

AMERICAN UNIVERSITY OF BEIRUT

BLOCKCHAIN DEVELOPMENT PLATFORMS:
PERFORMANCE COMPARISON

by
IMAN RABIE DERNAYKA

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
June 2020

AMERICAN UNIVERSITY OF BEIRUT

BLOCKCHAIN DEVELOPMENT PLATFORMS: PERFORMANCE
COMPARISON

by
IMAN RABIE DERNAYKA

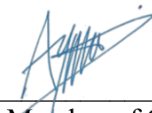
Approved by:



Dr. Ali Chehab, Professor
Electrical and Computer Engineering

Advisor

Dr. Ayman Kayssi, Professor
Electrical and Computer Engineering



Member of Committee

Dr. Imad Elhajj, Professor
Electrical and Computer Engineering



Member of Committee

Date of thesis defense: June 17, 2020

ACKNOWLEDGMENTS

Without the constant help from my supervisor, Professor Ali Chehab, this work would not been possible. I would like to thank him for always being supportive and cheerful while mentoring me throughout this research work.

Also, I would like to thank Mher Kazandjian and the IT team at AUB for providing so many hours of help concerning the experimentation on HPC and Azure.

Lastly, I would like to thank my family members and my dear husband for their nonstop love and encouragement.

AN ABSTRACT OF THE THESIS OF

Iman Rabie Dernayka for Master of Engineering
Major: Electrical and Computer Engineering

Title: Blockchain Development Platforms: Performance Comparison

In this master's thesis, two of the main Blockchain development platforms Ethereum and EOS.IO were compared. In the aim of helping developers choose between the platforms as the backend Blockchain for their apps, a decentralized application along with a corresponding smart contract was implemented on each of the platforms triggering basic operations and timing them. The simulation was tested on both AUB's High Performance Computing facility and Microsoft's Azure, running up to 150 Blockchain nodes while recording the user response time, the CPU utilization, and the totally used memory megabytes. The results in this study show that although recognized as a major competitor to Ethereum, EOS.IO fails to outperform the Ethereum platform, recording a very high response time in comparison to Ethereum.

CONTENTS

ACKNOWLEDGMENTS	V
ABSTRACT	VI
ILLUSTRATIONS	IX
TABLES	X
Chapter	
1. INTRODUCTION	1
1.1 Motivation	1
1.2 Related Work	3
1.3 Aims & Objectives	5
1.4 Limitations & Challenges	6
2. TECHNICAL BACKGROUND	8
2.1 Bitcoin	10
2.1.1 Bitcoin Structure	10
2.1.2 Bitcoin Scripting System	11
2.1.3 Consensus Algorithm: Proof-of-Work	12
2.1.4 Economic Incentive	15
2.1.5 Turing Incompleteness	16
2.2 Ethereum	17
2.2.1 Consensus	17
2.2.2 Resources	19
2.2.3 Smart Contracts & Code Execution	20
2.2.4 Storage	22
2.2.5 Economic Incentive	24
2.3 EOS.IO	24
2.3.1 Consensus	25
2.3.2 Resources	27
2.3.3 Smart Contracts & Code Execution	28
2.3.4 Storage	31
2.3.5 Economic Incentive	32
2.3.6 Permissions	32
2.4 Other Platforms	33
2.4.1 Blockstack	33
2.4.2 Hyperledger Fabric	36
3. METHODOLOGY & SIMULATION	40
3.1 Testing Environment	41
3.2 Singularity Containers	41
3.3 AUB HPC OCTOPUS Cluster	42
3.4 Azure HB-Series	42
3.5 Network Structure	43
3.6 Bash Scripting	43

3.7 Node.js promises and DApps.....	43
3.8 Samples.....	44
3.9 Performance Metrics.....	45
3.10 Testing Assumptions.....	47
3.11 Details of the Ethereum Setup.....	47
3.12 Details of the EOS Setup.....	49
4. RESULTS & ANALYSIS.....	52
More Analysis on Ethereum and EOS.IO Logs.....	66
5. CONCLUSION.....	70
6. BIBLIOGRAPHY.....	71

ILLUSTRATIONS

Figure 2: Ethereum Dapps Statistics [2]. Screenshot By Author.....	2
Figure 1 Eos Dapps Statistics [2]. Screenshot By Author.....	2
Figure 3: Blockchain Data Structure	8
Figure 4: Merkle Tree.....	10
Figure 5: Unspent Transaction Output	11
Figure 6: Double Spending Problem	12
Figure 7: Mining (Proof-Of-Work)	13
Figure 8: Bitcoin Reward Halving. Screenshot By Author.....	15
Figure 9: Ethash Steps	17
Figure 10: Gas Cost Example.....	19
Figure 11: Deploying An Ethereum Smart Contract Steps	20
Figure 12: Storing Two Words In The Same Trie (Ether & Ethereum), To The Left Is A Standard Trie And To The Right Is A Patricia Trie	22
Figure 13: Smart Contract Storage Resources	23
Figure 14: Top Producers On The Main Eos Network. Screenshot By Author.....	26
Figure 15: Eos Local Chain Example. Screenshot By Author.....	29
Figure 16: Eos Blockchain Structure.....	31
Figure 17: Eos Inflation Distribution	32
Figure 18: Kafka Consensus.....	37
Figure 19: Singularity Containers	41
Figure 20: Singularity Run Script	44
Figure 21: Dapp Steps	46
Figure 22: General Steps Of The Performance Experiment.....	47
Figure 23: Eos Hpc Results	57
Figure 24: Eos Azure Results.....	57
Figure 25: Ethereum Hpc Results	58
Figure 26: Ethereum Azure Results	58
Figure 27: Azure - Eos Vs Ethereum (Network Of 50 Nodes)	59
Figure 28: Azure - Eos Vs Ethereum (Network Of 10 Nodes)	59
Figure 29: Azure - Eos Vs Ethereum (Network Of 150 Nodes)	60
Figure 30: Azure - Eos Vs Ethereum (Network Of 100 Nodes)	60
Figure 31: Hpc - Eos Vs Ethereum (Network Of 50 Nodes)	61
Figure 32: Hpc - Eos Vs Ethereum (Network Of 10 Nodes)	61
Figure 33: Hpc - Eos Vs Ethereum (Network Of 150 Nodes)	62
Figure 34: Hpc - Eos Vs Ethereum (Network Of 100 Nodes)	62
Figure 35: Ethereum 50 Transactions Submission Logs.....	66
Figure 36: Ethereum 10 Transactions Submission Logs.....	66
Figure 37: Ethereum 100 Transactions Submission Logs.....	67
Figure 39: Eos 10 Transactions Submission Logs	68
Figure 38: Eos 50 Transactions Submission Logs	68
Figure 41: Eos 100 Transactions Submission Logs - 2	69
Figure 40: Eos 100 Transactions Submission Logs - 1	69

TABLES

table 1: Eos.io System Contracts And System Accounts	30
Table 2: Octopus Vs Azure Hb-Series Specs	42
Table 3: Summary Of Dapp Functions.....	51
Table 4: Eos Azure Summary	53
Table 5: Eos Hpc Summary.....	54
Table 6: Ethereum Azure Summary	55
Table 7: Ethereum Hpc Summary	56
Table 8: Low And High Values Of Eos On Hpc Vs Azure	63
Table 9: Low And High Values Of Ethereum On Hpc Vs Azure.....	64

CHAPTER 1

INTRODUCTION

1.1 Motivation

The Blockchain is bringing big things to the world, much more than cryptocurrencies, it's building the path for everyone to decentralized computing. The Blockchain along with open protocols, server-less applications and the right incentives are shaping the meaning of decentralized computing altogether. Blockchains became very popular because they made everyone discover the beauty of decentralization and transparency. It is now the beginning of a new era, the era of decentralization.

Many platforms arose in this new era, all are very innovative and very crucial for sculpting future coming Blockchains. Though the platforms are very competitive in the market, one should not deny the importance of their variety. We should all acknowledge that this technology is very new and appreciate the different paths to apply it. The first Blockchain platform ever is Bitcoin (2008), it is the Godfather of all Blockchains. It introduced us to this new era and forms till this day the most secure Blockchain.

In 2013, Ethereum was proposed as a programmable Blockchain, introducing smart contracts to the Blockchain community. Smart contracts are programs that exist on the blockchain and their state is part of the Blockchain state. One would invoke the operations of these programs via an API now so called DApps (decentralized Apps).

Ethereum’s concept definitely revolutionized the concept of a cryptocurrency Blockchain

#	Platform	Category	Users (24h)	Volume (7d)	Dev activity (30d)	User activity (30d)
401 ▲16	BLUEBET First cross-chain spinach application	EOS Gambling	169 +64.08%	1,249,310 EOS 4,853,569 USD -41.66%	-	
73 ▼4	WhaleEx Trade Cryptos on WhaleEx	EOS Exchanges	294 +33.03%	464,613 EOS 1,805,020 USD +117.42%	-	
398 ▼2	Dice EOS Betting platform	EOS Gambling	114 +7.55%	308,643 EOS 1,199,078 USD +136.34%	-	
1359 ▼3	EOSPLAY Lottery and dice games	EOS Gambling	19 0.00%	274,779 EOS 1,067,517 USD +15.80%	-	
466 ▼13	BetHash Bet on future blocks and earn a passive income	EOS Gambling	151 -17.49%	138,092 EOS 543,644 USD +36.17%	-	
781 ▼31	EQSHash The fairest crypto casino built on EOS	EOS Gambling	133 -21.30%	122,886 EOS 477,410 USD +100.66%	-	
558 ▼3	SportBet Sportsbook on EOS blockchain	EOS Gambling	50 -10.71%	47,088 EOS 182,627 USD -18.57%	0	
587 ▼1	Newdex EOS Decentralised Exchange	EOS Exchanges	67 +15.52%	42,008 EOS 163,199 USD -28.89%	-	

Figure 2 EOS DApps Statistics [2]. Screenshot by author.

#	Platform	Category	Users (24h)	Volume (7d)	Dev activity (30d)	User activity (30d)
1	MakerDAO Where you can interact with the Dai Credit System	Ethereum Finance	2,608 +10.59%	46,918 ETH 11,094,251 USD -30.18%	1,105 +25.00%	
31 ▼1	Compound Algorithmic money markets	Ethereum Finance	405 +12.81%	46,172 ETH 10,917,891 USD -13.83%	210 +12.30%	
9	Uniswap Protocol for automated token exchange	Ethereum Exchanges	838 +6.23%	32,317 ETH 7,641,645 USD -44.52%	267 0.00%	
5	KyberNetwork Enabling Token Swaps Everywhere	Ethereum Exchanges	1,141 +8.56%	26,300 ETH 6,219,061 USD -12.63%	1,380 +26.14%	
47 ▲3	imToken Tokenlon Iclick trading in your imToken wallet based on Ox	Ethereum Exchanges	295 +5.73%	16,036 ETH 3,791,914 USD -20.12%	213 -41.80%	
38 ▲1	1inch.exchange DEX Aggregator with the best prices on the market.	Ethereum Exchanges	243 +27.23%	14,620 ETH 3,457,085 USD +8.89%	200 +185.71%	
33 ▼2	IDEX Distributed exchange made of smart contracts	Ethereum Exchanges	418 -3.91%	8,493 ETH 2,008,274 USD -35.35%	1 0.00%	
629 ▼21	dice2win Simple and fair dice game	Ethereum Gambling	62 +29.17%	6,719 ETH 1,588,821 USD -66.38%	5 -28.57%	

Figure 1: Ethereum DApps Statistics [2]. Screenshot by author.

by adding on-chain computation. Nonetheless, many decentralized computing critics kept demanding better performance from decentralized platforms in order for them to replace centralized platforms. Another important platform that appeared as a competitor for Ethereum was EOS (initially released in 2018). EOS promised better performance than Ethereum. Till this day, Ethereum and EOS are the most popular decentralized computing

platforms. In the last decade, Bitcoin and its follow-ups were not just buzzwords, rather they were important projects funded by the biggest investors who believed in their power. A huge amount of money was spent for the sake of their growth. Ethereum for example was crowdfunded in 2014 with 18 million dollars, whereas EOS's ICO (Initial Coin Offering on Ethereum platform) raised 4 billion dollars in 2018 [1]. Figures 1 and 2 [2] show the Ethereum and EOS DApps statistics ranked by volume. The volume (7d) in the figures represent the amounts of Ethereum and EOS tokens that have been traded in the last seven days. Millions of dollars of cryptocurrencies are traded daily in thousands of DApps of different categories varying from finance and exchanges to games and gambling. This confirms the importance of understanding these platforms and their characteristics before developing DApps on them which will transfer lots of valuable assets.

Having this significant value, everyone is joining in on these platforms. From decentralized healthcare to transparent supply chains and much more, everyone is eager to implement their own decentralized version of the existing systems. Jumping into this world so enthusiastically with fear of missing out means fast decisions and fast results. One could not deny the importance of performance studies comparing the most well-known Blockchain platforms and the benefit it could provide for new developers and entrepreneurs.

1.2 Related Work

The tradeoffs between security and performance were studied in [3] regarding only PoW Blockchains via a quantitative framework. The authors aimed to analyze the effect of

changing several Blockchain parameters (block interval, block size...) on the risks of double-spending and selfish-mining attacks. The processes of double-spending and selfish-mining were each modeled into Markov Decision Process (MDP). The authors also built a Bitcoin Blockchain Simulator and changed its parameters in order to match the other PoW Blockchains (block interval, mining power, ...). In [4], the authors compared the time to process transactions on two well-known Ethereum clients: Geth and Parity. Each of the nodes was tested by itself only, not in a network with other nodes. The time to execute up to 10000 transactions while varying the RAM was compared in this paper. Others compared a single Ethereum Geth node and a Hyperledger Fabric network [5]. The consensus mechanism is not included in the study. A simple money transfer application was implemented on each platform. By varying the number of transactions sent to each of the networks, the throughput, latency, and execution time were compared. In [6], a comparative analysis was conducted between Ethereum, Hyperledger Fabric, and Corda.

The author of the Master Thesis in [7] discussed techniques to improve the performance and scalability in each of Bitcoin, Ethereum, and Hyperledger Fabric. Some tests have been run on Hyperledger Fabric varying the block size and the number of nodes and analyze the effect it has on the transaction throughput. Blockbench [8] is a framework to analyze the performance of private blockchains. The authors compared two Ethereum networks (Geth network and Parity network), and a Hyperledger Fabric network.

[9] provided a comparative analysis between Ethereum and Fabric. A smart contract was implemented on each and the programming techniques were explained. [10] measured

the performance of the Ethereum Blockchain on the proposed Whiteblock Blockchain testing platform. The study was done on a network of 10 Geth nodes, and several network conditions were varied and several attack scenarios were conducted.

Concerning EOS, [11] detailed the architecture of the EOSIO platform, and also conducted an EOS performance study on the Whiteblock testing framework. Various network conditions were varied upon an EOS network and the effects of the transaction throughput were analyzed.

The authors in [12] benchmarked the amount of computational resources needed per some Ethereum opcodes on different hardware computers. In [13], the most commonly used smart contracts on Ethereum were studied in order to compare the execution time in proportion to the amount of gas used, and thus conclude if the profits of miners are fair.

Another Blockchain benchmarking tool is Hyperledger Caliper [14], it is still under development at the time of writing this thesis. Also, OpBench [15] is a benchmarking platform and it is utilized in order to measure the CPU time required to execute opcodes in the Ethereum Virtual Machine.

Many have addressed Blockchain performance as seen. However, to the best of author's knowledge, there have been no direct comparison of Ethereum vs EOS under the same testing environment. Comparing such popular platforms is a very important study that the Blockchain community needs.

1.3 Aims & Objectives

The aim of this thesis is to compare the performance of the Ethereum and the EOSIO platforms by implementing a decentralized application along with a corresponding smart

contract on each of the platforms, triggering specific operations and measuring the time to complete these operations while varying some variables.

There are primary operations every DApp needs to perform. The research contribution of this thesis is to show the difference in the timing of these primary operations, and therefore help DApp developers choose between Ethereum and EOS as the backend Blockchain.

The study will be conducted according to these objectives:

- The study will be conducted on a network of nodes
- The study will include the load of the consensus, even if minimized
- The study will be conducted on the DApp layer in order to abstract all the differences and complexity of each platform
- The implementation of the two smart contracts and the two DApps should be as close as possible in order to obtain a fair comparison
- The study will compare Ethereum and EOS platforms, which are two public Blockchain platforms.

1.4 Limitations & Challenges

As the study is conducted on a single machine, we know that the study does not accurately simulate real-life behaviors, and that the numbers obtained in this research do not indicate real numbers if taken for each platform by itself. However, because the two platforms were put under same testing environment, and the same conditions were applied, this will give us a decent reflection of the relationships between the tested variables.

EOS's permissioned network is not part of the study, the network is implemented in a P2P fashion, similar to Ethereum.

One of the biggest challenges of this study is how fast the platforms evolve every day, which makes it very hard to keep up. The software versions used in this study are 1.9.1-stable for Geth (Go Ethereum Client), 0.4.25 for Solidity compiler (also the contract successfully compile by the latest 0.6.0 solc version), 1.6.6 for EOSIO and 1.6.1 for eosio.cdt (EOSIO Contract development kit).

CHAPTER 2

TECHNICAL BACKGROUND

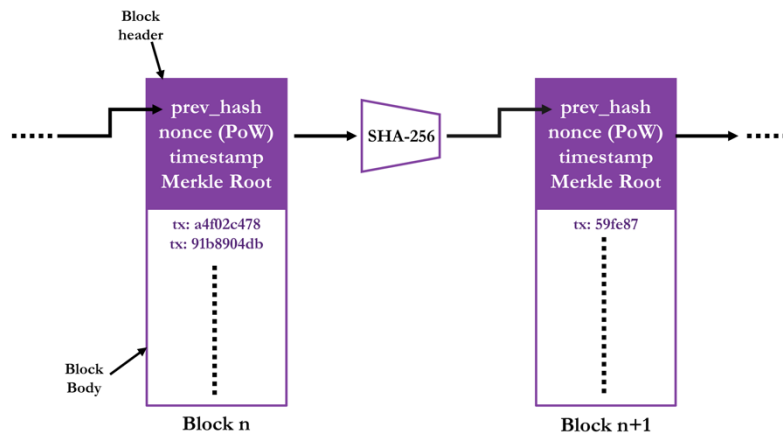


Figure 3: Blockchain Data Structure

Because of its popularity, a big number of people complicated the meaning of Blockchain to the point it is intimidating. But the Blockchain itself is something very simple. The Blockchain is a data structure that looks a lot like a linked list, but the bonds use cryptographic hashes instead of just normal pointers (See Fig.3). The elements in this structure are transactions holding digital assets; these transactions are grouped into blocks and then added, in a single write, as part of the chain. It grows in one direction only, meaning that data can be only appended to it. This data structure is synced between a network of nodes based on an ordering algorithm that ensures all nodes have the same data structure. The Blockchain's main function is to provide an immutable log of transactions. When in the context of financial transactions, the Blockchain is referred as an immutable ledger. Every Blockchain system is composed of 3 main components: the Blockchain by itself enforced by public-key cryptography (structure), the consensus

protocol (ordering), and an incentive mechanism. The order of blocks matters a lot in the Blockchain: The Blockchain is not maintained on a single computer, but a replica of it is available on not just few, but maybe thousands of other nodes. As such, it is crucial to have the same blocks in the same order on all nodes. This is where the role of the consensus comes in. A consensus algorithm makes sure all nodes agree on the next block, and if nodes agree every time on the next block, then they agree on the order of all blocks and stay synchronized. Lastly, something is needed to incentivize nodes to behave in a good way, this is where validators of the system gain some benefit. Some view the Blockchain as an infinite state-machine [16], and the consensus as the managing algorithm that validates every state-transition.

Centralization has always been the strategy in everything. This is how the human brain thought for years. If two complete strangers wanted to exchange some money, a middleman that both parties trust should be part of the transaction, like the bank. Even in public key cryptography, we need a CA that issues certificates proving the validity of public keys and identities. This middle man showed to be unreliable in several cases [17]. Some [18] proposed an interesting problem that represents the meaning of decentralization. This problem is The Byzantine Generals Problem (BGP). The problem states that a group of Byzantine Generals each along with their armies are waiting to attack a city. All armies should attack at the same time or the attack will fail. The issue is that amongst the generals, there are some traitors who want the attack to fail. So, the traitors try to alter the messages spread between the generals to change the time of the attack of some armies, and thus fail the attack.

A solution to this problem would be a method that guarantees reaching the agreement between the good generals even though there are some traitors trying to mess everything up. What we have been doing for years is put a general in the middle who everyone gets information from, and if this general got compromised in any way, the whole system fails. However, Bitcoin (2008) [19] has been the only real working solution to this problem with its Proof-of-Work consensus algorithm. In this chapter, a detailed background on Bitcoin, Ethereum, EOS.IO, and two other Blockchain platforms is presented.

2.1 Bitcoin

Bitcoin [19] is the first Blockchain ecosystem which introduced the best ever proposed decentralized cash system till this day. Satoshi Nakamoto aimed to design a cash system without a middleman, and with all the Blockchain components, proper consensus and proper incentive, he succeeded in creating the largest digital ledger ever made.

2.1.1 Bitcoin Structure

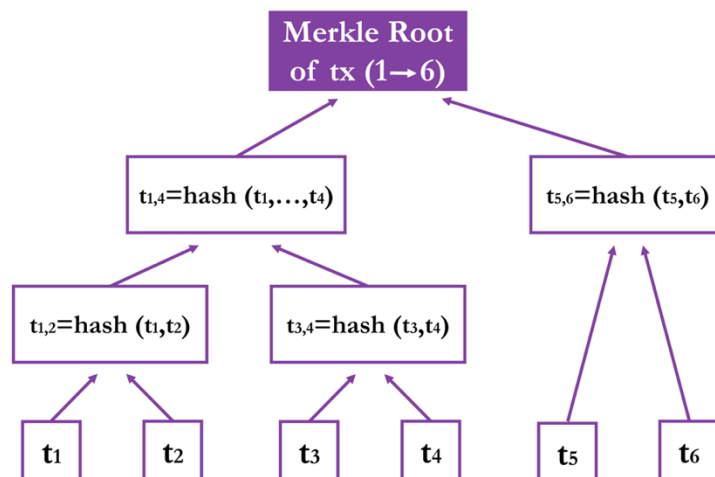


Figure 4: Merkle Tree

Transactions are grouped together into a Block following Merkle Trees structure (also known as a hash tree structure). The hash used in Bitcoin is SHA-256 which is considered a secure cryptographic hash function till this day. All transactions are hashed together until one hash is obtained at the end, the Merkle Root. Any bit change in any of the transactions will result in a very different root due to the Avalanche effect of cryptographic hash functions. Each block has a header and a payload. The transactions are stored in the body of the block. In each block header, several fields are stored, mainly the Merkle root, the timestamp, the SHA-256 hash of the previous block, and the Proof-of-Work nonce (see Section 2.1.D). A chain of blocks, that is where the name “Blockchain” came from. Blocks are chained together because each new block contains the hash of the Block before it. By linking blocks together through hashing functions, the integrity of all the blocks is safe-guarded. Changing a single bit in any block will result in a totally different cryptographically linked chain of blocks.

2.1.2 Bitcoin Scripting System

Bitcoin users are assigned addresses for identification, it is basically the hash of the public

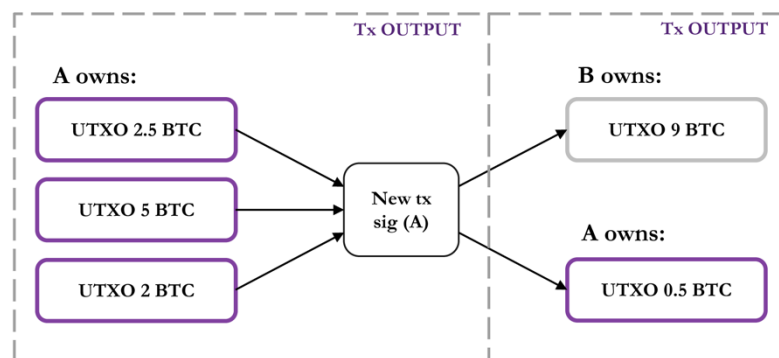


Figure 5: Unspent Transaction Output

key. Using the different Bitcoin wallets, it might seem that Bitcoin transactions happen

like normal bank transactions. However, the wallet processes Bitcoin transactions very differently in the background. Bitcoin transactions are actually scripts written in Bitcoin's scripting language called Script [20]. In order to check the validity of the transaction, the transaction script should exit with no error when executed by the validator. Bitcoin uses the model of Unspent Transaction Output (UTXO) to manipulate state. The state of Bitcoin is in specific the collection of all UTXOs, a full node has to have all UTXOs synced. If A wants to transfer 9 Bitcoins to B, A does not necessarily own 9 Bitcoins UTXO, it might own 3 UTXOs (2.5 Bitcoins, 5 Bitcoins, and 2 Bitcoins). To transfer the money, the wallet assembles the transaction in the background using the 3 UTXOs mentioned, and signs it with the private key of A (ECDSA) to make sure that only A made this transaction. The public key of B is specified in the output, and if the remaining 0.5 BTC are not redeemed, it will be counted as transaction fee and gained by the miner of the transaction. The model is called UTXO because it only uses the output of previous transactions to be consumed in new transactions and generate new UTXOs to be consumed later. All **unconsumed UTXOs** represent the state of the Bitcoin Blockchain.

2.1.3 Consensus Algorithm: Proof-of-Work

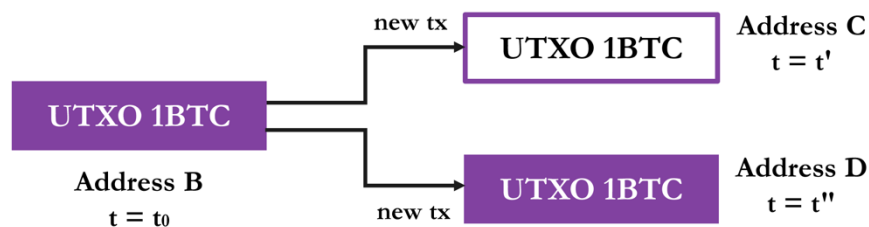


Figure 6: Double Spending Problem

To develop a digital currency, the biggest challenge is to make sure the double spending problem is solved. In simple words, if user x has 1 coin and spends it, the user should not be able to spend it again. This is not possible with normal physical currency, because when you spend it, you do not have it physically. Things in the digital world are different, a solution is needed to tell if a file of zeroes and ones representing an asset digitally is

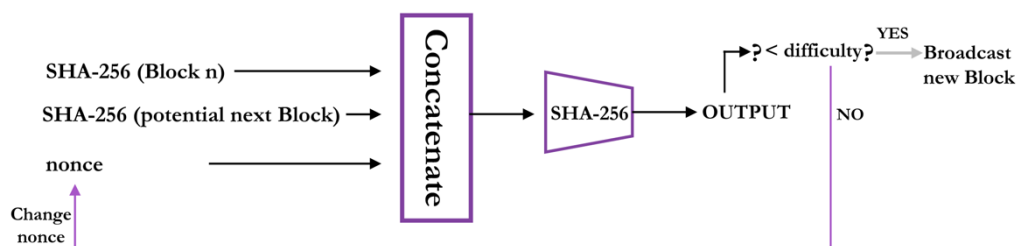


Figure 7: Mining (Proof-of-Work)

real or just a fake copy. Figure 6 shows the double-spending problem. For a coin to be spent once, the order of transactions is key. Ordering depends on the consensus mechanism adopted. In Bitcoin, Satoshi selected Proof-of-Work (PoW). A subset of peers (miners) deliberately choose to validate blocks. Miners are the ones who contribute the most in the consensus. Miner ‘A’ groups transactions into a ‘block x’ according to the timestamp of transactions and to the miner’s preference in transaction fees. Then, ‘A’ must compute a proof that involves a lot of processing (work) and compete with other miners. The miner who computes the proof first sends it to all other nodes in the network. The other nodes can quickly verify the correctness of the proof and add the block to their own copy of the Blockchain. The proof is computed using these steps:

- a) Check the latest difficulty value such as:

0x0000000000003ba27aa200b1cccaad478d2b00432346c3f1f3986da1afd33e506)

- b) Hash the current block concatenated with the potential new block, and a random value (nonce), result *OUTPUT*
- c) Compare the *OUTPUT* with the difficulty (is $OUTPUT < \text{difficulty?}$)
- d) If yes, stop and broadcast the new block with the random value and add it to the chain
- e) If not, repeat b) with a new random value until d) or until f)
- f) A broadcast received containing a new block with its nonce

The difficulty of the hash is adjusted according to the current hashing power of all nodes in such a way that one block is generated every 10 minutes.

Proof-of-Work requires a big hashing power in order to find a valid proof. Miners usually use GPUs and ASICs for mining Bitcoin. However, for non-miner nodes, verifying the proof of a block can be done on a regular processor, requiring only one hash per proof. Using cryptographic hashing as the work needed to validate a block is very powerful, because finding the proof is very hard and resource expensive, while verifying the proof is very easy and fast.

2.1.4 Economic Incentive

All miners are racing to compute the proof of the next block because there is an incentive behind it. In Bitcoin, those who work for the good of the network get a mining reward whenever they successfully mine a block and collect transaction fees for every transaction in the mined block. (See Fig. 8 [21])

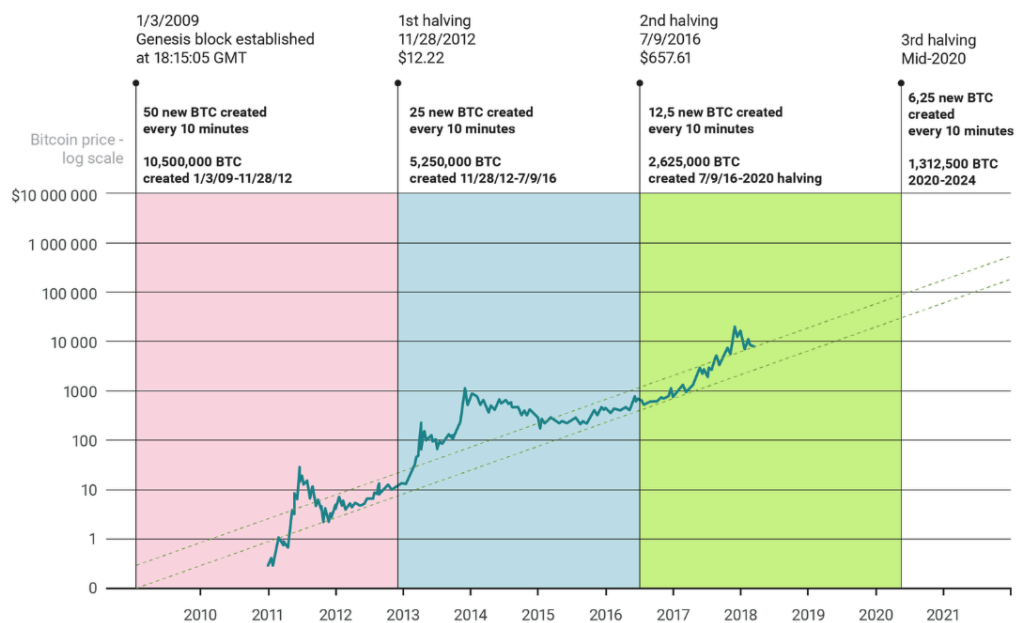


Figure 8: Bitcoin Reward Halving. Screenshot by author.

Bitcoin's supply is fixed to target 21 million Bitcoins. New Bitcoins can only be generated when mining a block. When Bitcoin first started in 2008, the reward was 50 Bitcoins per block mined. Satoshi implemented the protocol to half every 4 years [22]. As seen in Fig. 8, in 2008 the reward was 50 Bitcoins. In 2012, it halved to 25. Then, it halved again in 2016 to 12.5 Bitcoins. The reward will keep on halving until almost zero. Then, transaction fees would be the only incentive for miners. From here we understand why the process is called mining; because of the controlled supply and the increasing demand,

Bitcoin is very scarce, and its price is very high. It is often referred as digital gold, so it is logical that the name of the process by which one obtains digital gold is called mining. These decentralized ecosystems and the game theory involved in incentivizing benign participation have opened the doors to a new science of Cryptoeconomics. Indeed, Cryptoeconomics is needed in order to choose the right incentive mechanism, and thus motivate peers to cooperate and to ensure the truthfulness of the whole system. Damaging the system would mean damaging your own money. In order to break the Proof-of-Work system, a conspiracy between 51% of the mining power of the network is needed. This is not feasible however in a truly decentralized network. But many concerns have been raised about the centralization mining pools make as for hashing power [23]. This is why the cryptocurrency community is constantly searching for solutions to minimize the risks of such attacks or researching other consensus algorithms to completely replace Proof-of-Work.

2.1.5 Turing Incompleteness

Bitcoin's Scripting language is made non-turing complete [20] on purpose with no loops to avoid any mistake or vulnerability that could cause a transaction to run forever. When Bitcoin succeeded as the first functioning decentralized cash system, few proposals for different useful applications that could be useful in a decentralized network got implemented like Namecoin [24]. And because of the turing incompleteness of Script, all these proposals either started a new Blockchain or were built on top of the Bitcoin Blockchain without a good foundational layer.

2.2 Ethereum

Due to the Turing incompleteness issue, in late 2013 Vitalik Buterin proposed a new Blockchain platform [25] with a more flexible programming language to be the foundational layer of all new ideas. The ideas are implemented on this Blockchain as what is known today as Smart Contracts. A new Gas system was proposed to limit the processing on the network and put a bound on every transaction. The system was called Ethereum, and till this day, it is one of the most important existing Blockchain platforms.

2.2.1 Consensus

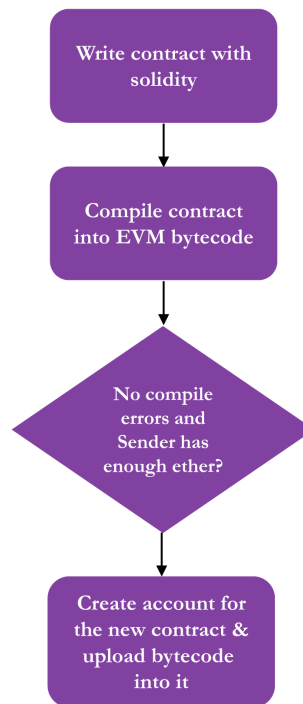


Figure 9: Ethash Steps

The current Ethereum consensus algorithm is called Ethash. Ethash is a proof-of-work algorithm as in Bitcoin, but it is made ASIC-resistant. This implies that mining with

ASICs will not add any extra benefit in Ethereum. Mining with ASICs on Bitcoin however is much better than mining with GPUs.

Though Ethash is a PoW algorithm as the consensus algorithm of Bitcoin, but the hashing process happens differently:

- First, compute the seed for the potential new block from all the previous block headers
- Second, compute a 16 MB pseudorandom cache from the seed
- Third, generate a 1 GB dataset from the cache
- Start the mining process by hashing random slices of the dataset

The 1 GB dataset will be updated once every 30000 blocks, not generated for every block mined. As in Bitcoin, the mining difficulty is adjusted according to the current hashing power of all nodes, but a block is generated every 10-20 seconds instead of every 10 minutes. Because Ethereum's consensus algorithm is a PoW algorithm, all transactions have probabilistic finality. In other words, the probability of a transaction being irreversible increases as the number of blocks chained after increases. Generally, a block in Ethereum or in Bitcoin is considered irreversible if six more blocks were committed after it. Ethereum 1.0 adopted Proof-of-Work for its consensus. However, very soon phase 0 of Ethereum 2.0 will start with a Proof-of-Stake consensus algorithm.

2.2.2 Resources

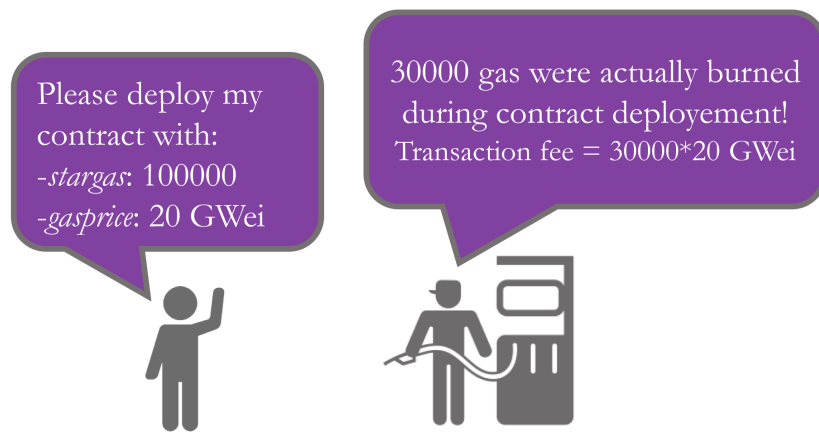


Figure 10: Gas Cost Example

Resources are protected from abuse on the Ethereum Blockchain through a Gas system as previously mentioned. As the name implies, users need to pay for the fuel necessary for their transactions to get verified by the miners. The amount of gas needed for the verification of a transaction totally depends on the operations the transaction triggers. Users need to specify two things in each transaction: *startgas*, and *gasprice*. *startgas* is the maximum amount of gas (computation steps) the sender of the transaction is willing to provide for the execution of his transaction. *gasprice* is the amount of Ethers (Ethereum Cryptocurrency) the sender of the transaction is willing to pay per one computational step (gas). The miner who validates the transaction gets a transaction fee equal to $ActualGasBurned * gasprice$ (See Fig.16). If for any reason the amount of Gas burned gets above the *startgas* specified in the transaction, the execution stops, and everything reverts back to its original state, and the miner still gets his transaction fee in order to make any attack costly. Resources in Ethereum are all monetized through the gas system; The amount of gas needed for the execution of a transaction varies according to the

resources it uses. A storing transaction needs more fuel than that only needing processing. Plus, for every byte in a transaction, there is a fee of 5 gas to also monetize bandwidth. As one can see, Ethereum's resource usage is somehow a rental mechanism. A user must pay the fees to rent the resources of the miner in order to validate the transactions. More info on Ethereum gas prices and price recommendations can be found here [26].

2.2.3 Smart Contracts & Code Execution

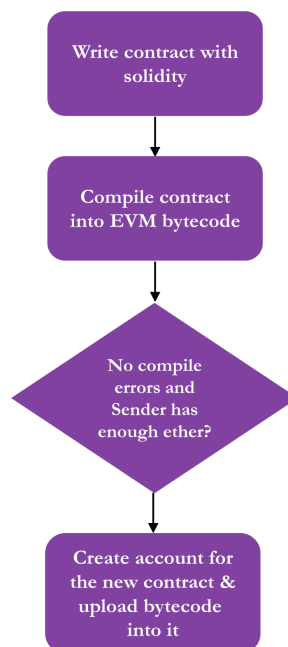


Figure 11: Deploying an Ethereum Smart Contract Steps

A Gas system cannot be implemented without a controlled environment with controlled operations. In order to achieve this over all computers which will be running Ethereum nodes and for better portability, a virtual machine implementation is a must. Ethereum implements the Ethereum Virtual Machine (EVM) which abstracts the whole code execution mechanism. The EVM language is stack-based, it has specific opcodes each requiring different amount of resource usage. Every opcode has a gas cost relative to its

resource consumption. Opcodes are encoded in bytecode in order to achieve better storage efficiency. The code running on EVM is completely isolated from the filesystem, network, and all possible processes. In order to write a smart contract, one must write the code in a high-level language (like Solidity and others). After that, the code should be compiled into the lower level bytecode and integrated in the transaction. If there are no compile errors, and the sender has enough Ether to reserve a place in the Blockchain for the code, a new contract account is created, and the bytecode is uploaded to it (See Fig.11). Solidity is a high-level language for developing Ethereum smart contracts, other languages exist but Solidity seem to be the most established. Solidity has been partially designed after ECMAScript, which makes it similar to Javascript [27]. It is also important to note that in Ethereum, identification is done through accounts, and there are two types of accounts: externally owned accounts and contract accounts. The former is the normal user account; it is identified by a public-key address and it is used to sign transactions. The second one is specific to smart contracts; it is also identified by a public-key address, plus it has the code field which holds the code of the smart contract program. In Ethereum, the term “transaction” is used when referencing signed data sent from an externally owned account, and the term “message” is used when referencing a smart contract call to another smart contract. If a transaction triggers the execution of smart contract A and A triggers smart contract B, the *startgas* specified in the transaction should be enough to execute A and everything A calls. Once a transaction is constructed from a local node, the node broadcasts this transaction to the entire network. Mining nodes have a transaction pool where a finite number of the broadcasted transactions get stored according to their

gasprice. Most miners assemble the highest paid transactions in a block and start solving the PoW puzzle. Once a miner succeeds to compute the proof, it broadcasts the block along with the proof to the whole network. Other nodes upon receiving this block execute all the transactions in this block, and thus sync their Blockchain accordingly.

2.2.4 Storage

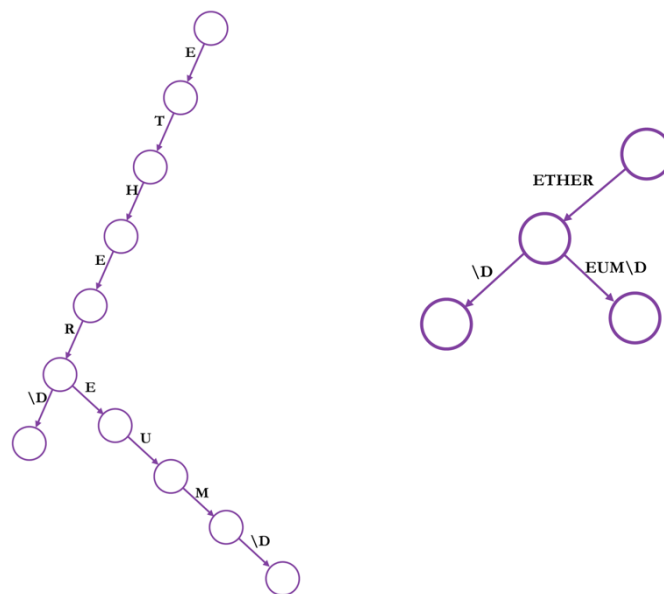


Figure 12: Storing two words in the same trie (Ether & Ethereum), to the left is a standard trie and to the right is a PATRICIA trie

Though all blockchains are basically similar to Bitcoin's Blockchain structure, Ethereum's Blockchain structure is a bit different. Ethereum utilizes Patricia Tries (*Practical Algorithm to Retrieve Information Coded in Alphanumeric*) for its persistent storage (See Fig. 12). In order to store the tries, the geth client uses Levelldb as its database software. Levelldb is a high speed google key/value store.

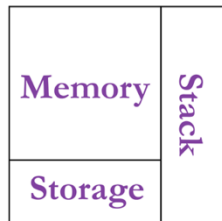


Figure 13: Smart Contract Storage Resources

There are four types of tries in Ethereum:

- Transaction Tries: every block contains a transaction trie storing its transactions.
- Transaction Receipt Tries: every block contains a transaction receipt trie which store the outcome of the transactions.
- Storage Tries: every account has its data stored in its own storage trie.
- State Trie: One and only state trie in Ethereum. It has the address of each account as a 'path' and the value is the RLP (Recursive Length Prefix) encoding of these four fields: nonce, balance, storageRoot, codeHash.

In every block header, the root hashes of the block's transaction trie, transaction receipt's trie, and state trie are included.

There are three storage resources utilized by every smart contract (See Fig. 13):

- Storage: Non-volatile memory storing the executable code of the contract, the Ether balance, persistent state variables and local variables. It is expensive to use. This is the storage used in the storage trie mentioned. The storage is a key/value store that maps 256-bit words to 256-bit words.
- Memory: Temporary infinitely expandable array to hold temporary variables, arrays, structures and function arguments. It gets reset after the execution ends. Of course, the cost of expansion must be paid in gas.

- **Stack:** Temporary non-modifiable call-stack where the EVM code execution happens and can only hold 1024 256-bit words. It gets reset after the execution ends.

2.2.5 Economic Incentive

Though Ethereum and Bitcoin have the same concept of mining rewards and transaction fees, some prefer mining Ethereum over mining Bitcoin. One could mine ~60 Ethereum blocks for every 1 Bitcoin block. This is due to the difficulty of mining which adapts in a way a block is generated every 10-20 seconds in Ethereum and every 10 minutes in Bitcoin. Also, because of the very easy to learn smart contract language, many people are more drawn to Ethereum and believe in its bright future. In addition, Ethereum incentivizes individual miners with not much hashing power to participate in the mining process by giving rewards to uncle blocks miners. Uncle blocks are generated when two valid blocks are mined simultaneously. One gets accepted as a valid block and the other becomes an uncle block. The miner of the uncle block receives 75% of the block reward. This not only incentivizes individual miners, but also increases the security of the Ethereum Blockchain by increasing the amount of work done on the main chain.

2.3 EOS.IO

Ethereum have added big things to the Blockchain community. But, because of several incidents, and scalability issues, many tried to start an alternative and more scalable decentralized platform that can compete with centralized platforms already implemented. EOS is the one of the most popular decentralized platforms and as mentioned before, it

received a funding of 18 million dollars. It was founded by Dan Larimer in 2018. EOS promised vertical and horizontal scaling to the Blockchain community.

2.3.1 Consensus

EOS uses not the well-known proof-of-work consensus, rather it uses a variation of Proof-of-stake algorithm known as Delegated Proof-of-Stake (DPoS). Plus, EOS adds another layer to the consensus known as the Asynchronous Byzantine Fault Tolerance layer (aBFT). Thus, the EOS consensus is composed of two layers: DPoS and aBFT [28]. The DPoS layer regulates all the processes related to staking tokens, voting, vote decay, vote recording, producer ranking, and inflation pay. On the other hand, the aBFT layer determines the finality of each block.

DPoS

Unlike PoW, block generators in PoS are called validators. In order for a node to become a validator, it has to stake a certain amount of coins. The more is the amount of coins staked, the higher the chance is to be selected as the next block validator. If the validator generates a faulty block, a part of its coin stake is lost. DPoS is a variation of PoS where 21 delegates are elected to be the block validators through a real-time voting process. The block validators in EOS are called producers. Token holders have to stake their tokens in order to vote for their preferred DPoS delegates. The more a user stakes, the more its vote has more weight. Also, the voting weight is base-2 exponentially proportional to the date January 1,2000. This makes the voting weight increase with time for the same number of tokens staked. Every stakeholder can vote for up to 30 block producers in one voting

round. Every voting round, the top 21 producers are selected as active producers and the others will be placed in a standby list (See Fig. 14 [29]). Votes are not recounted from zero every round but rather they are increased with the new votes. So, old votes are kept,

Name	Url	Votes	%	Daily Reward
eoshuobipool	https://www.eoshuobipool.com	321,076,424	2.71 %	897 EOS
eosnationfw	https://eosnation.io	320,908,689	2.71 %	897 EOS
eoseoul dotio	https://www.eoseoul.io	320,195,112	2.70 %	896 EOS
newdex.bp	https://newpool.io	319,719,489	2.70 %	895 EOS
big.one	https://eos.big.one	319,648,836	2.70 %	895 EOS
eosinfstones	http://infinitystones.io	319,436,448	2.70 %	894 EOS
atticlab eosb	https://atticlab.net	319,210,076	2.70 %	894 EOS
bitfinex eos1	https://www.bitfinex.com	318,972,494	2.69 %	894 EOS
blockpool eos	https://www.blockpool.com	318,869,803	2.69 %	893 EOS
starteosiobp	https://www.starteos.io	318,838,004	2.69 %	893 EOS

Figure 14: Top Producers on the main EOS network. Screenshot by author.

but the weight of the vote is replaced by the new voting weight of each voter. If the voting weight of a user increases, the votes of the user held by any producer are decayed, this is done in order to encourage more users to join in and participate in the voting and to give more weight for their newer choice of delegate. The order of the production order of the 21 producers is the alphabetical order of the elected producer names. Regardless of the voting ranking, all 21 producers have equal producing power. The process of producing blocks happens in rounds. Each round is 126 seconds composed of 21 time slots, where each producer gets a time slot of 6 seconds. Every producer is meant to produce 12 blocks in one time slot, so 2 blocks/second. If for any reason the producer did not produce during the time slot, which results in a gap in the blockchain, the producer is placed in the standby list. In order to prevent overloading the network with false transactions, all producing nodes re-validate transactions upon receiving them, so a false transaction would get dropped. From the very first block in a round, all producers receive the list of the

producers of the next scheduled round. When this block becomes irreversible (check aBFT), the proposed schedule becomes active in the next round. In each round, blocks pass through three phases: production, validation, and finality.

aBFT

This layer determines which blocks become final (irreversible) out of the ones synced between all the elected producers. It consists of two stages: first a block is proposed as a last irreversible block (LIB), and secondly the block is confirmed as final. The aBFT layer ensures algorithmic finality by verifying that the super majority of producers authorized for this scheduled session agree on this block. The super majority is considered to be 15 producers ($2/3 + 1$).

2.3.2 Resources

There are three system resources available on the EOSIO Blockchain:

- RAM: acts as the permanent space/storage of all data. It is a scarce resource and needs to be purchased. Its price is set according to the Bancor Liquidity algorithm [30].
- CPU: represents the processing time of actions available for a user to interact with contracts, it is measured in microseconds. In order to obtain CPU bandwidth on EOSIO, a user must stake tokens.
- NET: represents the amount in bytes of transactions that is available for a user to interact with contracts. As the CPU resource, one must stake tokens in order to obtain net bandwidth.

The resources in EOS are based on an ownership model unlike Ethereum. The user owns the CPU, Net, RAM provided and does not need to pay transaction fees or rent the resources as in the Ethereum's gas system. The clients of a business running on the Blockchain do not pay for the use of Blockchain because of the Receiver-pays model that EOS uses. The owners of a website do not make customers pay for visiting their website in order to cover hosting costs. In a similar way, developers pay for the best amount of bandwidth, CPU, and storage their application needs. Users use the applications for free.

2.3.3 Smart Contracts & Code Execution

Smart contracts in EOS are written in C++ as an `eosio::contract` class. The smart contracts are then compiled into lower level Web Assembly bytecode (Wasm) to run over the EOS virtual machine. Wasm is not developed by EOS but rather it has its own community improving it constantly. This removes the load of developing the virtual machine language from scratch and leaves it to the experts of the well-known and robust Wasm engines. Smart Contract functions are known in EOSIO as actions. A transaction could carry not one but a group of actions in an EOSIO DApp, the success of all the actions is a must for the success of the whole transaction. In case of a transaction failure, the blockchain state is reverted back to its state prior to the transaction processing. Participants in the EOSIO Blockchain and smart contracts are not identified by the public key addresses but rather by alphanumeric names of 12 characters max. Beside the account name, other variables are held for each participant in an account schema, such as RAM usage, CPU/NET limits, voting information... There are two types of actions in EOSIO:

- Explicit Actions: the regular actions which are included in a transaction and are synced throughout the network.
- Implicit (inline) Actions: the actions that happen implicitly and do not get included in a transaction. Implicit actions are called by explicit actions.

The EOSIO smart contracts do not interact with transactions, transactions are only at the application level. The incoming blocks and transactions are forwarded to the chain controller module. The chain controller is responsible for the execution on the local chain.

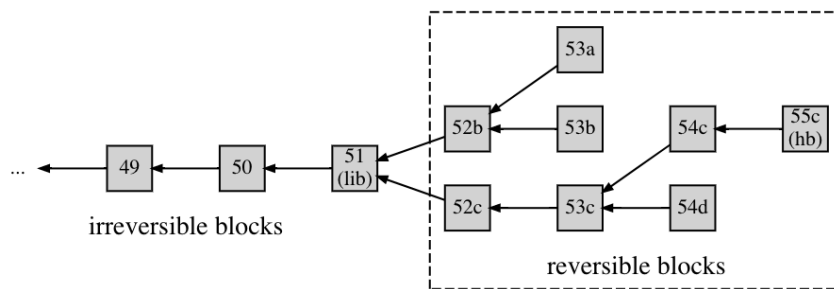


Figure 15: EOS Local Chain Example. Screenshot by author.

The local chain contains both the irreversible (immutable) blocks, and the reversible blocks. The reversible blocks are managed by the Fork Database (See Fig. 15 [31]). The Fork database is an internal part of the chain controller. The Chain controller relays all new blocks to the Fork Database which produces temporary mini forks until the LIB block advances, after the LIB advances, the invalid forks are purged.

EOSIO System Contracts

The core Blockchain features of EOSIO are all implemented in the EOSIO system contracts and thus can be easily modified depending on use case requirements.

At the genesis stage, the only account existing is a system account named eosio. Eosio then creates other system accounts, where some belong to system contracts (such as

eosio.token, eosio.msig) and others are just regular system accounts (such as eosio.prods, eosio.ramfee).

Account	Description
eosio	It contains the eosio.system contract and is the main account created at the genesis of the Blockchain
eosio.msig	It contains the eosio.msig contract which allows multi-sig transactions
eosio.wrap	It contains the eosio.wrap contract which simplifies superuser actions of block producers
eosio.token	It contains the eosio.token contract which creates, issues, and manages all tokens on the EOSIO Blockchain
eosio.names	It holds funds from namespace auctions
eosio.bpay	It pays block producers which produced blocks
eosio.prods	It holds the union of all current active block producers' permissions
eosio.ram	It keeps track of all bought/sold SYS tokens of RAM
eosio.ramfee	It holds the fees from trading RAM
eosio.saving	It holds 4% of network inflation
eosio.stake	It keeps track of all staked SYS tokens of CPU and NET
eosio.vpay	It pays block producers according to their received votes
eosio.rex	It keeps track of fees from REX related actions

Table 1: EOS.IO System Contracts and System Accounts

For transactions to get executed, a chain database session is started, and a snapshot is taken in order to be able to revert back in case of failures. A transaction context is generated in order to record the transaction state during execution. For every action, an action receipt and an action trace objects are generated. The action trace allows the action to be traced back to its actual transaction and block. After all the action receipts have been generated, the transaction receipt is generated and pushed to the block.

2.3.4 Storage

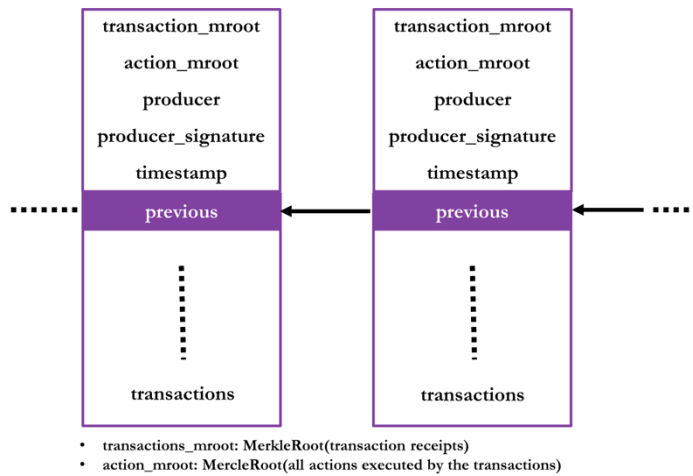


Figure 16: EOS Blockchain Structure

As for Blockchain storage, the chain plugin is responsible for aggregating data into the EOSIO Blockchain. The Blockchain structure in EOSIO is very close to the standard Blockchain structure (See Fig. 16). As for smart contract storage, the user should purchase RAM. EOS uses RAM as its storage (not disk) to achieve very fast read/writes. The data in each smart contract can be organized by the developer into multi-index tables and specifying the structure of each table. EOSIO developers wrote their own version of the Boost Multi-Index [32]. It is basically a C++ struct with multiple indexes. The approach of multi-indexing is very known in relational databases. The Boost Multi-index library borrows this approach in order to make more complex data structures similar to multiple indexed tables.

2.3.5 Economic Incentive

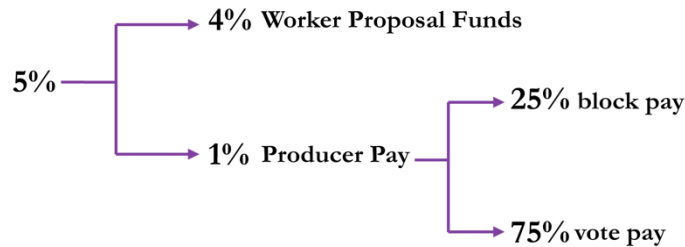


Figure 17: EOS Inflation Distribution

As in any Blockchain system, the block producers are incentivized by the token rewards. In EOS, there is no supply limit as in Bitcoin. The DPOS EOS system is based on inflation to incentivize block producers. Inflation refers to the issuance of new tokens into the circulating supply. Because there are no transaction fees, producers get paid by the internal inflation system which is 5% annually. A percentage of 4% of the newly minted tokens goes to worker proposal funds. The 1% left goes to producer pay. Not only the active producers get rewarded from the producer pay (25% of the producer pay), but also standby producers get rewarded according to the percentage of votes received (75% of the producer pay, see Fig. 17).

2.3.6 Permissions

The actions of an account in EOSIO must be authorized by each account's permissions. There is a permission structure that keeps record of the list of hierarchical named permissions, and each named permission is linked to an authority table. The authority table linked to each permission contains the factors that must be satisfied in order to authorize the action. The highest permission in the permission structure of each account

is the owner permission, which is used for recovering in case a key with lower permission has been compromised. The second highest permission is the active permission which is used for actions. More custom permissions can be created for other purposes such as friend's permission.

2.4 Other Platforms

In this section, other Blockchain development platforms are briefly presented.

2.4.1 Blockstack

Blockstack [33] was co-founded by Muneeb Ali and Ryan Shea in 2013. It is a decentralized computing platform totally different from Ethereum and forms a decent competition with it. Blockstack is well-known for its virtualchain technology, server-less applications, and decentralized naming system. Blockstack by itself is shaping the meaning of decentralization in a totally different manner than any other Blockchain system. It aims at re-decentralizing the Internet. The main reason behind the scalability issues in the existing Blockchain systems according to Blockstack, is that there is too much focus on the Blockchain by itself, the programs and their data are living on the Blockchain, and not so much concern is given outside of the Blockchain. Blockstack's perspective is to remove everything heavy from the Blockchain and keep it only for its key purpose, an immutable log of operations. Blockstack's team had scalability in mind since first started designing the system. Developers working with decentralized Apps on Blockstack have the ability to easily make them server-less, scalable, and they do not have to worry about authentication because it is built-in with the Blockstack Blockchain naming system (BNS). Not only that, but also developers do not need to worry about

storing user data. With Blockstack's decentralized Gaia Storage System, users bring their own cloud storage, each App saves user data in the user's cloud. Data can be stored encrypted in the user's cloud storage; their perspective is to treat the cloud as dumb hard drives that only store user's data.

Blockstack is composed of 4 layers [34]:

- Blockchain Layer
- Virtual chain Layer
- Discovery Layer
- Storage Layer

Control Plane

The lowest layer in their architecture is the Blockchain Layer. The Blockchain used in Blockstack is Bitcoin [35]. The team first built their architecture over Namecoin [35] but they faced some unusual and ambiguous behaviors. So, they migrated their system onto the first and most fault-tolerant Blockchain ever: The Bitcoin's Blockchain. This was the first successful cross-chain migration of a system.

Blockstack layers only refer to the Blockchain for reading and writing totally-ordered operations. Every other complex issue related to the Blockchain by itself is abstracted to the other Blockstack layers like the mining process and others... Though interacting with the Blockchain is abstracted, the Blockchain consists the basis of trust in this system, trust is bootstrapped up from it to all other layers.

Migrating from one Blockchain to another is much facilitated with the existence of Virtualchain layer. As virtual machines can run any operating system on top of any other

operating system or physical machine, virtualchains construct a somehow lighter chain on top of an existing chain. The state machine's new operations are introduced to the original Blockchain without the need of changing it or forking it. The virtualchain is only concerned with the transactions that carry BNS operations in their OP_RETURN field in Bitcoin (a field for metadata). However, virtualchain can be configured to work on top of any other Blockchain. Blockstack chose Bitcoin for their BNS (Blockchain Naming System) after trying it over Namecoin because they noticed several weaknesses and possible selfish-mining signs [35].

The biggest problem a chain could face is a fork, and the integrity of user's data in applications built on top of this chain need to be sustained. By building state machines on top of the abstracted Blockchain and securing it with a more efficient hash (called Consensus Hash), a higher-level consensus is achieved (called application-level consensus by the Blockstack team).

Data Plane

The discovery layer in Blockstack is used to separate the routes to data from the storage of data itself in order to allow the use of multiple storage services. The discovery layer in Blockstack uses a peer-to-peer network to store routes to storage files in form of zone files globally. Users do not need to trust the P2P network as they can verify the hash of the data record in the Blockchain. Blockstack team implemented their own enhanced peer-to-peer network called the Atlas network.

The storage layer consists of Blockstack's decentralized storage system: Gaia.

The state of BNS is realized and agreed upon in the control plane, the route of the data hashed in the state is present in the Atlas zone files, and the real data is stored signed and encrypted by choice of the user in his/her preferred cloud storage provider. This data represents data associated with a specific name.

Pricing functions are available to control the name registration process, and to start the first namespace of BNS, the Blockstack team paid the fee according to their pricing function (about 10 000\$ at the time), enforcing their view that true decentralization is achieved when also developers follow the rules of the Blockchain. This commitment to fairness and transparency is not found in all Blockchain communities.

2.4.2 Hyperledger Fabric

The previous platforms mentioned are either public Blockchains or based on a public Blockchain. Seeing the advantages of decentralized computing, several businesses were interested in benefitting from this technology. In result of that, the Linux Foundation started in 2015 the open-source Hyperledger projects [36], in aim of making Distributed Ledger Technologies (DLT) for businesses. The Digital Asset and IBM were the main contributors in one of their famous projects, the Hyperledger Fabric (HF). The Hyperledger Fabric [37] is one of the most important permissioned Blockchains out there. Permissioned Blockchains, in contrary to public Blockchains, not everyone can run the node and become part of the network. Only a set of members are authenticated to participate in a permissioned Blockchain network. Hyperledger fabric is based on channels between organizations. It has no cryptocurrency, but a token or a currency can be implemented. Every channel has a shared ledger, and it provides private, confidential

transactions between its members. The shared ledger is programmable through what is called “chaincode”, the code of the chain. The Hyperledger community find this name more meaningful than “smart contracts”. Authenticating the participation in the network is managed by CAs (Fabric-CA or any CA of choice) and Membership Service Providers (MSPs). MSPs are used to somehow abstract the process of obtaining cryptographic certificates from CAs in the point of view of an organization’s employees. They grant the users in an organization different roles and different access privileges depending on their

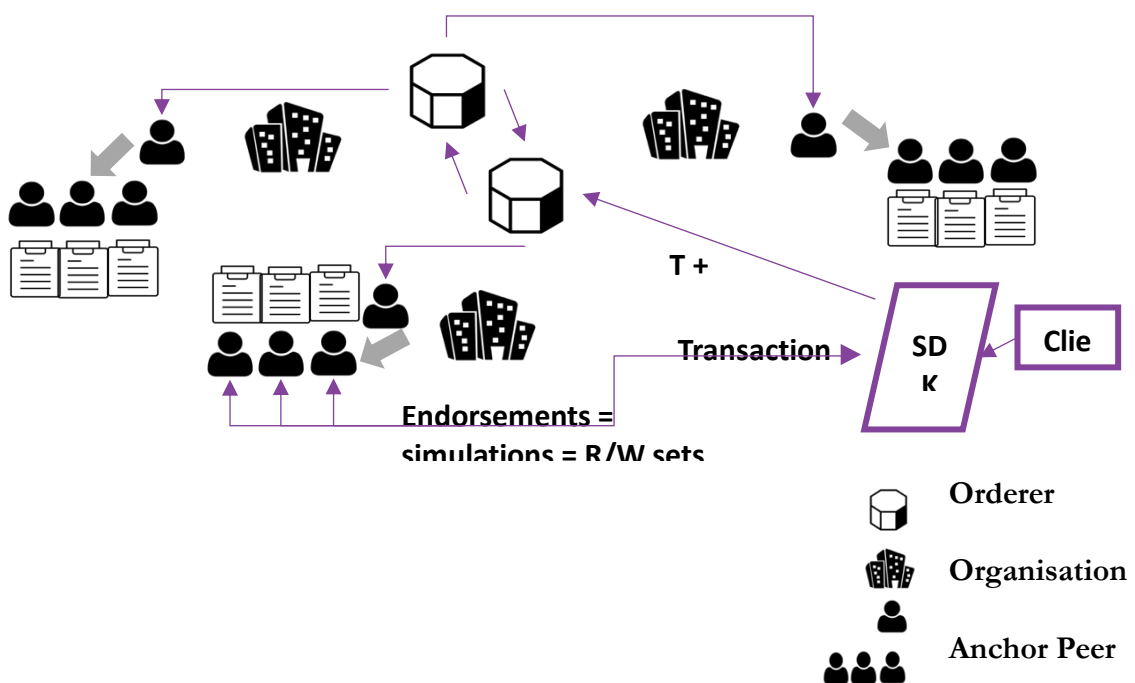


Figure 18: Kafka Consensus

role. Because the Hyperledger Fabric is providing the Blockchain for businesses, assets of organizations need to be defined in an HF channel as JSON files. The shared ledger consists of two important databases: the chain of cryptographically linked blocks (immutable transactional log), and the state database (representing all latest values and versions of assets, it can be recovered from the chain at any time).

There are three types of nodes in HF:

- Client (end-user that connects to the channel through a peer)
- Peer (normal peer who replicates the ledger or endorsing peer who in addition to his role endorses transactions)
- Orderer (ordering-service node OSN)

The consensus in HF is not composed of a single election-like algorithm (like PoW mining), but it is achieved by a series of sub-processes. The transactions are first invoked by clients and sent to a certain number of peers (based on the number of endorsers requested by the endorsement rules specified in the chaincode). After that, the endorsing peers simulate the transaction execution based on the current state of the ledger replica they have and generate Read/Write sets (endorsements). The client receives the endorsement of his/her transaction and sends it to the orderers. The transactions enter a queue when received by the orderer. The total-ordering of transactions is the most important goal of the consensus in a Blockchain. In HF, the orderers are responsible for this step. The orderers agree on the transactions based on the endorsements of peers, all endorsements must be the exact same set so that the orderers accept the transaction. If the orderer accepts the transaction, it exchanges the transaction with other orderer nodes through Apache Kafka. Apache Kafka [38] is a high-performance publish/subscribe messaging system and it is highly used in distributed systems. Each channel has a separate Kafka partition where orderers exchange transactions they accept. Here, two parameters can be changed in this step that affect the block generation process, the BatchSize and BatchTimeout. BatchSize consists the number of transactions in a Block, while

BatchTimeout is a timer that upon its expiration a Block should be generated. Finally, the ordered block is broadcasted by OSNs to peers who commit the block to their ledger.

Both Blockstack and Hyperledger Fabric are undoubtedly very innovative Blockchain systems. Both however have not been included in this study.

Hyperledger Fabric has not been included in this study as it is a permissioned Blockchain. It was involved however to enforce the view of how different it is from public Blockchains and thus, its performance should not be compared to the studied Blockchains.

Blockstack was part of the comparison, but it then got excluded for the following reasons:

- The computing done on Blockstack is not on-chain but rather client-side. On-chain computation is under development and their smart contracts language named “Clarity” is still in its beta version.
- Running a Blockstack network would require running a Bitcoin network in parallel, which is a big difference in the hierarchy of the implementation and thus comparing Ethereum with EOS would make much more sense.

CHAPTER 3

METHODOLOGY & SIMULATION

In this chapter, we discuss our performance methodology and experimentation in detail. Our main objective is to help developers get an idea of developing a DApp on Ethereum vs Developing a DApp on EOS, by giving an insight of the timing of the basic operations in each. Any performance comparison is meant to give an overview of the two compared systems, test out some basic operations, and distinguish between them.

In an attempt to compare two platforms, the implementation of the two simulations must be as similar as possible.

Note that in this experimentation, we do not consider block finality.

The Blockchain tools used in this study are:

- EOS.IO Tools:
 - Nodeos: The EOSIO core node
 - Cleos: The EOSIO command line interface that connects to a nodeos daemon and manages wallets
 - Keosd: The EOSIO keys storage
 - EOSIO.CDT: The EOSIO Smart Contract Development Kit
 - Eosjs: Nodejs library for DApp development
- Ethereum Tools:
 - Geth: The Ethereum node written in go
 - Solc: Solidity compiler
 - Web3: Nodejs library for Ethereum DApp development

3.1 Testing Environment

The performance comparison was studied on both Azure cloud and AUB HPC Octopus cluster. A portable and consistent testing environment was produced using Singularity containers, making the whole migration process mess free.

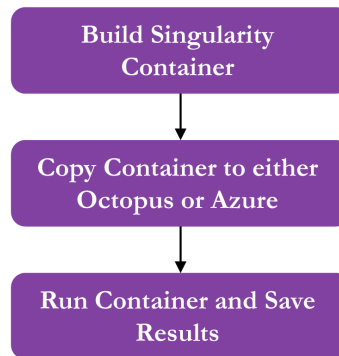


Figure 19: Singularity Containers

Two singularity containers were created, one for Ethereum and the other for EOS. To run the Ethereum or EOS experiment, start the run script of the corresponding singularity container.

3.2 Singularity Containers

Singularity containers [39] give mobility of compute with its image file format, able to move to any OS and run there as a lightweight container. It encapsulates all the software stack. Singularity is very known in HPC world, because though its competitor docker is very known and powerful, docker requires root access to run containers. And that is why HPC administrators do not allow it, and instead allow singularity containers. Singularity was developed by a collaboration of HPC admins and research scientist in aim to develop lightweight, portable, and reproducible environments. In this study, Singularity version 3.5.2 was used.

3.3 AUB HPC OCTOPUS Cluster

The American University of Beirut offers for its student a High-Performance Computing cluster named Octopus. Octopus is a mixed architecture Intel/AMD Beowulf virtualized cluster. It has 472 virtual CPUs of which 328 are AMD EPYC 7551p vCPUs logical cores and the others are Intel Xeon E5-2695 v4 vCPU logical cores. Octopus nodes operate on a Centos 7 Linux Operating System with a subset of the OpenHPC software stack. Slurm is the scheduler used in Octopus. The Octopus node used in this study belonged to the large partition with 64 vcpus and 256 GB RAM. More information on Octopus can be found here [40].

3.4 Azure HB-Series

In aim of checking if a cloud platform will influence the results, Azure HB-series has been selected in order to compare with Octopus. The HB-series uses 60 physical processor cores of type AMD EPYC 7551, which is the same type used in the Octopus nodes which we tested. As for RAM, the HB-series offer 240 GB of RAM. As seen, the HB-series VM was selected because of the closeness of its specs to the Octopus nodes in CPU and RAM (See Table 2).

	Octopus	Azure HB-series
Processor Type	AMD EPYC 7551p	AMD EPYC 7551
Type of Cores	Virtual	Physical
# of Cores	64	60
RAM	256 GB	240 GB

Table 2: Octopus vs Azure HB-series Specs

3.5 Network Structure

Many samples were generated out of each Blockchain network, each was randomly connected, but all were connected as a Minimum Spanning Tree with a maximum degree of 6. The network algorithm was written in Nodejs. It starts by generating a random tree of N nodes. Then, the MST of the random tree is calculated. Two Nodejs packages are used: ‘random-tree’ [41] and ‘js-graph-algorithms’ [42]. After that, the nodes are connected statically in bash according to the network structure generated from Node.js. For Ethereum nodes to be connected statically, one must use the “enode format” of each node. The enode of a node identifies it in a form of a URL. More information on the enode format can be found here [43].

3.6 Bash Scripting

For long years, Bash has been used in various Linux distributions and in Apple’s macOS. It is widely used amongst developers and it is a very powerful language. Many may argue the need of replacing Bash with a more elegant language such as Python, but Bash is very robust and can finish some things with a few lines of code compared to other languages. Bash has been selected to be used in this study along with Node.js in order to develop the experiments’ scripts. Most of the work was done in bash, except the DApps and the network algorithm were written in Node.js.

3.7 Node.js promises and DApps

Though only 10 years old, Node.js has killed it with over a billion downloads [44]. It is a very fast, lightweight, and cross-platform Javascript runtime environment, allowing Javascript to run both client and server side. It surely deserves the attention it got, along

with its Node Package Manager (NPM) offering lots of ready-to-use tools and modules to include in the developed applications. Mostly all Blockchain platforms offer their DApp development tools and modules in Node.js, and that is why it was selected for the Ethereum and EOS DApps in this study. Node.js is very known with its promise objects. Promises are basically asynchronous structures that could get resolved or get rejected. The functions in the Ethereum Node.js package ‘web3’ [45, p. 3] and in the EOS Node.js package ‘eosjs’ [46] are all implemented as promises. In order to time the promises, the ‘timely’ package [47] was used with its method to time promises named ‘timely.promise’.

3.8 Samples

There are four testing networks: 10, 50, 100, 150. In each network, the nodes are connected randomly 30 different times through the network algorithm, in order to have 30 samples of each network. On Octopus or on Azure each random sample is run

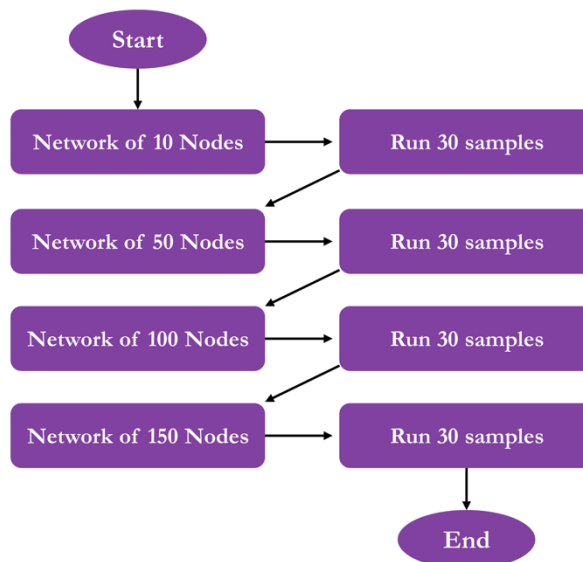


Figure 20: Singularity Run Script

individually, executing the DApp and recording the results before moving on to the next sample.

We increased the number of samples to 30 to have a better generalization of the final results. Fig. 20 shows what the Blockchain container in Fig 19 does when executed. Note that for each sample in Fig. 20, a random network connectivity is achieved according to the network algorithm.

3.9 Performance Metrics

While varying the number of transactions submitted simultaneously from the same node and varying the number of nodes in the network, the user response time, the CPU usage of each Blockchain node instance, and the totally used memory megabytes were measured.

The study is mostly focused on the user response time of the basic smart contract operations triggered from a DApp. The five operations that were studied on both platforms are:

1. Depositing native tokens into the smart contract
2. Withdrawing native tokens from the smart contract
3. Reading the native token balance of the smart contract
4. Storing data in the contract (100 bytes)
5. Reading data from the contract (100 bytes)

These operations represent the basic kind of interaction with a Blockchain node.

On the other hand, the number of transactions submitted from the node running the DApp (Node 5) was varied between 1, 10, 50, 100 and 150. The multiple transactions were all

submitted in batches at the same time. So, in the case of the 100 transactions, all 100 transactions were submitted at the same time, and when all transactions get a response, the response time is recorded. This indicates the transaction throughput.

In addition, the CPU usage of each Blockchain node has been measured with 'ps' unix command along with the totally used memory megabytes caught from 'node-os-utils' Node.js module in order to give an insight about cpu/memory usage of each platform.

The following flowchart summarizes the steps executed by each DApp:

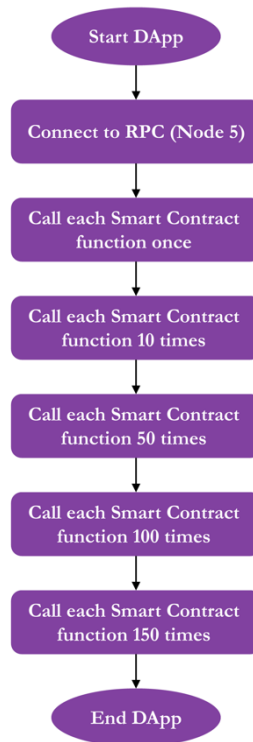


Figure 21: DApp Steps

Both platforms experiments were implemented following the steps in the figure below.

Each of the 30 samples in Fig. 20 performs the steps in Fig. 22.

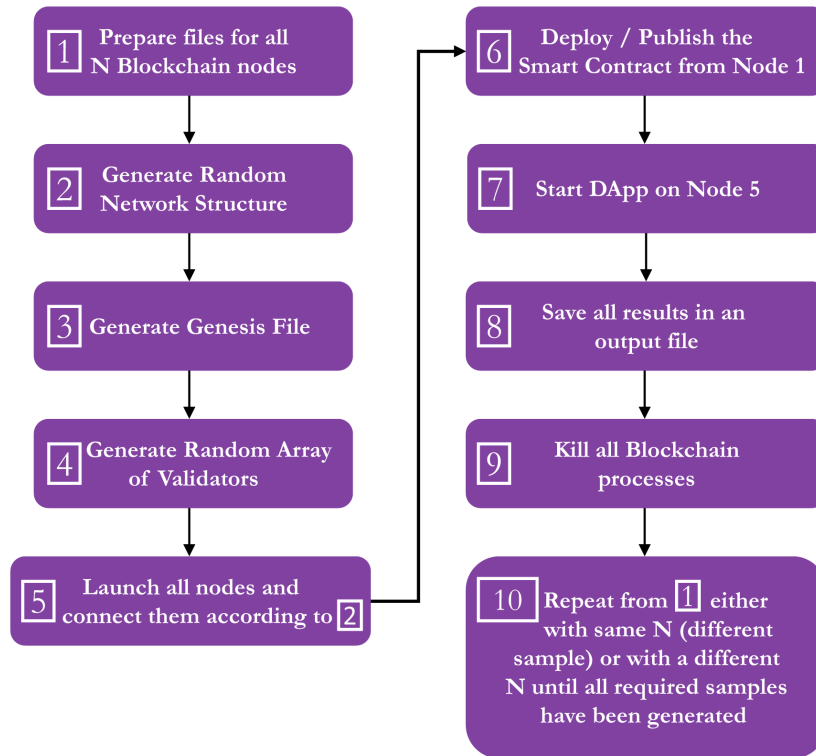


Figure 22: General Steps of the Performance Experiment

3.10 Testing Assumptions

- Mining/Producing Percentage: 25% of all nodes in the network (In EOS only a maximum of 21 producers are chosen for the actual block producing process, but votes are casted for 25%)
- Maximum number of peers: 6

3.11 Details of the Ethereum Setup

The Ethereum smart contract has been written in a few lines of solidity v0.4.25. The smart contract is then compiled once and set aside for use after the network launch.

A script written in bash launched the testing network. The script steps are:

1. Clean any residue file or processes from previous sample
2. Generate the random network structure by running the Node.js file "generate-mst.js". It also generates a Graphviz file in order to visualize the network later.
3. Input generated random tree back into the bash script in the form of an associative array where the keys are all the nodes and the value for each is a string of connected nodes.
4. Generate the genesis file of the testing Blockchain and allocating Ethers for all testing accounts
5. Prepare a directory for each of the N geth nodes and generate required testing accounts on each of the nodes.
6. Launch all nodes according to the randomly generated array of 25% miners.
7. Connect nodes statically according to the associative array generated in step 3. To achieve that in Ethereum, the following method was chosen:
 - a. Run the written "getenode.js" file which connects the RPC of each node and saves its enode in the corresponding geth node directory as "enode.txt".
 - b. Kill all the geth nodes processes
 - c. Loop over the associative array from step 3.
 - d. For each key, get all the enodes of the connected geth nodes and insert them in static-nodes.json file in the key geth directory.
 - e. Restart all geth nodes

8. Deploy the Ethereum smart contract from Node 1.
9. In order to call this smart contract from the DApp in the following steps, the address of the deployed smart contract had to be saved in an environment variable.
10. Run the DApp and save the results. The DApp steps are presented in Fig. 20.
11. Draw the graph of the network structure of this sample using the Graphviz software [48].

3.12 Details of the EOS Setup

The script steps follows the BIOS Boot Sequence tutorial [49] steps to launch your own EOS.IO Blockchain. The script does the following steps:

1. Clean any residue file or processes from previous sample
2. Generate the random network structure by running the Node.js file "generate-mst.js". It also generates a Graphviz file in order to visualize the network later.
3. Input generated random tree back into the bash script in the form of an associative array where the keys are all the nodes and the value for each is a string of connected nodes.
4. Start Cleos wallet and create public/private key pairs for all testing accounts
5. Generate the genesis file of the testing Blockchain
6. Prepare a directory for each of the N nodeos nodes which contains the start command that connects each node to its peers according to the associative array in step 3
7. Launch the genesis node "eosio"

8. Create system accounts and setup system contracts such as eosio.token and eosio.msig
9. Create system token “SYS” and allocate tokens for eosio
10. Create all testing accounts and allocate for each one CPU, NET, and RAM.
More resources were allocated for all producers’ accounts.
11. Register producers according to the randomly generated array of 25% producers and launch all nodes
12. Vote for producers
13. Publish the developed smart contract
14. Resign eosio and system accounts
15. Run the DApp and save the results. The DApp steps are presented in Fig. 20.
16. Draw the graph of the network structure of this sample using the Graphviz software [48].

Note that according to this answer [50] on the official EOSIO github repository, the purchased RAM should be $10 * \text{wasm size}$. Way more RAM than the recommended amount was allocated in the implementation.

Table 3 summarizes and explains the DApp functions’ notations and purposes which are used in Chapter 4.

Deposit	Issue a transaction calling the Deposit function of the smart contract
Withdraw	Issue a transaction calling the Withdraw function of the smart contract
ReadBalance	Locally call the Blockchain node's database to return the balance of an account
StoreInContract	Issue a transaction calling the Storing function of the smart contract to store 100 bytes
ReadFromContract	Locally call the Blockchain node's database to read the data stored in the smart contract (100 bytes)
Deposit10	The Deposit function of the smart contract is called 10 times for 10 different accounts (batch of 10 transactions)
Withdraw10	The Withdraw function of the smart contract is called 10 times for 10 different accounts (batch of 10 transactions)
ReadBalance10	The ReadBalance function of the smart contract is called 10 times for 10 different accounts (batch of 10 local reads)
StoreInContract10	The StoreInContract function of the smart contract is called 10 times for 10 different accounts (batch of 10 transactions)
ReadFromContract10	The ReadFromContract function of the smart contract is called 10 times for 10 different accounts (batch of 10 local reads)
Deposit50	The Deposit function of the smart contract is called 50 times for 50 different accounts (batch of 50 transactions)
Withdraw50	The Withdraw function of the smart contract is called 50 times for 50 different accounts (batch of 50 transactions)
ReadBalance50	The ReadBalance function of the smart contract is called 50 times for 50 different accounts (batch of 50 local reads)
StoreInContract50	The StoreInContract function of the smart contract is called 50 times for 50 different accounts (batch of 50 transactions)
ReadFromContract50	The ReadFromContract function of the smart contract is called 50 times for 50 different accounts (batch of 50 local reads)
Deposit100	The Deposit function of the smart contract is called 100 times for 100 different accounts (batch of 100 transactions)
Withdraw100	The Withdraw function of the smart contract is called 100 times for 100 different accounts (batch of 100 transactions)
ReadBalance100	The ReadBalance function of the smart contract is called 100 times for 100 different accounts (batch of 100 local reads)
StoreInContract100	The StoreInContract function of the smart contract is called 100 times for 100 different accounts (batch of 100 transactions)
ReadFromContract100	The ReadFromContract function of the smart contract is called 100 times for 100 different accounts (batch of 100 local reads)
Deposit150	The Deposit function of the smart contract is called 150 times for 150 different accounts (batch of 150 transactions)
Withdraw150	The Withdraw function of the smart contract is called 150 times for 150 different accounts (batch of 150 transactions)
ReadBalance150	The ReadBalance function of the smart contract is called 150 times for 150 different accounts (batch of 150 local reads)
StoreInContract150	The StoreInContract function of the smart contract is called 150 times for 150 different accounts (batch of 150 transactions)
ReadFromContract150	The ReadFromContract function of the smart contract is called 150 times for 150 different accounts (batch of 150 local reads)

Table 3: Summary of DApp functions

CHAPTER 4

RESULTS & ANALYSIS

In this chapter, the results of the experimentations on both Azure and AUB's HPC are presented. After that, for better comprehension and detection of patterns, the results are visualized using both Microsoft Excel and Python Matplotlib. Lastly, the study findings and the analysis of the obtained results are concluded.

Note that all the information gathered from the running samples of EOS and Ethereum on both Azure and AUB's HPC were organized through a Python script into a Microsoft Excel sheet. This is done in order to avoid any human errors while inputting data manually into a Microsoft Excel document. The Python script uses NumPy and Pandas libraries.

Figures 28 to 35 were generated using Python Matplotlib. While figures 24, 25, 26, and 27 were generated using Microsoft Excel.

Tables 4 to 7 represent the average values of the 30 samples tried for each scenario.

Functions	10	50	100	150
Deposit (ms)	300.466667	306.133333	316.666667	311.033333
Withdraw (ms)	234.133333	202.2	216.2	205.666667
ReadBalance (ms)	1.86666667	2.03333333	2.1	2.1
StoreInContract (ms)	191.066667	222.7	218.966667	205.1
ReadFromContract (ms)	2.06666667	2.16666667	2.2	2.43333333
Deposit10 (ms)	1950.13333	1892.53333	1939.3	1873.03333
Withdraw10 (ms)	1914.23333	1958.2	1907.8	1904.06667
ReadBalance10 (ms)	10.1	10.1333333	10.3333333	10.1
StoreInContract10 (ms)	1935.23333	1949.73333	1890.6	1912.26667
ReadFromContract10 (ms)	7.46666667	7.33333333	7.43333333	7.46666667
Deposit50 (ms)	9204.36667	9345.9	9324.63333	9402.36667
Withdraw50 (ms)	9336.13333	9323.73333	9332.2	9425.73333
ReadBalance50 (ms)	27.3333333	27.3	27.2	27.9666667
StoreInContract50 (ms)	9512.53333	9513.06667	9532.96667	9435.3
ReadFromContract50 (ms)	29.6333333	30.3333333	30.5	31.4
Deposit100 (ms)	18559.1	18696.2333	18605.0333	18619.7667
Withdraw100 (ms)	18641.3333	18607.1667	18687.3667	18792.0333
ReadBalance100 (ms)	44.9333333	45.9	45.3333333	46.0666667
StoreInContract100 (ms)	19398.6333	19582.3	19681.4333	19525.3333
ReadFromContract100 (ms)	43.0333333	43.7666667	43.8666667	42.4333333
Deposit150 (ms)	27687.0667	27917.0667	27837.4667	27857.7333
Withdraw150 (ms)	27640.2333	27867.2667	27862.5333	27734.7
ReadBalance150 (ms)	55.9	54.7666667	54.4	54.7666667
StoreInContract150 (ms)	27839.1	28193.1	27946.5667	27995.5
ReadFromContract150 (ms)	54.8666667	55.1	54.8666667	54.5333333
mem_min_nodejs (MB)	4618.087	6090.98833	7971.95067	9791.063
mem_max_nodejs (MB)	4676.787	6194.349	8134.06367	10020.525
cpu_min_ps (%)	0.80666667	0.66	0.5	0.47
cpu_max_ps (%)	2.73333333	1.98	1.52333333	1.3
totalSeconds (s)	299.166667	429.233333	590.5	752.066667

Table 4: EOS Azure Summary

Functions	10	50	100	150
Deposit (ms)	417.033333	390.133333	334.166667	377.5
Withdraw (ms)	251.4	268.166667	250.566667	264.3
ReadBalance (ms)	3.96666667	3.4	3.76666667	6.9
StoreInContract (ms)	246.966667	226	236.833333	227.633333
ReadFromContract (ms)	4.06666667	3.33333333	2.4	3.23333333
Deposit10 (ms)	2323.3	2116.8	1965.03333	2198.03333
Withdraw10 (ms)	2153.66667	2106.96667	2035.43333	2121.86667
ReadBalance10 (ms)	17	13.8	15.3333333	11.6333333
StoreInContract10 (ms)	2357.03333	2083.26667	2181.76667	2067.4
ReadFromContract10 (ms)	19.5	15.5	14.3	12.8
Deposit50 (ms)	10568.8333	10586.4667	10063.9	10513.9
Withdraw50 (ms)	11875.2333	10480.6667	10070.6667	10348.5333
ReadBalance50 (ms)	36.6666667	35.2	55.2666667	43.8333333
StoreInContract50 (ms)	11782.3333	10197.1	10158.3	10318.2667
ReadFromContract50 (ms)	47.8666667	57.1666667	57.5333333	57.6
Deposit100 (ms)	22524.7	20474	20276.7667	19899.4
Withdraw100 (ms)	23277.0667	20732.9667	19929.8	20245.6667
ReadBalance100 (ms)	119.166667	84.3666667	87.3666667	79.2666667
StoreInContract100 (ms)	22592.2333	20508.4667	20514.2667	19934.7667
ReadFromContract100 (ms)	135.7	61.9	77	73.3333333
Deposit150 (ms)	32133.3	30574.8667	29546.2333	30350.4
Withdraw150 (ms)	33424.2667	30358.1	30421.9333	30957.7667
ReadBalance150 (ms)	205.766667	139.833333	66.3	183.333333
StoreInContract150 (ms)	32111.8333	30508.5333	29973.3333	30172.8
ReadFromContract150 (ms)	101	102.766667	82.9333333	218.833333
mem_min_nodejs (MB)	2922.12867	4478.92	6527.63267	8517.833
mem_max_nodejs (MB)	3000.211	4539.206	6629.63467	8643.53133
cpu_min_ps (%)	1.17	0.24333333	0.14666667	0.03
cpu_max_ps (%)	4.37	3.03666667	3.04666667	4.26333333
totalSeconds (s)	388.7	497.1	658.666667	855.233333

Table 5: EOS HPC Summary

Functions	10	50	100	150
Deposit (ms)	4423.66667	1789.36667	2262.4	2646.86667
Withdraw (ms)	4513.23333	1877.83333	5055.33333	1929.33333
ReadBalance (ms)	3.53333333	3.76666667	4.56666667	7.43333333
StoreInContract (ms)	4114.4	1780.06667	1587.3	1660.1
ReadFromContract (ms)	4.93333333	5.46666667	6.46666667	9.03333333
Deposit10 (ms)	3377.86667	1818.96667	1782.4	1681.3
Withdraw10 (ms)	3772.16667	1939.83333	1690.43333	1718.73333
ReadBalance10 (ms)	23.6333333	25.3	30.3666667	63.3
StoreInContract10 (ms)	3408.03333	1775.13333	1697.73333	1749
ReadFromContract10 (ms)	19.4333333	20.7333333	27.3	43.5333333
Deposit50 (ms)	3835.76667	2065.73333	1995.63333	2425
Withdraw50 (ms)	3649.73333	2104.76667	1700.06667	2409.03333
ReadBalance50 (ms)	47.6333333	50.2	63	145.066667
StoreInContract50 (ms)	3469.63333	1976.1	2247.76667	2281.46667
ReadFromContract50 (ms)	62.1333333	66.1666667	91.8333333	147.033333
Deposit100 (ms)	3790.7	2029.56667	2145.93333	2659.53333
Withdraw100 (ms)	3659.56667	2128.33333	2221.23333	2645.83333
ReadBalance100 (ms)	76.8	83.0666667	101.666667	190.033333
StoreInContract100 (ms)	4331.03333	2213.5	2397.23333	2965.33333
ReadFromContract100 (ms)	94.6333333	99.5	148.9	220.966667
Deposit150 (ms)	3577.26667	2381.2	2640.46667	2893.76667
Withdraw150 (ms)	4233.63333	2127.86667	2458.66667	2890.8
ReadBalance150 (ms)	85.6333333	102.3	145.966667	248.966667
StoreInContract150 (ms)	3822.46667	2615.23333	2934.13333	4104.2
ReadFromContract150 (ms)	133.266667	143.433333	168	302.233333
mem_min_nodejs (MB)	8364.30467	27328.994	45207.2547	83585.325
mem_max_nodejs (MB)	9978.60233	32701.552	59849.283	116119.207
cpu_min_ps (%)	4.2	1.28333333	5.22333333	18.4633333
cpu_max_ps (%)	105.27	138.533333	139.71	147.86
totalSeconds (s)	164.6	341.733333	566.733333	867.333333

Table 6: Ethereum Azure Summary

Functions	10	50	100	150
Deposit (ms)	5996	2108.4	2025.6	1955.8
Withdraw (ms)	4565.7	1993.9	2133.53333	1850.56667
ReadBalance (ms)	7.26666667	5.33333333	14.3666667	18.0666667
StoreInContract (ms)	5169.06667	2061.06667	1865.2	1999.6
ReadFromContract (ms)	12.9666667	11.4	23.8333333	26.1333333
Deposit10 (ms)	4362.86667	2437.53333	2241.8	2713.16667
Withdraw10 (ms)	5898.3	2056	2367.83333	2557.2
ReadBalance10 (ms)	33.3333333	36.8	51.7333333	85.7333333
StoreInContract10 (ms)	4866.56667	2466.93333	2208.46667	2703.46667
ReadFromContract10 (ms)	36.9	41.9	60.5333333	117.8
Deposit50 (ms)	6311.26667	2974.93333	3312	4358.9
Withdraw50 (ms)	5542.76667	2518.56667	3261.43333	4397.7
ReadBalance50 (ms)	111.833333	107.233333	151.666667	322.8
StoreInContract50 (ms)	5073.36667	2775.96667	3427.6	4624
ReadFromContract50 (ms)	140.166667	111.233333	192.666667	435.166667
Deposit100 (ms)	5944.63333	3543.23333	4602.73333	6832.13333
Withdraw100 (ms)	5122.93333	3544.1	4106.13333	6482.63333
ReadBalance100 (ms)	156.566667	134.3	205.9	374.4
StoreInContract100 (ms)	5966.66667	3582.7	5667.7	8334.4
ReadFromContract100 (ms)	179.633333	210.133333	249.8	409.9
Deposit150 (ms)	5988.8	3674.76667	6182.3	9086.1
Withdraw150 (ms)	5894.73333	3801.23333	5951.23333	8930.06667
ReadBalance150 (ms)	177.633333	164.666667	245.733333	321.1
StoreInContract150 (ms)	6078	4391.76667	7690.73333	11655.86667
ReadFromContract150 (ms)	220.366667	214.166667	252.366667	486.666667
mem_min_nodejs (MB)	7227.28533	25766.8783	43305.8537	60634.8603
mem_max_nodejs (MB)	8424.83433	30294.0963	54490.758	78175.541
cpu_min_ps (%)	11.9766667	6.20333333	9.23	10.3266667
cpu_max_ps (%)	114.1	132.533333	157.466667	148.866667
totalSeconds (s)	220.4	402.7	709.866667	1049.03333

Table 7: Ethereum HPC Summary

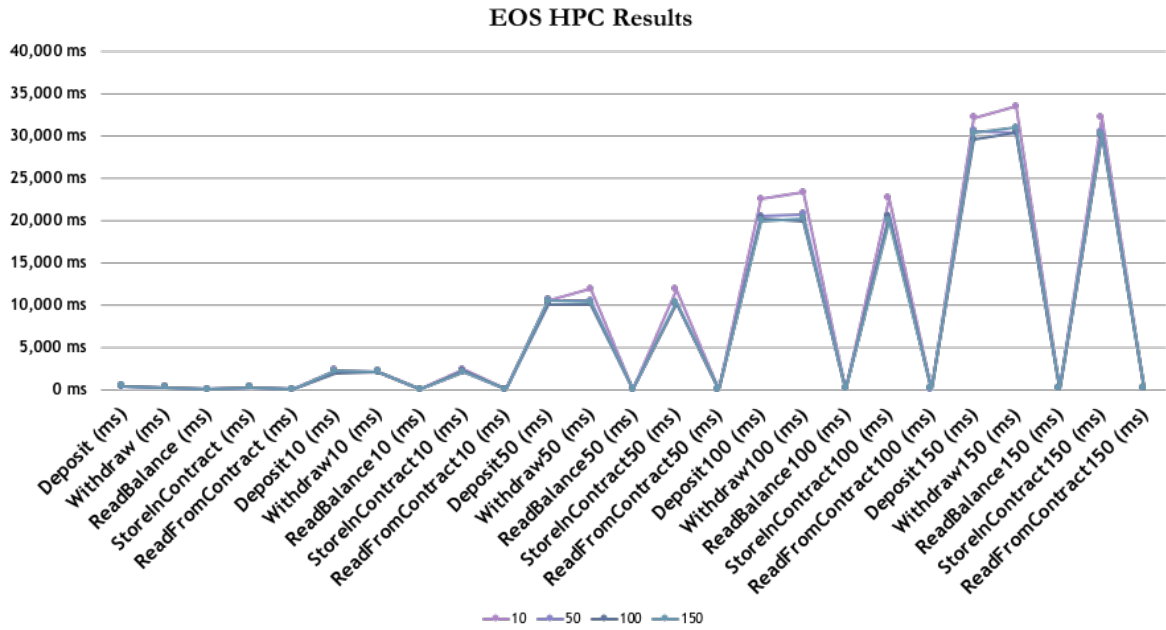


Figure 23: EOS HPC Results

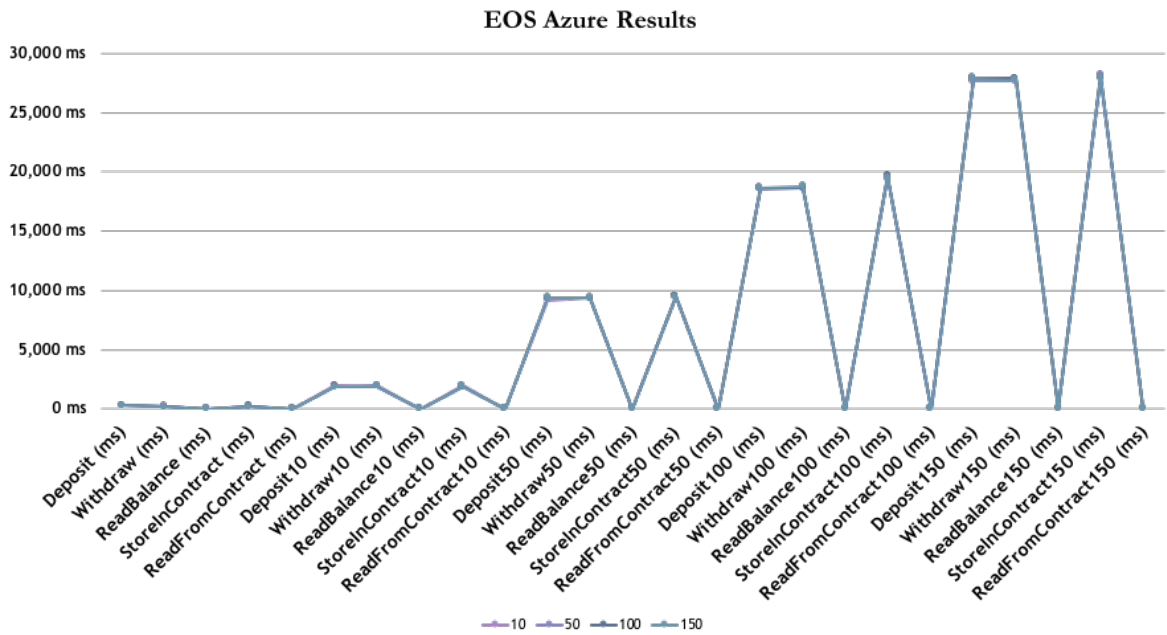


Figure 24: EOS Azure Results

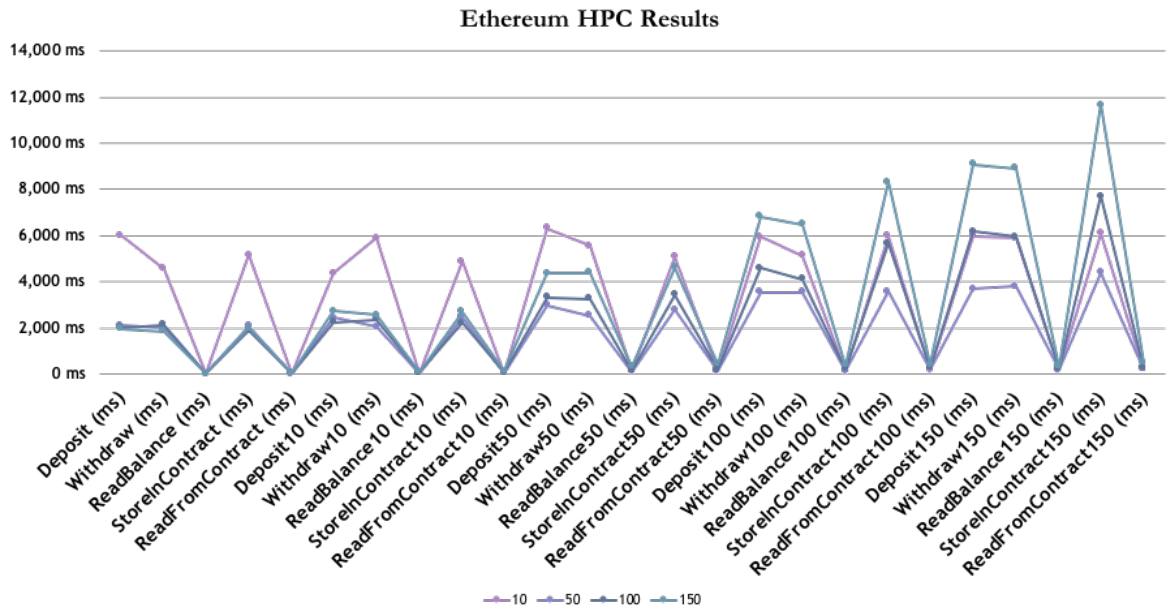


Figure 25: Ethereum HPC Results

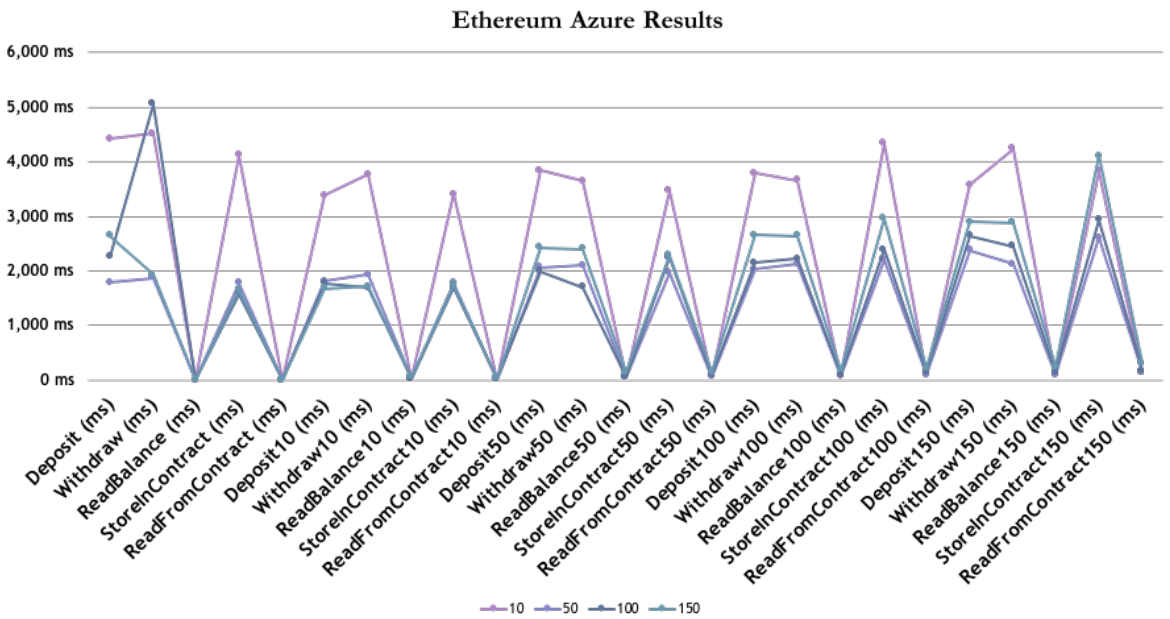


Figure 26: Ethereum Azure Results

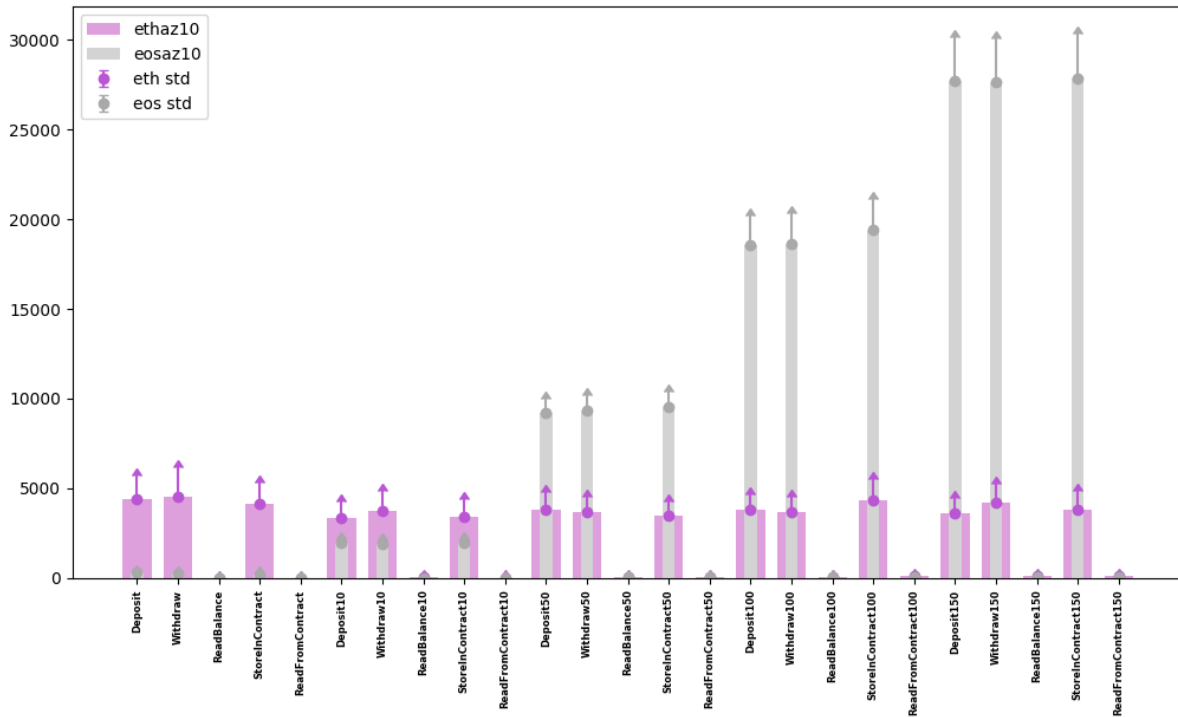


Figure 28: Azure - EOS vs Ethereum (Network of 10 nodes)

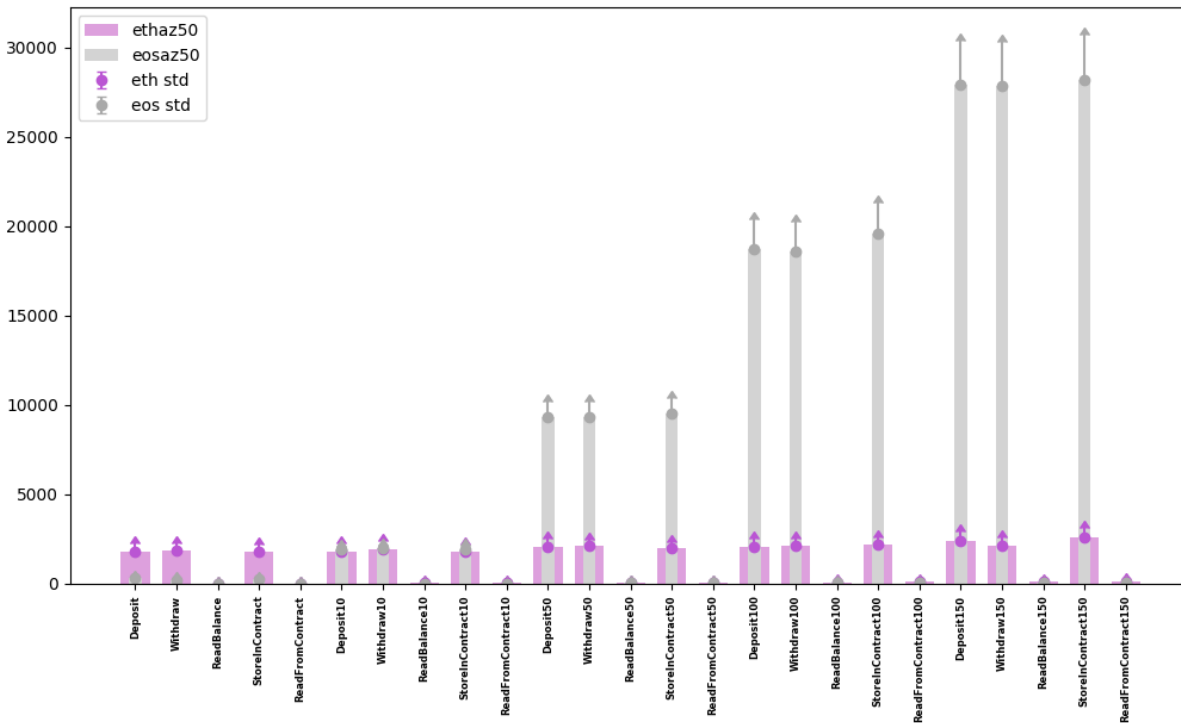


Figure 27: Azure - EOS vs Ethereum (Network of 50 nodes)

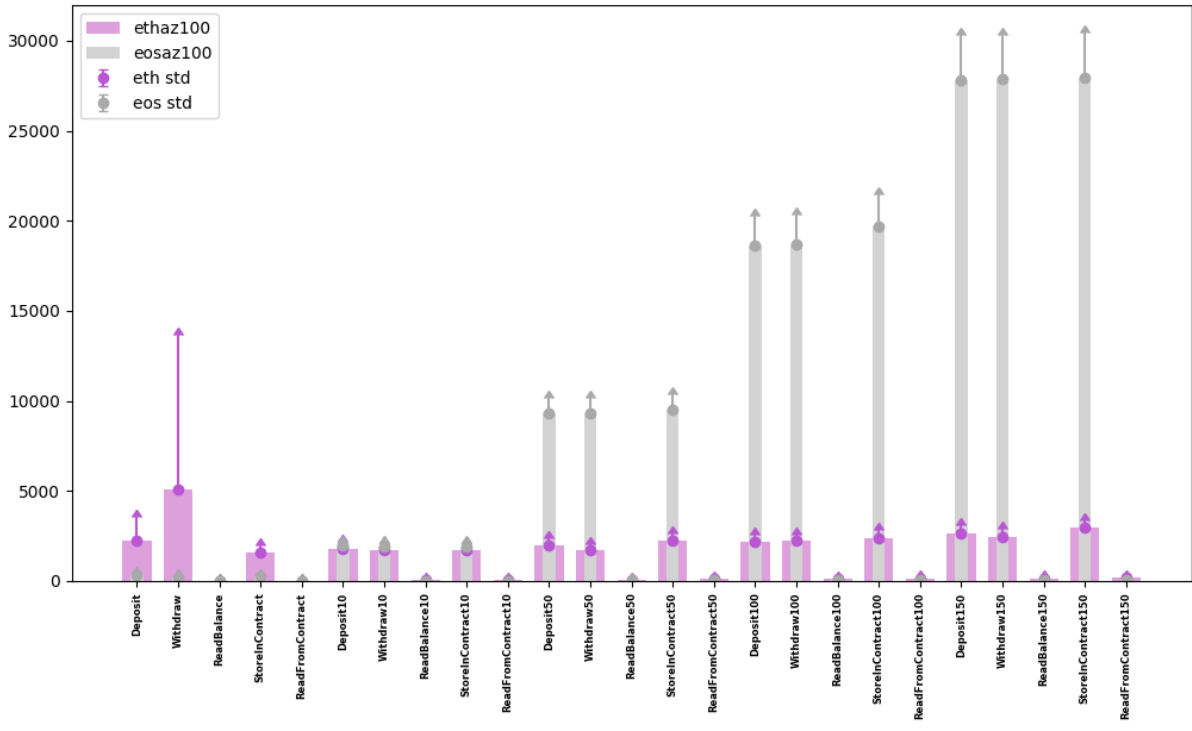


Figure 30: Azure - EOS vs Ethereum (Network of 100 nodes)

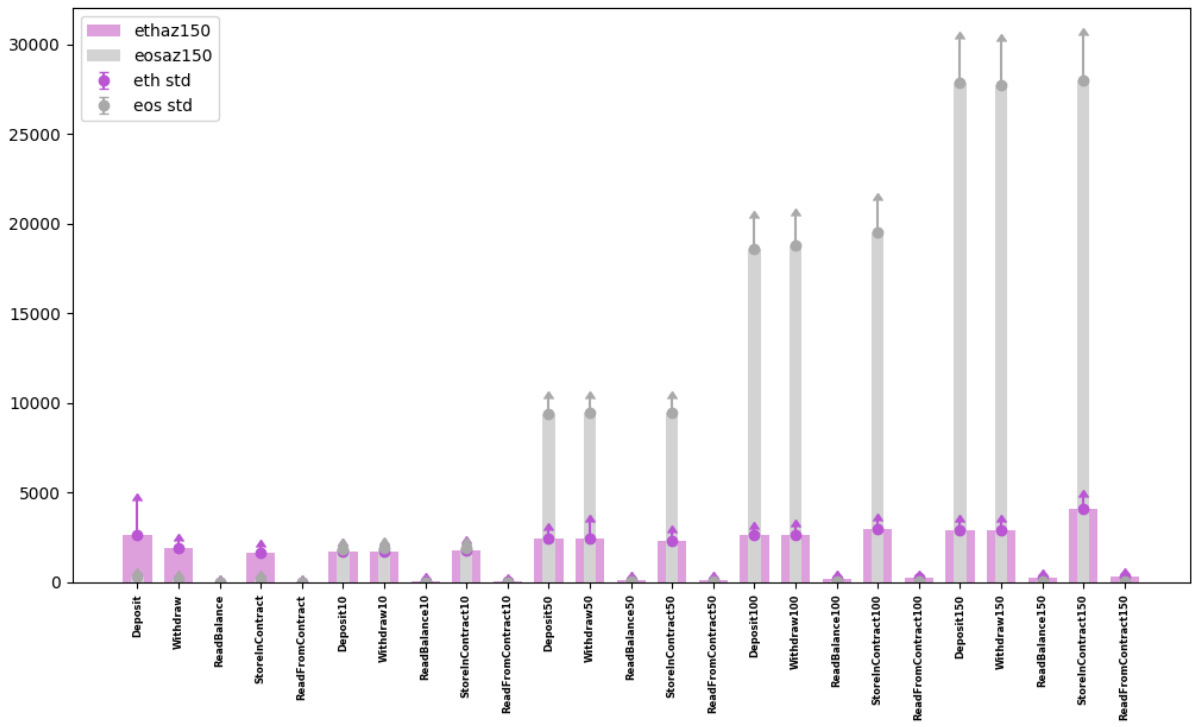


Figure 29: Azure - EOS vs Ethereum (Network of 150 nodes)

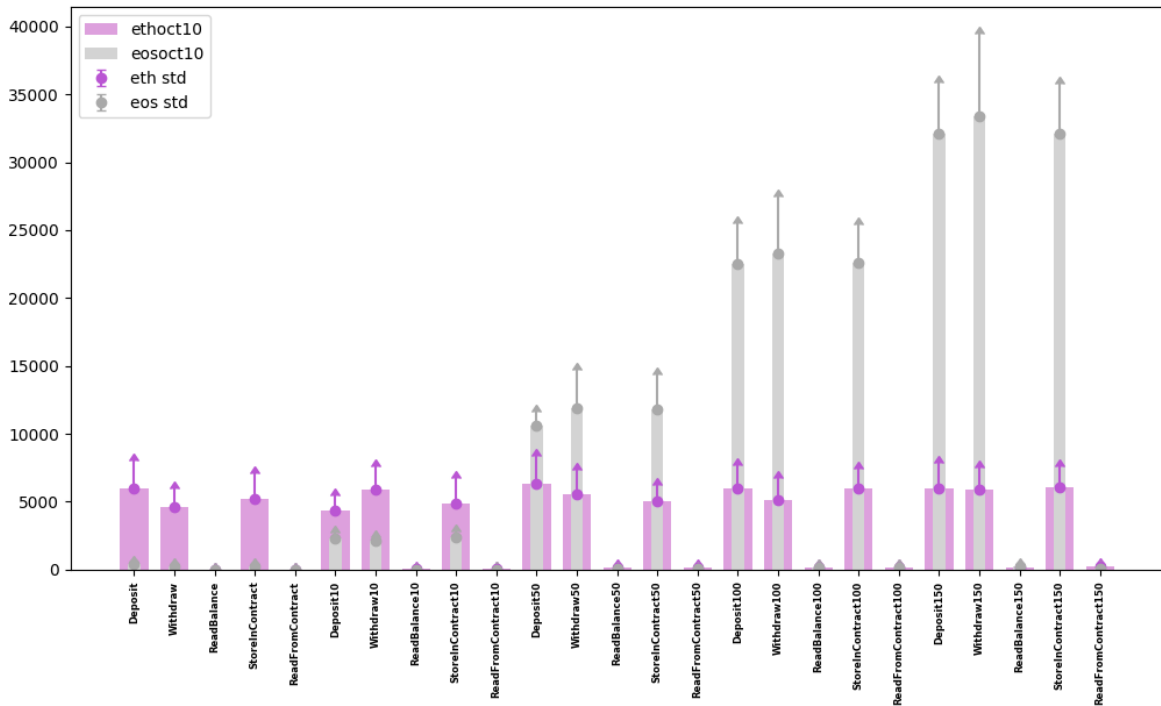


Figure 32: HPC - EOS vs Ethereum (Network of 10 nodes)

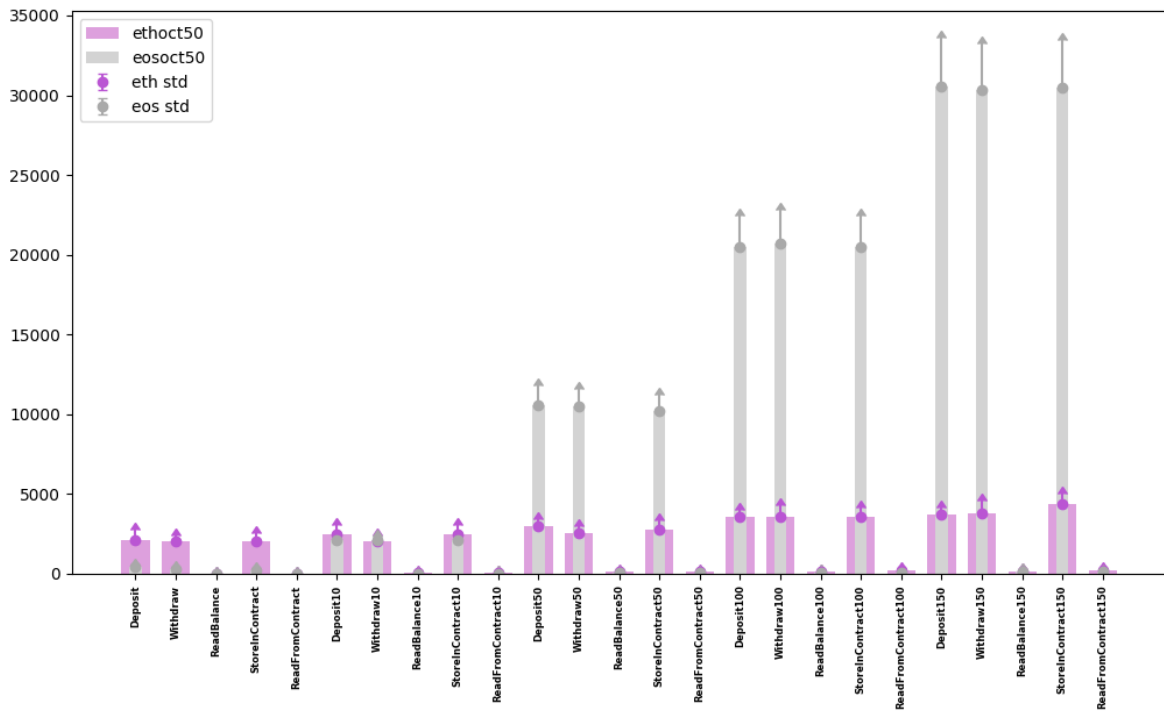


Figure 31: HPC - EOS vs Ethereum (Network of 50 nodes)

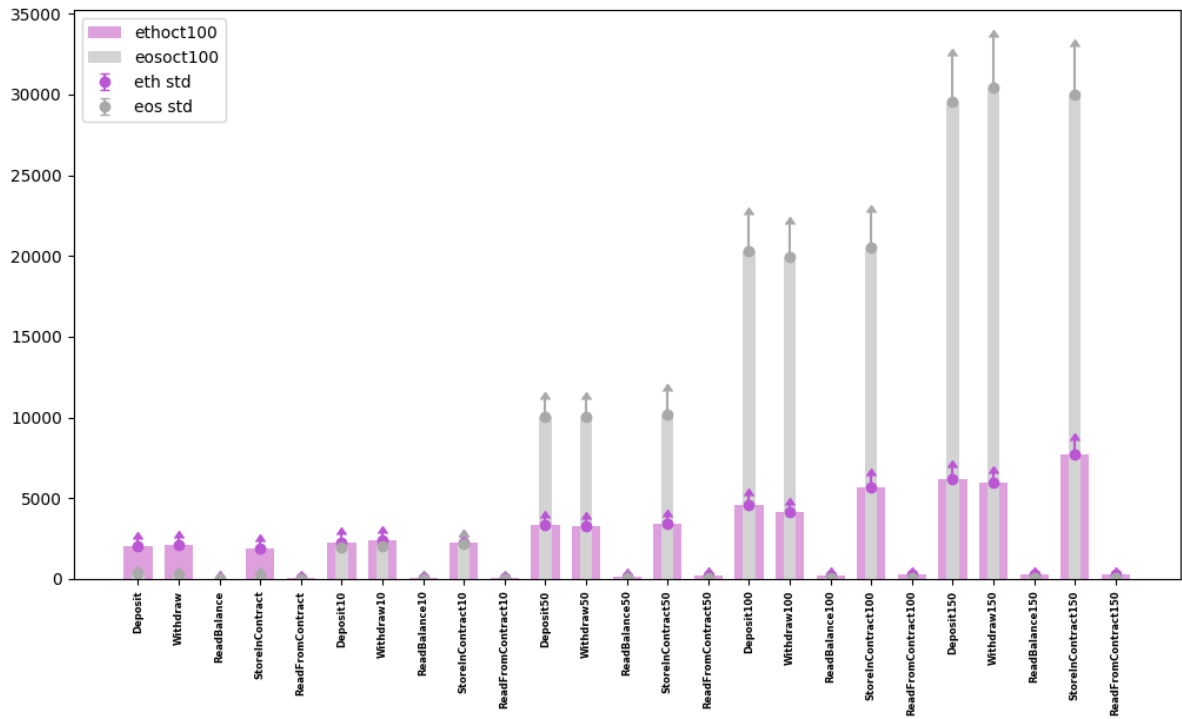


Figure 34: HPC - EOS vs Ethereum (Network of 100 nodes)

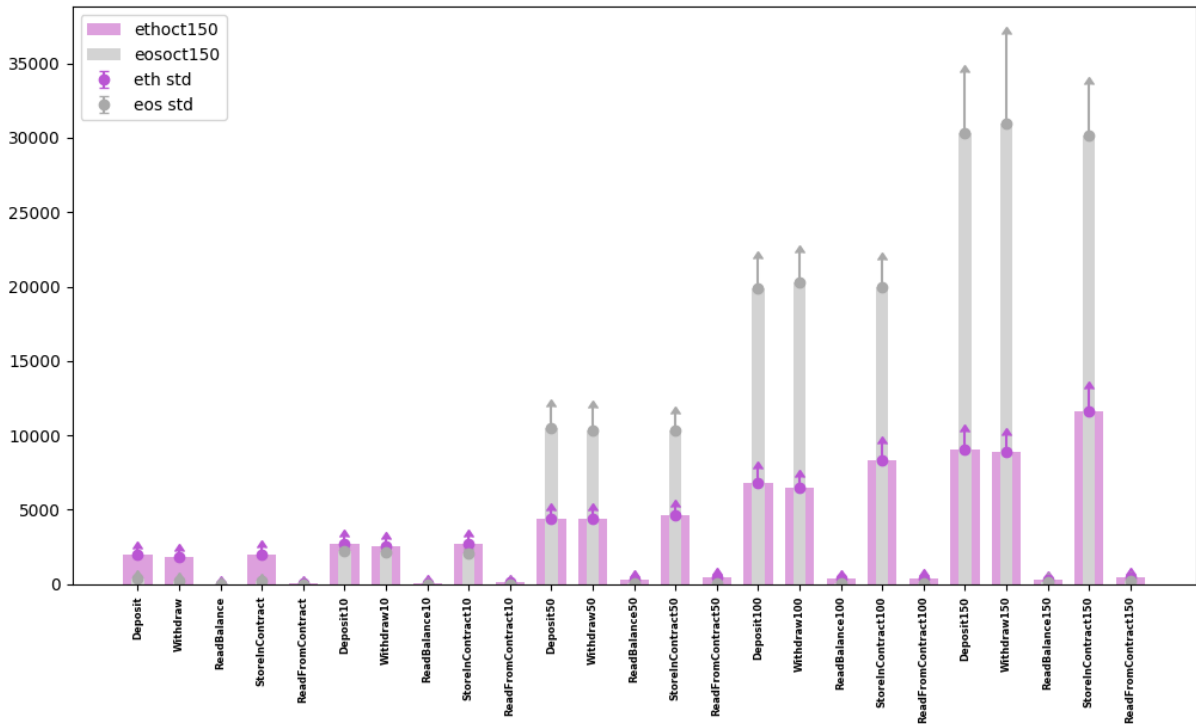


Figure 33: HPC - EOS vs Ethereum (Network of 150 nodes)

From the simulation of each platform on both Azure and AUB's HPC, we can evaluate the following:

	EOS HPC		EOS Azure	
	Low	High	Low	High
Deposit (ms)	334.17	417.03	300.47	316.67
Withdraw (ms)	250.57	268.17	202.20	234.13
ReadBalance (ms)	3.40	6.90	1.87	2.10
StoreInContract (ms)	226.00	246.97	191.07	222.70
ReadFromContract (ms)	2.40	4.07	2.07	2.43
Deposit10 (ms)	1965.03	2323.30	1873.03	1950.13
Withdraw10 (ms)	2035.43	2153.67	1904.07	1958.20
ReadBalance10 (ms)	11.63	17.00	10.10	10.33
StoreInContract10 (ms)	2067.40	2357.03	1890.60	1949.73
ReadFromContract10 (ms)	12.80	19.50	7.33	7.47
Deposit50 (ms)	10063.90	10586.47	9204.37	9402.37
Withdraw50 (ms)	10070.67	11875.23	9323.73	9425.73
ReadBalance50 (ms)	35.20	55.27	27.20	27.97
StoreInContract50 (ms)	10158.30	11782.33	9435.30	9532.97
ReadFromContract50 (ms)	47.87	57.60	29.63	31.40
Deposit100 (ms)	19899.40	22524.70	18559.10	18696.23
Withdraw100 (ms)	19929.80	23277.07	18607.17	18792.03
ReadBalance100 (ms)	79.27	119.17	44.93	46.07
StoreInContract100 (ms)	19934.77	22592.23	19398.63	19681.43
ReadFromContract100 (ms)	61.90	135.70	42.43	43.87
Deposit150 (ms)	29546.23	32133.30	27687.07	27917.07
Withdraw150 (ms)	30358.10	33424.27	27640.23	27867.27
ReadBalance150 (ms)	66.30	205.77	54.40	55.90
StoreInContract150 (ms)	29973.33	32111.83	27839.10	28193.10
ReadFromContract150 (ms)	82.93	218.83	54.53	55.10

Table 8: Low and High values of EOS on HPC vs Azure

	ETH HPC		ETH Azure	
	Low	High	Low	High
Deposit (ms)	1955.80	5996.00	1789.37	4423.67
Withdraw (ms)	1850.57	4565.70	1877.83	5055.33
ReadBalance (ms)	5.33	18.07	3.53	7.43
StoreInContract (ms)	1865.20	5169.07	1587.30	4114.40
ReadFromContract (ms)	11.40	26.13	4.93	9.03
Deposit10 (ms)	2241.80	4362.87	1681.30	3377.87
Withdraw10 (ms)	2056.00	5898.30	1690.43	3772.17
ReadBalance10 (ms)	33.33	85.73	23.63	63.30
StoreInContract10 (ms)	2208.47	4866.57	1697.73	3408.03
ReadFromContract10 (ms)	36.90	117.80	19.43	43.53
Deposit50 (ms)	2974.93	6311.27	1995.63	3835.77
Withdraw50 (ms)	2518.57	5542.77	1700.07	3649.73
ReadBalance50 (ms)	107.23	322.80	47.63	145.07
StoreInContract50 (ms)	2775.97	5073.37	1976.10	3469.63
ReadFromContract50 (ms)	111.23	435.17	62.13	147.03
Deposit100 (ms)	3543.23	6832.13	2029.57	3790.70
Withdraw100 (ms)	3544.10	6482.63	2128.33	3659.57
ReadBalance100 (ms)	134.30	374.40	76.80	190.03
StoreInContract100 (ms)	3582.70	8334.40	2213.50	4331.03
ReadFromContract100 (ms)	179.63	409.90	94.63	220.97
Deposit150 (ms)	3674.77	9086.10	2381.20	3577.27
Withdraw150 (ms)	3801.23	8930.07	2127.87	4233.63
ReadBalance150 (ms)	164.67	321.10	85.63	248.97
StoreInContract150 (ms)	4391.77	11655.87	2615.23	4104.20
ReadFromContract150 (ms)	214.17	486.67	133.27	302.23

Table 9: Low and High values of Ethereum on HPC vs Azure

By examining each platform by itself on both Azure and AUB's HPC (table 8-9), one can notice that the user response time ranges are noticeably lower on Azure in both EOS and Ethereum platforms. However, by looking at the visualized graphs in figures 24 to 27, we can see that the pattern is the same for each platform on both Azure and AUB's HPC.

In EOS's case in Fig. 25, all the networks (10, 50, 100, 150) have a perfectly stacked line graph. This shows more stable time ranges with the variation of the number of nodes in the network and thus more stability on Azure.

In Ethereum's case too in Fig. 27, all the networks except the network of 10 nodes are closer to each other than the lines in Fig. 26. The network of 10 nodes in Ethereum takes a longer time to respond than a network with more nodes because it has only two mining nodes with each having one mining thread only. This makes the mining puzzle harder to resolve and thus takes more time on both AUB's HPC and on Azure.

Looking at all the visualized graphs (figures 24 to 35), it is very clear that:

- With a very low throughput and with the variation of the number of nodes in the network, EOS's numbers (~200ms) are way lower than Ethereum's numbers (~2s).
- With the increase of the number of transactions submitted, Ethereum keeps a steady range in comparison to EOS. EOS's response time augments rapidly starting when the number of transactions submitted is 50 and keeps growing up, while Ethereum's time response takes 2 to 4 seconds on Azure at max. At 150 transactions submitted, EOS's response time is almost 6 times Ethereum's response time.
- One can also notice that Ethereum's storing function takes slightly more time than deposit/withdraw transactions.
- Local call functions take less time on EOS
- CPU and RAM usage averages of Ethereum were way higher than those of EOS due to the heaviness of Proof-of-Work
- The experimentation on Azure achieved shorter response time than that on Octopus but guarding the same patterns.

More Analysis on Ethereum and EOS.IO Logs:

In this section, a small network is generated for each of Ethereum and EOS.IO in order to analyze the logs.

```
INFO [03-21|08:05:37.601] Imported new chain segment blocks=1 txs=1 mgas=0.155 el
INFO [03-21|08:05:43.868] Imported new chain segment blocks=1 txs=0 mgas=0.000 el
INFO [03-21|08:05:50.215] Submitted transaction fullhash=0xdcea3c62243a7eabe
INFO [03-21|08:05:50.215] Submitted transaction fullhash=0xaf98be0c67f93ff50
INFO [03-21|08:05:50.218] Submitted transaction fullhash=0x756304c77b09636e4
INFO [03-21|08:05:50.221] Submitted transaction fullhash=0x68df14d01c79322a6
INFO [03-21|08:05:50.222] Submitted transaction fullhash=0xb28be4da071f650de
INFO [03-21|08:05:50.225] Submitted transaction fullhash=0xb7601f25e9e2b1b9b
INFO [03-21|08:05:50.228] Submitted transaction fullhash=0xcb71fb37c2298e069
INFO [03-21|08:05:50.229] Submitted transaction fullhash=0x9a253d4a8b0fa162c
INFO [03-21|08:05:50.230] Submitted transaction fullhash=0x3a3039fa28352462e
INFO [03-21|08:05:50.231] Submitted transaction fullhash=0x1cfb38cfd0002f664
INFO [03-21|08:05:51.760] Imported new chain segment blocks=1 txs=0 mgas=0.000 el
INFO [03-21|08:05:53.196] Imported new chain segment blocks=1 txs=10 mgas=0.271 e
INFO [03-21|08:05:53.531] Submitted transaction fullhash=0xb307f6314205d04
```

Figure 36: Ethereum 10 Transactions Submission Logs

```
INFO [03-21|08:06:08.949] Submitted transaction fullhash=0xda591a132bfab2755
INFO [03-21|08:06:08.964] Submitted transaction fullhash=0x3952bce840294df73
INFO [03-21|08:06:08.965] Submitted transaction fullhash=0x3c4371c4eca040b51
INFO [03-21|08:06:08.966] Submitted transaction fullhash=0xe1cc6d23409c1e2b6
INFO [03-21|08:06:08.977] Submitted transaction fullhash=0xf2f08276b294e1736
INFO [03-21|08:06:08.979] Submitted transaction fullhash=0x5f95fbbc3f82b09d9
INFO [03-21|08:06:08.985] Submitted transaction fullhash=0xe90a23bfa1cb166b4
INFO [03-21|08:06:08.986] Submitted transaction fullhash=0xab211ebbb6cca66835
INFO [03-21|08:06:08.988] Submitted transaction fullhash=0x55fa6210525e2a3ca
INFO [03-21|08:06:08.990] Submitted transaction fullhash=0x59c1d2a16ecfd1bd0
INFO [03-21|08:06:08.992] Submitted transaction fullhash=0x7a11da63de2fec354
INFO [03-21|08:06:08.993] Submitted transaction fullhash=0xa296a651db522e3cb
INFO [03-21|08:06:08.994] Submitted transaction fullhash=0xd5d74d07f902d51af
INFO [03-21|08:06:08.995] Submitted transaction fullhash=0xc233aa2391055ae2f
INFO [03-21|08:06:08.996] Submitted transaction fullhash=0x773fe96d6e24f8c3c
INFO [03-21|08:06:09.078] Submitted transaction fullhash=0xe268ebbfc86b0bca2
INFO [03-21|08:06:09.090] Submitted transaction fullhash=0x6cc25efc8502d5dfb
INFO [03-21|08:06:09.094] Submitted transaction fullhash=0xcb6d0e50647627bd4
INFO [03-21|08:06:09.104] Submitted transaction fullhash=0x1ea7e0b5765d373b2
INFO [03-21|08:06:09.117] Submitted transaction fullhash=0x60d44ce99f22199c1
INFO [03-21|08:06:09.118] Submitted transaction fullhash=0x12236097acd60bda6
INFO [03-21|08:06:09.119] Submitted transaction fullhash=0xf5c1843b78ee9583c
INFO [03-21|08:06:09.156] Submitted transaction fullhash=0x9eb897a78850b226e
INFO [03-21|08:06:09.159] Submitted transaction fullhash=0xf2a10341601c299d1
INFO [03-21|08:06:09.160] Submitted transaction fullhash=0xc0c3780cb940a49b5
INFO [03-21|08:06:09.166] Submitted transaction fullhash=0xa5e7cd6af3addefac
INFO [03-21|08:06:09.167] Submitted transaction fullhash=0xe023664286abc640b
INFO [03-21|08:06:09.168] Submitted transaction fullhash=0xdd70a1c325f05ac34
INFO [03-21|08:06:09.169] Submitted transaction fullhash=0x356a4defe105d38e7
INFO [03-21|08:06:21.883] Imported new chain segment blocks=1 txs=50 mgas=1.354 e
INFO [03-21|08:06:23.961] Submitted transaction fullhash=0xe662803fa4397ab84
```

Figure 35: Ethereum 50 Transactions Submission Logs

```

INFO [03-21|08:06:59.851] Submitted transaction fullhash=0x8ab08a9a23df9a04e0
INFO [03-21|08:06:59.863] Submitted transaction fullhash=0x7f25d9625c7b29cadb
INFO [03-21|08:06:59.864] Submitted transaction fullhash=0x978a05214ee6c0c635
INFO [03-21|08:06:59.865] Submitted transaction fullhash=0xe8a837e5960a3a7459
INFO [03-21|08:06:59.866] Submitted transaction fullhash=0x8cafa8c3cc3366013
INFO [03-21|08:06:59.871] Submitted transaction fullhash=0x9961132e7ca70af293
INFO [03-21|08:06:59.872] Submitted transaction fullhash=0x65700801c4f3b36d28
INFO [03-21|08:06:59.873] Submitted transaction fullhash=0xa8fb2ada0e40e3e410
INFO [03-21|08:06:59.874] Submitted transaction fullhash=0xe4ed6a41eacefb1032
INFO [03-21|08:06:59.875] Submitted transaction fullhash=0xba5fdf480555975b07
INFO [03-21|08:06:59.876] Submitted transaction fullhash=0xf6696cb2ce688ac437
INFO [03-21|08:06:59.891] Submitted transaction fullhash=0xcb72f95d55200bcfcc
INFO [03-21|08:06:59.892] Submitted transaction fullhash=0x067db9f5385e5c6d1f
INFO [03-21|08:06:59.893] Submitted transaction fullhash=0x6a86f6a02b7ea4972d
INFO [03-21|08:06:59.894] Submitted transaction fullhash=0x50b828e5fc383b87ac
INFO [03-21|08:06:59.895] Submitted transaction fullhash=0xbe37a76c7d992c1690
INFO [03-21|08:06:59.899] Submitted transaction fullhash=0x9d1f427ee130c95ee9
INFO [03-21|08:06:59.900] Submitted transaction fullhash=0x10c3539e8c9a1292b6
INFO [03-21|08:06:59.901] Submitted transaction fullhash=0xd0a2589c6cf5ca1203
INFO [03-21|08:06:59.901] Submitted transaction fullhash=0x664d0800329d7c11cc
INFO [03-21|08:06:59.902] Submitted transaction fullhash=0x5ad784c0cfe76c1801
INFO [03-21|08:06:59.903] Submitted transaction fullhash=0x202d4fe65873e0b958
INFO [03-21|08:06:59.987] Submitted transaction fullhash=0xe23a26d78b76465ecb
INFO [03-21|08:07:00.002] Submitted transaction fullhash=0xcfb89f14c20a3efaec
INFO [03-21|08:07:00.003] Submitted transaction fullhash=0x3a50e695bdbbe6bedc
INFO [03-21|08:07:00.105] Submitted transaction fullhash=0x6e6001f978c73e54e6
INFO [03-21|08:07:00.106] Submitted transaction fullhash=0x198db073ec0a56038b
INFO [03-21|08:07:00.113] Submitted transaction fullhash=0x0f8dee2bc4e9a22048
INFO [03-21|08:07:00.116] Submitted transaction fullhash=0x51923f62e6733d7dcd
INFO [03-21|08:07:00.836] Imported new chain segment blocks=1 txs=0 mgas=0.000 eL
INFO [03-21|08:07:07.220] Imported new chain segment blocks=1 txs=100 mgas=2.707 eL
INFO [03-21|08:07:08.627] Submitted transaction fullhash=0x70a0baace7ddf6b6b2

```

Figure 37: Ethereum 100 Transactions Submission Logs

After having a look at the Geth node logs (Fig. 35,36,37), we can clearly conclude that Ethereum nodes are packing all the transactions in one block. This explains why the user response time of 10, 50, 100, and 150 calls in Ethereum are very close to the single call timing.

As for Nodeos logs (Fig. 38 -> 41), one can notice how EOS.IO transactions are spread over multiple blocks. Even if it is the same producer (check signed by field in figures), they still are parted in multiple blocks. Also, the number of empty blocks in between multiple blocks is large.

Several reasons could result in the transaction grouping of a certain block. Latency to process/broadcast transactions, or some kind of bottleneck in the network protocol could

be the problem. EOS.IO need to resolve this issue in order to compete with Ethereum and all other decentralized computing platforms.

```

accountc
File Edit View Search Terminal Help
<->Info 2020-03-21T06:37:35.508 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block db2b136b4b56c295... #1
95 @ 2020-03-21T06:37:35.500 signed by account [trxs: 0, lib: 171, conf: 0, latency: 8 ms]
<->Info 2020-03-21T06:37:36.006 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 5d182875c0ad2b97... #1
96 @ 2020-03-21T06:37:36.000 signed by account [trxs: 0, lib: 183, conf: 12, latency: 6 ms]
<->Info 2020-03-21T06:37:36.509 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 6aa5f88599fd2e0f... #1
97 @ 2020-03-21T06:37:36.500 signed by account [trxs: 0, lib: 183, conf: 0, latency: 9 ms]
<->Info 2020-03-21T06:37:37.005 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 0e4c09c0d1c2c841... #1
98 @ 2020-03-21T06:37:37.000 signed by account [trxs: 0, lib: 183, conf: 0, latency: 5 ms]
<->Info 2020-03-21T06:37:37.506 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 0515ce04e119a505... #1
99 @ 2020-03-21T06:37:37.500 signed by account [trxs: 0, lib: 183, conf: 0, latency: 6 ms]
<->Info 2020-03-21T06:37:38.005 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 8bae6a58a5b0a2e4... #2
00 @ 2020-03-21T06:37:38.000 signed by account [trxs: 0, lib: 183, conf: 0, latency: 5 ms]
<->Info 2020-03-21T06:37:38.506 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 71875e7d99283314... #2
01 @ 2020-03-21T06:37:38.500 signed by account [trxs: 0, lib: 183, conf: 0, latency: 6 ms]
<->Info 2020-03-21T06:37:39.007 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block dcf4d49bad766de8... #2
02 @ 2020-03-21T06:37:39.000 signed by account [trxs: 0, lib: 183, conf: 0, latency: 7 ms]
<->Info 2020-03-21T06:37:39.507 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block df176c47bb6fb6c2... #2
03 @ 2020-03-21T06:37:39.500 signed by account [trxs: 1, lib: 183, conf: 0, latency: 7 ms]
<->Info 2020-03-21T06:37:40.046 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 7434b9e4b36c6453... #2
04 @ 2020-03-21T06:37:40.000 signed by account [trxs: 3, lib: 183, conf: 0, latency: 46 ms]
<->Info 2020-03-21T06:37:40.551 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block d9654f1322a65c18... #2
05 @ 2020-03-21T06:37:40.500 signed by account [trxs: 6, lib: 183, conf: 0, latency: 51 ms]
<->Info 2020-03-21T06:37:41.000 signed by account [trxs: 6, lib: 183, conf: 0, latency: 4 ms]
06 @ 2020-03-21T06:37:41.000 signed by account [trxs: 6, lib: 183, conf: 0, latency: 93 ms]
<->Info 2020-03-21T06:37:41.543 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 0499739b245a798d... #2
07 @ 2020-03-21T06:37:41.500 signed by account [trxs: 4, lib: 183, conf: 0, latency: 43 ms]
<->Info 2020-03-21T06:37:42.006 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 0a49028bb475c4de... #2
08 @ 2020-03-21T06:37:42.000 signed by account [trxs: 0, lib: 195, conf: 12, latency: 6 ms]
<->Info 2020-03-21T06:37:42.593 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block b85ccb061afa0d75... #2
09 @ 2020-03-21T06:37:42.500 signed by account [trxs: 6, lib: 195, conf: 0, latency: 6 ms]
<->Info 2020-03-21T06:37:43.000 signed by account [trxs: 0, lib: 195, conf: 0, latency: 5 ms]
10 @ 2020-03-21T06:37:43.000 signed by account [trxs: 0, lib: 195, conf: 0, latency: 5 ms]
<->Info 2020-03-21T06:37:43.506 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 6c7f664c7e5d03ad... #2
11 @ 2020-03-21T06:37:43.500 signed by account [trxs: 1, lib: 195, conf: 0, latency: 6 ms]
<->Info 2020-03-21T06:37:44.006 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 594de508ae8df7e... #2
12 @ 2020-03-21T06:37:44.000 signed by account [trxs: 0, lib: 195, conf: 0, latency: 6 ms]
<->Info 2020-03-21T06:37:44.525 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 1e8292c111b273b... #2
13 @ 2020-03-21T06:37:44.500 signed by account [trxs: 0, lib: 195, conf: 0, latency: 25 ms]

```

Figure 38: EOS 10 Transactions Submission Logs

```

accountc
File Edit View Search Terminal Help
<->Info 2020-03-21T06:38:26.004 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 3e65c1ce25e657ab... #2
96 @ 2020-03-21T06:38:26.000 signed by account [trxs: 0, lib: 279, conf: 0, latency: 4 ms]
<->Info 2020-03-21T06:38:26.504 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 2b2268719d32dfbb... #2
97 @ 2020-03-21T06:38:26.500 signed by account [trxs: 0, lib: 279, conf: 0, latency: 4 ms]
<->Info 2020-03-21T06:38:27.005 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 4c653f6caa76697... #2
98 @ 2020-03-21T06:38:27.000 signed by account [trxs: 0, lib: 279, conf: 0, latency: 5 ms]
<->Info 2020-03-21T06:38:27.532 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block b79efafb3fbf4e86... #2
99 @ 2020-03-21T06:38:27.500 signed by account [trxs: 0, lib: 279, conf: 0, latency: 32 ms]
<->Info 2020-03-21T06:38:28.042 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 246ab01b51371e3e... #3
00 @ 2020-03-21T06:38:28.000 signed by account [trxs: 4, lib: 279, conf: 0, latency: 42 ms]
<->Info 2020-03-21T06:38:28.507 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 583b5a2b5052e231... #3
01 @ 2020-03-21T06:38:28.500 signed by account [trxs: 0, lib: 279, conf: 0, latency: 7 ms]
<->Info 2020-03-21T06:38:29.005 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 71ee78d596cda22e... #3
02 @ 2020-03-21T06:38:29.000 signed by account [trxs: 0, lib: 279, conf: 0, latency: 5 ms]
<->Info 2020-03-21T06:38:29.508 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 197bd737279ab252... #3
03 @ 2020-03-21T06:38:29.500 signed by account [trxs: 0, lib: 279, conf: 0, latency: 8 ms]
<->Info 2020-03-21T06:38:30.005 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block f3492eadb790cf73... #3
04 @ 2020-03-21T06:38:30.000 signed by account [trxs: 0, lib: 279, conf: 12, latency: 5 ms]
<->Info 2020-03-21T06:38:30.507 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 130f8a8e8a9f71bd... #3
05 @ 2020-03-21T06:38:30.500 signed by account [trxs: 0, lib: 279, conf: 0, latency: 7 ms]
<->Info 2020-03-21T06:38:31.090 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 61fd734dff3371d0... #3
06 @ 2020-03-21T06:38:31.000 signed by account [trxs: 24, lib: 91, conf: 0, latency: 90 ms]
<->Info 2020-03-21T06:38:31.510 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block b7564441496c0462... #3
07 @ 2020-03-21T06:38:31.500 signed by account [trxs: 0, lib: 279, conf: 0, latency: 10 ms]
<->Info 2020-03-21T06:38:32.007 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 7a7fe7e1d061626c... #3
08 @ 2020-03-21T06:38:32.000 signed by account [trxs: 0, lib: 279, conf: 0, latency: 7 ms]
<->Info 2020-03-21T06:38:32.509 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block fd84db9169eb66c... #3
09 @ 2020-03-21T06:38:32.500 signed by account [trxs: 0, lib: 279, conf: 0, latency: 9 ms]
<->Info 2020-03-21T06:38:33.011 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block e4d27aab729dfc5... #3
10 @ 2020-03-21T06:38:33.000 signed by account [trxs: 0, lib: 279, conf: 0, latency: 11 ms]
<->Info 2020-03-21T06:38:33.548 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 3fed084f87ab8e9... #3
11 @ 2020-03-21T06:38:33.500 signed by account [trxs: 22, lib: 91, conf: 0, latency: 48 ms]
<->Info 2020-03-21T06:38:34.008 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 37711268f391084... #3
12 @ 2020-03-21T06:38:34.000 signed by account [trxs: 0, lib: 279, conf: 0, latency: 8 ms]
<->Info 2020-03-21T06:38:34.509 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block c189624c912a294f... #3

```

Figure 39: EOS 50 Transactions Submission Logs


```

accountc
File Edit View Search Terminal Help
<<-info 2020-03-21T06:39:03.005 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 26e058f901907025... #3
70 @ 2020-03-21T06:39:03.000 signed by accounti [trxs: 0, lib: 351, conf: 0, latency: 5 ms]
<<-info 2020-03-21T06:39:03.507 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 94a955a1267ef736... #3
71 @ 2020-03-21T06:39:03.500 signed by accounti [trxs: 0, lib: 351, conf: 0, latency: 7 ms]
<<-info 2020-03-21T06:39:04.007 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block a7f42d81a0541e64... #3
72 @ 2020-03-21T06:39:04.000 signed by accounti [trxs: 0, lib: 351, conf: 0, latency: 7 ms]
<<-info 2020-03-21T06:39:04.521 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 5f7ec2ae4814e0ac... #3
73 @ 2020-03-21T06:39:04.500 signed by accounti [trxs: 0, lib: 351, conf: 0, latency: 21 ms]
<<-info 2020-03-21T06:39:05.008 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block aed606c83e7ca003... #3
74 @ 2020-03-21T06:39:05.000 signed by accounti [trxs: 0, lib: 351, conf: 0, latency: 8 ms]
<<-info 2020-03-21T06:39:05.511 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block dee34767706aeac... #3
75 @ 2020-03-21T06:39:05.500 signed by accounti [trxs: 0, lib: 351, conf: 0, latency: 11 ms]
<<-info 2020-03-21T06:39:06.005 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 6c511ea850cb0871... #3
76 @ 2020-03-21T06:39:06.000 signed by accounti [trxs: 0, lib: 363, conf: 12, latency: 5 ms]
<<-info 2020-03-21T06:39:06.505 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 9c1c782e02ba67c6... #3
77 @ 2020-03-21T06:39:06.500 signed by accounti [trxs: 0, lib: 363, conf: 0, latency: 5 ms]
<<-info 2020-03-21T06:39:07.024 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 650f784e87efb523... #3
78 @ 2020-03-21T06:39:07.000 signed by accounti [trxs: 0, lib: 363, conf: 0, latency: 24 ms]
<<-info 2020-03-21T06:39:07.505 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 6f5d28ed8db1274... #3
79 @ 2020-03-21T06:39:07.500 signed by accounti [trxs: 0, lib: 363, conf: 0, latency: 5 ms]
<<-info 2020-03-21T06:39:08.005 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block a55e65be88a86f1b... #3
80 @ 2020-03-21T06:39:08.000 signed by accounti [trxs: 0, lib: 363, conf: 0, latency: 5 ms]
<<-info 2020-03-21T06:39:08.505 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 92b9840bf7d0a86a... #3
81 @ 2020-03-21T06:39:08.500 signed by accounti [trxs: 0, lib: 363, conf: 0, latency: 5 ms]
<<-info 2020-03-21T06:39:09.059 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block fb764503b77d5515... #3
82 @ 2020-03-21T06:39:09.000 signed by accounti [trxs: 1, lib: 363, conf: 0, latency: 59 ms]
<<-info 2020-03-21T06:39:09.664 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block f5e41445a8e2684a... #3
83 @ 2020-03-21T06:39:09.500 signed by accounti [trxs: 67, lib: 363, conf: 0, latency: 164 ms]
<<-info 2020-03-21T06:39:10.027 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 1cc0b1f85a8606f3... #3
84 @ 2020-03-21T06:39:10.000 signed by accounti [trxs: 0, lib: 363, conf: 0, latency: 27 ms]
<<-info 2020-03-21T06:39:10.505 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 6a10a1af58c830b3... #3
85 @ 2020-03-21T06:39:10.500 signed by accounti [trxs: 0, lib: 363, conf: 0, latency: 5 ms]
<<-info 2020-03-21T06:39:11.004 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 1696f0d191dde6ac... #3
86 @ 2020-03-21T06:39:11.000 signed by accounti [trxs: 0, lib: 363, conf: 0, latency: 4 ms]
<<-info 2020-03-21T06:39:11.505 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 902873c6ac8713fa... #3
87 @ 2020-03-21T06:39:11.500 signed by accounti [trxs: 0, lib: 363, conf: 0, latency: 5 ms]
<<-info 2020-03-21T06:39:12.006 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 0be19f42636735da... #3
88 @ 2020-03-21T06:39:12.000 signed by accounti [trxs: 0, lib: 375, conf: 12, latency: 6 ms]

```

Figure 41: EOS 100 Transactions Submission Logs - 1

```

accountc
File Edit View Search Terminal Help
<<-info 2020-03-21T06:39:28.525 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block f28777e141883df4... #4
21 @ 2020-03-21T06:39:28.500 signed by accounti [trxs: 0, lib: 399, conf: 0, latency: 25 ms]
<<-info 2020-03-21T06:39:29.039 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block a7a26ac781771048... #4
22 @ 2020-03-21T06:39:29.000 signed by accounti [trxs: 0, lib: 399, conf: 0, latency: 39 ms]
<<-info 2020-03-21T06:39:29.518 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 9e9897637616ea7e... #4
23 @ 2020-03-21T06:39:29.500 signed by accounti [trxs: 0, lib: 399, conf: 0, latency: 18 ms]
<<-info 2020-03-21T06:39:30.022 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block e4e91e7caa8a0ce6... #4
24 @ 2020-03-21T06:39:30.000 signed by accounti [trxs: 0, lib: 411, conf: 12, latency: 22 ms]
<<-info 2020-03-21T06:39:30.509 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block a166270f19bae8... #4
25 @ 2020-03-21T06:39:30.500 signed by accounti [trxs: 0, lib: 411, conf: 0, latency: 9 ms]
<<-info 2020-03-21T06:39:31.000 signed by accounti [trxs: 6, lib: 411, conf: 0, latency: 41 ms]
26 @ 2020-03-21T06:39:31.000 signed by accounti [trxs: 6, lib: 411, conf: 0, latency: 41 ms]
<<-info 2020-03-21T06:39:31.517 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block ab8a13a0f04d4485... #4
27 @ 2020-03-21T06:39:31.500 signed by accounti [trxs: 0, lib: 411, conf: 0, latency: 17 ms]
<<-info 2020-03-21T06:39:32.019 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block abd68e315b5639c... #4
28 @ 2020-03-21T06:39:32.000 signed by accounti [trxs: 0, lib: 411, conf: 0, latency: 19 ms]
<<-info 2020-03-21T06:39:32.500 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block d9ecb280f9b4e5e... #4
29 @ 2020-03-21T06:39:32.500 signed by accounti [trxs: 0, lib: 411, conf: 0, latency: 8 ms]
<<-info 2020-03-21T06:39:33.007 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 3018f8b967497eb7... #4
30 @ 2020-03-21T06:39:33.000 signed by accounti [trxs: 0, lib: 411, conf: 0, latency: 7 ms]
<<-info 2020-03-21T06:39:33.511 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block d096b3c1c2ee00b0... #4
31 @ 2020-03-21T06:39:33.500 signed by accounti [trxs: 0, lib: 411, conf: 0, latency: 11 ms]
<<-info 2020-03-21T06:39:34.306 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block e2d647520ef2e048... #4
32 @ 2020-03-21T06:39:34.000 signed by accounti [trxs: 25, lib: 411, conf: 0, latency: 306 ms]
<<-info 2020-03-21T06:39:34.673 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 029b4d2856158267... #4
33 @ 2020-03-21T06:39:34.500 signed by accounti [trxs: 51, lib: 411, conf: 0, latency: 173 ms]
<<-info 2020-03-21T06:39:35.082 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 03f89b0b3e5362d1... #4
34 @ 2020-03-21T06:39:35.000 signed by accounti [trxs: 18, lib: 411, conf: 0, latency: 82 ms]
<<-info 2020-03-21T06:39:35.506 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block a84e61305c490b35... #4
35 @ 2020-03-21T06:39:35.500 signed by accounti [trxs: 0, lib: 411, conf: 0, latency: 6 ms]
<<-info 2020-03-21T06:39:36.006 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 8ee6bbdb98a2af19... #4
36 @ 2020-03-21T06:39:36.000 signed by accounti [trxs: 0, lib: 423, conf: 12, latency: 6 ms]
<<-info 2020-03-21T06:39:36.507 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block cf5b46d6b89e2cee... #4
37 @ 2020-03-21T06:39:36.500 signed by accounti [trxs: 0, lib: 423, conf: 0, latency: 7 ms]
<<-info 2020-03-21T06:39:37.007 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 1539ca2f6d5a25fa... #4
38 @ 2020-03-21T06:39:37.000 signed by accounti [trxs: 0, lib: 423, conf: 0, latency: 7 ms]
<<-info 2020-03-21T06:39:37.510 thread-0 producer_plugin.cpp:345 on_incoming_block ] Received block 607b02c9f496f0c5... #4
39 @ 2020-03-21T06:39:37.500 signed by accounti [trxs: 0, lib: 423, conf: 0, latency: 10 ms]

```

Figure 40: EOS 100 Transactions Submission Logs - 2

CHAPTER 5

CONCLUSION

In conclusion, two of the main Blockchain development platforms were studied in this masters' thesis. Providing a detailed performance study is needed in order to guide developers in this recently found technology. A decentralized application along with a corresponding smart contract were written for each of the platforms. The DApps measure the timing of the basic operations one could use when interacting with a Blockchain. Each of the DApps was tested along with randomly generated networks on both Microsoft Azure and the HPC cluster Octopus of the American University of Beirut. Though Azure's numbers are lower, the experimentation in both environments show that Ethereum performs better than EOS with the increase of the number of transactions submitted to the network.

In a time where the markets of cryptocurrencies are constantly fluctuating between bear and bull markets, marketing plays a big role. Though EOS.IO has a huge potential in the crypto community and is highly marketed, the results show that EOS.IO does not outperform Ethereum.

Both platforms however contributed a lot to the innovation of Blockchain systems. Decentralized Computing is quite a new field, and so many researches are still needed in order to better assess performance and scalability in Blockchain systems.

BIBLIOGRAPHY

- [1] “List of highest-funded crowdfunding projects - Wikipedia.” https://en.m.wikipedia.org/wiki/List_of_highest-funded_crowdfunding_projects (accessed Feb. 24, 2020).
- [2] “State of the DApps — A list of 3,118 blockchain apps for Ethereum, Steem, Hive, EOS, and more.” <https://www.stateofthedapps.com/> (accessed Apr. 06, 2020).
- [3] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, “On the Security and Performance of Proof of Work Blockchains,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, Oct. 2016, pp. 3–16, doi: 10.1145/2976749.2978341.
- [4] S. Rouhani and R. Deters, “Performance analysis of ethereum transactions in private blockchain,” in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Nov. 2017, pp. 70–74, doi: 10.1109/ICSESS.2017.8342866.
- [5] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, “Performance Analysis of Private Blockchain Platforms in Varying Workloads,” in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, Jul. 2017, pp. 1–6, doi: 10.1109/ICCCN.2017.8038517.
- [6] M. Valenta and P. Sandner, “comparison of ethereum hyperledger fabric and corda,” 2017.
- [7] M. Scherer, “Performance and Scalability of Blockchain Networks and Smart Contracts,” p. 46.
- [8] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, “BLOCKBENCH: A Framework for Analyzing Private Blockchains,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, Chicago, Illinois, USA, May 2017, pp. 1085–1100, doi: 10.1145/3035918.3064033.
- [9] K. Veskus and F. Milani, “Ethereum versus Fabric – A comparative analysis,” 2018.
- [10] Q. T. Zhong and Z. Cole, “Analyzing the Effects of Network Latency on Blockchain Performance and Security Using the Whiteblock Testing Platform.”
- [11] B. Xu, D. Luthra, Z. Cole, and N. Blakely, *Eos: An architectural, performance, and economic analysis*. Bitmex. Retrieved from <https://www.whiteblock.io/library/eos-test-report.pdf>, 2018.
- [12] A. Aldweesh, M. Alharby, and A. van Moorsel, “Performance Benchmarking for Ethereum Opcodes,” in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, Oct. 2018, pp. 1–2, doi: 10.1109/AICCSA.2018.8612882.
- [13] A. Aldweesh, M. Alharby, E. Solaiman, and A. van Moorsel, “Performance Benchmarking of Smart Contracts to Assess Miner Incentives in Ethereum,” in *2018 14th European Dependable Computing Conference (EDCC)*, Sep. 2018, pp. 144–149, doi: 10.1109/EDCC.2018.00034.
- [14] “Hyperledger Caliper – Hyperledger.” <https://www.hyperledger.org/projects/caliper> (accessed Feb. 24, 2020).

- [15] A. Aldweesh, M. Alharby, M. Mehrnezhad, and A. Van Moorsel, “OpBench: A CPU Performance Benchmark for Ethereum Smart Contract Operation Code,” in *2019 IEEE International Conference on Blockchain (Blockchain)*, Jul. 2019, pp. 274–281, doi: 10.1109/Blockchain.2019.00043.
- [16] “White Paper · ethereum/wiki Wiki · GitHub.” <https://github.com/ethereum/wiki/wiki/White-Paper#ethereum> (accessed Apr. 06, 2020).
- [17] “Cert Spotter - Timeline of PKI Security Failures.” <https://ssllmate.com/certspotter/failures> (accessed Apr. 07, 2020).
- [18] “The Byzantine Generals Problem | ACM Transactions on Programming Languages and Systems.” <https://dl.acm.org/doi/10.1145/357172.357176> (accessed Apr. 07, 2020).
- [19] “Nakamoto, Satoshi. ‘Bitcoin: A peer-to-peer electronic cash system.’ (2008). - Google Search.” [https://www.google.com/search?q=Nakamoto%2C+Satoshi.+%22Bitcoin%3A+A+peer-to-peer+electronic+cash+system.%22+\(2008\).&oq=Nakamoto%2C+Satoshi.+%22Bitcoin%3A+A+peer-to-peer+electronic+cash+system.%22+\(2008\).&aqs=chrome..69i57.168j0j9&sourceid=chrome&ie=UTF-8](https://www.google.com/search?q=Nakamoto%2C+Satoshi.+%22Bitcoin%3A+A+peer-to-peer+electronic+cash+system.%22+(2008).&oq=Nakamoto%2C+Satoshi.+%22Bitcoin%3A+A+peer-to-peer+electronic+cash+system.%22+(2008).&aqs=chrome..69i57.168j0j9&sourceid=chrome&ie=UTF-8) (accessed Apr. 07, 2020).
- [20] “Script - Bitcoin Wiki.” <https://en.bitcoin.it/wiki/Script> (accessed Apr. 11, 2020).
- [21] “Chart of the Day: Bitcoin Reward Halving and Price History | Infographics | ihodl.com.” <https://ihodl.com/infographics/2018-04-09/chart-day-bitcoin-reward-halving-and-price-history/> (accessed Apr. 11, 2020).
- [22] “Controlled supply - Bitcoin Wiki.” https://en.bitcoin.it/wiki/Controlled_supply (accessed Apr. 11, 2020).
- [23] “The Dangers of Mining Pools: Centralization and Security Issues.” <https://cointelegraph.com/news/the-dangers-of-mining-pools-centralization-and-security-issues> (accessed May 02, 2020).
- [24] “Namecoin.” <https://www.namecoin.org/> (accessed Apr. 11, 2020).
- [25] “Ethereum - Wikipedia.” <https://en.wikipedia.org/wiki/Ethereum> (accessed May 03, 2020).
- [26] “ETH Gas Station | Consumer oriented metrics for the Ethereum gas market.” <https://ethgasstation.info/index.php> (accessed Mar. 10, 2020).
- [27] “Solidity - Wikipedia.” <https://en.wikipedia.org/wiki/Solidity> (accessed May 03, 2020).
- [28] “Consensus Protocol | EOSIO Developer Docs.” https://developers.eos.io/welcome/latest/protocol/consensus_protocol (accessed Mar. 06, 2020).
- [29] “EOS Tracker | Real time viewer for EOSIO Blockchains.” <https://eostracker.io/producers> (accessed Mar. 06, 2020).

- [30] “EOSIO RAM Market & Bancor Algorithm - Daniel Larimer - Medium.” <https://medium.com/@bytemaster/eosio-ram-market-bancor-algorithm-b8e8d4e20c73> (accessed May 11, 2020).
- [31] “Network Peer Protocol | EOSIO Developer Docs.” https://developers.eos.io/welcome/latest/protocol/network_peer_protocol (accessed Mar. 08, 2020).
- [32] “Boost.MultiIndex Documentation - Index - 1.62.0.” https://www.boost.org/doc/libs/1_62_0/libs/multi_index/doc/index.html (accessed May 12, 2020).
- [33] “Blockstack.” <https://blockstack.org/> (accessed May 30, 2020).
- [34] “<https://muneebali.com/thesis>.” <https://muneebali.com/thesis>.
- [35] “Blockstack,” *Ali, Muneeb, Jude C. Nelson, Ryan Shea, and Michael J. Freedman. “Blockstack: A Global Naming and Storage System Secured by Blockchains.” In USENIX Annual Technical Conference, pp. 181-194. 2016.*
- [36] “Hyperledger – Open Source Blockchain Technologies.” <https://www.hyperledger.org/> (accessed May 30, 2020).
- [37] “A Blockchain Platform for the Enterprise — hyperledger-fabricdocs master documentation.” <https://hyperledger-fabric.readthedocs.io/en/release-1.3/> (accessed May 30, 2020).
- [38] “Apache Kafka.” <https://kafka.apache.org/> (accessed May 30, 2020).
- [39] “User Guide — Singularity container 3.5 documentation.” <https://sylabs.io/guides/3.5/user-guide/> (accessed May 18, 2020).
- [40] “Octopus user guide (Mixed architecture virtualized Beowulf cluster) — hpc user guide master documentation.” https://hpc-aub-users-guide.readthedocs.io/en/latest/octopus/octopus_index.html (accessed May 15, 2020).
- [41] “random-tree - npm.” <https://www.npmjs.com/package/random-tree> (accessed May 18, 2020).
- [42] “js-graph-algorithms - npm.” <https://www.npmjs.com/package/js-graph-algorithms> (accessed May 18, 2020).
- [43] “enode url format · ethereum/wiki Wiki · GitHub.” <https://github.com/ethereum/wiki/wiki/enode-url-format> (accessed May 20, 2020).
- [44] “More Than A Billion Downloads of Node.js 🎉 - Node.js - Medium.” <https://medium.com/@nodejs/more-than-a-billion-downloads-of-node-js-952a8a98eb42> (accessed May 16, 2020).
- [45] “web3 - npm.” <https://www.npmjs.com/package/web3> (accessed May 18, 2020).
- [46] “eosjs - npm.” <https://www.npmjs.com/package/eosjs> (accessed May 18, 2020).
- [47] “timely - npm.” <https://www.npmjs.com/package/timely> (accessed May 18, 2020).
- [48] “Graphviz - Graph Visualization Software.” <https://www.graphviz.org/> (accessed May 20, 2020).
- [49] “BIOS Boot Sequence | EOSIO Developer Docs.” <https://developers.eos.io/welcome/latest/tutorials/bios-boot-sequence/> (accessed Mar. 02, 2020).

[50] “How do I know how much RAM I need to deploy a contract? · Issue #4979 · EOSIO/eos · GitHub.” <https://github.com/EOSIO/eos/issues/4979> (accessed May 21, 2020).