Mystiko—Blockchain Meets Big Data

ERANGA BANDARA*, WEE KEONG NG*, KASUN DE ZOYSA†, NEWTON FERNANDO*,
SUPUN THARAKA* P. MAURAKIRINATHAN†, NAMAL JAYASURIYA*
* {eranga, awkng, ofernando, mhstharaka, namal}@ntu.edu.sg
School of Computer Science and Engineering
Nanyang Technological University, Singapore
† kasun@ucsc.cmb.ac.lk, maurakiri@gmail.com
University of Colombo School of Computing, Sri Lanka

***Abstract*—Blockchain is a peer-to-peer distributed storage that stores chronological series of transactions in a tamper-resistant manner. Blockchain became popular in various industries due to its decentralized trust ecosystem. When integrating blockchain with big data, one encounters many challenges. Current public blockchain does not support high transaction throughput; it does not scale in terms of big data storage and management; it does not provide keyword-based search and retrieval; and so on. As a result, it is hard to incorporate existing blockchain systems for big data applications. In this research, we propose a new blockchain storage "Mystiko" that is built over the Apache Cassandra distributed database to incorporate big data. Mystiko supports high transaction throughput, high scalability, high availability and full text search features. With Mystiko, we make big data more secure, structured and meaningful, and allows further data analytics on big data to be more easily performed.**

***Keywords*-Blockchain; Consensus; Cassandra; Kafka; Big-Data; Immutability; Paxos; Raft; BigTable**

I. Introduction

Blockchain provides a tamper-evident, shared digital ledger that records data in a public or private peer-to-peer network. Blockchain is also a form of distributed peer-to-peer storage. Each node in the blockchain has the exact same copy of data that is immutable. Since blockchain is a distributed storage, it uses a consensus algorithm to distribute and maintain consistency of data among the nodes.

Currently, there are various blockchain platforms in the market. Bitcoin [1], Ethereum [2], Bigchaindb [3] are some examples. Some of these blockchains are used mostly for electronic currencies, such as Bitcoin. Ethereum-like storage goes beyond crypto-currency to support different kind of storage models.

Blockchain vs Enterprise Big Databases

When one compares enterprise distributed database (sometimes called "big data" databases) with blockchain, one sees that blockchain has some attractive features. However, blockchain also lacks some features that are found in big data databases. Here are some of them:

1) Current blockchains are not storage scalable, unlike distributed databases.
2) Current blockchains do not support fast real-time transactions—Bitcoin takes 10 minutes to confirm a transaction.
3) Current blockchains do not have high transaction write throughput—Bitcoin supports only 7 transactions per second.
4) They do not have data querying features; Ethereum, Bitcoin, and Litecoin do not come with querying features.
5) Current blockchains are not ideal for keeping large data payloads.

As such, current blockchains are not immediately suited for the big-data world. In big data, one deals with massive amount of data and the need to perform analytics on them. Data storage must support high transaction throughput, high scalability and have search-retrieve features. Much research have been conducted to solve the performance and scalability issues in blockchain [3]–[5]. Attempts have been made to implement scalable and high write-throughput enabled blockchain storage. Although there has been some degree of success, one cannot directly take them to the big data world. The main reason is that the blockchains are still not built for big-data-scalability, high transaction throughput and search requirements in the big data context.

Mystiko

In an attempt to improve the performance of blockchain, most work tried to add enterprise big data features into existing blockchains. We have adopted a different approach. Instead of adding big data features, we add blockchain features to existing enterprise-level big data storage. We have chosen the Apache Cassandra distributed database [6] as the underlying storage platform. We have added blockchain properties on top of Apache Cassandra and built a new blockchain system from it. We called it Mystiko. Mystiko targets private blockchain usage, where peers are trusted and know one another. When designing Mystiko, we follow the keep-it-simple principle so that the system is easy to understand, easy to configure, and easy to deploy. It is built with solid concepts and has a beautiful architecture. It is simple, but stylish.

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

The followings are our main contributions in this paper in building the Mystiko blockchain:

1) Increased scalability and transaction throughput (Section IV).
2) Microservices [7]-based architecture adopted. All services on blockchain are Dockerized [8] and available for deployment using Kubernetes [9] (Section IV).
3) New federated consensus implemented using Cassandra Paxos consensus [10] and Apache Kafka [11] (Section V).
4) Communication performance and network overhead of the blockchain reduced by avoiding full node replication. Instead, sharding-based data replication is used (Section V).
5) Full text search of blockchain is made possible by building indices on the blocks/transactions using Elastic Search [12] (Section V).

II. Related Work

Much research have been conducted to add scalability, high availability, high-transaction-throughput-like enterprise database features into blockchains. In this section, we outline the main features and architecture of these research projects.

BigchainDB [3] is an enterprise blockchain database built on top of MongoDB [13]. The consensus is handled by the underlaying MongoDB. They have added blockchain features (decentralized control, immutability, movement of digital asserts) into MongoDB to make it a scalable blockchain database. The major contribution of BigchainDB is enabling scalability of blockchain using a concept called blockchain pipelining. With blockchain pipelining, block validation does not happen when a block is added to the network. It is eventually done by the voting process among the nodes. When a block is added to the network, all nodes start to validate the transactions and vote for a block (valid or invalid). When the majority nodes voted for a block, the block is considered as a valid block. This process yields high transaction throughput for BigchainDB.

HBasechainDB [4] is similar kind of blockchain database to BigchainDB. Instead of MongoDB, they used the Apache HBase with Hadoop. Like Bigchaindb, HbasechainDB uses blockchain pipelining and federated consensus (majority of nodes vote for a block) to generate the blocks. Since it uses HBase with Hadoop, it has linear scalability. It is also capable of analyzing data that are present on the blockchain.

Chain [14] is a blockchain storage that mainly targets private blockchains. It uses federated consensus. The majority of the nodes need to vote for a block in order to add to the ledger. Unlike other blockchains, it validates transactions in the blocks concurrently. All the nodes do not keep the entire state of the ledger. Instead it uses a sharding-based approach. Concurrently validating transactions and having sharding-based data replication allow the Chain blockchain to scale up.

Hyperledger Fabric [15] is a permissioned blockchain system using modular design approach that allows scalability, extensibility and flexibility. It comes in different consensus algorithm which can be configured `Kafka, RBFT, Sumeragi, and PoET`. Hyperledger comes with a smart contract [2] platform called `Chaincode`. Users can write chaincode contracts with Golang [16]. Hyperledger uses Apache Kafka to facilitate private communication channels between the nodes. It can achieve up to 3,500 transactions per second in certain popular deployments.

Rapidchain [5] is the first sharding-based public blockchain protocol that is resilient against Byzantine faults [17] [18]. It partitioned the data and distributed them into multiple `committer` nodes (sharding). Using an efficient cross-shard transaction verification technique, RapidChain avoids gossiping transactions to the entire network. Rapidchain evaluations suggest that it can process (and confirm) more than 7,300 transactions per second.

RSCoin [19] is a sharding-based blockchain protocol to enable scalability for centrally-banked crypto-currencies. It comes with centralized monetary supply and distributed transaction ledger. A set of authorities called `mintettes` perform validation (double-spend checking) of the transactions. By having a centralized monetary authority, RSCoin addresses scalability issues in decentralized crypto-currencies. RSCoin uses a simple and fast mechanism for double-spending check and two-phase commit to maintain the integrity of the transaction ledger. RSCoin guarantees that it can process 2,000 transaction per second.

BitcoinNG [20] is a scalable blockchain protocol based on BFT consensus. It focused on improving the scalability of Bitcoin by using the same trust model as Bitcoin. BitcoinNG's latency is limited only by the propagation delay of the network, and its bandwidth is limited only by the processing capacity of the individual nodes. BitcoinNG achieves this performance improvement by decoupling Bitcoin's blockchain operation into two planes: leader election and transaction serialization. BitcoingNg achieves significantly higher throughput and lower latency than Bitcoin while maintaining the Bitcoin trust assumptions.

III. Mystiko Architecture

Most current blockchain systems are built as monolithic systems. A single program/service on the blockchain handles all the features in the blockchain. This includes handling consensus, maintaining the decentralized ledger, broadcasting transactions, checking double spends [1], etc. We believe this is not an ideal design for a distributed system environment. With a monolithic system approach, one needs to build everything using a single programming language. When the code base grows, it becomes unwieldy. Since only one service is available, it is not possible to scale. As such, we build Mystiko with a microservice architecture, solving all the aforementioned problems. In Mystiko all the functionalities

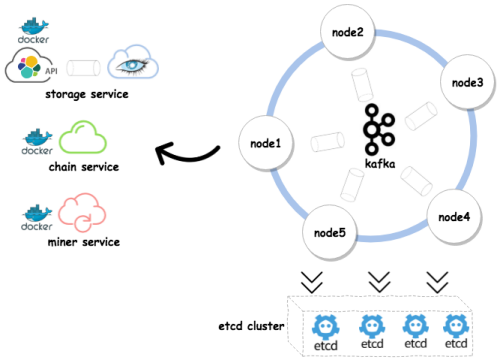


Figure 1. MystiokoDB microservices based architecture

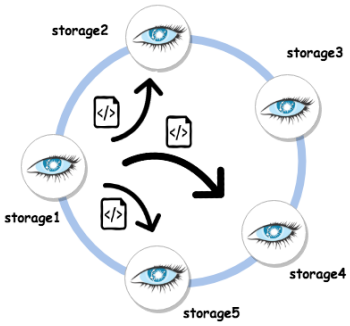


Figure 2. Mystiko Storage Service, replicating data on three nodes with replication factor 3

```
{
    "id": "<transaction uuid>",
    "timestamp": "<transaction unix timestamp>",
    "creator": "<transaction creating client>",
    "signature": "<creators digital signature>",
    "transfer": {
        "from": "<transaction sending party>",
        "to": "<transaction receiving party>",
        "amount": "<transaction amount>"
    }
}
```

Figure 3. MystiokoDB Transaction Table structure in igift application

are implemented as small services (micro service). Different programming languages and enterprise level distributed tools are used to build the Mystiko blockchain platform. Figure 1 shows the architecture of Mystiko. It contains the following services/components:

1) Storage Service (block and transaction storage)
2) Miner Service (implemented with Scala)
3) Chain Service (implemented with Golang)
4) Kafka (used to inter node communication)
5) Etcd (user as service registry)

Each node in the blockchain has their own Storage Service instance, Miner Service instance, and Chain Service instance. All the services are built as Docker containers and deployed via Kubernetes.

Storage Service

Storage Service is the place where all the data (blocks and transactions) on blockchain exist. We have chosen Cassandra [6] as the underlying storage service as this allows us to write to any node in the Cassandra cluster. In other database systems, one can only write to the master node. There is no master node on Cassandra; it is a master-less ring architecture. In a blockchain, every node should have write capability to the data storage. We also use Cassandra for its scalability and high write throughput. Cassandra supports up to one million writes per second, which is an ideal data storage for a high transaction throughput environment (e.g., banking applications).

Each node in the blockchain runs its own Cassandra node. As shown in Figure 2 these nodes connected with each other in the Cassandra ring cluster. Any node may write to the cluster. The order of the data will be decided by Cassandra's Paxos consensus algorithm. The data will be replicated to other nodes according to the replication factor of Cassandra (Mystiko does not use full node replication like Bitcoin or other blockchains). For example in a 10-node cluster; if the replication factor is 3, transactions will be replicated in 3 nodes.

The Storage Service comes in the form of a single Cassandra keyspace and two tables (transactions and blocks). The Transaction Table keeps all the transactions of the blockchain since genesis. When a client sends a transaction to the blockchain, the blockchain first checks for double spending of the transaction (see Section III under Chain Service), and add the transaction to the Transaction Table. As in other blockchains, these transactions are on a pending state until they are blocked. The content of the Transaction Table can be configured. Unlike Bitcoin which only target crypto-currencies, Mystiko is able to keep any type of data with large payloads. Users are free to define their own transaction structure. Figure 3 shows the example transaction structure that is used in the igift application [21]—an electronic money transferring application which we have built for the Sampath Bank in Sri Lanka (see Section V).

The Block Table keeps the block information. The Mining Service gets the pending transactions in a give time and creates a block (see Section III Miner service) Figure 4 shows the structure of the Block Table in the igift application.

Chain Service

Each blockchain node has it is own Chain Service written in Golang [16]. The Chain Service comes with a REST API where clients are able to connect in the blockchain network. As clients submit transactions to the blockchain, the Chain Service checks the validity of the transactions (whether the

```
{
    "id": "<block uuid>",
    "timestamp": "<block unix timestamp>",
    "creator": "<miner id>",
    "header": {
        "merkel_root": "<merkel root of the transactions>",
        "no_of_transactions": "<transaction count>",
        "previous_block": "<previous block hash>",
        "block_hash": "<current block hash>",
        "signatures": "<digital signatures of miners>"
    },
    "transactions": "<list of transaction>"
}
```

Figure 4. Mystiko Block Table structure in igift application

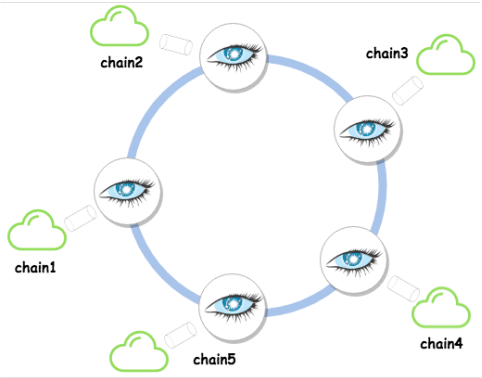


Figure 5. Mystiko Chain Service architecture

transaction is already spent/double spend). If it is valid, the Chain Service digitally signs the transactions and inserts it into the Transaction Table (via the Storage Service). As mentioned previously, these transactions are on a pending state until the Miner put them into a block. Figure 5 shows the architecture of the Chain Service.

Miner Service

The Miner Service is written in Scala [22] with the Akka framework [23]. The number of Miner Services in the network is configurable. Each blockchin node may run its own Miner Service. As shown in Figure 6 all the Miners in the network are connected with one another via Apache Kafka [11]. Every Miner has a Kafka topic which other Miners may send messages. The main functionality of the Miner Service is to create a new block from the pending transactions. At any given time, the Miner picks the pending transactions from the Transaction Table and do the following operations:

1) Validate digital signature of transactions
2) Check double spending of transactions
3) Generate Merkle root [24] and block header
4) Add miners' digital signature into the block header
5) Broadcast the block to other Miners via Kafka

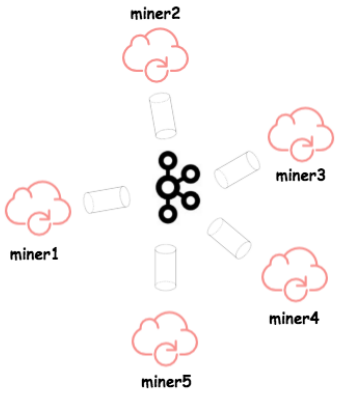


Figure 6. Mystiko Miner Service architecture

These block are not added to the Block Table yet. Other miner nodes need to approve the block before it is added to the Block Table. This is the place where our own federated consensus protocol comes into action.

Apache Kafka-based Federated Consensus

We have implemented federated consensus with Kafka to guarantee the order of the new blocks. When a Miner creates a block (block creating Miner identified as Block Creator), it broadcasts the block to other Miners (other Miners identified as Validators) via Kafka. Validators validate the transactions in the block and block header. If it is valid, they add their digital signature to the block header and send the response back to the Miner who generated the block. When a majority of Miners approved (signed) the block, the block is considered a valid block and is inserted into the Block Table. For example, assume that there are n Miners in the network. One Miner creates a block and broadcasts it to the other $n-1$ Miners. For the block to be considered valid, $n/2+1$ miners need to approve (sign) that block.

At any given time (Miner execute time decided by a distributed scheduler, see Section IV) one miner creates a block and other miners validate the created block. The Block Creator is determined in a round robin manner using a distributed scheduler. This scheduler is implemented by using the Etcd distributed key-value pair storage. Consider the scenario in Figure 7, which has 3 miners. Assume that the first block is created by Miner A, the second block will be created by Miner B and the third block is by Miner C. This goes on repeatedly.

Etcd Service Registry

As Mystiko is built using a microservice architecture, there is a need for a Service Registry to do record keeping of information of the services. We use Etcd [25] as it is a Raft consensus-based distributed key-value pair storage. We use Etcd for two main purposes:

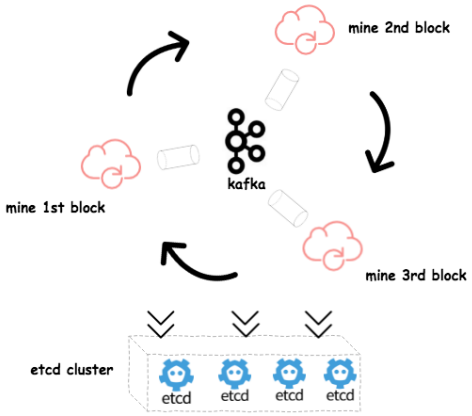


Figure 7. Deciding block creator with Etcd-based distributed scheduler

1) Service health check—In a distributed system, we need to check the health status of all the running services (whether they are up and running).
2) Distributed scheduler to determine Block Creator—Block Creator is determined by the Etcd-based distributed round robin scheduler.

IV. Mystiko Characteristics

Consensus

Blockchain uses a consensus algorithm to guarantee the order of the blocks. The consensus algorithm used by traditional blockchains is Proof of Work [1] or Proof of Stake [2]. In this approach, the blockchain miners performing time and resource consuming brute-force computational task to decide the next block. In Mystiko, we have used Cassandra's built-in Paxos [10] as the consensus algorithm. On top of Paxos, we have implemented a new federated consensus algorithm to further verify the blocks and guaranteeing the order of blocks. This federated consensus algorithm is implemented using Apache Kafka.

Data Replication

Traditional blockchains replicate data (transactions/blocks) in each and every peer. If 1,000 nodes exist, the replication factor is 1,000. It kills network bandwidth and time and directly affects scalability [26]. In Mystiko, instead of using full node replication, we use sharding-based replication, which reduces network overhead and broadcast time.

Full Text Search

By default, Cassandra does not support full text search. It only supports search via primary keys or secondary keys. To achieve full text search on Mystiko, we use Elasticsearch [12]. We have integrated Elasticsearch into Cassandra via, Elassandra [27]. With Elassandra, all data written to Cassandra is automatically indexed in the Elasticseach. In our case, all the blocks, transactions that are written to Cassandra are automatically indexed so that one may search for any field in the transactions or blocks.

Mystiko Features

By turning Cassandra into a blockchain database, we have achieved the following features on both Mystiko:

1) High scalability and high availability
2) High transactions write throughput via Cassandra
3) Fast block creation with federated consensus
4) Low overhead due to avoiding full node replication and broadcast communication
5) Full text search on blocks and transactions
6) Decentralized control of asserts
7) Create transfer assets without reliance on a central entity
8) Data immutability
9) Support to store large data payloads with Cassandra
10) Supports for data mining with the data indexed in elastic search

Integration with Big Data

When data is huge, blockchain has to be able to cope with massive amount of data and yet afford high transaction throughput. Cassandra is a big data friendly data storage system. It supports up to one million transaction writes per second. By enabling blockchain features into Cassandra, one achieves high transaction throughput and scalability.

Anyone dealing in big data also needs big data analytics. As current blockchains are not search-and-retrieve friendly, they are not suitable for analytics. By integrating elasticsearch, a high featured search API, we build Mystiko as a search-and-retrieve friendly blockchain storage.

By enabling high transaction write throughput, scalability and full text search, Mystiko is a big-data friendly blockchain storage. As Mystiko has blockchain properties, we added data immutability and more security into big data. With Mystiko, big data management is more secure, structured, meaningful, and easier for further analytics.

V. Mystiko Application

We have integrated Mystiko into enterprise grade applications in the banking and financial sectors. In this section, we introduce one such application igift which we have built for the Sampath Bank in Sri Lanka. igift is an electronic currency application where users transfer money as gift vouchers between their friends/contacts. Gift vouchers are electronic, so they come with the double spend problem. To check double spend, we store all igift electronic transactions in the Mystiko blockchain. igift is the first blockchain-based banking solutions in Sri Lanka [21].

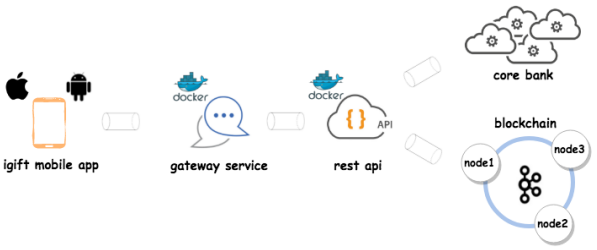


Figure 8. igift application architecture

A. *igift Architecture*

Figure 8 shows the high level architecture of the igift application. It has the following components:

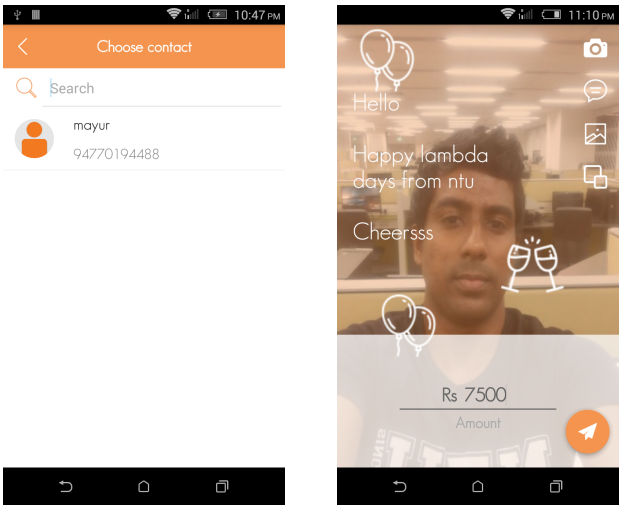
1) igift mobile application : Available as an Android/iOS app to bank users. With this app, users can generate igifts which are electronic gift vouchers and send to their friends.
2) Gateway service: This is the gateway service that links the bank application and mobile app.
3) REST API - This is the entry point to the banking application. It is a SSL-enabled secure API. All the data from mobile app to REST API is encrypted over SSL.
4) Blockchain: All igift electronic transactions are stored in the blockchain. We use blockchain to check the double spend problem of using igifts.

Mobile app generates igifts with receiver information and gift amount. igift first sends to the bank via HTTPS REST API. Then the bank API redirects the request to blockchain to check potential double spending in the igift transaction. Blockchain records all the information about transactions. If the transaction does not have double-spending issue, it calls the core banking API and do the banking transactions. Finally, igift forwards to the receiving user. At some point, blockchain miner takes the pending transactions, validate them and creates the blocks.

B. *igift Usage Scenario*

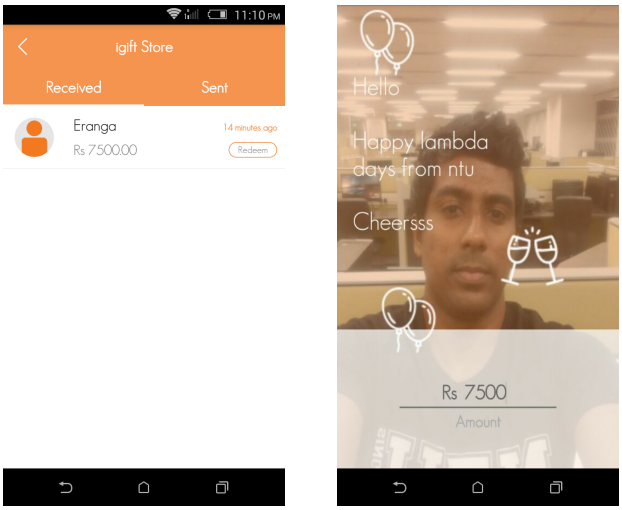
Let us assume that user `Eranga` and user `Mayur` are using the igift app. Eranga wants to send an igift to Mayur. First he needs to create an igift with the amount. Then he sends it to the bank via the REST API. The request payload contains a `JOSN` object with `id, amount, sender, receiver, igift image payload, digitalsignature`. Sending igift is similar to paying electronic money to a receiver. Figure 9 shows how an igift design is created in the mobile app.

When the bank API receives the igift, it checks for potential double-spending of the igift and adds a transaction entry to the blockchain if there is no double spending. Finally the bank calls the core bank API and forwards the igift to the receiver Mayur.



(a) igift contact list (b) design igift

Figure 9. Send igift to mayur



(a) received igift (b) igift preview

Figure 10. Received igift from eranga

As shown in Figure 10 Mayur receives the igift into his igift mobile app. He can view a preview of igift. Then he can redeem the igift amount to his bank account or forward the igift to another user (which means paying to any other user via the igift electronic currency).

C. *igift Future Work*

Currently igift only transfers money between users. In future, the bank plans to facilitate the purchasing of goods from Sri Lankan shops by paying with igifts.

VI. Mystiko Performance Evaluation

We have done a performance evaluation of Mystiko. The results are in four areas. The evaluation is performed based on the blockchain deployed in the Sampath Bank Sri Lanka.

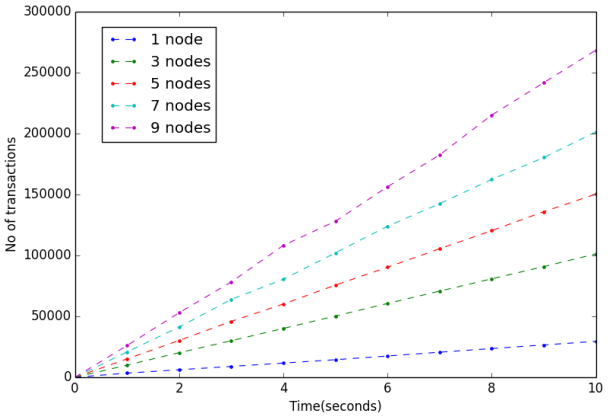


Figure 11. Number of transaction over time (seconds)

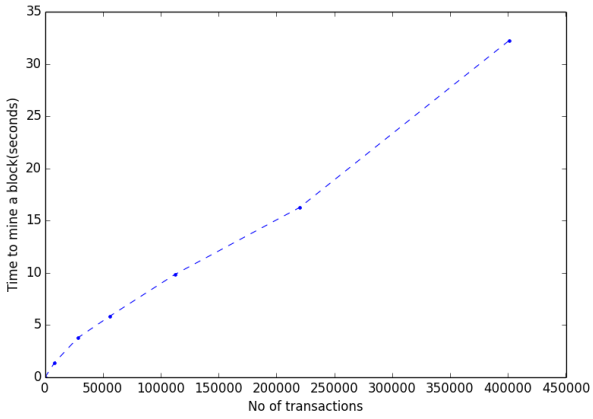


Figure 13. Block mining time vs the number of transactions in the block

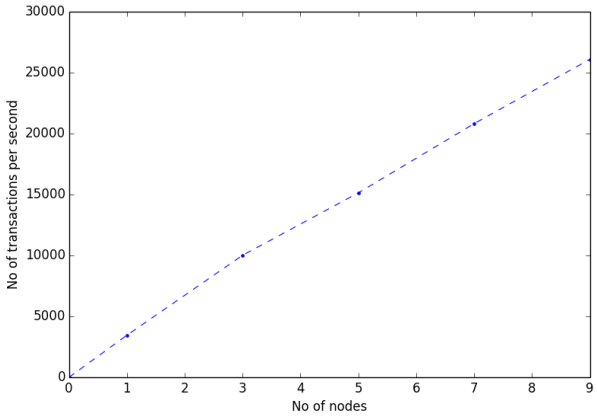


Figure 12. Number of transactions vs number of Mystiko nodes in the cluster

A. Transaction Throughput Over Time

We have setup 1-node, 3-node, 5-node, 7-node and 9-node Mystiko clusters in Cassandra and collected transactions throughput over the time.

As shown in Figure 11 the blockchain showed consistent and linear transaction growth. The reason for the transaction growth is the underlaying Apache Cassandra database.

B. Transaction Throughput vs Number of Nodes

For this scenario, we deployed a 1-node Mystiko and increased the number of nodes in the cluster every 10 seconds. Then we recorded the transaction throughput (transactions per second).

As shown in Figure 12, as we increase the number of Mystiko nodes in the cluster, the transaction throughput doubles. The main reason for this is the underlying database master-less ring architecture. In Cassandra, all blockchain nodes have write capability. So when adding a node to the cluster, it linearly increases the transaction throughput.

C. Block Mining Time

The time to mine a new block is recorded with respect to the number of transactions in the blocks. The time for mining a block include:

1) Transaction validations time $\times$ No of transactions
2) Consensus time
3) Block replication time

For this setup, we use a 9-node Mystiko cluster. As shown in Figure 13, when the transaction count increases, the mining time/latency also increases. The main reason for this observation is the time taken to validate the transactions. When transaction count increases, the validation time also increases correspondingly.

D. Search Performance

In Mystiko, we index all the transactions/blocks using Elasticsearch via Elassandra. This allows one to search data in transaction/block using Elasticsearch. For this evaluation, we issued a search query (which we are using to check the double spend) into Elasticsearch and compute the search time. Double-spending checking query (with different parameters) is issued 20 times with a given transaction set and the average search time taken. We have done this test with 10 different transaction sets. We have started with 136,182 transaction set and increased the transaction set up to 2,044,267.

As shown in Figure 14, we achieved super fast search performance (few milliseconds time). The Apache Lucene index-based Elasticsearch storage is the main reason that yields super fast search in Mystiko.

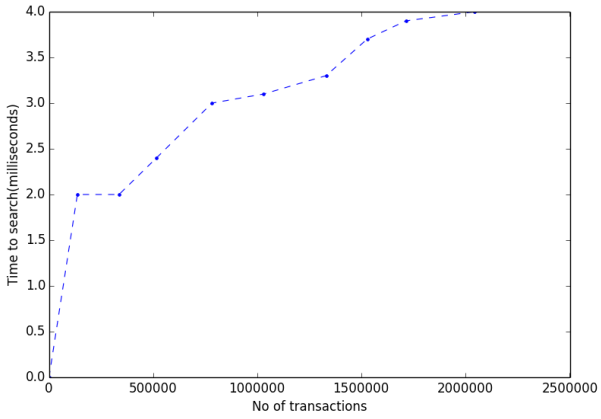


Figure 14. Time to execute double spend search with Elasticsearch

VII. Conclusions and Future Work

With Mystiko, we have built a blockchain with high transaction throughput, high scalability, and high availability. It uses the Apache Cassandra distributed storage as the underlying storage platform. We have added full text search capability to Mystiko by indexing the transactions and blocks on Elasticsearch. Our microservice-based architecture with Docker and Kubernetes enables easy deployment and easy scalability. With these features Mystiko is a big-data friendly blockchain storage system.

We have proven the scalability and transaction throughput features with empirical evaluations. We have integrated Mystiko into production grade applications in the banking and financial sectors. The deployments are votes of confidence for Mystiko as an ideal blockchain system for big data and cloud storages.

We have released Mystiko version 1.0. We follow the Continuous Delivery approach when building and releasing the product. We release a new version into production every month. The following are features we will release in future releases.

1) Functional-programming-based smart contract platform.
2) New BFT consensus algorithm instead of Cassandra's Paxos-based consensus.
3) Incorporate homomorphic encryption [28] to provide privacy and confidentiality features in the blockchain.

Currently we are customizing Mystiko for an online 3D-printing market place and a music share market place. These projects will be released into production soon.

Acknowledgements

This work is supported in part by NRF grant #NCR2016NCR-NCR002-004 and NAMIC grant #2018022.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, 2014.

[3] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, "Bigchaindb: a scalable blockchain database," *white paper, BigChainDB*, 2016.

[4] M. Subhankar Sahoo and P. K. Baruah, "Hbasechaindb a scalable blockchain framework on hadoop ecosystem," 03 2018, pp. 18–29.

[5] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: A fast blockchain protocol via full sharding," *IACR Cryptology ePrint Archive*, vol. 2018, p. 460, 2018.

[6] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[7] "Microservices pattern: Microservice architecture pattern." [Online]. Available: http://microservices.io/patterns/microservices.html

[8] "Docker documentation," Aug 2018. [Online]. Available: https://docs.docker.com/

[9] "Kubernetes documentation." [Online]. Available: https://kubernetes.io/docs/home/?path=users&persona=app-developer&level=foundational

[10] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998.

[11] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[12] "Elastic stack and product documentation — elastic." [Online]. Available: https://www.elastic.co/guide/index.html

[13] "Mongodb for giant ideas." [Online]. Available: https://www.mongodb.com/

[14] "Chain protocol whitepaper." [Online]. Available: https://chain.com/docs/1.2/protocol/papers/whitepaper

[15] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 30.

[16] "The go programming language." [Online]. Available: https://golang.org/

[17] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[18] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[19] G. Danezis and S. Meiklejohn, "Centrally banked cryptocurrencies," *arXiv preprint arXiv:1505.06895*, 2015.

[20] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.

[21] "Blockchain gifting app launched by sri lanka's sampath bank," Jun 2018. [Online]. Available: https://blocktribune.com/blockchain-gifting-app-launched-by-sri-lankas-sampath-bank/

[22] "The scala programming language." [Online]. Available: https://www.scala-lang.org/

[23] "Akka documentation." [Online]. Available: https://doc.akka.io/docs/akka/2.5/actors.html

[24] M. S. Niaz and G. Saake, "Merkle hash tree based techniques for data integrity of outsourced data." in *GvD*, 2015, pp. 66–71.

[25] Coreos, "coreos/etcd," Aug 2018. [Online]. Available: https://github.com/coreos/etcd

[26] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

[27] Strapdata, "strapdata/elassandra," Jul 2018. [Online]. Available: https://github.com/strapdata/elassandra

[28] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1192, 2015.