes.

4.5.3 Web hosting

For web hosting it is possible to check the integrity of the Task with an uptime robot. Uptime robot (https://uptimerobot.com/) offers 4 ways to ensure that a website is up:

- HTTP(S) requests: that's perfect for website monitoring. The service regularly sends requests (which are the same as if a visitor is browsing your website) to the URL and decides if it is up or down depending on the HTTP statuses returned from the website. (200-success, 404-not found, etc.)
- Ping: this is good for monitoring a server. Ping (ICMP) requests are sent and up/down status is decided "if responses are received or not". Ping is not a good fit for monitoring websites as a website (its IP) can respond to ping requests while it is down (which means that the site is down but the server hosting the site is up)
- Keyword: checks if a keyword exists or not exists in a web page.
- Port: good for monitoring services like smtp, dns, pop as all these services run from a specific port and Uptime Robot decides their statuses if they respond to the requests or not.

Each worker would be checked every minute, whether they are completing their Task or not. The results of the checks will be placed in the private blockchain. If a worker fails to keep the website running, it will be kicked out of the network, and other workers will take its place. There should be redundant backup for the website. Uptime robot could be hosted on the network, as part of the Task, or using a 3rd party company to execute this Task. Clients that are planning to visit a website will be redirected to the closest host of the website. This can be handled by the DNS servers.

Host a server for 1 hour

1. Task owner sends the web-app repository to masternodes.
2. Masternodes find suitable workers for the Task (mainly based on physical location)
3. Private blockchain will be created.
4. HTTP request, ping, keyword and port checking will take place
5. Task owner can always check the content of that private blockchain and see if the majority of the network is still result positive on his challenges.

4.6 Strength and Weaknesses of the whole system

Strength:

- New workers can join in at any time, using the sub solution, they can pick it up from where the rest of them are.
- Workers can leave at any time without affecting the fact that the completed task is available to its owner and still get paid for the subTasks completed, thanks to smart contracts.

Weaknesses:

- 51% attack on the blockchain could lead to fraudulent answer getting accepted.
- Privacy issues may occur if a single user solves all subTasks, while they get access to "un-encrypted" files, they could potentially have the whole solution to themselves. This however should never happen. Un-encrypted files should not be part of the system.

4.7 Payment

The financial aspect of this project is outside of the scope of this work. However in the future, workers and masternodes should be financially compensated for their work in the future, which should be paid by the Task owner. To ensure that the workers and masternodes will get paid, all Tasks should have a smart contract, and the funds allocated for this Task at an escrow.

Proof of Concept

5.1 Feasibility study for data storage use case

Due to time constraints and limited resources, a complete investigation of the system is not viable. Hence the feasibility study conducted will only focus on storing a directory as a proof of concept. Due to the fact that this is only a feasibility study, an assumption is that existing technologies will work as they intend to without extraneous issues or errors. A feasibility study has 6 parts; The **technical component** (i.e. can it be built?), financial component (i.e. costs and benefits), the legal component (i.e. copyrights and government rules), **operational component** (i.e. will it work?), the schedule component (i.e. how long will it take to build?), and the **resource requirements** (i.e. what and how much resources need?) [9] [33]. However, the financial perspective falls outside of the scope of this project, so it will not be part of this feasibility study. I will take a look at four different technologies and analyze them individually to provide insight into whether or not the proposal will be suitable based on their successes or failures. These four will be:

- Bitcoin
- Ethereum
- Hyperledger Fabric
- IOTA

These four types of cryptocurrencies will be further discussed in the subsection about scalability (subsection 5.2.4). Its not necessary to discuss them before, because parts of their technical characteristics (performance and efficiency, ease of deployment and operational characteristics) are quite similar to each other.

5.2 Technical Feasibility

In this section, the basic question of whether the DGCS proposed in the previous chapters can be built or not will be answered. Due to the generality of this project, the technologies involved will be different for different types of applications. The three distinct applications were defined earlier in Section 4.5. As mentioned above, due to resource and time limitations, the only type that will be considered in this study is the storage solution. Based on the following analysis, the right blockchain platform should be selected or decided whether

a custom blockchain development would be necessary. For the analysis the selected properties below will be studied:

- Performance and efficiency
- Ease of deployment
- Operational characteristics (can it run 7 days a week, 24 hours a day?)
- Scalability

5.2.1 Performance and efficiency

We need to ensure acceptable performance of the network, no matter the file sizes uploaded to them. Performance of the network can be measured by a simple question:

How much data should the system need to be able to handle?

Upload, Download and Distribution

Based on Ookla, the global internet download speed averages out at 45 Mb/s ($\approx$ 5MB/s) with upload speed of 22 Mb/s ($\approx$ 2.5 MB/s) [50]. This means that each node can approximately handle two uploads simultaneously. On an average day there are 2.5 Exabytes of data generated over the world [20]. Most of this data is generated by cloud companies (such as Microsoft, Amazon, Oracle Google), a $75 Billion industry [40]. Many data analysts suggest the digital universe will be 40 times bigger by 2020 [20]. This means, that there is definitely a market for whoever is going to try to sell their free hard drive capacity. In order for a system to be feasible, we shall assume that the **goal** for a decentralized storage solution *should* be to store 0.1% ($\approx$ 25 Petabyte) of all data generated on a daily bases. In each day we have 86400 seconds, which means with an average of 5 MB/s download speed a single node can download 420 GB of that data. To achieve the required 25 PB, the number of nodes in the system would need to reach roughly 62,415. (While it could handle twice this many ($\approx$ 124,830) uploaders).

Based on the information provided above we need to ensure that our blockchain solution can handle 60.000+ nodes at the same time with a potential option of a 120.000+ task owners.

Once the upload from the task owner is finished, the nodes will need to distribute their share of the data to other nodes, thus generating more traffic at a lower level. Therefore the numbers above are purely estimates. To keep our calculations simplistic, we should assume that redundancy is an *extra*, and not *requirement*.

Based on previously mentioned BOINC project, they have got over 800,000 computers. Therefore 60000+ computers in a distributed network is not an unreasonable request. BOINC would be able to handle this amount of data flow every day. Therefore, the feasibility of such system is plausible.

5.2.2 Ease of deployment

Deploying such a system should be rather straight forward. The development and deployment would be hosted as a git repository; perhaps at GitHub for

example. (This would provide make deployment available free of charge to many nodes at the same time.) Deploying a large database (such as blockchain) can be a rather tricky problem. A single blockchain can hold an size of the data type used for pointing to the previous block. For a 64-bit pointer there could be $2^{64} - 1$ blocks. (However, as computers keep improve larger data types could be implemented for such pointers, allowing even larger number of block count). At the beginning, a single block will have a very small size. As time passes by it is going to be harder and more expensive to become a masternode. A masternode would need to keep track of all possible tasks that were ever executed by holding a copy of a top level blockchain. The only realistic approach is to destroy each solution after it has been computed, and only store essential information (such as who did what task, and who paid who for that certain task). Otherwise, the deployment of even a couple day old system would require databases full of hard drives just to record the full history. The removal of old low level blockchain would help to fulfill the **efficiency** property of the system.

5.2.3 Operational characteristics

Storing and uploading large files from and to a network is not computationally heavy. Therefore, we can assume that the machines can run the program as a background process, which would not affect the user's experience even when the computer isn't at idle. *Note this part would have different results in different type of tasks. For example when rendering we should assume that a computer only get utilized when the original owner of the computer is not using it.*

5.2.4 Scalability

One of the main limitations of blockchain technologies is scaling. Scaling means that a decentralized solution might work perfectly fine for small transaction throughput while it would struggle when more users join to the network. Currently most blockchains (such as Bitcoin and Ethereum) can handle around 10-20 transactions/second. This means it is not a ready for world adoption. Hyperledger Fabric and IOTA could handle more transactions / second in theory. Hence, this makes the scalability objectives less realistic.

Bitcoin

Bitcoin's block size is 1MB-2MB, and each block gets mined in roughly 10 minutes [14]. Also, bitcoin does not support smart contracts, so it is not suitable for being the blockchain technology for DGCS. There are ongoing developments going on to improve scaling for this network, but they are currently in beta. Therefore, we won't include it in this feasibility study [16].

Ethereum

Ethereum main blockchain follows Bitcoin's example and is mainly meant for processing transactions. However, Ethereum makes it possible to build your own software on top of its platform. This opens up an infinite number of possibilities for software developers. Instead of block size, Ethereum has gas limit. This gas limit is a cap on both processing and bandwidth due to that the cost of a

transaction if fixed in units of "gas" for each type of instruction. Miners of the network then decide what gas limit they are willing to accept for a transaction. If DGCS were to be implemented on top of an Ethereum blockchain then it would be possible to create blocks of any size. There is no hard cap for the block size. Ethereum tries to solve scalability issues with Plasma [51], Plasma Cash & Sharding. Also developing project Casper [13], which meant to replace part of PoW with PoS consensus, allowing less miners and more stakeholders to validate the blockchain [47]. However, Plasma and Sharding is not yet implemented on the Ethereum blockchain, hence the outcome of this is yet to be decided.

IOTA

IOTA is counting on people to run their own full node at home. This allows them to make free transactions within the network. By creating a new payment each full node will need to verify two additional transactions. However this only scales if we can assume that most users will run a full node at their home. We cannot make such an assumption. IOTA was primary created for Machine to Machine (M2M) transactions, where a chip inside of the machine could fulfill these criteria. As mentioned earlier, IOTA has a project ongoing called Quibic. *"Quibic offers a great incentive for people to run nodes: let others use spare computational power or storage capacity, thus providing a useful service, and get paid to do so. In the future, Qubic will leverage this world-wide unused computation capacity to solve all kinds of computational problems. "[43]* However, this is all a concept at this point, so I can't take it as a solution to the scalability problem. IOTA also does not have support for smart contracts either, making it unsuitable for the financial aspect of the project.

Hyperledger Fabric

From the technical side, Hyperledger Fabric uses a well-known database system. The consensus-mechanism utilizes Apache Kafka[46], which is a well-established stream processing platform. Hyperledger Fabric is a modular platform for a distributed ledger solution. Due to its modularity it can deliver great levels of *confidentiality, resiliency, flexibility and scalability* [24]. Their target market is the enterprise level companies who require hundreds of thousands of transactions every second. With initial testing, IBM was able to achieve 3560 transactions per second [29]. While this might still not be enough for world wide adoption, it is definitely a better start compared to other blockchains.

With (other) current blockchain/tangle technology (such as Ethereum, or IOTA), such a high transaction throughput is not (yet) possible [54][27].

Due to the fact that IBM's main target with this project is enterprises who require a private blockchain solution, their intention was not to create a widely distributed solution for the public.

Using Hyperledger Fabric for this project *could* be a possibility, as it offers a solution to each requirements of the system.

Hyperledger can achieve higher performance by reading data from the state database instead of reading it directly from the ledger. Such a database is based on CouchDB[37] technology. Such components are known from classic distributed database projects, so it also imports the security features into the already known distributed data storage. Scaling such systems would be possible

by distributing task solving to different nodes. By far, Hyperledger Fabric offers the best scaling solution over other similar distributed platforms such as Bitcoin, Ethereum, and IOTA. Smart contracts are also supported by Hyperledger.

5.3 Method of production and Operational Feasibility

Based on the conclusion of Subsection 5.4.4, the only two methods that are feasible to provide a platform for this work are Ethereum and Hyperledger Fabric. Bitcoin and IOTA need to be excluded, due to the fact that neither of them have support for smart contracts. However, neither Hyperledger Fabric nor Ethereum provide an all in one solution to the problem, their platform allows modifications to private solutions, allowing users to create a software for their own purpose.

Self made blockchains

One might think of creating a blockchain from scratch, that would allow to implement only required properties. It would be the ideal solution, but this is not a feasible approach. This would require too many resources and too much time to make a reasonable solution as opposed to retrofitting Ethereum or a Hyperledger. This cuts the count down to two options.

	Hyperledger Fabric	**Ethereum**
Data storage	Data is distributed to the members of the network, and decentralization is limited to such members	Ethereum provides all the required functions for decentralized applications (dapps)
Security & Privacy	You need to trust the owner of the blockchain	Ethereum offers less privacy options then Hyperledger. Evan.network[39] and Quorum[45] are possibilities.
Smart Contracts	Smart contracts are present and they are called chaincode. Offers a very fast but hard to handle all in one docker solution.	Smart contracts are core functionality of Ethereum. They are distributed and operated by all nodes. They can be run on the Ethereum Virtual Machine (EVM)
Immutability	Hyperledger Fabric is a distributed ledger system, and all distributed ledger systems ensure that the data cannot be changed.	Ethereum makes it possible to build a truly decentralized consortium chain.

Table 5.1: Ethereum vs Hyperledger Fabric

Based on this comparison it is fair to say, that both Ethereum as well as Hyperledger Fabric can be a suitable platform for DGCS.

The basic properties of Hyperledger Fabric and Ethereum are close to what DGCS would require. However, there are a few modifications that would need to take place in order to accommodate the desired design previously envisioned in Section 4.2.

When comparing Ethereum public blockchain network with a permissioned blockchain such as Hyperledger Fabric, it is clear to see that as a system gets less decentralized, the scalability and performance gets greater. However, sacrificing some decentralization means that trust needs to come from outside the network.

5.4 Modification required in comparison to the conceptual model

The conceptual model has 7 main steps that are a requirement for this program.

1. The task owner (Alice) selects a file, encrypts it and shards it to prepare the file for submission.
2. Alice submits a task to the network.
3. A set of nodes (Masternodes) take a copy of the task on a top level blockchain.
4. Masternodes distribute the task to workers.
5. Alice can send out challenges periodically.
6. Workers solve the task, most common answers will be placed into the low level blockchain due to the consensus.
7. Return solution to Alice.
8. Destroy work level blockchain.

5.4.1 Modifications required for Ethereum

Encryption and sharding

For distributing files on a decentralized storage system, it is required that the system be able to handle a storage solution. Encryption and file storing properties are not default feature of the Ethereum blockchain. Therefore, in case the selected platform would be Ethereum, an additional layer of applications would need to be built on top of it. A suitable option would be InterPlanetary File System (IPFS) [44]. IPFS can provide a P2P distributed file system. Combination of IPFS and Ethereum, a version of DGCS could be created. Instead of location based addressing (look up a certain location such as http://website.com/picture.jpg) IPFS uses content based addressing (for example https://gateway.ipfs.io/ipfs/QmRAQB6YaCyidP37UdDnjFY5vQuiBrcqdyoW1CuDgwxkD4).

Content based addressing uses the hash of a file instead of it's location. When you look for a resource, you will need to define what it is you want to find instead of defining where to find it. Note that the resource defined above is not stored on IPFS's web server, but rather on someone's computer. IPFS

only provides a gateway to that file. Each file has a unique hash, which can be compared to a fingerprint. When you want to download a file, you will need to ask who has the file with a certain fingerprint and the node who has it will provide it to you. This ensures the integrity of the file, and also ensures that no-one can tamper with your files.

IPFS also allows encryption of files. After encrypting of the files, they can be sharded. The sharding process has already been described in Section 4.3. Following sharding, the task owner can upload such shards to a set of masternodes.

Sharding is not yet supported by the Ethereum blockchain. The lead developer of Ethereum, Vitalik Buterin, says that sharding is under development [35]. He has not confirmed when the sharding process would be available for public use. Therefore, we cannot assume that this technology will be available in the foreseeable future.

Combining Ethereum based sharding and the IPFS file system would allow users to share files in a discrete manner.

Also, due to the fact that all personal and non personal data would be encrypted, it is possible to fulfill the **security** and **privacy** desired properties listed in Section 3.3.

Creation of top and low level Blockchain

A set of masternodes will need to run a top level blockchain where they have a copy of all available tasks to be solved, and all the tasks that have been solved. This allows anyone to check and ensure that their task was indeed executed. Assuming that the chosen solution would be implemented on Ethereum the top level blockchain would need to run separate from the Ethereum main net. This would require a side chain off of Ethereum. The idea of creating side chain off of the main net was introduced by Joseph Poon and Vitalik Buterin in Aug. 2017 [51]. Plasma is a proposed framework that allows a way to scale computation on a blockchain by creating an economic incentive to operate the chain.

However, Plasma is under development at this point. Therefore, we cannot assume that this technology will be available in the foreseeable future just like sharding.

These two technologies allow scaling possibilities for Ethereum chain. *"Incredibly high amount of transactions can be committed on this Plasma chain with minimal data hitting the root blockchain." [51]* The creation of a lower level blockchain could be implemented as a child of the top level blockchain. This would require the same properties as creation of the higher level blockchain.

Plasma is one promising technology that might validate the concept of nested blockchains. Although this project hasn't been publicly released, this is a starting point whether to assess whether this requirement is feasible. There does not seem to be any conceptual problem when creating nested blockchains. The original conceptual model is based on such design.

Sending a task to the network and distributing tasks to the workers

We will need to assume that sharding and plasma will be available for the public soon. Combining sharding, plasma and previously mentioned IPFS such technologies could handle file distributing on a decentralized manner. One of the

main challenges that will be encounter is to combine all these technologies. Due to the fact that the Ethereum developers are currently working on implementing sharding and Plasma we can assume that the combination of that with Ethereum blockchain will be feasible. According to Micheal Chan, combination of IPFS with Ethereum is also possible [36].

Challenges

Challenges are not yet supported on the Ethereum blockchain, and there is no reference of it being under development either. Challenges would need to be implemented on top of Ethereum. A possibility of how challenges could be handled is described in Sub-Section 5.8.1.1.

Consensus during task solving

The current consensus algorithm of Ethereum is PoW, which requires lots of computing power in order to verify blocks. Such a computationally demanding algorithm would be unnecessary because each task will be encrypted by the task owner. There is no need for the masternodes to run endless hashing algorithms, trying to find nonces. A version of PoI could create a vote based system, where the most important nodes decide what really has happened. When the vote about a certain task reaches over 50% of the whole network, a new block will be created and the task will be placed into the task queue. This process can repeat on the lower level blockchain. Multiple nodes would need to compute the right solution. Once that is done the right solution will be selected based on whichever results has the majority in the whole network. This is how the right solution would be selected.

Destruction of a Blockchain

Destruction of an Ethereum blockchain is not something that is supported. However, this can be achieved when all worker nodes remove their copy of a chain. This can be done by the command `"geth removedb"`. This command allows to keep workers' resources freed up after a task is completed. This function could be automated by the software to ensure of automatization of the system.

Conclusion

With currently available technologies, the Ethereum blockchain would not be suitable to implement the DGCS. However, there are lots of developments currently ongoing, which could potentially make this computer system possible to create in the near future.

5.4.2 Modifications required for Hyperledger Fabric

Encryption and sharding

Hyperledger Fabric takes into consideration various aspects of data privacy. It allows users to create channels where they can authorize certain participants to see the data for the chaincodes which are running on the channel. Such chaincode initializes and manages ledger state through transactions submitted

by applications. It is also possible to change visibility of a channel. Encryption and or hashing of data is also possible before calling a chaincode. The ledger data can also be encrypted via file system encryption [23]. Due to the properties of channels it is possible to fulfill the **security** and **privacy** which were the desired properties listed in Section 3.3.

Sharding at this point is not implemented, however based on the feedback of Hyperledger Fabric contributor: "Sharding is certainly a technique we can apply to the Fabric..." [18].

Task submission and Task Distribution

A task submission follows after encryption and sharding of the data is completed.

The creation of a multi level blockchain solution on Hyperledger Fabric would require the creation of a new channel. A Hyperledger Fabric channel is a private 'subnet' of communication between two or more members of the network, This subnet would enable the task owner to jump straight to the workers, leaving out the masternodes for the task distribution. This feature is useful when a task owner already has established certain trusted workers. This private channel could allow a safer environment for the task owner (assuming that all workers are trusted by the task owner). To create this new channel, the client calls or creates a configuration system chaincode (smart contract) and references properties, such as the initial peers and members, who can be part of the network. This will then create a genesis block for the channel ledger that will store the configuration of the channel. This configuration file might include information such as channel policies, members, and initial peers (so called anchor peers). Adding a new member to an existing channel would require the peers to share the genesis block with the new member.

The use of channels would create a much simpler overall system. Also this is an already implemented functionality of Hyperledger Fabric. Therefore, the use of channels is inevitable in case the selected platform would be Hyperledger.

As I mentioned above, the use of channels would only be possible if the task owners already have an existing trusted set of workers. If a task owner would only use channels to solve his/her tasks then the use of channels would lead to a more centralized solution.

The idea of Hyperledger Fabric is built for industrial applications, where a centralized solution is not an issue. However, to get a fully decentralized general propose computer, the use of channels should be disabled. This would make Hyperledger Ledger a very featureless system without channels.

Challenges

Real time challenges are not yet supported by Hyperledger Fabric. At this point there is no any sign of it being under development either. Challenges would need to be implemented as a plugin feature for Hyperledger Fabric. However, at this point it is unclear whether this would be something that the Fabric could natively support.

Consensus

Hyperledger Fabric supports modular consensus protocols [29]. This enables participants get tailored protocols. This property would allow DGCS to have

different consensus algorithms for each of its task types.

Removal of a blockchain/chaincode

Currently the removal of a chaincode is not supported. However, it seems[26] like it is most likely will be released in a future version of Fabric. This would enable users to free up resources that are not needed anymore.

5.4.3 Feasibility compared to conceptual model

The system I have proposed is required to perform "any" task in a distributed manner. This means that it needs a **controlled environment**, an **ability to install applications and tools** that a task might need, a means of **distributing and collecting results** of a task, and a **coordinating program to manage this functionality autonomously**. These requirements are defined in Section These properties are created based on the desired properties of the system (Section 3.3). The above mentioned requirements will be further discussed in Subsections 5.4.3.1-5.4.3.4.

5.4.3.1 Controlled Environment

The use of docker [38] containers is proposed for this system. Docker allows a safe environment for both task owner as well as workers and masternodes.

Docker is currently a widely available application. Therefore, the portability of such software is guaranteed. Using a controlled environment such as docker, the user should have the option to select the resources he/she is willing to "share". A controlled environment enables greater security and privacy over a non controlled environment. A controlled environment would also enable users to create an efficient system, that only uses resources when it needs. This will partially fulfill the **efficiency** part of the desired properties.

5.4.3.2 Ability to install Applications and Tools

The required applications and tools can be installed in a docker image. Some projects are specialized for certain tasks but nobody seems to have made a general application that can solve all type of tasks. Due to this, the feasibility of creating a general application is undetermined at this point. However, creating the application and tools for a specific task only has been done before by others, therefore for some specific tasks it is feasible.

Creating a new type of task would require the task owner to create a plugin that does the task. Then this task can be tested in his/her own container. After testing it could be pushed (as a modification to the current source code) as a new task that the application can work with. This suggestion then can be approved or discarded by the public.

With the ability to install different tools and applications by use of plugins, the users can expand the functionality of the system in any way they wish. This would fulfill **generality** part of the desired properties.

5.4.3.3 Distributing and Collecting results

After a task is encrypted and sharded, it can be distributed on the network. Distributing on the network would involve a P2P connection between the task owner and a set of masternodes. The masternodes will download the task from the task owner. This task will be then placed in the top level blockchain, which is kept on all masternodes.

P2P connections allow great efficiency when it comes to data distribution. With direct connection between peers the only limiting factor will be internet speed. P2P connections will fulfill partially a desired property which is **efficiency**.

Distributing and collecting results can be handled by creating P2P connections between nodes. This is further explained in Section 5.7.

5.4.3.4 Autonomous coordinating program

The distribution and completion of a task should be handled autonomously. This means that application should be able to run without any user input. This shall be true for all parties involved in the process (task owner, masternodes, and workers). This means that a marketplace for tasks should be created and an autonomous matchmaking should take place. This matchmaking should find the most suitable deals for the task owner as well as for the workers. The hosting of the marketplace could be also implemented as a task and fed to the network of workers, hence allowing a decentralized marketplace allowing fair trades for everyone.

Due to the automatization of the program the software should become very easy to handle. This will fulfill the **ease of use** desired property declared in Section 3.3.

5.4.4 Legal Feasibility

Each block can be encrypted by the task owner. Due to this, sensitive data is hidden from the public. Encryption algorithms used today (such as AES256 [2] are considered safe but with technological improvements such algorithm can be broken or compromised). Hence, there are no conflicts created with any legal requirements. The software is meant to be implemented by students and released to public as an open source project.

Based on newly introduced General Data Protection Regulation (GDPR) any organization or business that operates in the European Union (EU) or handles data relating to people living in the EU have responsibilities. Such responsibilities can be found in the following link: https://gdpr-info.eu/art-24-gdpr/

A decentralized computer system would not be considered as a business or and organization. Therefore, at this point it is unclear whether this regulation would be relevant for DGCS. Once a blockchain is created and some information has been published on it, it is not possible to remove this afterwards from it. Therefore, if someone would have uploaded some personal information, which later would like to remove from the chain, it would not be possible. This could potentially lead to legal issues.

5.5 Schedule and Resource feasibility

As mentioned earlier in this thesis, a number of similar projects are currently being researched. Many of these projects have full time employees who are working on creating a platform for outsourced computation/storage. This project is not a one man job by any means, and a lot of development still need to take place before a fully working product can be created.

Previously mentioned Project Golem has 32 full time employees, and started developing Golem around April 2016. Two years later they released their first beta: Golem Brass.[52]. Golem Brass is specialized for a **single purpose** computation which is video rendering task. In order to develop software for *any* task type, the scale of Golem Brass would need to up-scaled by a lot. To put into perspective how enormous this project is, in order to develop a platform similar/competitive to Golem Brass, a large group of software developers (30+) and 2 years of free time would be required. This makes a full implementation unrealistic for this project's scope.

5.6 Challenges with DGCS

Although the concept of DGCS has many advantages over centralized solutions, there are still many challenges to be addressed.

- Motivation for usage: Unless there is a financial benefit for people to adopt the system, adoption will be low. A financial motive for people to join the network is a must. Without financial compensation, no system could ever compete with centralized systems. Users will also need to find better pricing plans for decentralized solutions than centralized ones in order to take advantage of it on a large scale.
- Handling a decentralized marketplace like that proposed for DGCS would have economic and regulatory barriers. As people from anywhere could potentially join this network, a set of regulations world wide would need to be applied to this system. This makes the overall system a much more complex in the eyes of lawyers.
- Expanding to do any task is still uncertain. As initial development begins, milestones can be laid down and a better understanding achieved.

5.7 Sketch of implementation

Due to time constraints mentioned above, the full implementation of such a system for a single person within three month is unreasonable. Based on the feasibility study, a sketch of an implementation is provided. This should give a rough idea what the basics of an DGCS should include. This also includes a very basic task and block implementation. It also provides a better understanding of how blocks of a blockchain are built up, and how queues of tasks *could* be represented.

5.7.1 Technical perspective

The Sub-Sections 5.10.1 - 5.10.1.4 will refer to a Docker container, which is the hypothetical system that will be running on the host and client's computer. The idea is that a single docker image can handle all communication that would be required by the computers. Once the Docker container is downloaded by the user, he/she could select what type of user he/she will become. After selecting the type (such as task owner, node or masternode), the container should download the necessary components based on which type the user has selected. A user should be able to be select an individual type or multiple different types of accounts.

5.7.1.1 Task owner

Initial connection and configuration

In case of the task owner, we will need to assume that an established network of masternodes already exist.

Masternodes create an open network which the task owner can access by downloading a docker image and running this image on his/her computer. This docker image will initiate a connection to the group of masternodes that create the top layer of the network. To initiate such connections, a custom software will need to be implemented.

By running this image, the task owner will also start hosting a local website. Each user can become a task owner. Hence all users will have this option. However, to become a worker or masternode, an external step should be taken. This website will need to have different options to create a new task, manage existing task, or end task. For initial connections to the top layer of the network, it is required to have a couple of static nodes. Due to the fact that running a static node is not a computationally heavy process, these could be hosted by the developers.

Once the initial connection is established, the task owner's computer should search for further nodes in the top layer network. The addresses of the further nodes will be available for the user once it established the connection between itself and the initial nodes.

Submitting a file to the network, sharding, creating challenges, and uploading

Using a user interface, the users will need to have different options. The task owner will have an option to select a new file/folder, and select what kind of task type it is. In this example we are looking at data storage, therefore we assume that the task owner wishes to backup a file/folder on this distributed network. We should assume that there are two files, one is an image with a size of 2 MB and a video file of size 1022 MB. Once these two files are selected they will be placed into so called shards. These shards have a verifiable size and it is up to the task owner to decide how big he/she wants them. The more shards there are the more difficult it is to back it up, while with smaller blocks data corruption and data loss has a higher probability. For this example we assume that the user will have fewer, large shards.

Once the sharding process is finished, the custom software automatically encrypts the task shards. Using AES256 [2] encryption on the data, it is kept secure and private. At this point the user will have 32 encrypted shards of 32 MB each. At this point, the user can create some challenges. A challenge will be created in such a way:

$$Challenge_n = Hash(shard_m + salt)$$

(Note that there can be an infinite number of challenges, due to the fact that the salt number can be any random number, and it is possible to create multiple different challenges for the same shard)

After generating all the challenges, the Merkle root can be calculated by the following way:

```
for (i = 0; i < numOfChallanges; i++){
    root = root + hash(challange_i);
}
```

After adding a new challenge the Merkle root will need to be recalculated. The set of challenges and the Merkle root should be kept secret by the task owner. Also the task owner should ensure that each challenge is unique, to ensure that no worker can copy an existing solution and feed that back to the task owner.

Now the user can send the shards out to the network, and can send challenge (consists of a request of a hashed value of the shard with the sent salt).

Uploading the shards to the network can be done by a basic Unix command called rsync [58]. Rsync is a copying tool. It can copy locally, to/from host(s) on a remote shell. Its a versatile tool, that offers lots of different options.

5.7.1.2 Masternodes

Initial connection and configuration

To join to a set of masternodes, the same docker image referred above needs to be downloaded. Once the image is running, the user should be able to open a website. After establishing a connection between multiple nodes (explained above) the user should click on a button, which says connecting to nodes. After filling out some specific information, like limiting power usage, and scheduling, it should send its address to the list of masternodes, and start synchronizing the top level blockchain. Synchronizing the blockchain is a CPU intensive process. The CPU will verify each block as they are downloaded. It must hash the blocks and check the proof of work, hash the data in the block and compare that with existing blocks.

Once the blockchain is synced up with the rest of the network it will be ready to receive new tasks, and create blocks.

Getting a task

The masternodes will start receiving the shards from the task owner. At this point the masternodes will create a new task. A task contains the following properties:

```
class Task {
    constructor(data, id, type, numOfSubTasks,
                active){
        this.data = data;
        this.id = id;
        this.type = type;
        this.numOfSubTasks = this.numOfSubTasks;
        this.active = this.active;
    };
};
```

The `data` has the actual shard(s) in it (example of data:[$\texttt{shard}_1$, $\texttt{shard}_2$ ... $\texttt{shard}_{numOfSubTasks}$]), the `id` will be a unique identification for the task. The `type` identifies the task's type in our case this will be storage, and the `numOfSubTasks` will let the masternodes know how many shard to expect from the task owner. `active` will be an internal check for the workers whether the task is being executed already by another worker or not.

Task Queue

Once the transfer of all shards are completed, the task will be pushed on the `taskQueue`. Example of the `taskQueue` would look before a task is in:

```
taskQueue[
    task[data[shard_1, shard_2... shard_numOfSubTasks],
         id = uniqueID,
         type = "anytype",
         numOfSubTasks = x
         ]

]
```

And this is how one would look after the task being added:

```
taskQueue[
    task[data[shard_1, shard_2... shard_numOfSubTasks],
         id = 42,
         type = "storage",
         numOfSubTasks = 32
         ]
    task[data[shard_1, shard_2... shard_numOfSubTasks],
         id = uniqueID,
         type = "anytype",
         numOfSubTasks = x
         ]

]
```

Each modification of the `taskQueue` or the `solutionQueue` will mean that a new block will need to be created. This block is the data base of all tasks, to be completed and already completed.

Block

Each block needs to have the following properties:

```
class Block {
    constructor(index, previousHash, timestamp,
    taskQueue, solutionQueue, hash) {
        this.index = index;
        this.previousHash = previousHash.toString();
        this.timestamp = timestamp;
        this.taskQueue = taskQueue;
        this.solutionQueue = solutionQueue;
        this.hash = hash.toString();
    };
};
```

The `index` of the block shows the height of the blockchain, the `previousHash` gets the previous block's hashed value, the `timestamp` gets the time of the block creation, and the `taskQueue` and `solutionQueue` will have the tasks and solutions in them.

Management of workers

Masternodes will select the workers for a specific task. Selection of workers for a task can be done in different ways:

- First come first serve: The workers that report for a task the fastest will be selected.
- Closest neighbour/Lowest latency: The masternodes will select nodes that are physically close to the task owner. This can be done by `traceroute`, or with `ping`. Low latency is not required for most task, but for some specific task it might be more suitable (such as web hosting task).
- Most reliable: Masternodes could track workers, and set a level of importance to each of them based on how long they been on the network, what is their `uptime` percentage, or how much computing power did they dedicate to task solving. By creating level of importance, masternodes could create a hierarchy, and offer most reliable nodes to those who require it.

After the masternodes select the most suitable workers based on the task owner's request, they can send the task off to them.

Forwarding a task to the workers

Once a new task appears in the `taskQueue`, the masternodes will notify the workers and tell them that there is a new task to be solved. The first worker who responds to the masternodes will take the task and create a low level blockchain for that specific task and set the `active` property of the task to `true`.

These `subTaskQueue` elements can be then copied to the local machines of the workers. See more on this below.

5.7.1.3 Workers

Initial connection and configuration

Similarly to the first two user types, the worker will need to download the same docker image, run it and get connected to the masternodes. Once the connection is established, the user will need to select that it would like to become a worker type. Similarly to the masternode version, the user will have the option to specify specifics about its system, such as how long and when the system should run. This enables a fully autonomous system, and ensures that the computer will be maximized as the user wishes.

Once the user is connected to the masternodes, it will either start a new set of worker (if there are non existing) or join to the group of workers. The addresses of the workers will be stored and backed up by the masternodes too.

Getting a task from masternodes and solving a particular task

When a worker goes idle, it will send requests for a new task to the masternodes. If the masternodes see that there is a new task available in the `taskQueue`, then the worker gets notified about an incoming task, and it can get started with it.

Creating a low level blockchain

To create the lower level blockchain the worker will need to initialize a new genesis block for the task. This can be done in the following manner:

```
//Create the genesis block
var createGenesisBlock = (taskID) => {
    var tID = taskID;
    var i = 1;
    var t = Math.floor(Date.now() / 1000);
    var subTaskQueue = [];
    var subSolutionQueue = [];
    var hash = SHA256(tID + t).toString();
    return new Block(tID, i, "0", t, subTaskQueue,
    subSolutionQueue, hash);
};

//Create a blockchain
var lowBlockchain = [createGenesisBlock(taskID)];
```

Once a blockchain has been created, each shard from the `taskQueue` of the high level blockchain gets copied to the `subTaskQueue`. The worker then starts working on the specific task that is given to it. At any point more workers should be able to join to this process.

5.7.1.4 Limitations of the system

Using the above mentioned technologies would lead to a number of limitations.

- Docker *preferably* runs on unix based systems, while windows does support it, its not as straight forward as it is on linux.

- For each task, a minimum of 3 workers are required to verify that the solution is indeed the correct one.
- We need to make an assumption that there are more honest than dishonest nodes.

5.8 Conclusion of the feasibility study

The purpose of this feasibility study was to determine the feasibility of various alternatives to implement a blockchain based distributed supercomputer, where a task owner can check in real time whether their task is being executed or not. The goal of the feasibility study was to find whether the existing blockchain platforms would fulfill the desired properties. Based on this study, some properties could be fulfilled at this moment, but others could not. There are still lots of active development going on on both of these chains. Both the Ethereum blockchain and Hyperledger Fabric supports a wide variety of tools, but lack of the possibility to send out real time challenges for to check task execution. This would need to be implemented in both cases.

Based on the project specifications and the availability of current technologies, the project could be feasible, however it is not recommended unless time and resources are not a limiting factor.

Closing Thoughts

6.1 Conclusion

This project started out with an assessment of different blockchain technologies, where I tried to provide sufficient background information for it to be easily be understood even by people who are not familiar with blockchain. This was followed by the problem definition ("How to ensure of task execution in a distributed computer system in a trustless manner?"), along with its corresponding properties. The problems are met with a potential way of solving them. A conceptual model was then introduced, which was explained in more detail with sequence diagrams. The sequence diagrams led to potential use cases, which were evaluated in a step-by-step process. Finally, a proof of concept in the form of a feasibility study was created. Possible solution(s) to the research problem were then obtained. The feasibility study established that two main blockchain technologies (Hyperledger Fabric and Ethereum) have the potential of being a suitable platform for a DGCS. Both of them have their advantages (modular channels for Hyperledger Fabric and public trustless blockchain for Ethereum) and disadvantages (no challenges possible for either, and low transaction throughput for Ethereum) compared to each other. Only time can tell which will be sufficient for implementing DGCS.

6.2 Evaluation and Future Work

Based on the feasibility study provided in the previous chapter there are clearly different bottlenecks that still need to be solved before someone could start the implementation of such system. These bottlenecks (lack of challenges, lack of scaling for (ethereum only), lack of sharding) are mainly created due to software limitations of Ethereum and Hyperledger Fabric. It is clear that both of these platforms could one day provide a great base for creating a decentralized computer system, but at this point our technology has not reached this level.

There are plenty of additions to the work that has been done here. The first step for creating such a system would require a group of programmers to create a MVP for a single specific task.

- Implementation of a minimum viable product (MVP) for a specific task.
- Adding marketplace and match-making between task owners and workers.
- Adding priority queue system with weights, allowing certain tasks to be executed before others.

- Adding option to make a project public/private. Open projects could get funded. Such fund would go towards the overall reward for such task, forcing it higher in the priority queue.
- Adding a public voting option for deciding who can stay and who has to leave the network.
- Support for multi platform computer systems.

Bibliography

[1] N. Szabo. (1994). Smart contracts, [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

[2] J. Daemen and V. Rijmen, "Aes proposal: Rijndael", 1999. [Online]. Available: https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf.

[3] I. Foster and C. Kesselman, *The Grid: Blueprint for a new computing infrastructure*. 1999. [Online]. Available: https://books.google.nl/books/about/The_Grid.html?id=ONRQAAAAMAAJ&redir_esc=y.

[4] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric", *International Workshop on Peer-to-Peer Systems*, pp. 53–65, 2002. [Online]. Available: https://link.springer.com/chapter/10.1007%5C%2F3-540-45748-8_5.

[5] M. Stamp, *Information Security: Principles and Practice*. Wiley; 2 edition, 2005, ISBN: 978-0470626399.

[6] Us-cert. (2009). Understanding denial-of-service attacks | us-cert, [Online]. Available: https://www.us-cert.gov/ncas/tips/ST04-015.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[8] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake", 2012. [Online]. Available: https://peercoin.net/assets/paper/peercoin-paper.pdf.

[9] W. Y. Arms. (2014). Feasibility studies, [Online]. Available: https://www.cs.cornell.edu/courses/cs5150/2014fa/slides/C1-feasibility.pdf.

[10] M. N. Durrani and J. A.Shamsi, "Volunteer computing: Requirements, challenges, and solutions", *Journal of Network and Computer Applications*, vol. 39, pp. 369–380, 2014. DOI: https://doi.org/10.1016/j.jnca.2013.07.006.

[11] D. Sandholm. (2014). Amazon's 'predatory pricing' questioned, [Online]. Available: https://www.the-blockchain.com/2018/03/01/400-percent-growth-decentralized-marketing-technology-projects-report/.

[12] D. Vorick and L. Champine, "Sia: Simple decentralized storage", 2014. [Online]. Available: https://sia.tech/sia.pdf.

[13] B. Group. (2015). Proof of stake versus proof of work, [Online]. Available: https://bitfury.com/content/downloads/pos-vs-pow-1.0.2.pdf.

[14] A. Hayes. (2016). The three major bitcoin protocols explained, [Online]. Available: https://www.investopedia.com/news/three-major-bitcoin-protocols-explained/.

[15] I. Kurochkin and A. Saevskiy, "Boinc forks, issues and directions of development1. procedia computer science", *Procedia Computer Science*, vol. 101, pp. 369–378, 2016. DOI: https://doi.org/10.1016/j.procs.2016.11.043.

[16] R. Russell. (2016). #bitcoin-lightning: Things to know, [Online]. Available: https://medium.com/@rusty_lightning/bitcoin-lightning-things-to-know-e5ea8d84369f.

[17] Unknown, "Golem white paper", 2016. [Online]. Available: https://golem.network/doc/Golemwhitepaper.pdf.

[18] Vukolic. (2016). Hyperledger sharding future possibilities, [Online]. Available: https://github.com/hyperledger-archives/fabric/issues/1631.

[19] S. Wilkinson, T. Boshevski, J. Brandoff, J. Prestwich, G. Hall, P. Gerbes, P. Hutchins, and C. Pollard, "A peer-to-peer cloud storage network", 2016. [Online]. Available: https://storj.io/storj.pdf/.

[20] I. M. Cloud. (2017). 10 key marketing trends for 2017, [Online]. Available: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WRL12345USEN.

[21] B. Darrow. (2017). Gartner cloud rankings, [Online]. Available: http://fortune.com/2017/06/15/gartner-cloud-rankings/.

[22] D. J. Hosp, *Crypto Currencies; Bitcoin, Ethereum, Blockchain, ICO's & Co. simply explained*. Julian Hosp Coaching LTD, 2017, ISBN: 978-9881485083.

[23] Hyperledger. (2017). Hyperledger fabric faq, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.1/Fabric-FAQ.html.

[24] ——, (2017). Hyperledger-fabric, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.1/.

[25] S. Mitchell. (2017). Amazon control the underlying infrastructure of our economy, [Online]. Available: https://motherboard.vice.com/en_us/article/7xpgvx/amazons-is-trying-to-control-the-underlying-infrastructure-of-our-economy.

[26] M. Parzygnat. (2017). Delete / remove chaincode, [Online]. Available: https://jira.hyperledger.org/browse/FAB-1149.

[27] D. Sonstebo. (2017). Iota development roadmap, [Online]. Available: https://blog.iota.org/iota-development-roadmap-74741f37ed01.

[28] Amazon. (2018). Cloud object storage | store & retrieve data anywhere | amazon simple storage service, [Online]. Available: https://aws.amazon.com/s3/.

[29] E. Androulaki, A. Barger, V. Bortnikov, C. C. andKonstantinos Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains", p. 30, 2018. [Online]. Available: https://arxiv.org/pdf/1801.10228.pdf.

[30] A. Behrens. (2018). 400 percent growth in decentralized marketing technology projects: Report, [Online]. Available: https://www.the-blockchain.com/2018/03/01/400-percent-growth-decentralized-marketing-technology-projects-report/.

[31] I. BitTorrent. (2018). Bittorentt, [Online]. Available: https://www.bittorrent.com.

[32] Blockgeeks. (2018). Basic primer: Blockchain consensus protocol, [Online]. Available: https://blockgeeks.com/guides/blockchain-consensus/.

[33] F. Blom, "A feasibility study of blockchain technology as local energy market infrastructure", 2018. [Online]. Available: https://brage.bibsys.no/xmlui/bitstream/handle/11250/2502356/18359_FULLTEXT.pdf?sequence=1&isAllowed=y.

[34] BOINC. (2018). Boinc statistics of the day, [Online]. Available: https://boinc.berkeley.edu/.

[35] V. Buterin. (2018). Ethereum sharding, [Online]. Available: https://twitter.com/vitalikbuterin/status/991021062811930624?s=21.

[36] M. Chan, "Build a simple ethereum + interplanetary file system (ipfs)+ react.js dapp.", 2018. [Online]. Available: https://itnext.io/build-a-simple-ethereum-interplanetary-file-system-ipfs-react-js-dapp-23ff4914ce4e.

[37] A. CouchDB. (2018). Apache couchdb, [Online]. Available: https://couchdb.apache.org.

[38] Docker. (2018). Docker, [Online]. Available: https://www.docker.com.

[39] evan.network. (2018). Evan.network is an initiative to build a european blockchain, [Online]. Available: https://evan.network.

[40] B. Evans. (2018). Why microsoft is ruling the cloud, ibm is matching amazon, and google is $15 billion behind, [Online]. Available: https://www.forbes.com/sites/bobevans1/2018/02/05/why-microsoft-is-ruling-the-cloud-ibm-is-matching-amazon-and-google-is-15-billion-behind/%5C#191d12991dc1.

[41] B. Foundation. (2018). Blender.org - home of the blender project - free and open 3d creation software, [Online]. Available: https://blender.org.

[42] A. Gal. (2018). The tangle: An illustrated introduction, [Online]. Available: https://blog.iota.org/the-tangle-an-illustrated-introduction-4d5eae6fe8d4.

[43] IOTA. (2018). Iota project qubic protocol, [Online]. Available: https://qubic.iota.org/intro.

[44] ipfs/ipfs. (2018). Interplanetary file system (ipfs), [Online]. Available: https://github.com/ipfs/ipfs.

[45] jpmorganchase/quorum. (2018). Quorum, [Online]. Available: https://github.com/jpmorganchase/quorum.

[46] A. Kafka. (2018). Apache kafka, [Online]. Available: https://kafka.apache.org.

[47] G. Konstantopoulos. (2018). The state of ethereum scaling, [Online]. Available: https://medium.com/loom-network/the-state-of-ethereum-scaling-march-2018-74ac08198a36.

[48] kushsharma1998. (2018). On sharding blockchains, [Online]. Available: https://github.com/ethereum/wiki/wiki/Sharding-FAQs.

[49] Luxcorerender.org. (2018). Luxcorerender - open source physically based renderer, [Online]. Available: https://luxcorerender.org/.

[50] Ookla. (2018). Global internet speeds, [Online]. Available: http://www.speedtest.net/global-index.

[51] J. Poon and V. Buteri, "Plasma: Scalable autonomous smart contracts", 2018. [Online]. Available: http://plasma.io/plasma.pdf.

[52] G. Project. (2018). Ready, set, launch!, [Online]. Available: https://blog.golemproject.net/ready-set-launch-b14d73ca5400.

[53] Unknown, "Nem whitepaper", p. 26, 2018. [Online]. Available: https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf.

[54] J. Young. (2018). Vitalik buterin: Ethereum will eventually achieve 1 million transactions per second, [Online]. Available: https://www.ccn.com/vitalik-buterin-ethereum-will-eventually-achieve-1-million-transactions-per-second/.

[55] Certificate-transparency.org. (). How log proofs work - certificate transparency, [Online]. Available: http://www.certificate-transparency.org/log-proofs-work.

[56] Gilc. (). Privacy and human rights - overview, [Online]. Available: http://gilc.org/privacy/survey/intro.html.

[57] N. I. of Standards and Technology, "Descriptions of sha-256, sha-384, and sha-512", [Online]. Available: http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf.

[58] A. Tridgell and P. Mackerras. (). Rsync, [Online]. Available: https://linux.die.net/man/1/rsync.

[59] Unknown. (). Proof of burn, [Online]. Available: https://en.bitcoin.it/wiki/Proof_of_burn.

[60] T. Zafer. (). Why client-side encryption is the next best idea in cloud-based data security, [Online]. Available: http://www.infosectoday.com/Articles/Client-Side_Encryption.htm%5C#.W05hKdUzaUk.