Proof of Stake Blockchain: Performance and Scalability for Groupware Communications

Conference Paper · November 2017

DOI: 10.1145/3167020.3167058

2 authors:

Jason Spasovski
IT University of Copenhagen
1 PUBLICATION **3** CITATIONS
SEE PROFILE

Peter W. Eklund
Deakin University
211 PUBLICATIONS **1,845** CITATIONS
SEE PROFILE

Proof of Stake Blockchain: Performance and Scalability for Groupware Communications

Jason Spasovski and Peter Eklund
IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen, Denmark
jspa@itu.dk, petw@itu.dk

ABSTRACT

A blockchain is a distributed transaction ledger, a disruptive technology that creates new possibilities for digital ecosystems. The blockchain ecosystem maintains an immutable transaction record to support many types of digital services. This paper compares the performance and scalability of a web-based groupware communication application using both non-blockchain and blockchain technologies. Scalability is measured where message load is synthesized over two typical communication topologies. The first is 1 to n network – a typical client-server or star-topology with a central vertex (server) receiving all messages from the remaining $n-1$ vertices (clients). The second is a more naturally occurring scale-free network topology, where multiple communication hubs are distributed throughout the network. System performance is tested with both blockchain and non-blockchain solutions using multiple cloud computing configurations. We analyze the empirical results from each configuration to identify the costs and overhead of blockchain technology.

General Terms

Groupware communication, distributed ledger ecosystems, blockchain, scale-free networks, distributed & cloud computing.

1. INTRODUCTION

Security and scalability are chief concerns when providing data storage and communication solutions in a transactional digital ecosystem. Recent security vulnerabilities exposed in well-established companies, e.g. ebay [1] and Sony [2] demonstrate the consequences of security failures in commercial communication software. The blockchain, or distributed ledger, is a way of decentralizing database control. Likewise, the importance of scalability is particularly visible in web-based applications, system performance is critical to the growth of a user base [14].

Collaboration tools are on-line services used by large audiences and are characterized by countless database changes, with the addition of other special requirements, such as the need for low response times and confidential data storage. These requirements prove to be a challenge to both security and scalability.

One such collaboration tool is the web application DALLR [3]. The first author Spasovski (with Andreassen and Lyck) developed DALLR to create a simple intuitive platform with a flat learning curve to target small to medium sized companies. DALLR is a groupware communication application that provides the platform for our empirical comparison.

A major threat to any database are *tampering* attacks. In such attacks the database is partly or completely controlled by an unauthorized third party. Database tampering can be performed in many ways, each reflecting the possible severity of the attack, depending on the level of privileges held by the adversary. As a concrete example, if an adversary were to use SQL injection, they would most likely only be able to tamper with the database with minimal privileges, whereas if an adversary gains access to an administrator account, the entire database could be deleted.

A common approach to deal with database tampering is auditing [12], a collection of methods to verify the integrity of the database by simple procedures such as inserted checksums. The main concern with auditing is that it does not protect against an adversary gaining access to the administrator privileges. An alternative to deal with database tampering is to use an immutable database, such as Datomic[4]. Immutable databases lift the modify and delete functionality, thus making them immutable by design and allow for extreme scalability and performance.

The current trending method to deal with data tampering is by using a decentralized *blockchain* [15] – a distributed cryptographic ledger. Blockchain is a distributed database consisting of linked blocks that contain timestamped and valid transactions. These blocks are linked using the hash

[1] "eBay faces investigations over data breach", BBC News, May, 2014, see `http://www.bbc.com/news/technology-27539799`

[2] "Sony PlayStation suffers massive data breach", Reuters, April, 2011, see `http://www.reuters.com/article/us-sony-stoldendata-idUSTRE73P6WB20110426`

[3] J. Spasovski, A. Lyck, J. Andreassen. *Collaboration without a learning curve* [Software], see http://www.dallr.com.

[4] Pavlou, Kyriacos E., and Richard T. Snodgrass *The fully transactional, cloud-ready, distributed database.* [Software], see `http://www.datomic.com/`.

MEDES 2017

of the previous block, creating a chain of blocks where each block reinforces the integrity of its predecessor. Chaining blocks in this cryptological manner prevents data stored in the blocks – the blockchain – to be easily tampered. Decentralization of the blockchain adds an extra layer of security by distributing the blockchain over multiple decentralized participating network nodes, each containing a copy of the blockchain. In this way, the blockchain cannot be controlled or exploited through a single point of attack on any single node.

Blockchain has few benefits over traditional databases while having multiple trade offs, therefore it is important that the motivation for using a blockchain is worth the trade offs. Our motivation for using a blockchain is primarily to protect against database tampering while other use cases could be motivated by the decentralized blockchain's ability to remove the need for a middle man to authorize transactions.

A large body of work [15, 10, 7] demonstrates that decentralized blockchain allows for strong irrefutability and auditable data storage. In most cases, when using a blockchain to increase the security of a web application, the scalability and performance of the application is significantly decreased [6]. This is due to the overhead of the different consensus algorithms that secure the blockchain.

This paper is structured as follows. Section 2 presents a survey of the basic theory behind blockchain, as well as describing some of the mechanisms used in implementations. Section 3 presents the technologies and architecture used and give details of the implementation of both blockchain and non-blockchain versions of DALLR. In Section 4, we present the test scenarios and cloud environments used throughout the tests. In Section 5 experimental results are presented focusing particularly on response time and throughput. In Section 6 we confront expectations with the experimental findings, thereafter we analyze each of the results attained. Lastly, in Section 7 we conclude with a brief summary of the experimental results and analysis.

2. BLOCKCHAIN

Consensus algorithms are used within blockchains to ensure that the participating nodes in the distributed system agree with the current state of the blockchain before each new block is added [5]. Blockchain consensus algorithms are *Byzantine Fault Tolerant* meaning no single machine can succeed in being malicious against the distributed system [9].

Bitcoin is the prime example of a cryptocurrency powered blockchain that allows users to submit transactions without the need for a central trusted organization; this is achieved using a consensus algorithm called *Proof of Work* [10]. This algorithm uses mining to prove consensus, which requires a large amount of energy and therefore enforces slow transaction times.

Proof of Stake is a more efficient alternative to Proof of work where no mining is required to achieve consensus but instead the next node to create a block is selected proportional to its stake in the blockchain. The stake can vary depending on how the blockchain has implemented its proof. In cryptocurrency blockchains, the stake is normally the number of coins a node holds. In non-cryptocurrency blockchains alternative approaches such as voting are required as no stake exists by default.

Merkle trees are a common data structure for storing blockchain data and implemented using an AVL tree, also refereed to as a self balancing binary search tree. Merkle trees have one main difference from normal tree-based data structures: a value in the tree can always be verified by rehashing all the way from the selected node upwards to the root of the tree, which will take O(log n) time. On every insert the the trail of the inserted node is rehashed from where the node is inserted all the way to the root node.

All blockchains fall into one of two categories, namely Public or Private. Public-blockchains are those that allow all users on the network to view the data you store, while private blockchains hide all data stored on the blockchain between its permissioned participants. Private blockchains can be either fully private or consortium blockchains. Read and write permissions of a fully private blockchain are controlled by a single organization, which can be viewed as a centralized database with added cryptographic security. Consortium blockchains are where the read and write permissions are distributed across a permissioned consortium which adds the extra security of decentralization.

3. IMPLEMENTATION

Figs. 1 and 2 show User 1 sending a message to User 2. Fig. 1 does not use a blockchain while in Fig. 2 a blockchain is used.

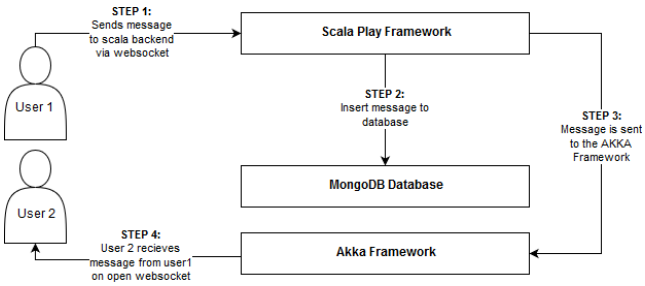

Figure 1: Flow diagram of User 1 sending a message to User 2 without a blockchain.

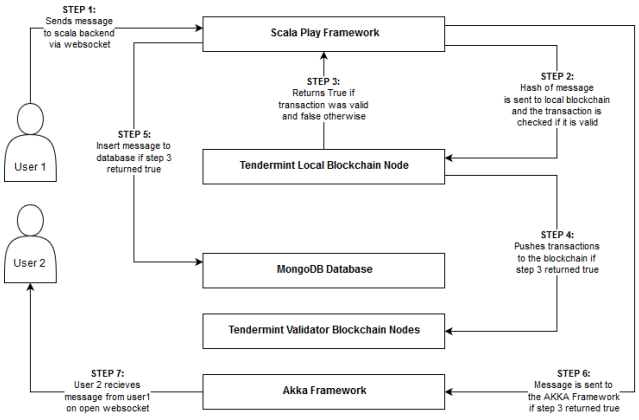

Figure 2: Flow diagram of User 1 sending a message to User 2 with a blockchain.

The blockchain used in our implementation is the private consortium blockchain named Tendermint[5]. Tendermint is

[5]`http://tendermint.com` version 0.9.

a relatively lightweight blockchain solution since its consensus method is proof of stake using a voting mechanism [2] as opposed to the more computational expensive proof of work [8].

Two different types of nodes exist in the tendermint blockchain, namely validator nodes and non-validator nodes. Validator nodes are part of the consortium which vote to agree upon consensus while non-validator nodes are restricted to reading and proposing transactions on the blockchain. All nodes in the tendermint blockchain communicate over a persistent encrypted TCP p2p gossip protocol.

From a design perspective, a straight-forward method of using a blockchain is to store all data in the blockchain. Storing all data in a blockchain has one large drawback; namely if all data stored on the blockchain is immutable, consequently no data can ever be removed. This in turn causes the blockchain to grow so large that it can become impractical to store.

Growth of the blockchain is inevitable but the rate of growth can easily be managed. All data in our implementation is stored in the document database MongoDB database[6], while only a hash of the data entries are saved in the blockchain. The hashes saved in the blockchain are used as a checksum to confirm the validity of the data stored in the database. This method of maintaining all data in the database has some vulnerabilities which are not addressed in this paper. For each individual document retrieved from the database, the blockchain is queried with the according hashes, thus confirming the validity the data retrieved from the database. The blockchain validity of data is always sent to the user alongside the data retrieved from the database.

In order for the application to communicate with the blockchain validator nodes each application server runs a local version of the blockchain. This local version is in a non-validator node which is synchronized with the blockchain and pushes new transactions to validator nodes, but is not involved in consensus. We chose this design to allow the validator nodes to focus on adding transactions with consensus rather than responding to blockchain queries, which would otherwise slow the consensus process. We adopted an asynchronous approach when pushing new transactions to the blockchain which allows users the ability to view transactions before consensus is agreed upon by the validators. When adding a new transaction a constant time function is run on the local blockchain returning true if the local blockchain verifies the transaction as valid. The local blockchain node gossips with the blockchain validators and – normally after a second – consensus will be agreed and the new transaction will be added to the blockchain, hereafter the user will receive confirmation that the transaction has been validated [3].

4. TESTING

4.1 Testing Scenarios

The two testing scenarios have the primary goal of simulating a large number of users simultaneously communicating via the application. The difference between the two scenarios lies in the form in which communication between the users is initiated. The first test uses a realistic messaging pattern based on a scale-free network graph. Network theory and its application have shown that communication in communities in the real-world follow a scale-free communication pattern [1]. The second test simulates a pathological client-server scenario where a single node (server or authority) receives messages from the entire client-network. We run each test scenario for a total of 60 seconds including a 10 second ramp-up period in which all threads are started and hereafter run concurrently. The timeout limit for all requests is 20 seconds throughout all tests. *Response time* and *throughput* are the chosen metrics, when testing both `SendMessage` and `RetrieveAllMessages` functions. Both the non-blockchain and blockchain implementations are tested for each of the two testing scenarios.

Scale-Free Network Topology

We create the first testing scenario by applying network theory, more specifically scale-free network theory to produce scale-free networks. A graph is scale-free if the degree distribution of the neighbor list of its vertices is roughly power-law. As a concrete example, the French instant messaging community NIOKI [1] proved to be scale-free.

We use the Barabasi-Albert model to build our produce power-law graphs. The Barabasi-Albert Model is described by the formula:

$$p_i = \frac{k_i}{\sum_j k_j}$$

This formula states that the probability of a newly created vertex being connected to vertex $[i]$ is the degree of vertex $[i]$ divided by the sum of the degree of every individual remaining vertex. [7].

Each user is represented as a vertex and a message sent from one user to another is represented as an (undirected) edge. A node with relatively high degree is denoted as a *hub*. Illustration of hubs (marked by a black dot) can be seen in in Fig. 3. A hub in scale-free network can be thought of as role in an organization: hubs may represent the CEO of a company, the vice-chancellor of a University or the faculties manager of a building.

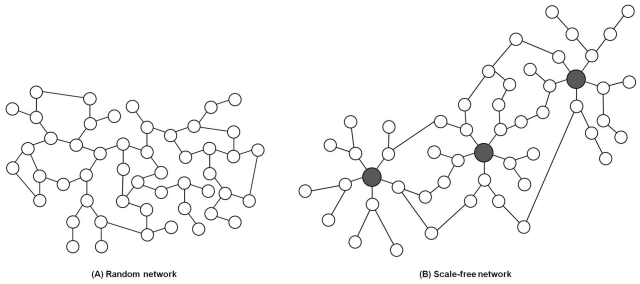

Figure 3: Random network vs Scale-free network

The test runs for 60 seconds in which it receives the number of users as input and creates that many threads (users) that run in parallel. The Scale-Free Network graph is divided over multiple files which are concurrently read by each user, these files contain a single edge on each line. For each line in the graph file, the user sends a message to the according user using the `SendMessage` function. Thereafter

[6] `https://www.mongodb.com`

[7] We produce the scale-free network graphs using a Java implementation of the Barabasi-Albert Model named graphstream `http://graphstream-project.org/`

the user has a 10% chance to retrieve all messages using the `RetrieveAllMessage` function, followed by a 300ms second delay. If the graph file reaches the end it will wait for (users $\times$ 2)ms before returning to the start of the file, this is done to reduce the throughput of users with fewer edges.

1 to n Topology

We now describe the second testing scenario, namely 1 to n (or star graph). By forcing all users to send their messages to a single user (User 1), we replicate a very heavy load being put onto a single user – as may occur in DoS or DDoS attack. By comparing the load on the heavily loaded user with the remaining users on both implementations, we emphasize the blockchain's ability to handle centralized stress.

Similarly to the previous case, the test runs for 60 seconds. In that timeframe, the test takes the number of users as input and creates that many threads (users) that run in parallel. Each user concurrently sends a message to User 1 using the `SendMessage` function, followed by 300ms delay. Hereafter the user has a 10% chance to retrieve all messages using the `RetrieveAllMessage` function, followed by a 300ms second delay.

4.2 Testing Environment

As seen in Figure 4 all requests are sent using 10 slave testing machines controlled by the Master testing machine. Each slave testing machine concurrently sends requests to the Application Load balancer which distributes the requests across all the application servers. The blockchain implementation will run a local blockchain that is used to communicate and keep in sync with the validator blockchains. A single database server is shared between all application servers.

Testing Machines

The tests[8] run up to 800 concurrent users for over a minute, thus testing on a single machine will break the tests as the JVM garbage collector (GC) is likely to run mid test and pollute the test results [9]. All tests are distributed over 10 slave test machines. All test machines are hosted on the Azure cloud within the same subnet and the master machine is controlled via remote desktop.

Each slave test machine is an Azure D13 instance running with 8 Intel Xeon E5-2660 2.2 GHz Cores and 56GB RAM running Debian version 8.7. The master test machine runs on a smaller D3 instance, as less resources are required to control the slaves. The resources of the testing machines are monitored throughout all tests to ensure they do not cause inconsistent test results.

Blockchain Validator Nodes

Throughout our tests the 10 blockchain validator nodes are spread over three different geographic regions (Amsterdam, London and Frankfurt) on the Digital Ocean cloud. By doing so we attempt to replicate a real-world scenario where the blockchain nodes would be distributed between a consortium of organizations. The validator nodes are hosted on dual core virtual machines with 2GB of RAM, SSD storage and running Ubuntu version 14.04. In addition, we monitored the CPU and RAM usage to ensure that the servers hosting the blockchains are appropriately provisioned.

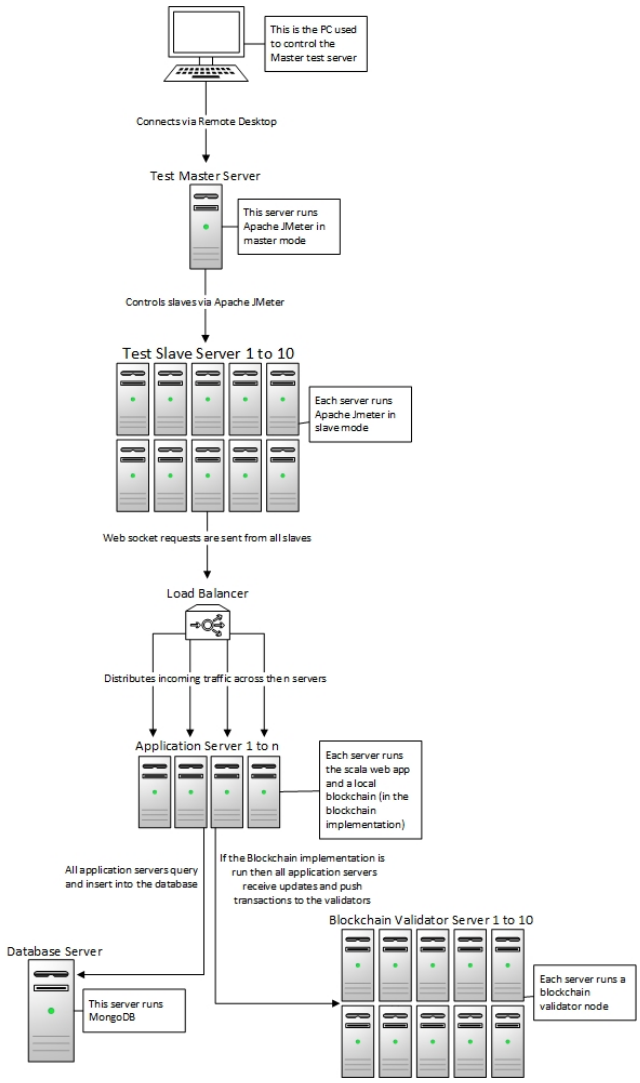

Figure 4: Overview of testing environment.

Buchman [3] ran detailed tests on tendermint which showed that adding more validators both lowers the throughput and increases the latency. Buchman first ran 64 validators and reached a throughput of 4,000 transactions per second with latencies of 2 seconds, and then 8 validators with 9,000 transactions per second and latencies of 1.5 seconds. The reason for the high latencies in his tests is due to the high spread of locations of the testing validators. Due to the asynchronous architecture of our application, our tests are not affected by validator latencies. It follows that changing the number of validators does not directly affect our test results but could affect the overall user experience.

[8] We ran the tests using Apache JMeter, see `http://jmeter.apache.org`

[9] The min and max heap of the JVM is set to be 52GB and the young generation (Eden) of the heap to be as large as possible to ensure the GC does not pollute the results. A major GC won't trigger until Eden starts to get full. The verbose gc setting was used throughout all tests to output all garbage collection information.

Cloud Computing Configurations

The Microsoft Azure cloud computing service is used for all tests to host both the application, test and database servers instances. The number of instances that run the application and local blockchain are the only factors that change within the testing environment. Throughout all tests we use a single Azure G4 instance to run our database which has a 16 core Intel Xeon E5 v3 2.0 GHz processor,224GB of RAM and SSD storage.

The Azure scale set is used to scale the application servers using replica instances which run the DALLR application, alongside a local version of the blockchain (only if the blockchain implementation is being tested) [10].

The Azure scale set uses D3 instances that have 4 Intel Xeon E5-2660 2.2 GHz cores and 13GB of ram and use SSD hard drives. The D3 instance was chosen because the application and local blockchain would not run adequately on the smaller D1 or D2 instances due to lack of hardware provision.

The tests are started initially with one instance and a gradually increasing load until the test results show average response times of over a second. After these average response times of over a second appear the number of instances is doubled and the tests are restarted and run until an average response time of over a second appears again. This process is be repeated until an average response time of over 1 second appears on 8 test instances. Both implementations are tested on the four cloud configurations consisting of 1, 2, 4 and 8 instances.

5. EXPERIMENTAL RESULTS

This section presents experimental results in the form of line graphs focusing on response time and throughput. Ramsay et. al [13] noted negative user behavior with response times of 200ms or higher. Throughout all our results we focus on maintaining a response time of less than 200ms, and decide not to display results over 350ms averaged.

In Section 5.1 we present experimental results measuring response time over various cloud configurations. In Section 5.2 we present experimental results measuring throughput on a single cloud instance. Lastly, in Section 5.3 we present experimental results measuring response time of heavy stress loads through a single user.

5.1 Response Time

This section contains line graphs comparing the two implementations (blockchain and non-blockchain) in terms of different user loads using the scale-free network and 1 to n test scenarios. The two messaging functions tested in each testing scenario are the `RetrieveAllMessages` function and the `SendMessage`. These two functions combined with the two implementations give four different combinations to test. The metric to measure scalability in this section is the Average response time over a minute.

5.2 Throughput

This section focuses on one of the two messaging functions `RetrieveAllMessages` and `SendMessage` while combining the blockchain implementation & non-blockchain implementation with the scale-free testing scenario & 1 to n testing scenario to give four different testing combinations for each line graph. The metric used to measure scalability in this section is the average transactions per second over a minute.

[10]All instances on the Azure scale set are running the latest stable release of Debian version 8.7.

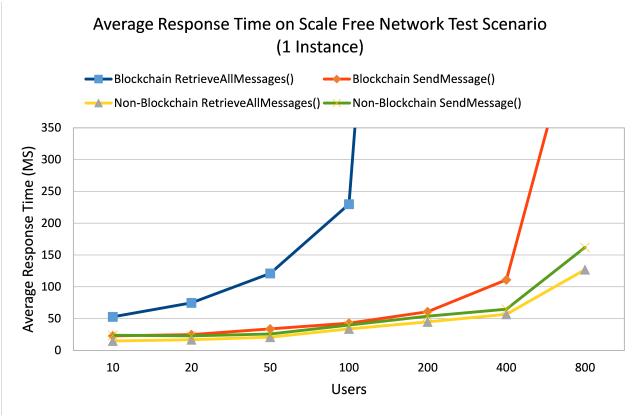

Figure 5: Scale-Free Network Test: compares `RetrieveAllMessages` against `SendMessage` for both implementations on a single D3 (4 Core 13GB RAM) instance

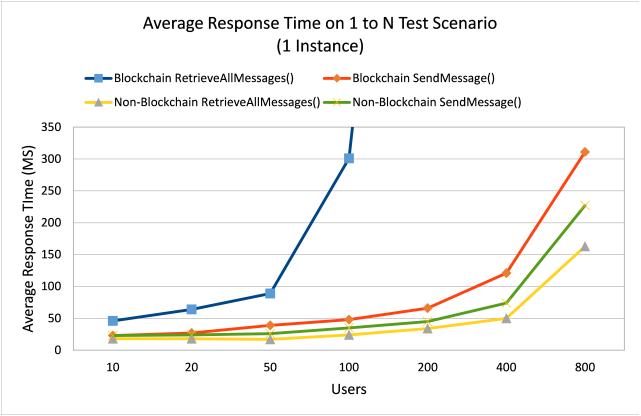

Figure 6: 1 to n Test: compares `RetrieveAllMessages` against `SendMessage` on both implementations (blockchain & non-blockchain) on a single D3 (4 Core 13GB RAM) instance

5.3 User 1 vs User n

In this section *User 1* represents the heavily loaded user that receives all messages in the 1 to n test scenario while *User n* represents each of the remaining $n-1$ users. The users in this test are represented as a rooted tree where each user is a node and each edge is a message sent. The user receiving all messages is the the root node of the rooted tree and all the remaining users are the leaf nodes. In this section we will use the terminology of a *root node* to represent User 1 and a *leaf node* to represent the remaining $n-1$ users. This test compares the average response times of the `RetrieveAllMessages` function being called on the blockchain and non-blockchain implementations. This test proves how the two implementations handle the stress of a single heavily loaded user.

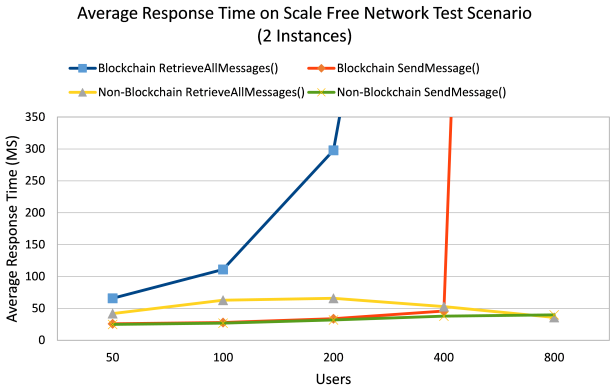

Figure 7: Scale-Free Network Test: compares `RetrieveAllMessages` against `SendMessage` on both implementations (blockchain & non-blockchain) on 2 x D3 (4 Core 13GB RAM) instances

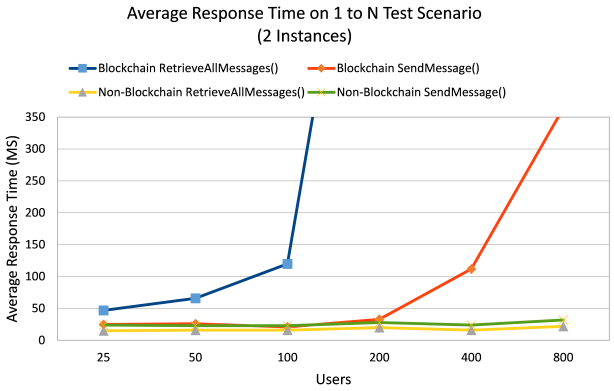

Figure 8: 1 to n Test: compares `RetrieveAllMessages` against `SendMessage` on both implementations (blockchain & non-blockchain) on 2 x D3 (4 Core 13GB RAM) instances

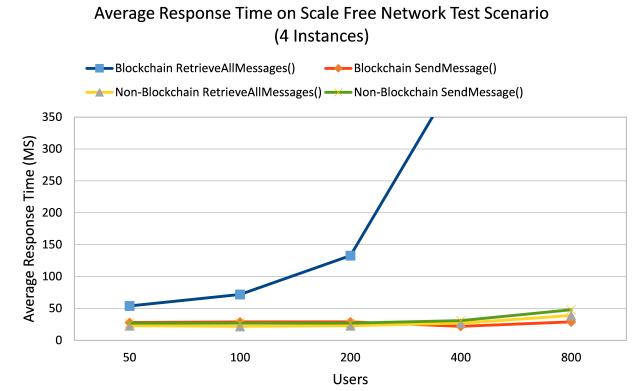

Figure 9: Scale-Free Network Test: compares `RetrieveAllMessages` against `SendMessage` on both implementations (blockchain & non-blockchain) on four x D3 (4 Core 13GB RAM) instances

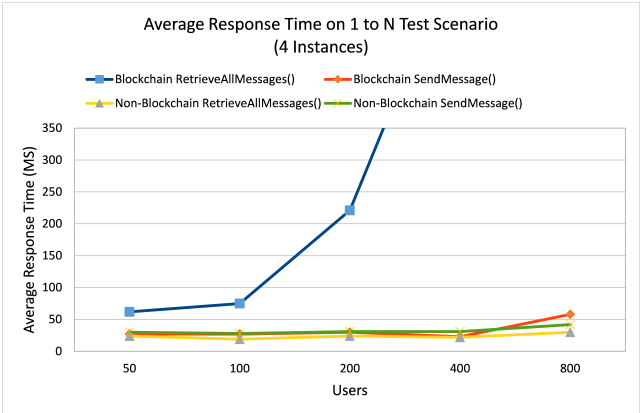

Figure 10: 1 to n Test: compares `RetrieveAllMessages` against `SendMessage` on both implementations (blockchain & non-blockchain) on four x D3 (4 Core 13GB RAM) instances

6. ANALYSIS

6.1 Expectations

We anticipated that the function `RetrieveAllMessages` would be the bottleneck on the blockchain implementation. This is due to the large number of queries the blockchain performs within the function. The blockchain data is stored using merkle tree data structure, and thus it takes $O(log\ n)$ [3] time to search a block containing n transactions. The function `RetrieveAllMessages` retrieves all the users messages (m) from the database, and then queries the blockchain on each of them. This implies a running time of $O(m\ log\ n)$ added to the database query.

The send message function performs a single insertion to the database, as well as a single call to the function `InsertToBlockchain`, the latter being a constant time operation [3]. Due to the asynchronous nature of how the implementation adds messages to the blockchain, we expected the sending of messages to perform well. Insertion to the database is expected to be much slower than querying due to write locks on the database but these locks [11].

We did not expect to return consistent results due to unpredictable network traffic on the azure cloud that can cause unexpected response times with inconsistent latencies. Evidence of unpredictable network traffic on the azure cloud is visible in [4] where the latencies throughout the tests randomly double over a testing period of 45 seconds.

6.2 Response Time

It is visible from Figures 5 to 12 that the results scale linearly until they reach their threshold, and then have a seemingly exponential rate of growth. This is due to the time outs that occur when the implementation can no longer handle the load. In most tests, the majority of reliable results are those under 200ms on average. From these figures it is clear that the scalability of both implementations is linear. As concrete example, in figures 5 and 11 a single instance of `RetrieveAllMessages` has a threshold of 50 users, whereas 8 instances can handle 400.

We also find that the non-blockchain implementation can

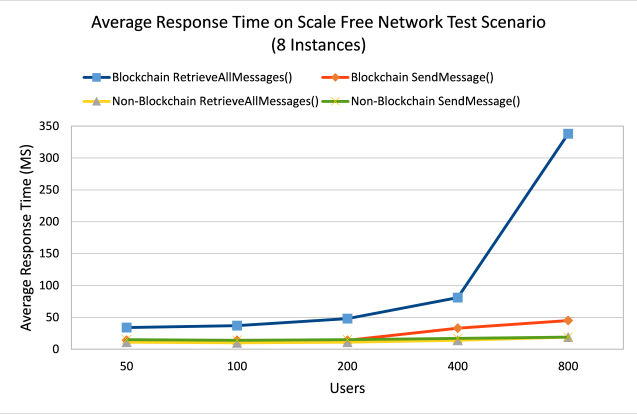

Figure 11: Scale-Free Network Test: compares `RetrieveAllMessages` against `SendMessage` on both implementations (blockchain & non-blockchain) on eight x D3 (4 Core 13GB RAM) instances

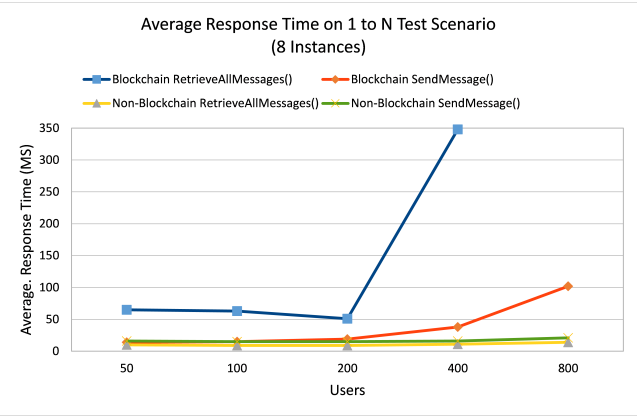

Figure 12: 1 to n Test: compares `RetrieveAllMessages` against `SendMessage` on both implementations (blockchain & non-blockchain) on eight x D3 (4 Core 13GB RAM) instances

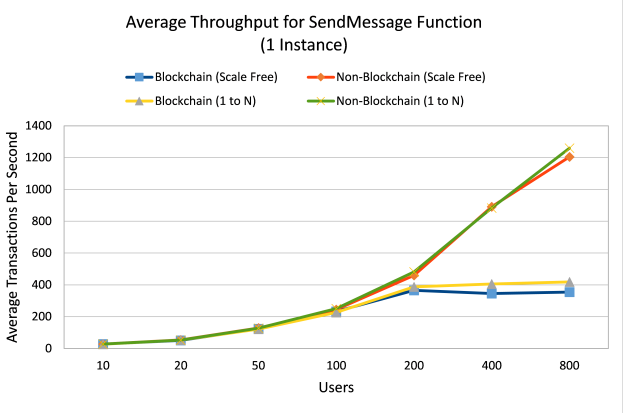

Figure 13: Comparing the `SendMessage` function on both tests (Scale-Free & 1 to n) and implementations (blockchain & non-blockchain) on a single D3 (4 Core 13GB RAM) instance

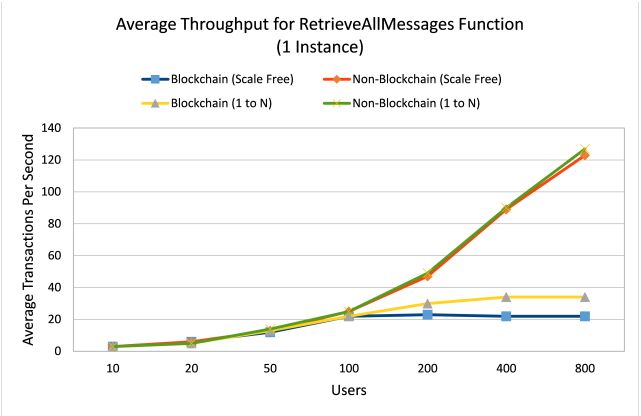

Figure 14: Comparing the `RetrieveAllMessages` function on both tests (Scale-Free & 1 to n) and implementations (blockchain & non-blockchain) on a single D3 (4 Core 13GB RAM) instance

handle 4 to 8 times the load as the blockchain implementation before reaching its threshold, while the blockchain implementation incur a penalty of multiplicative 2 to 4 on the response time. Figures 5 6 7 8 9 10 11 12 also show that the function `SendMessage` in the blockchain implementation keeps pace with the non-blockchain implementation. This is due to the asynchronous non-blocking design pattern choice that our application follows. The `SendMessage` function does not wait until each message is validated and added to the blockchain before returning.

We can also conclude that the `RetrieveAllMessages` function on the blockchain implementation has a harder time coping with the same user load on the 1 to N test scenario as the scale-free network test scenario. All querying of the blockchain occurs through the local blockchain which cannot handle the heavy load directed on a single user in the 1 to n test scenario.

6.3 Throughput

Figures 13 and 14 imply that the non-blockchain implementation has linear throughput growth throughout all tests while the blockchain implementation grows linearly until a threshold is reached. The throughput is throttled on both functions of the blockchain implementation due to the `RetrieveAllMessages` function timing out. The testing software is built to run on threads which disallow the test to continue until the time out is finished. As seen in figures 5 and 6 the response time of the `RetrieveAllMessages` function starts increasing dramatically from 100 users onwards which is where the linear growth of the throughput stops.

It is valid to note that the transactions per second on the RetrieveAllMessages is a scaled down representation of the actual reads that occur on the blockchain. As a specific example, when testing with 100 users on the user 1 to n test scenario from Figure 14, user 1 queries the blockchain over 10000 times each time the RetrieveAllMessages function is called within the last 10 seconds of this test.

The scale-free test scenario has an extra delay of $(2n)$ MS (where n is the number of users), which the 1 to n test

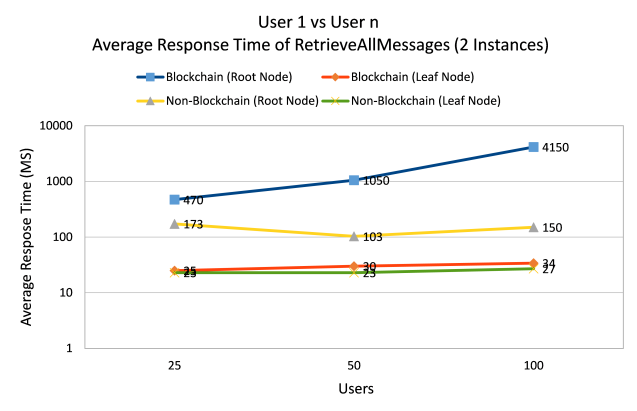

Figure 15: Comparing the `RetrieveAllMessages` function on the root node against the leaf nodes on both implementations (blockchain & non-blockchain) on a 2 x D3 (4 Core 13GB RAM) instances

scenario does not [11]. The additional delay slightly lowers the throughput of the scale-free test scenario, which is visible in Figures 13 and 14.

6.4 User 1 vs User N

In Figure 15, a blockchain root node has a polynomial rate of growth while the non-blockchain root node seems to be constant. This type of behavior was expected after analyzing the running time of the `RetrieveAllMessages` function on the blockchain implementation in Section 6.1. Figure 15 shows that the leaf nodes are almost identical regardless of the implementation. This proves that blockchains are capable of keeping up with non-blockchain solutions given they are not under heavy load.

7. CONCLUSION

We have shown that the blockchain implementation scales linearly as the number of instances is increased, until we reach the throughput threshold mentioned in Section 6.3. We learned also from Section 6.2 that using the tendermint blockchain incurs a 4-8 multiplicative factor on scalability, and a multiplicative factor of 2-4 on the average response times. These two findings imply that high throughput and low response times can still be achieved if enough servers are deployed. For well-established companies concerned with transaction immutability and ledger security, paying 4 to 8 times more for cloud computing costs is unlikely to be any concern. We have also shown in Section 6.4 that blockchains perform particularly poorly when handling heavy traffic load through single users. This implies that such solutions are more vulnerable to cyber-attacks such as denial of service. In future work we will be measuring the performance and scalability of a other proof of stake blockchain implementations.

8. ACKNOWLEDGEMENT

This research was supported by the European Blockchain Center `http://www.ebccenter.eu`

[11]This delay is described in Section 4.1.

9. REFERENCES

[1] Albert-László Barabási, Réka Albert, and Hawoong Jeong. Mean-field theory for scale-free random networks. *Physica A: Statistical Mechanics and its Applications*, 272(1):173–187, 1999.

[2] Jörg Becker, Dominic Breuker, Tobias Heide, Justus Holler, Hans Peter Rauer, and Rainer Böhme. Can we afford integrity by proof-of-work? scenarios inspired by the bitcoin currency. In *The Economics of Information Security and Privacy*, pages 135–156. Springer, 2013.

[3] Ethan Buchman. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. PhD thesis, 2016.

[4] Atusa Mohammadian Casper Frimodt, Maya Nordhild. Implementing scalability and security features for the first development cycles of a web application. *`http://spasovski.dk/wp-content/uploads/2017/06/Implementing%20scalability%20and%20security%20features.pdf`*, 2016.

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

[6] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. BLOCKBENCH: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1085–1100, 2017.

[7] Justin Fisher and Maxwell Henry Sanchez. Authentication and verification of digital data utilizing blockchain technology, u.s. patent application no. 15/083,238., 2016.

[8] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *`https://archive.org/details/PPCoinPaper`*, 2012.

[9] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[10] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *`https://bitcoin.org/bitcoin.pdf`*, 2008.

[11] Suyog S Nyati, Shivanand Pawar, and Rajesh Ingle. Performance evaluation of unstructured nosql data over distributed framework. In *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on*, pages 1623–1627. IEEE, 2013.

[12] Kyriacos E. Pavlou and Richard T. Snodgrass. Forensic analysis of database tampering. *Transactions on Database Systems (TODS)*, 33(4), 2008.

[13] Judith Ramsay, Alessandro Barbesi, and Jenny Preece. A psychological investigation of long retrieval times on the world wide web. *Interacting with computers*, 10(1):77–86, 1998.

[14] Matei Ripeanu, Ian T. Foster, and Adriana Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *CoRR*, cs.DC/0209028, 2002.

[15] Guy Zyskind, Oz Nathan, and Alex Pentland. Decentralizing privacy: Using blockchain to protect personal data. *2015 IEEE Security and Privacy Workshops*, pages 180–184, 2015.