Journal of Systems Architecture 116 (2021) 102061

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Rahasak—Scalable blockchain architecture for enterprise applications

Eranga Bandara [a,*], Xueping Liang [b], Peter Foytik [a], Sachin Shetty [a], Nalin Ranasinghe [c], Kasun De Zoysa [c]

[a] Old Dominion University, Norfolk, VA, USA
[b] University of North Carolina at Greensboro, NC, USA
[c] University of Colombo School of Computing, Sri Lanka

ARTICLE INFO

Keywords:
Blockchain
Big data
Cloud computing
Edge computing
IoT

ABSTRACT

Blockchain-based decentralized infrastructure has been adapted in various industries to handle the sensitive data in a privacy-preserving manner without trusting third parties. However, integrating state-of-the-art blockchain platforms with the scalable, enterprise-level applications result in several challenges. Current blockchain platforms do not support high transaction throughput, lack high scalability, and cannot provide real-time transaction processing and back-pressure operation handling in high transaction throughput applications (e.g Big data, IoT). In this paper, we propose a novel permissioned blockchain platform "Rahasak" for highly scalable, enterprise applications. Rahasak blockchain adopts the Apache Kafka-based consensus on top of a "Validate-Execute-Group" blockchain architecture to handle real-time transaction execution on the blockchain. The architecture is equipped with a functional programming and actor-based smart contract platform that enables concurrent execution of transactions in the blockchain. Rahasak supports high transaction throughput, high scalability, concurrent transaction execution, data analytics features. With Rahasak, we make blockchain more scalable, secure, structured and meaningful for further data analytics.

1. Introduction

Enterprise-level applications in various industries operate in highly scalable environments and cope with privacy-sensitive data. To handle the security and privacy concerns of the data in these applications, blockchain-based infrastructure can be adopted. Then data sharing between different components, data aggregation, monitoring, decision-making functions can be securely facilitated with blockchain-based architecture. However, to efficiently handle the high data load and make analytics with them, the underlying data storage must support high transaction throughput, high scalability and have search-retrieve features. When integrating existing blockchain platforms with this type of scalable applications, one encounters many challenges. Currently, existing blockchain platforms (1) are not scalable [1], (2) do not support real-time transaction processing [1,2], (3) do not support high transaction write throughput [3], (4) cannot handle back-pressure operations in high transaction throughput enable applications [4] in high transaction throughput enabled applications, (5) do not support full-text search query APIs to search the data in blockchain [2,5], (6) and do not support data analytics and machine learning features. As such, current blockchains are not immediately suited for the scalable enterprise-level applications.

There are three major performance bottlenecks in the existing blockchain platform (1) "storage model", (2) "Order-Execute architecture" and (3) "imperative-style smart contracts" [1,5,14]. In existing blockchains, each peer in the network maintains their ledger. Data on the network replicates throughout all nodes(full-node data replication). Unlike the distributed database, there is no sharding to improve performance [3]. Existing blockchains cannot execute the transactions when a peer submits a transaction to the network. To validate and execute a transaction, it needs to wait until a block is created [1]. Existing blockchain smart contract platforms [1,15,16] do not come with concurrency control. Due to this reason, concurrent transactions are not supported in existing blockchain platforms. All transactions are executed sequentially and so is the ledger update. This results in low transaction throughput and latency [12]. Much research has been conducted to solve these major performance bottlenecks on blockchain [1–3,7] and integrate them with scalable, enterprise-level applications. Table 1 summarizes how these performance bottleneck features are solved on existing blockchain platforms and smart contract platforms. According to the Table 1 we have observed that there are no current blockchain solutions that address any of the two conditions out of

* Corresponding author.
E-mail addresses: cmedawer@odu.edu (E. Bandara), x_liang@uncg.edu (X. Liang), PFoytik@odu.edu (P. Foytik), sshetty@odu.edu (S. Shetty), dnr@ucsc.cmb.ac.lk (N. Ranasinghe), kasun@ucsc.cmb.ac.lk (K. De Zoysa).

https://doi.org/10.1016/j.sysarc.2021.102061
Received 7 October 2020; Received in revised form 20 January 2021; Accepted 12 February 2021
Available online 17 February 2021

Table 1
Features identified in existing blockchain platforms which affects to scalability and transaction throughput.

Blockchain	Real-time transactions	Concurrent smart contracts	Sharding based replication
Rahasak [3]	✓	✓ (functional semantic)	✓
BigchainDB [2]	✓ (quasi-real-time)	✗	✓
HbasechainDB [6]	✓ (quasi-real-time)	✗	✗
Hyperledger [1]	✗	✗	✗
Chain [7]	✗	✗	✓
RapidChain [3]	✗	✗	✓
RSCoin [8]	✗	✗	✓
Lightchain [9]	✗	✗	✓
LSB [9]	✗	✗	✗
Bitcoing-NG [10]	✗	✗	✗
Sensor-Chain [11]	✗	✗	✗
Simplicity [12]	✗	✗ (functional semantic)	✗
Kadena(Pact) [13]	✗	✗ (functional semantic)	✗
Rchain(Rholang) [14]	✗	✓ (functional semantic)	✗

(i.e., real-time transactions with O-E model, concurrent execution of smart contracts and sharded replications), let alone all three.

BigchainDB [2], HbasechainDB [6] blockchains provides Quasi-real-time transactions(not full real-time transactions) with using novel blockchain-pipelined architecture. They do not support sharded data replication or concurrent transaction execution of smart contracts. The Hyperledger Fabric [1] which is the most popular private blockchain platform currently available proposed "MVCC" [17] concurrency control-based novel blockchain architecture "Execute-Order-Validate" to reduce the overhead of the "Order-Execute" architecture. Even though Hyperledger Fabric gains considerable performance gain with using this novel architecture, it does not yet support the real-time transactions. Also, its Chaincode smart contract platform does not support concurrent transaction execution since it used imperative style based programming. Chain [7], Rapidchain [3], RSCoin [8], LightChain [9] kind of blockchains support sharded data replication to get rid of full-node replication. But they do not support real-time transactions processing or concurrent transaction execution of smart contracts. Simplicity [12], Scilla [16], Rholang [14] kind of smart contract platforms trying to support side-effects less functional programming-based smart contracts. But they do not support real-time transaction execution on the blockchain or sharded data replications. Due to these reasons, one cannot directly use most of the existing blockchain platforms and smart contract platforms with scalable, enterprise-level, financially critical application domains [18]. In this research, three major performance bottlenecks(i.e., real-time transactions with O-E model, concurrent execution of smart contracts and sharded replications) are addressed on existing blockchain platforms and a scalable blockchain system called "Rahasak" is built as a proof-of-concept prototype. Rahasak targets private blockchain usage where peers are trusted and are known to one another. Following are our main contributions of this research.

1. Real-time transaction enabled, "Validate-Execute-Group" blockchain architecture is introduced with Apache Kafka-based consensus. The proposed architecture reduces the overhead of "Order-Execute" architecture.
2. Functional programming and Actor based smart contract platform is integrated to achieve concurrent transactions in the blockchain.
3. Instead of full-node data replication, sharding has been used to reduce the network and communication overhead in the blockchain.
4. Microservices-based architecture is introduced to build scalable blockchain applications.
5. Back-pressure operations on high transaction throughput enabled applications handles with Apache Kafka and Akka streams.
6. Full-text search of blockchain data is made possible by building indices on the blocks/transactions/assets using Apache Lucene index-based API.

The rest of the paper has been organized as follows. Sections 2 and 3 discusses the background information and Validate-Execute-Group architecture of Rahasak blockchain. Section 4 discusses the architecture and implementation of the Rahasak blockchain. Section 5 performance evaluation. Section 6 Related works. Section 7 conclusion and future works.

2. Background information

2.1. Full node replication

In current blockchain systems, all the peers maintain their own local ledger (local storage). They use full node replication to replicate the data among the nodes [3]. For example, when a block gets created it will broadcast to other peers. Then other peers will execute the transaction to update their local ledger [19]. Even though this works fine with untrusted public blockchain ledgers, there is no point of having full node replication on private/permissioned blockchains. There are various drawbacks on full node replicated systems. They use gossip protocols to broadcast blocks among the nodes. It will add additional network bandwidth and time when transport block information between all the peers. When storing all the data on local ledgers it will consume a large amount of storage. Since data exists different local nodes, current blockchain systems cannot provide a way to effectively search the data from the ledger.

2.2. Order-execute architecture

Most existing blockchains follow the Order-Execute architecture [1]. In this architecture, transactions are executed in the following order:

1. Peers generate transactions. Different transaction parameters exist in different blockchains. In Bitcoin-like blockchains, inputs and outputs are transaction parameters. In Hyperledger-like smart contract-based blockchain, transactions are generated with smart contract function name and function parameters. Until a new block is created, these transactions are not executed; they are pending.

2. Using some consensus algorithm, a new block will be generated from the pending transactions. There are various consensus protocols used by different blockchains. Bitcoin and Ethereum-like public blockchains use POW; Tendermint, Chain, or Quorum-like private blockchain use BFT consensus [20,21].
3. Finally, the generated block is broadcast to all peers via a gossip protocol. Peers that receive a block sequentially execute the transactions on the block and update their local ledger status.

The Order-Execute architecture is simple and works well for public blockchains. However, using Order-Execute for private blockchains has several drawbacks. Transactions are not immediately executed after submission by peers. This adds considerable lag between transaction submit time and transaction execution time. As a result, clients need to wait until the transaction is completed. After the transactions are added to a block, all transactions will be serially executed; there is no concurrent transaction execution. This added further latency. All transactions are executed in all peers. For an instance, if there are 10 peers in the network, a single transaction will be executed 10 times in each peer. This is a major drawback on resources and time. As an alternative to the Order-Execute architecture, Hyperledger Fabric proposed the Execute-Order-Validate architecture [1]. It has gained success to some extent; however, it is not sufficient for high transaction throughput and scalability requirements on enterprise-level applications.

2.3. Imperative-style smart contracts

We have seen blockchain platforms that have introduced programming interface called "smart contracts" to interact with the blockchain ledger. Users do not need to execute queries to save or retrieve data from the blockchain. Instead smart contracts provide a programming interface to interact with the underlying blockchain storage models. Smart contracts can be introduced as a database abstraction layer for blockchain. It is similar to the Object Relational Mapping (ORM) tools in traditional programming frameworks. Unlike traditional ORMs, smart contracts are capable of defining the business logic of the application. With smart contracts, business logic on the application layer can be moved to the blockchain layer.

There are various smart contract platforms. Ethereum has the Solidity platform [15], Hyperledger has the Chaincode smart contract in the Fabric framework [1], Zilliqa has Scilla [16], Kadena has Pact [13], RChain has Rholang [14], etc. Most of these platforms follow the imperative programming style. There is no concurrency-control mechanism in them; they do not support concurrent execution of transactions [5]. As a result, there is considerable latency and scalability suffers.

3. Rahasak

3.1. Overview

Rahasak is a permissioned Blockchain system which is targeted for scalable, enterprise-level applications such as big data, cloud computing, edge computing, IoT. It utilizes Apache Kafka [22] as the underlying consensus platform and eventual consistent distributed database [23,24] as the blockchain asset storage. Eventual consistent databases produce high transaction write throughput [23] when compared to other databases. That is the main reason to use an eventually consistent database as the underlying asset storage in Rahasak [25]. Rahasak uses Apache Kafka message broker [22] and reactive streams [26] methodology to handle back-pressure operations [4] on high transaction throughput enabled applications. To facilitate the full-text search on Blockchain data, Rahasak utilizes the Lucene index [27] based API. Rahasak addressed three main performance bottlenecks on existing Blockchain platforms, namely, Order-Execute architecture, Full node data replication and Imperative style smart contracts.

To address the issues with existing imperative style smart contracts, we have introduced functional programming-based [28] and actor [29,30] based Aplos smart contract platform [5]. With Aplos smart contracts, all Blockchain functions (smart contracts) are written using actors. Different actors may interact with one another via message passing. Aplos smart contract platform supports concurrent execution of transactions; this yields high transaction throughput and scalability.

All blocks, transactions and asset information are stored in a distributed database (e.g. in tables) in Rahasak. In Rahasak every Blockchain peer comes with a distributed database node; these nodes are connected as a ring cluster. After executing a transaction, state update in a peer is distributed and replicated with other peers via underlying distributed databases' sharding algorithm. With this approach, we avoid full node data replication issues which exist in traditional blockchains [3,31] on Rahasak.

To address the issues in "Order-Execute" architecture, we designed a new blockchain architecture "Validate-Execute-Group". Unlike Order-Execute-based systems, Validate-Execute-Group systems can validate and executes transactions concurrently when peers submit them to the network. The client does not need to wait until a block is created to approve transactions. Transactions will be executed only in one peer; With Aplos smart contracts, the transactions can be executed concurrently. Once transactions get executed, the state update will distribute and replicate to other peers in the cluster by underlying distributed database storage. Following are the main functionalities of Validate, Execute and Group phases. The workflow of each phase described in Fig. 1.

3.2. Validate phase

Client submits transactions to Rahasak blockchain via Apache Kafka. Each blockchain peer has separate Kafka topic which stores and order the transactions(Apache Kafka is the consensus service on Rahasak blockchain). The transactions in the Kafka topic consumed by blockchain peers and executes them. Unlike other blockchain systems, Rahasak validates the transactions (double-spending check) when a client submits them to the blockchain network. Upon a client's request, the blockchain peer first checks whether this transaction is double-spent or not (check replay attacks). To check double-spending, we use the Transaction ID and the sending user's ID. When a transaction is invoked, we need to check whether the given user has already sent a transaction with the same transaction ID. This Transaction ID is a sequence number.

To validate the transactions when the client submits them to the blockchain, the underlying blockchain asset storage needs to have linearizable consistency [32,33]. Since we have used an eventually consistent distributed database as the underlying storage, we need to find a way to achieve linearizable consistency from the distributed database. Most of the existing distributed databases follow AP architecture in CAP theorem [34]. The default consistency model(eventual consistency and strong consistency) in AP based databases does not guarantee linearizable consistency. The main reason is that the default consistency model does not consider pending writes when querying data. When two transactions execute at the same time, one transaction could override the older one. This is not an issue in the AP based distributed databases, it is the intended way that they have built to achieve high transaction throughput [35,36]. To achieve linearizable consistency from AP based distributed database, Rahasak introduces a distributed cache-based mechanism [37,38]. The architecture is shown in Fig. 3. In this architecture, we have added a distributed cache between a blockchain peer and distributed database nodes. All the recent transaction execution information is added to the distributed cache. The double-spending check happens in two phases now (Fig. 3).

In the first phase, when a transaction is proposed, a peer checks the validity of the transaction by scanning the underlying distributed database. The underlying distributed database keeps all the executed

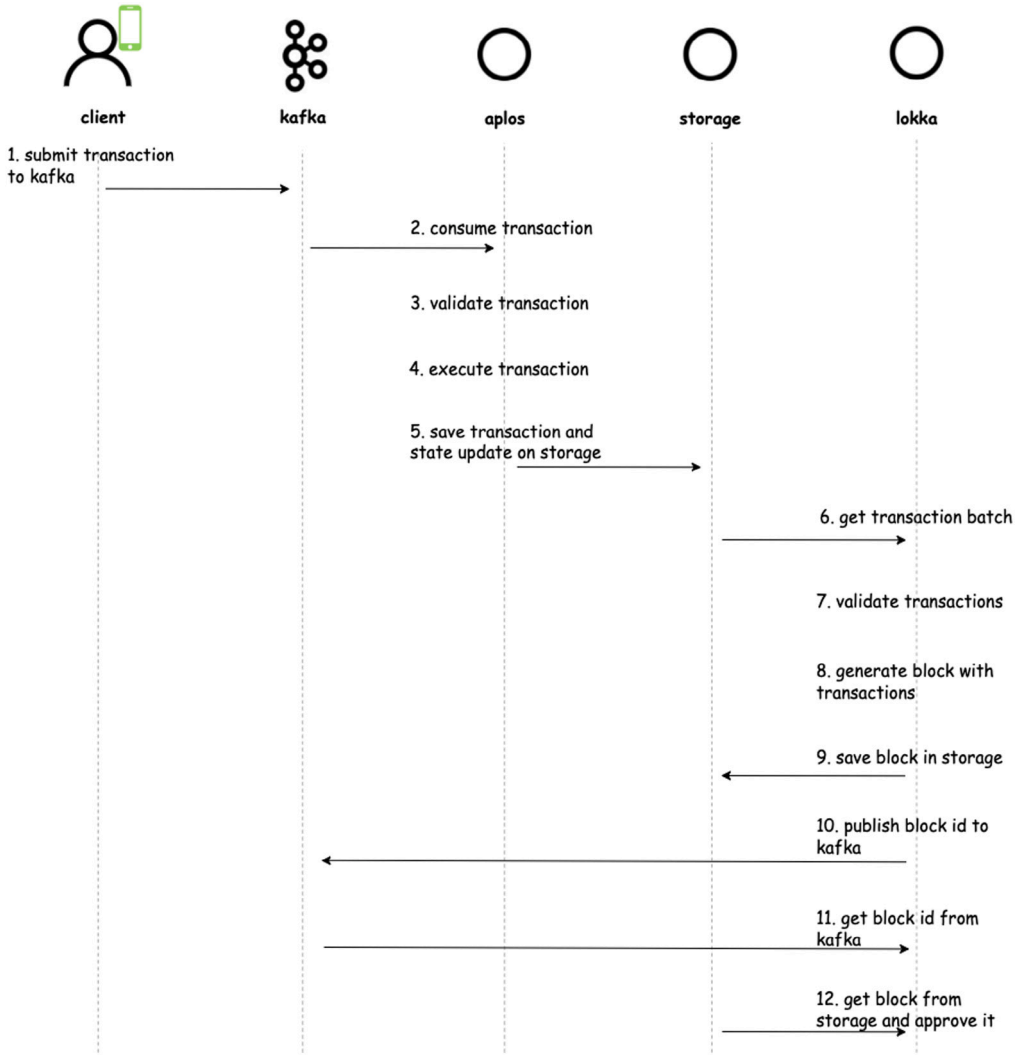

Fig. 1. Rahasak blockchain Validate-Execute-Group architecture workflow.

transactions from blockchain genesis. If this transaction is not included in the distributed database, a second validation phase with the distributed cache is used. Since we have used eventually consistent distributed storage when validating transactions, the pending transactions are not checked [36]. Pending transactions are those that have not been written to the transaction table yet. If an identical transaction with the same transaction ID comes from the same user before the pending transaction complete, it results in a double-spend scenario. To overcome this issue, we have introduced a distributed cache to store recently executed transactions (all the recently executed transaction ids and transaction sending user ids are stored in the distributed cache). If the first validation phase succeeds, Rahasak checks whether the given transaction (with its ID and sending user ID) exists in the distributed cache. If it does not exist, it means complete validation process is accomplished. Then it writes the transaction ID and transaction sender user IDs into the cache. The second validation phase guarantees double spend does not occur with concurrent transactions.

3.3. Execute phase

If both validations (two-step double-spend check) are successful, the transaction is executed and the ledger asset is updated. This transaction is saved in the transaction table. The order of the transactions and ledger assets are guaranteed by the underlying distributed databases consensus algorithm. Unlike the Order-Execute approach, a transaction will be executed only once. After executing a transaction, state update in a peer is distributed and replicated with other nodes using the underlying distributed database's sharding algorithm.

3.4. Group phase

Now that a transaction is validated and executed, the executed transaction is replicated and made available to other peers through the distributed database, the recently executed transaction entries are on the distributed cache. At any given time, a special service called "Lokka" takes all transaction entries in the distributed cache and creates a block. Lokka's main functionality is to create blocks and approve the blocks. There could be multiple Lokka nodes in the blockchain network.

When creating a block, Lokka takes all the transactions that correspond to the cache entries from the underlying distributed database and add them to a block. Block is also stored in the underlying distributed database. Once the block is created it needs to be approved by other Lokka services in the network. The block ordering process is done via majority vote federated consensus [39,40] implemented between Lokka services. There is an endorsement policy defined among the Lokka

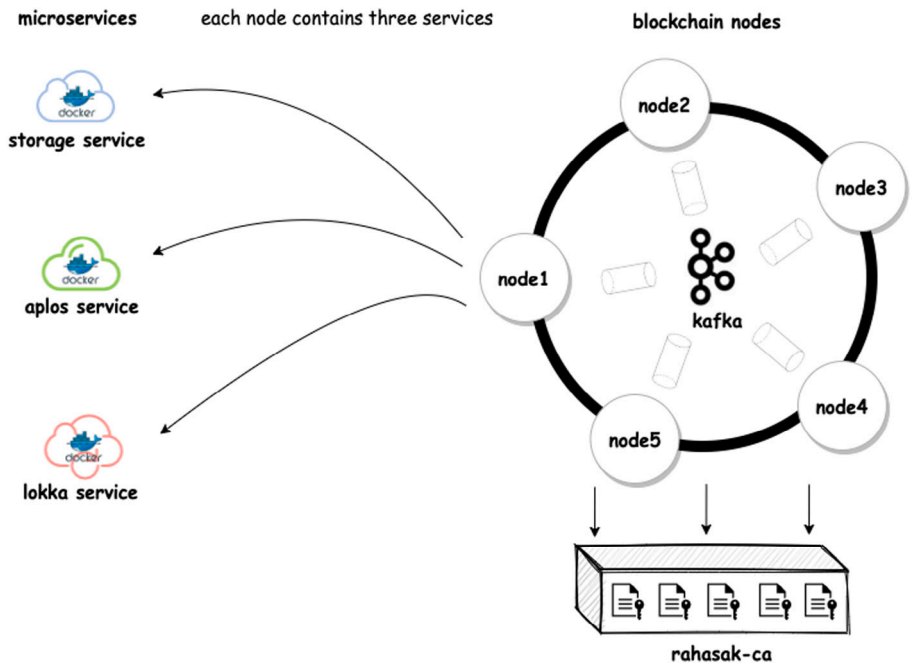

Fig. 2. Rahasak blockchain microservice-based architecture. Each blockchain node contains three services, (1) Storage service, (2) Lokka service (3) Aplos service.

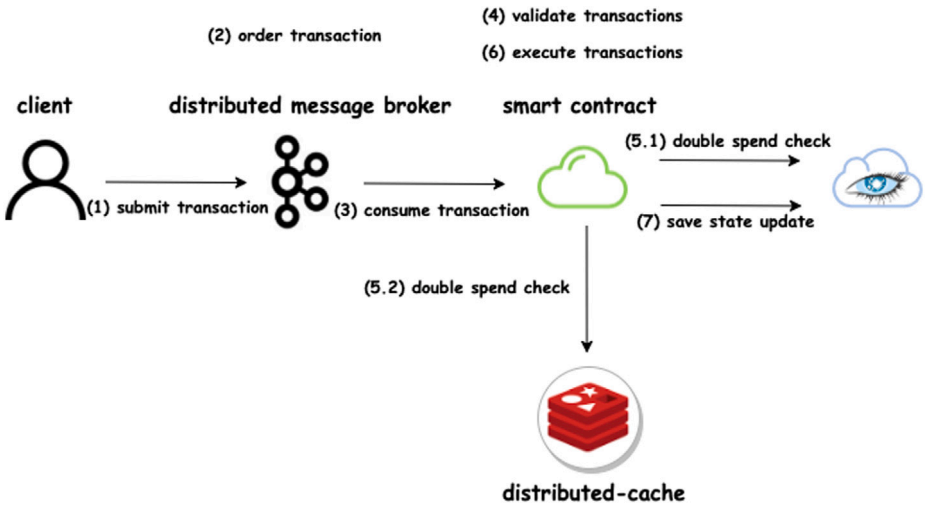

Fig. 3. Rahasak Aplos service architecture.

nodes. This policy defines how many Lokka nodes that need to vote to approve a block. Lokka service creates blocks either interval-based or volume-based; i.e., based on the number of transactions in the cache (e.g. 100 transactions in cache).

It is important to note that the transaction execution process and the grouping process (block creation) are two separate processes. Clients do not need to wait until block creation to confirm a transaction. Since transactions have already been validated and executed, the Grouping phase proceeds fully asynchronously.

4. Rahasak architecture

4.1. Overview

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 [19], etc. We believe that this is not an ideal design for a distributed system environment. In 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 Rahasak using a microservice architecture [41], solving all the aforementioned problems. In Rahasak, all the functionalities are implemented as small services (microservices). All of these services are Dockerized [42] and available for deployment using Kubernetes [43]. Fig. 2 shows the architecture of Rahasak. It contains the following services/components:

1. Aplos service — Smart contract service implemented using Scala [44,45] functional programming language and Akka actors [46]
2. Storage service — Apache cassandra [23] based block, transaction and asset storage service
3. Lokka service — Block creating service implemented using Scala and Akka streams [26]

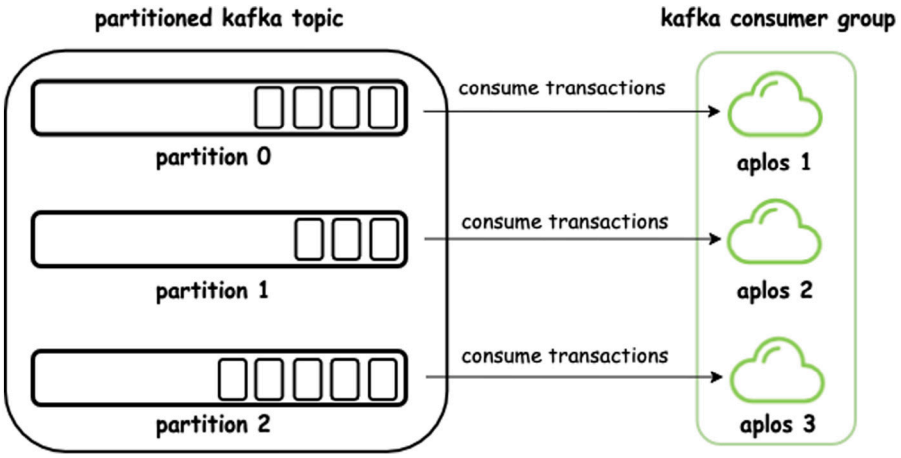

Fig. 4. Paritioned kafka topic.

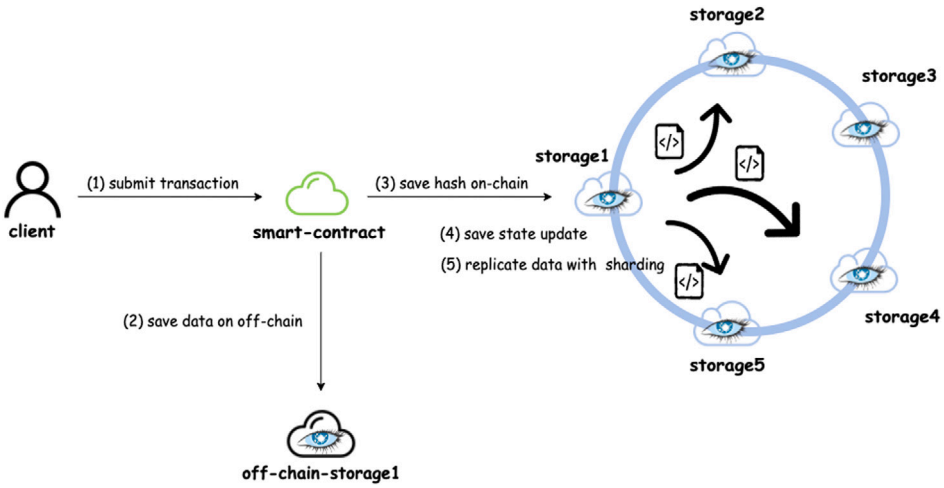

Fig. 5. Rahasak storage service, replicating data on three nodes with replication factor 3.

4. Kafka message broker — Kafka [22] based distributed publisher/subscriber service
5. Rahasak-CA — Certificate authority in Rahasak blockchain.

4.2. Aplos service

The Aplos service is the actor-based smart contract service in Rahasak [5]. All blockchain-based software programs and the messages that pass between them are written as Akka actors and saved in the Aplos service. Clients send transaction requests to this service with the actor name and its message. Based on that, the Aplos service finds the corresponding actor and passes the message to that actor. Then the actor validates and executes the transaction message. Based on the validate/execute outcome, it creates a transaction and updates the asset state in the underlying asset storage, as shown in Fig. 3. Finally, the transaction ID saves on the distributed cache. Based on the transactions IDs on the cache, Lokka service will be created the blocks in Group phase. The Aplos service has been implemented using the Scala functional programming languages. The Akka actor framework is used to build smart contract programs.

Each node in the network runs its own Aplos service. This service consumes transaction messages from Apache Kafka. There is a Kafka topic which the service listens to. A client publishes transaction messages to this Kafka topic. These Kafka topics can be partitioned and operate as a Kafka consumer group. Then Kafka handles the message partitioning and message broadcasting between the topic, guaranteeing total order(provide total order by sending a message only to one consumer by topic partitioning [22]). With partitioned topic Rahasak can run the multiple Aplos services in a single blockchain peer depends on the transaction load. Each Aplos service consumes transaction only once and executes them. As shown in Fig. 4, they can work independently and execute transactions concurrently. When a message is received by the Aplos service, it delegates the message to the corresponding actor based on transaction parameters, validation phase, and execution phases performed by the actor.

4.3. Storage service

Storage is the place where the blocks, transactions and assets are stored in Rahasak. We have used eventually consistent distributed database as the asset storage in Rahasak. Each peer in Rahasak blockchain comes with two types of storage, off-chain storage and on-chain storage. Off-chain storage stores the actual data generated by the peers. The hash of these data published to on-chain storage and shared with other peers. As shown in Fig. 5, the on-chain nodes are connected in a ring cluster. Once Aplos service executes transactions, it will write the state updates and transactions into its storage service instance (distributed database node). Then the saved data will be distributed to other nodes via the underlying distributed database.

Rahasak blockchain does not use full node data replication like Bitcoin or other blockchain platforms. Instead, Rahasak adopts a sharding-based data replication mechanism. In Rahasak blockchain, data replications are handled by the underlying distributed database. When one

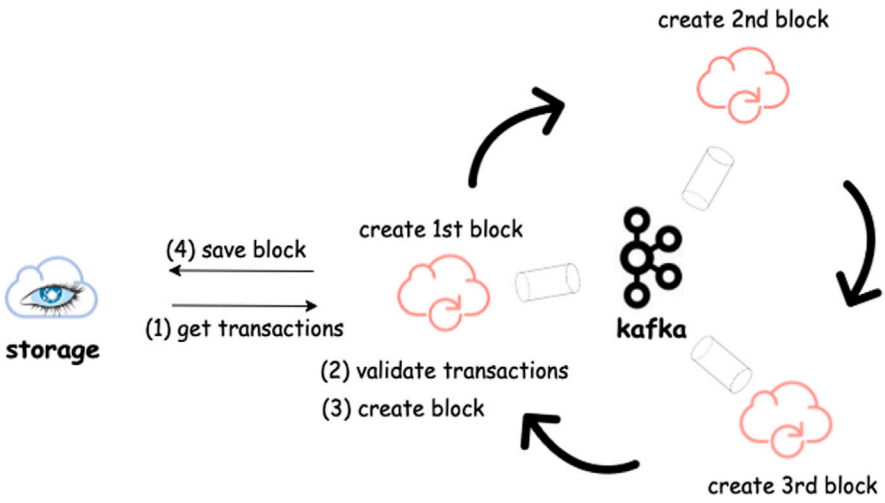

Fig. 6. Rahasak Lokka service architecture.

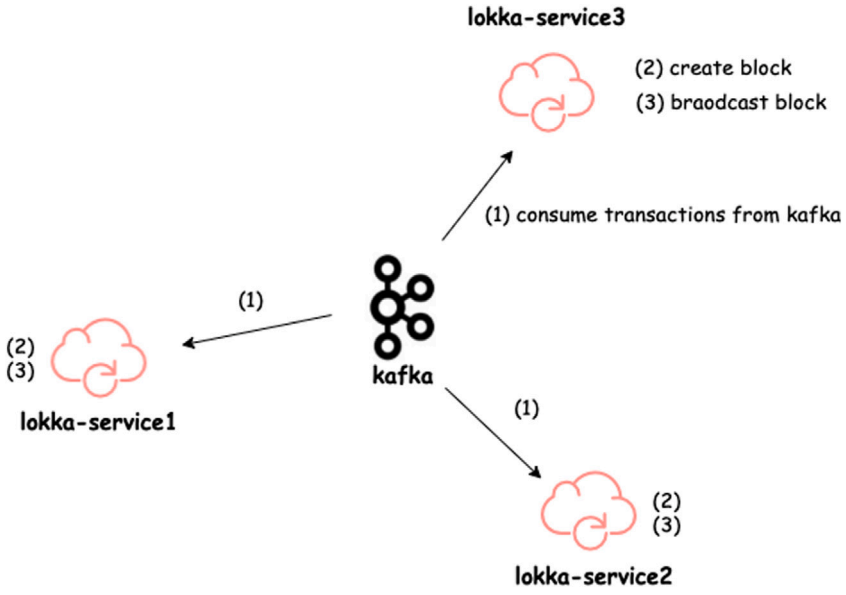

Fig. 7. Lokka services communicate via kafka.

blockchain node writes data to its storage service, the data will be replicated to only a certain number of blockchain nodes depending on the replication factor defined in the data replication procedure of the distributed database. For instance, in a 10-node blockchain cluster, if the `replication_factor` is 3, transactions will be replicated in 3 nodes.

We have chosen Apache Cassandra [23] as our storage service as we can write to any node in the Cassandra cluster [25]. In other database systems, one can only write to the master node. There is no master node on Cassandra, which has a masterless ring architecture. In the blockchain, every node should have equal write ability 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). All the data in `Transactions`, `Blocks` and `Asset` tables will be indexed in Apache Lucene index-based API [27,47] to achieve full-text search capability of the Rahasak blockchain.

4.4. Lokka service

In the Group Phase, the Lokka service creates block based on the transaction IDs in the cache. When creating a block, it generates the block hash and adds the transaction list to the block as a Cassandra user-defined type list. Block hash contains the previous block hash and Merkle root hash of the transaction list. Finally, the block is saved in the Blocks table. The newly created block ID (primary key of the block in Blocks table) is broadcast to other Lokka services via Kafka (each Lokka service has their own Kafka topic for communication). Then other Lokka services take the block from the Blocks table, verify the transactions in the block and digitally sign the block. When signing, they digitally sign the block hash and add the signature into the block header. The block generation and signing process happen in a fully asynchronous way.

There are multiple Lokka services in the system (each blockchain node may run their own Lokka service). Lokka services create blocks based on a time interval (e.g., create a block every second) or based on the number of transactions in the Redis cache (e.g., after every 100

transactions in Redis). The Block Creator is determined in a round-robin distributed scheduler. Consider the scenario in Fig. 6, which has 3 Lokka services. Assume that the first block is created by Lokka A, the second block will be created by Lokka B and the third block is by Lokka C. This goes on repeatedly.

4.5. Apache Kafka message broker

Apache Kafka is used as the consensus platform and message broker in Rahasak blockchain. All transactions published by the clients will be stored and ordered in a Kafka message broker. Aplos services take the ordered transactions from a Kafka message broker, execute them with smart contracts. Kafka message brokers in Rahasak focus on two main scenarios. In the first scenario, the client publishes transaction messages to the blockchain node via Kafka message broker. There is a separate Kafka message topic for each Aplos service in the blockchain peers. Clients publish transaction messages to these Kafka topics, Fig. 3. By using Kafka for client-to-blockchain communication, we can handle back-pressure operations [4] with handling a high transaction load in the scalable application. The second scenario is communication between Lokka services. When a Lokka service generates a block and saves it in the Blocks table, the block ID is broadcast to other Lokka services via Kafka for approval. As shown in Fig. 7 all communication between Lokka services happens through Kafka. We run 3 Kafka broker nodes with 3 Zookeeper nodes in Rahasak.

4.6. Rahasak-CA

Rahasak-CA is the certificate authority in Rahasak blockchain. The blockchain peers and clients public key certificates are issued by the Rahasak-CA. Apache Cassandra storage has been used as the certificate storage in Rahasak-CA. All the certificates issued by the Rahasak-CA will be stored in the Apache Cassandra storage. When bootstrapping a blockchain peer, it will generate a public–private key pair and submits the public key into Rahasak-CA. Then Rahasak-CA will digitally sign that public key and store it in the certificate storage. Then these certificates will be available for other peers and client in the blockchain network. Rahasak-CA exposes REST based API and Apache Kafka based streaming API to communicate. There are two main APIs available, (1) certificate publish API, (2) certificate search API. Blockchain peers and clients submit their public keys into the certificate store via certificate publish API. The certificates in the Rahasak-CA can be searched via the certificate search API.

4.7. Microservice deployment

Each blockchain node in the network runs their own Aplos, Storage and Lokka services. All these services are Dockerized and available to deploy with Kubernetes container orchestration system. The administrator for each node (e.g system admin of the organization) needs to configure and deploy their own services. Kafka cluster needs to be deployed independently from the Aplos, Storage and Lokka services. For the service deployments, Rahasak provides docker-compose based deployment scripts as well as Kubernetes helm chart based deployment scripts. The Kubernetes container orchestration system will manage the termination and recovery after the sudden failure of the microservices in each of the blockchain nodes.

5. Rahasak use case

As a use case of Rahasak blockchain, a connected vehicle (e.g containers and trucks) tracking platform with integrated 5G network is presented. The proposed platform is developed for ports to track their containers and trucks using GPS enabled 5G mobile phones. As shown in Fig. 8 the trucks and containers in the port travel to different locations. Each driver in the truck is equipped with a GPS enabled 5G mobile device. These 5G mobile devices will communicate with the 5G base stations. All 5G based stations are connected with Rahasak blockchain nodes in different ports(The Rahasak blockchain is deployed in each port in the network). The truck driver identity and their location-tracking information are both stored in the Rahasak blockchain. The 5G mobile device identity includes `Device Name`, `Device Address`, `Communication Port`, `Communication Protocol`, etc. There are two separate smart contracts in Rahasak blockchain to handle device identities and locations tracking functions. The `Device` smart contract implements the 5G mobile device identity handling functions(e.g device identity creation and device identity search functions) of the truck owners. The `Trace` smart contract implements the location tracking functions(location create and location search functions) of the 5G mobile devices of the truck owners. With this infrastructure, the base station tracks the location of each 5G device it communicates with and submits its location information to the Rahasak blockchain. Initially, 5G mobile device identities(which are used by the truck drivers in the port) need to be added(registered) to the blockchain. This is done via the web-based admin interface. When registering devices, the admin interface invokes `createDevice` function on `Device` smart contract. The registered devices can be searched via executing `searchDevice` functions on Device smart contacts. When a 5G enabled device transmits 5G packets to the base station, the based station extracts the location data from the packet and create location records which correspond to the 5G device in the blockchain. In this scenario the base station will invoke `traceLocation` function in the `Trace` smart contract.

This blockchain infrastructure can be used to detect suspicious trucks/containers and the locations of the trucks/containers which operates in different ports. All 5G mobile device identities of the truck owners are registered in the blockchain when they start the transportation. Then these device locations are continuously recorded in the Rahasak blockchain via the 5G base stations. The 5G mobile devices transmit their information to the base station as 5G packets. The packet attribute contains various information about the 5G device (e.g `Device Address`, `Frame Port`, `Frame Counter`, `Message Type` etc.). The `Device Address` field in the packet used to find the device identity of the 5G packet from the blockchain. The `Frame Counter` fields are used to check for de-duplication of transactions in the blockchain. When the 5G packet is received at the base station, it first extracts `Device Address` of that packet. Based on the `Device Address` in the packet, base station searches corresponding 5G device identity saved in the blockchain. It can be done via `searchDevice` function on `Device` smart contract. If the 5G device identity which corresponds to the packet exists on the blockchain, it is recognized as a valid packet(generated from the verified 5G device which belongs to the truck owner). If the device identity does not exist, it is recognized as a suspicious packet generated from an unauthorized device. Once a suspicious packet is found it can notify the corresponding authority. Each time a 5G packet is received at the base station, it extracts the location attributes from the packet. Then these location data will be feed into the Rahasak blockchain along with the `Device Address` field of the 5G device. In this way, all location records of the trucks and containers will be recorded in the Rahasak blockchain. When trucks move to different ports, they may be connected to the network via different base stations. This location tracking function is handled with `traceLocation` function in `Trace` smart contract. The proposed platform is operating in a highly scalable environment. The truck location information will be received to the base station as continuous streams. So high transaction load needs to be handled. The real-time transaction support Validate-Execute-Group blockchain architecture, Apache Kafka based back-pressure operation handling, Concurrent transaction execution of Aplos smart contracts enables Rahasak blockchain to operate in such a highly scalable application environment with supporting high transaction load.

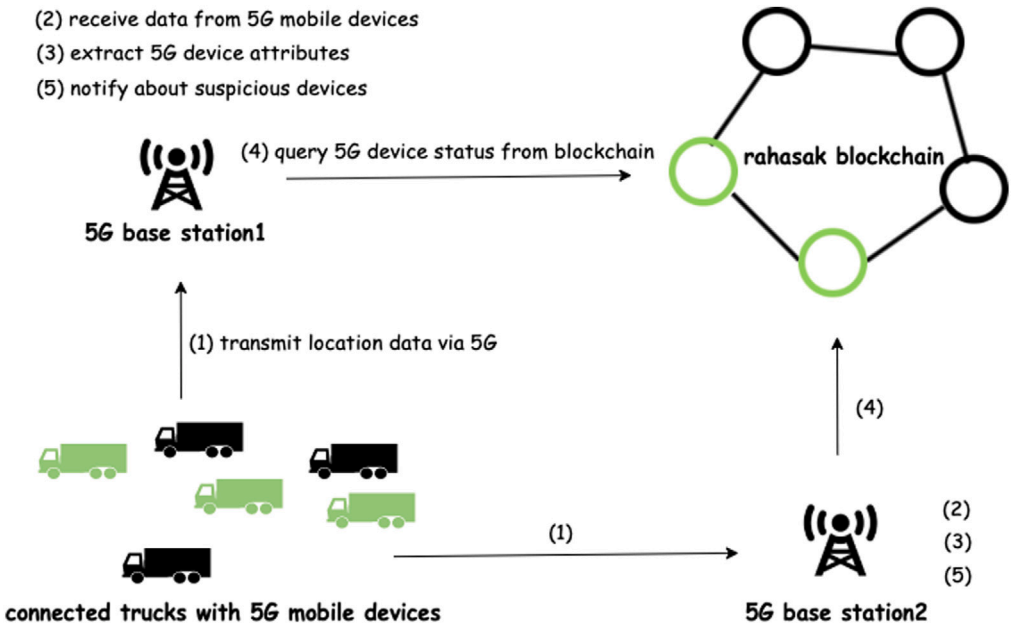

Fig. 8. Rahasak blockchain architecture of connected truck/container tracking platform in ports. 5G gateways deployed in different ports. These gateways can be connected to different blockchain nodes.

6. Rahasak performance evaluation

Performance evaluation of Rahasak is completed and discussed comparing Hyperledger Fabric and BigchainDB blockchains. To obtain the results Rahasak, Hyperledger Fabric and BigchainDB blockchains deployed on AWS cluster(AWS 2xlarge instance with 16GB RAM and 8 CPUs). Rahasak blockchain is set up to run with 4 Kafka nodes, 3 Zookeeper nodes and Apache Cassandra [23] as the state database. The smart contracts on the Rahasak blockchain are implemented with Scala functional programming and Akka actors. Hyperledger Fabric is set up to run with a Kafka based consensus utilizing 3 Orderer nodes, 4 Kafka nodes, 3 Zookeeper nodes and LevelDB [1] as the state database. BigchainDB blockchain is set up to run with Tendermint consensus [39] and MongoDB [48] as the state database. The evaluation tests performance for a varying number of blockchain peers (1 to 15 peers) and records the following results:

1. Transaction throughput
2. Transaction scalability
3. Transaction execution rate
4. Search performance
5. Block generation performance

6.1. Transaction throughput

For this evaluation, we recorded the number of invoke transactions and the number of query transactions that can be executed in each Rahasak blockchain peer. Invoke transactions update the status of the assets. Query transactions just read the status from the ledger without creating a transaction in the ledger or updating the asset statuses. We issued concurrent invoke, query transactions for each blockchain peer and recorded the number of executed transactions. As shown in Fig. 9, we compared the invoke transaction/query transaction throughput and obtained consistent throughput in each peer on the Rahasak blockchain. Since query's are not updating the ledger status, it has high throughput(2 times) compared to invoke transactions. Then as shown in Fig. 10, the invoke transaction throughput of Rahasak is compared to BigchainDB and Hyperledger Fabric. Rahasak performs with a higher transaction throughput than BigchainDB and Hyperledger Fabric. Hyperledger Fabric comes with Multi-Version Concurrency Control (MVCC [17]) based on Execute-Order-Validate blockchain architecture [1], BigchainDB comes with Blockchain-Pipeline architecture. Both of these architectures do not support real-time transactions. But Rahasak blockchain comes with real-time transaction supporting "Validate-Execute-Group" architecture. The smart contracts on both Hyperledger Fabric and BigchainDB do not support concurrent transaction execution since they are designed based on imperative style programming. The Aplos smart contract platform in Rahasak blockchain supports concurrent transaction execution using a functional programming based paradigm and actor-based concurrency handling system. Moreover, Rahasak supports Akka streams and Kafka streams based back-pressure operation handling in high transaction load scenarios. Due to these reasons("Validate-Execute-Group" architecture, functional programming-based smart contract platform, reactive streaming-based back-pressure handling) Rahasak blockchain produces higher transaction throughput than Hyperledger Fabric and BigchainDB blockchain systems. Similar to invoke transactions, the Query operation throughput of Rahasak is compared with BigchainDB and Hyperledger Fabric, Fig. 11. Rahasak serves its query API via Apache Lucene index-based search API. BigchainDB facilitates the query API with MongoDB and Hyperledger Fabric with CouchDB. Query transaction throughput of Rahasak is higher than both BigchainDB and Hyperledger Fabric. As mentioned above the real-time transaction enabled "Validate-Execute-Group" blockchain architecture, concurrent transaction execution of smart contracts and Akka streams based back-pressure handling are the main reasons for the improved results.

6.2. Transaction scalability

To evaluate transaction scalability, we recorded the number of invoke transactions and query transactions (per second) over the number of blockchain peers in the network. We issued concurrent transactions in each blockchain peer and recorded the number of executed transactions. Fig. 12 shows the transaction scalability comparison of Rahasak blockchain. When adding a node to the cluster, it nearly linearly increases the transaction throughput. Which means the transaction latency will decrease when adding blockchain peers to the cluster. Then as shown in Fig. 13 the transaction scalability of Rahasak is compared with BigchainDB and Hyperledger Fabric. Rahasak has higher scalability than BigchainDB and Hyperledger Fabric blockchains. Both

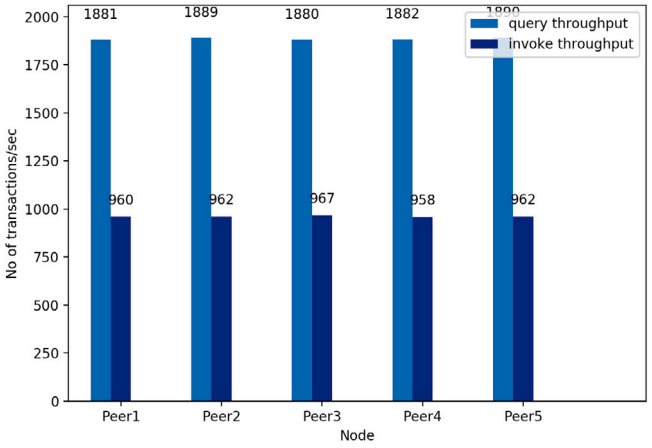

Fig. 9. Transaction throughput of Rahasak blockchain.

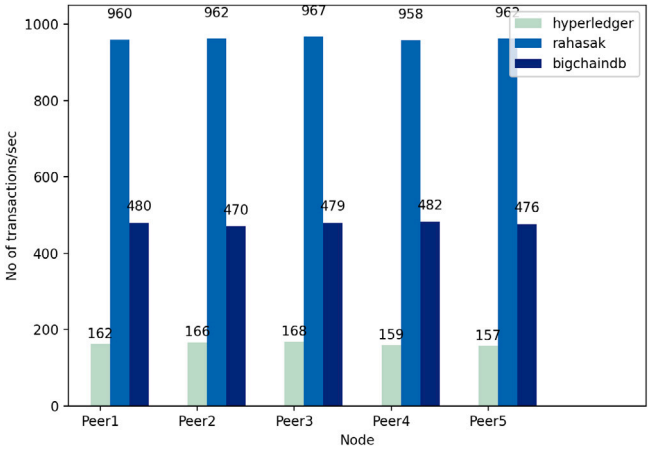

Fig. 10. Invoke transaction results of Rahasak, BigchainDB and Hyperledger Fabric.

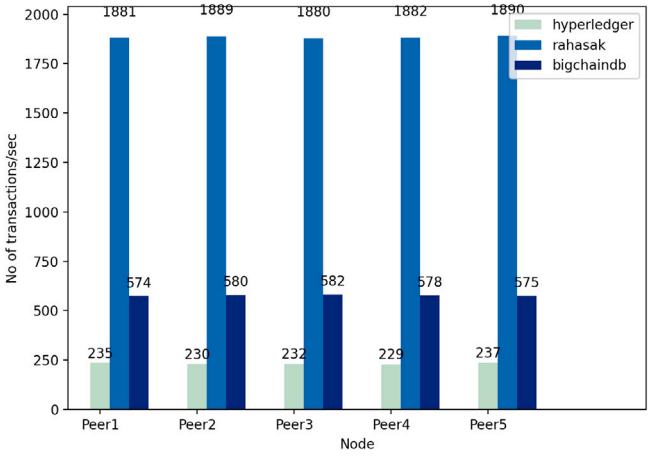

Fig. 11. Query transaction results of Rahasak, BigchainDB and Hyperledger Fabric.

Hyperledger Fabric and BigchainDB blockchains do not support real-time transaction enabled blockchain architecture. Hyperledger Fabric comes with Multi-Version Concurrency Control (MVCC [17]) based on Execute-Order-Validate blockchain architecture [1], BigchainDB comes with Blockchain-Pipeline architecture. Both architectures do not support real-time transactions. But Rahasak blockchain supports real-time transaction enabled "Validate-Execute-Group" blockchain architecture.

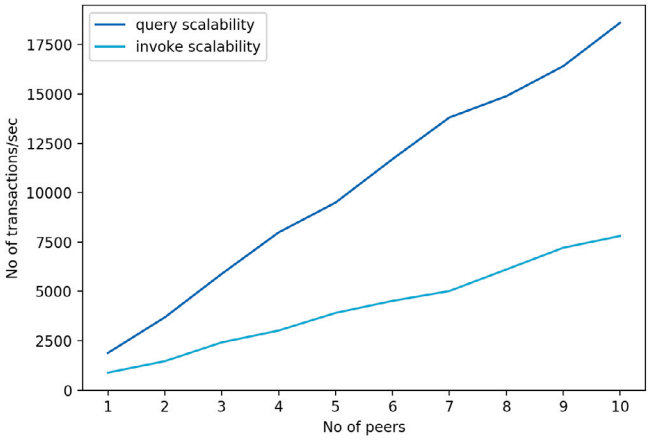

Fig. 12. Transaction scalability of Rahasak blockchain.

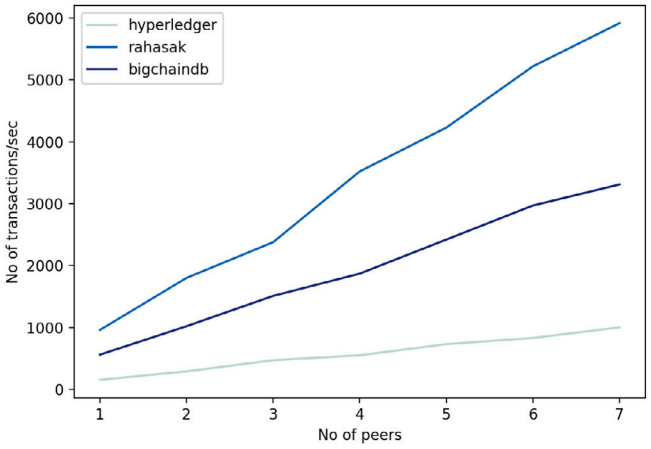

Fig. 13. Transaction scalability results of Rahasak, BigchainDB and Hyperledger Fabric.

With the "Validate-Execute-Group" blockchain architecture, the Aplos smart contract services in each Rahasak blockchain node can process/execute transactions independently with using partitioned Kafka topics. When adding nodes to the cluster, it nearly doubles the transaction throughput since Aplos services can process transactions concurrently. For example, one Aplos service processes 1000 transactions per second, two Aplos services can process approximately 2000 transactions per second. Due to this reason when the number of peers increases, the rate of executed transactions increase relatively. So when adding new nodes to the cluster, Rahasak linearly increases the transaction throughput.

6.3. Transaction execution rate

Next, we evaluate the transaction execution rate in the Rahasak platform. We tested the number of submitted transactions and executed transactions in different blockchain peers recording the time. Fig. 14 shows how transaction execution rate varies when having a different number of blockchain peers in the Rahasak(statistics observed up to 7 peer blockchain cluster). We have recorded the number of executed transactions with different no of peers against the time. When the number of peers increases, the rate of executed transactions is increased relatively. On other hand, we observed that Rahasak has consistent transaction throughput(with few ups and downs) against the time. Fig. 15 shows the number of executed transactions and submitted transactions in a single Rahasak blockchain peer. There is a gap between the rates of submitted transactions and executed transactions known

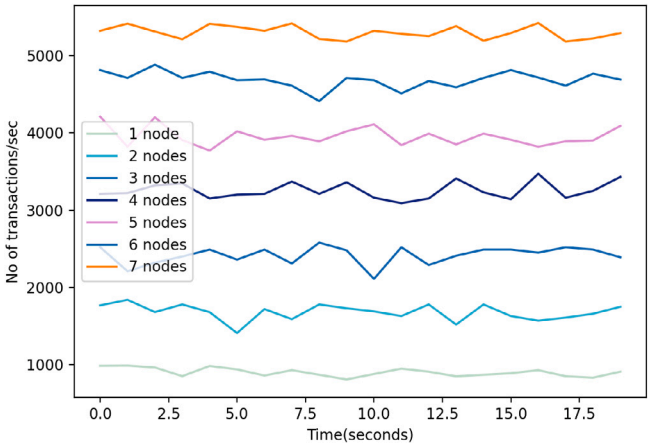

Fig. 14. Transaction execution rate with number of peers in the Rahasak blockchain.

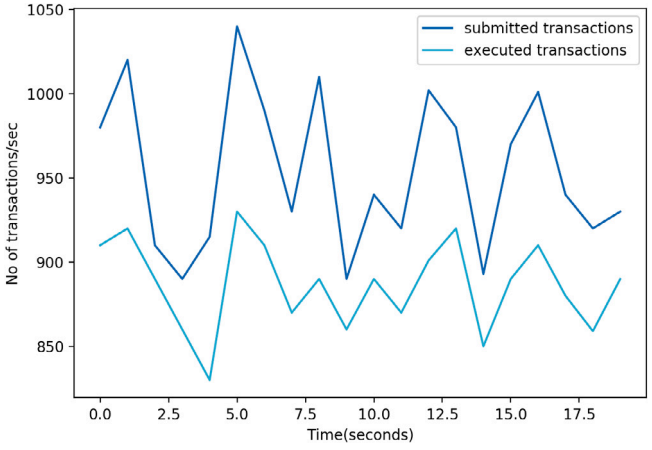

Fig. 15. Transaction execution rate and transaction submission rate in a single blockchain peer.

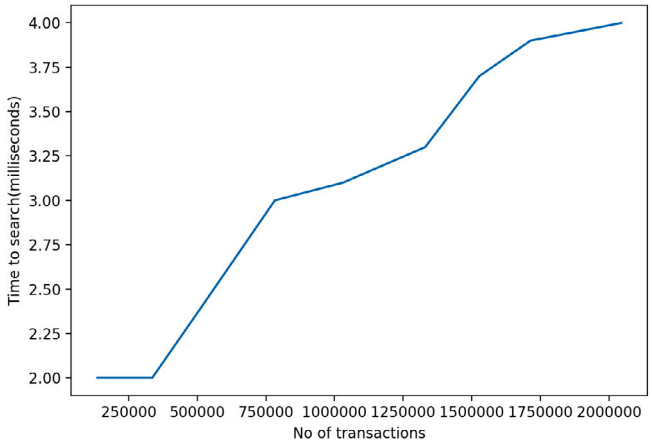

Fig. 16. Search performance of Rahasak blockchain.

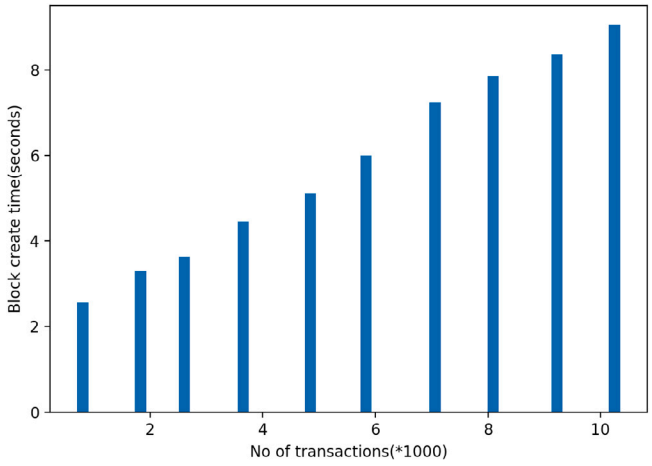

Fig. 17. Block generation time against the number of transactions in the block.

as the transaction back-pressure [4]. The transaction execution rate is lower than the transaction submission rate. A reactive streaming-based approach with Apache Kafka is used to handle back-pressure operations in the Rahasak blockchain.

6.4. Search performance

Rahasak allows one to search for data in the transaction/block/asset tables using its Apache Lucene index-based search API. Rahasak stores all its data(blocks, transactions, blockchain assets) on Apache Cassandra based Elassandra storage. Elassandra adds Apache Lucene index-based search API(e.g Elasticsearch API) into Cassandra storage. With Elassandra we automatically indexed all the data in Rahasak Cassandra storage on Elasticsearch API. The full-text search of the Rahasak blockchain facilitates with this Elasticsearch API. For this evaluation, we issued concurrent transaction search requests to Rahasak blockchains search API and computed the search time. Different transaction data sets are used in this experiment and the time to search a single transaction record from each transaction data set is calculated. Shown in Fig. 16, to search a transaction record from 2 million transaction data set, it took only 4 ms. Search performance of the Apache Lucene index-based API and concurrent transaction execution of the Aplos smart contract service are the main reasons yielding faster search in Rahasak. These search performance results obtained from a seven node Rahasak blockchain cluster.

6.5. Block generate time

In this evaluation, the time taken to generate blocks in the Rahasak blockchain is evaluated. First, the block generation time statistics is recorded against the number of transactions in the block from a seven peer Rahasak blockchain cluster. Block generation time depends on three main time factors (a). data replication and broadcast time between peers (b). Merkel proof/block hash generate time (c). transaction validation time. When the transaction count increases in the block, these factors will be increased. Due to this reason, when the transaction count increases, block generation time also increases correspondingly. As shown in Fig. 17 to increase a block when having 10k transaction, it takes 8 s. Next, evaluated block generation time against the number of blockchain peers in the cluster is evaluated. A block with a 2000 transaction set is used and the time to generate the block with the different number of blockchain peers(up to 7 peers) is calculated and evaluated. When adding peers to the cluster the above mentioned time factors(data replication and broadcast time between peers, Merkel proof/block hash generate time, transaction validation time) will be increased since each peer to need to validate transactions in the block and recalculate block header. Due to this reason when adding peers to the cluster block generation time also increases correspondingly in Rahasak blockchain, Fig. 18.

Table 2
Blockchain platform comparison.

Blockchain	Public/Private	Architecture	Consensus	Scalability	Smart contract	Full text search	Concurrent transactions	Sharding
Rahasak	Private	Validate-Execute-Group	Kafka	High	Yes	Yes	Yes	Yes
BigchainDB [2]	Both	Blockchain-Pipeline	Tendermint	High	Yes	Yes	No	Yes
HbasechainDB [6]	Private	Blockchain-Pipeline	ZAB	High	No	Yes	No	Yes
Hyperledger [1]	Private	Execute-Order-Validate	Kafka/ZAB	Mid	Yes	No	No	No
LSB [49]	Private	Order-Execute						
with Cluster Heads	LC	High	No	No	No	No		
Lightchain [9]	Public	Order-Execute						
with Light Block	SMP	Mid	No	No	No	Yes		
Chain	Public	Order-Execute	Federated	Mid	No	No	Yes	Yes
RapidChain [3]	Public	Order-Execute	Federated	Mid	No	No	No	Yes
RSCoin [8]	Private	Order-Execute	2PC variant	Mid	Yes	No	No	Yes
Bitcoin-NG [8]	Public	Order-Execute	PoW	Mid	No	No	No	No
Sensor-Chain [11]	Public	N/A	N/A	Mid	No	No	No	Yes

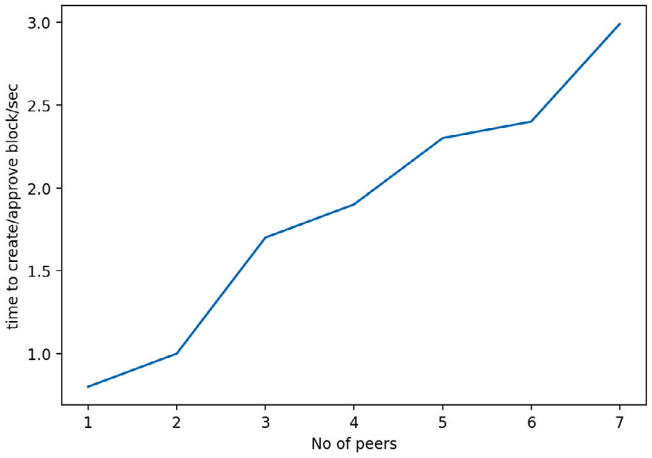

Fig. 18. Block generation time against the number of peers in the cluster.

7. Related work

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

BigchainDB [2] is an enterprise blockchain database built on top of MongoDB [48]. The consensus is handled by the underlying MongoDB. They have added blockchain features (decentralized control, immutability, movement of digital assets) into MongoDB to make it a scalable blockchain database. The major contribution of BigchainDB is to enable blockchain scalability 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 [6] is a similar kind of blockchain database to BigchainDB. Instead of MongoDB, they used the Apache HBase with Hadoop. Like Bigchaindb, HbasechainDB uses blockchain pipelining and majority vote federated consensus 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.

Hyperledger Fabric [1] is a permissioned blockchain system using a modular design approach that allows scalability, extensibility and flexibility. It comes in different consensus algorithms which can be configured `Kafka`, `RBFT`, `Sumeragi`, and `PoET`. Hyperledger comes with a smart contract [15] platform called `Chaincode`. Users can write chaincode contracts with Golang [50]. Hyperledger uses Apache Kafka to facilitate private communication channels between the nodes. It can achieve up to 3500 transactions per second in certain popular deployments.

LSB — Lightweight Scalable Blockchain [49] is a lightweight and scalable blockchain storage that is optimized for IoT requirements. It solves five main challenges when integrating existing blockchain platforms within the IoT environment. These challenges are complex consensus algorithms, Scalability and overheads, Latency, Security overheads, and Throughput. It uses a Smart home setting for illustrative purposes for LSB blockchain in IoT. LSB is application agnostic and well-suited for diverse IoT applications. To meet the requirements of blockchain for IoT, LSB incorporates several optimizations which include a Lightweight Consensus(LC) algorithm, a distributed trust method, a distributed throughput management strategy, and a separation of the transaction traffic from the data flow.

Lightchain [9] is a lightweight blockchain system built for power-constrained IIoT use cases providing a resource-efficient solution. It focuses on improving by reducing computing power consumption, storage space usage, and network resources usage. The consensus is handled by the protocol Synergistic Multiple Proof (SMP) for stimulating the cooperation of IIoT devices. For it to consider the limitation on the storage resource, it utilizes a novel Unrelated Block Offloading Filter to prevent an enormous growth of the ledger without affecting blockchain's traceability. This is accomplished with a lightweight data structure called LightBlock (LB) providing the ability to streamline broadcast content.

Chain [7] is blockchain storage that mainly targets private blockchains. It uses federated consensus. The majority of the nodes need to vote for a block to add to the ledger. Unlike other blockchains, it validates transactions in the blocks concurrently. All 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.

RapidChain [3] is the first sharding-based public blockchain protocol that is resilient against Byzantine faults [20,21]. 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 7300 transactions per second.

RSCoin [8] is a sharding-based blockchain protocol to enable the scalability of centrally-banked crypto-currencies. It comes with a centralized monetary supply and distributed transaction ledger. A set of authorities called `mintettes` perform validation (double-spend checking) of 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 maintaining the integrity of the transaction

ledger. RSCoin guarantees that it can process 2000 transactions per second.

Bitcoin-NG(Next Generation) [10] 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. Bitcoin-NG'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. Bitcoin-NG achieves this performance improvement by decoupling Bitcoin's blockchain operation into two planes: leader election and transaction serialization. It divides time into epochs, where each epoch has a single leader. As in Bitcoin, leader election is performed randomly and infrequently. Once a leader is chosen, it is entitled to serialize transactions unilaterally until a new leader is chosen, marking the end of the former's epoch. With this approach, Bitcoin-Ng achieves significantly higher throughput and lower latency than Bitcoin while maintaining the Bitcoin trust assumptions.

Sensor-Chain [11] is a lightweight blockchain framework for mobile IoT. To the reduction in resource consumption, it breaks downs global blockchain into smaller "local" blockchains in the spatial domain and limiting their size through a temporal constraint. The proposed work is independent of any particular ledger platform. Thus, it can be implemented with any blockchain platform (e.g. Ethereum, hyperledger, and so on) for IoT. With the proposed architecture Sensor-Chain blockchain framework consumes little storage space on the IoT sensor devices and is scalable with the increase in network size.

The comparison summary of these blockchain platforms and Rahasak platform is presented on Table 2. It compares Blockchain type (public, private), Architecture, Consensus, Scalability level, Smart contract support, Full-text search support, Concurrent transaction support and Sharding details. Table 1 summarizes how performance bottleneck features(i.e., real-time transactions with O-E model, concurrent execution of smart contracts and sharded replications) are solved on existing blockchain platforms and smart contract platforms. Compared with other Blockchain platform implementations, Rahasak is a permissioned Blockchain system which is targeted for scalable, enterprise-level applications such as big data, cloud computing, edge computing, and IoT. Meanwhile, the smart contract function, the full-text search, concurrent transactions and sharding capabilities are supported by Rahasak and make it a compelling solution for practical integration of Blockchain technology in real-world scenarios.

8. Conclusions and future work

With Rahasak, a blockchain for highly scalable, concurrent applications such as big data, cloud computing, edge computing, IoT, edge computing etc is developed and presented. Rahasak exhibits high transaction throughput, high scalability, and high availability. Validate-Execute-Group blockchain architecture to achieve real-time transaction from the blockchain is introduced. The new blockchain architecture is designed with Apache Kafka as the consensus platform. The functional programming and actor based smart contract platform in Rahasak supports concurrent execution of transactions. Full-text search capability is implemented into Rahasak by indexing transactions and blocks on Apache Lucene index-based search API. Rahasak blockchain's microservice-based architecture with Docker/Kubernetes enables easy deployment and easy scalability. Rahasak makes blockchain data more secure and meaningful where real-time data analytic and anomaly detection can be easily performed. The scalability and transaction throughput features with empirical evaluations is shown. Rahasak is integrated into production-grade applications in the banking and financial sectors where the deployments are evidence for Rahasak as an ideal blockchain system for highly scalable, enterprise-level applications. Future developments include implementing the following features in Rahasak. (a) Support Self Sovereign Identity [51,52] with Zero-Knowledge Proof [53] in Rahasak blockchain, (b) Do a formal verification of the Aplos smart actor platform and its security design, (c) Implement Zero Trust Authentication with Rahasak-CA.

CRediT authorship contribution statement

Eranga Bandara: Methodology, Software implementation, Writing, Draft preparation. **Xueping Liang:** Investigation, Writing - reviewing and editing, Supervision. **Peter Foytik:** Writing - reviewing and editing. **Sachin Shetty:** Supervision, Investigation, Writing - reviewing and editing. **Nalin Ranasinghe:** Supervision, Investigation. **Kasun De Zoysa:** Supervision, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was funded by the Department of Energy (DOE) Office of Fossil Energy (FE), USA (Federal Grant #DE-FE0031744).

References

[1] 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.

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

[3] M. Zamani, M. Movahedi, M. Raykova, RapidChain: A fast blockchain protocol via full sharding, IACR Cryptol. ePrint Arch. 2018 (2018) 460.

[4] A. Destounis, G.S. Paschos, I. Koutsopoulos, Streaming big data meets backpressure in distributed network computation, in: IEEE INFOCOM 2016-the 35th Annual IEEE International Conference on Computer Communications, IEEE, 2016, pp. 1–9.

[5] E. Bandara, W.K. NG, K. De Zoysa, N. Ranasinghe, Aplos: Smart contracts made smart, in: BlockSys'2019, 2019.

[6] M.S. Sahoo, P.K. Baruah, HBasechainDB–a scalable blockchain framework on Hadoop ecosystem, in: Asian Conference on Supercomputing Frontiers, Springer, 2018, pp. 18–29.

[7] Chain Protocol Whitepaper. URL https://chain.com/docs/1.2/protocol/papers/whitepaper.

[8] G. Danezis, S. Meiklejohn, Centrally banked cryptocurrencies, 2015, arXiv preprint arXiv:1505.06895.

[9] Y. Liu, K. Wang, Y. Lin, W. Xu, LightChain: A lightweight blockchain system for industrial Internet of Things, IEEE Trans. Ind. Inf. 15 (6) (2019) 3571–3581.

[10] I. Eyal, A.E. Gencer, E.G. Sirer, R. Van Renesse, Bitcoin-NG: A scalable blockchain protocol, in: NSDI, 2016, pp. 45–59.

[11] A.R. Shahid, N. Pissinou, C. Staier, R. Kwan, Sensor-Chain: a lightweight scalable blockchain framework for internet of things, in: 2019 International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2019, pp. 1154–1161.

[12] R. O'Connor, Simplicity: A new language for blockchains, in: Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security, ACM, 2017, pp. 107–120.

[13] S. Popejoy, The pact smart contract language, 2016, June-2017.[Online]. Available: http://kadena.io/docs/Kadena-PactWhitepaper.pdf.

[14] E. Eykholt, G. Meredith, J. Denman, Rchain architecture documentation, 2017.

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

[16] I. Sergey, A. Kumar, A. Hobor, Scilla: a smart contract intermediate-level language, 2018.

[17] P. Di Sanzo, B. Ciciani, F. Quaglia, P. Romano, A performance model of multi-version concurrency control, in: 2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems, IEEE, 2008, pp. 1–10.

[18] E. Bandara, W.K. Ng, K.D. Zoysa, N. Fernando, S. Tharaka, P. Maurakirinathan, N. Jayasuriya, Mystiko - Blockchain meets big data, in: IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018, pp. 3024–3032.

[19] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System. Working Paper, 2008.

[20] L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401.

[21] M. Castro, B. Liskov, et al., Practical Byzantine fault tolerance, in: OSDI, vol. 99, 1999, pp. 173–186.
[22] 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.
[23] A. Lakshman, P. Malik, Cassandra: a decentralized structured storage system, Oper. Syst. Rev. 44 (2) (2010) 35–40.
[24] L. Lamport, The part-time parliament, ACM Trans. Comput. Syst. (TOCS) 16 (2) (1998) 133–169.
[25] J.a. Louren¸co, B. Cabral, P. Carreiro, M. Vieira, J. Bernardino, Choosing the right NoSQL database for the job: a quality attribute evaluation, J. Big Data 2 (2015) 18, http://dx.doi.org/10.1186/s40537-015-0025-0.
[26] Akka Streams Documentation. URL https://doc.akka.io/docs/akka/2.5/stream/.
[27] A. Białecki, R. Muir, G. Ingersoll, L. Imagination, Apache lucene 4, in: SIGIR 2012 Workshop on Open Source Information Retrieval, 2012, p. 17.
[28] J. Hughes, Why functional programming matters, Comput. J. 32 (2) (1989) 98–107.
[29] C. Hewitt, Actor model of computation: scalable robust information systems, 2010, arXiv preprint arXiv:1008.1459.
[30] C.A.R. Hoare, Communicating sequential processes, Commun. ACM 21 (8) (1978) 666–677.
[31] J.C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J.J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al., Spanner: Google's globally distributed database, ACM Trans. Comput. Syst. (TOCS) 31 (3) (2013) 8.
[32] I.L. Traiger, J. Gray, C.A. Galtieri, B.G. Lindsay, Transactions and consistency in distributed database systems, ACM Trans. Database Syst. 7 (3) (1982) 323–342.
[33] N.S. Barghouti, G.E. Kaiser, Concurrency control in advanced database applications, ACM Comput. Surv. 23 (3) (1991) 269–317.
[34] E. Brewer, Towards robust distributed systems, in: PODC, 2000, p. 7.
[35] How do I accomplish lightweight transactions with linearizable consistency? URL https://rb.gy/nh4kjp.
[36] A. Kurath, Analyzing Serializability of Cassandra Applications (Master's thesis), ETH Zürich, 2017.
[37] Coreos, Coreos/etcd, 2018, URL https://github.com/coreos/etcd.
[38] G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, A. Edmonds, An architecture for self-managing microservices, in: Proceedings of the 1st International Workshop on Automated Incident Management in Cloud, 2015, pp. 19–24.
[39] E. Buchman, Tendermint: Byzantine Fault Tolerance in the Age of Blockchains (Ph.D. thesis), 2016.
[40] C. Cachin, M. Vukolić, Blockchain consensus protocols in the wild, 2017, arXiv preprint arXiv:1707.01873.
[41] J. Thönes, Microservices, IEEE Softw. 32 (1) (2015) 116.
[42] D. Merkel, Docker: lightweight linux containers for consistent development and deployment, Linux J. 2014 (239) (2014) 2.
[43] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Borg, omega, and kubernetes, Queue 14 (1) (2016) 70–93.
[44] The Scala Programming Language. URL https://www.scala-lang.org/.
[45] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, M. Zenger, An Overview of the Scala Programming Language, Tech. Rep., 2004.
[46] Akka Documentation. URL https://doc.akka.io/docs/akka/2.5/actors.html.
[47] C. Gormley, Z. Tong, Elasticsearch: The Definitive Guide: a Distributed Real-Time Search and Analytics Engine, " O'Reilly Media, Inc.", 2015.
[48] V. Abramova, J. Bernardino, NoSQL databases: MongoDB vs cassandra, in: Proceedings of the International C* Conference on Computer Science and Software Engineering, 2013, pp. 14–22.
[49] A. Dorri, S.S. Kanhere, R. Jurdak, P. Gauravaram, Lsb: A lightweight scalable blockchain for iot security and privacy, 2017, arXiv preprint arXiv:1712.02969.
[50] F. Schmager, N. Cameron, J. Noble, GoHotDraw: Evaluating the Go programming language with design patterns, in: Evaluation and Usability of Programming Languages and Tools, ACM, 2010, p. 10.
[51] A. Mühle, A. Grüner, T. Gayvoronskaya, C. Meinel, A survey on essential components of a self-sovereign identity, Comp. Sci. Rev. 30 (2018) 80–86.
[52] D. Baars, Towards Self-Sovereign Identity Using Blockchain Technology (Master's thesis), University of Twente, 2016.
[53] N. Kulabukhova, Zero-knowledge proof in self-sovereign identity, in: CEUR Workshop Proceedings, vol. 2507, RWTH Aahen University, 2019, pp. 381–385.

Eranga Bandara worked as a Senior Research Scientist at the Virginia Modeling Analysis and Simulation Center (VMASC) Virginia USA. His research interests include Distributed Systems, Blockchain, Big Data, Actor based Systems and Functional programming. He worked as a Lead Engineer at Pagero AB Sweden. With Pagero AB he was involved with research and developments in Distributed Systems, Functional Programming, Big Data, Actor based systems and DevOps.

Xueping Liang is an Assistant Professor in the Department of Information Systems and Supply Chain Management at the University of North Carolina at Greensboro. Before that, she served as a cybersecurity research scientist at Old Dominion University's Virginia Modeling, Analysis, and Simulation Center and a security research analyst at Tennessee State University's Tiger Institute. Her main research focus involves security and privacy, trusted computing, distributed architecture, and blockchain. She has worked on several research and application projects using Intel SGX and blockchain, and one of her papers has been awarded the Top 50 Most Influential Papers of Blockchain in 2019. Dr. Liang received her Ph.D. in Cyber Security from the Institute of Information Engineering at the University of Chinese Academy of Sciences in 2019.

Peter Foytik is currently a Lead Project Scientist at the Virginia Modeling Analysis and Simulation Center (VMASC) Virginia USA. His work has included modeling and simulation applications in transportation, department of defence, and cybersecurity. His research interests include Modeling and Simulation, Distributed Systems, Blockchain, and Analysis of Complex Systems. He is currently working on his Ph.D. research which is targeted on the use of modeling and simulation to explore problem spaces.

Dr. Sachin Shetty is an Associate Director in the Virginia Modeling, Analysis and Simulation Center at Old Dominion University and an Associate Professor with the Department of Computational Modeling and Simulation Engineering. Sachin Shetty received his Ph.D. in Modeling and Simulation from the Old Dominion University in 2007. His research interests lie at the intersection of computer networking, network security and machine learning. Recently, he has been involved with developing cyber risk/resilience metrics for critical infrastructure and blockchain technologies for distributed system security. His laboratory has been supported by the National Science Foundation, Air Office of Scientific Research, Air Force Research Lab, Office of Naval Research, Department of Homeland Security, and Boeing. He has published over 150 research articles in journals and conference proceedings and four books. He is the recipient of Commonwealth Cyber Initiative Research Fellow, Fulbright Specialist award, EPRI Cybersecurity Research Challenge award, DHS Scientific Leadership Award and has been inducted in Tennessee State University's million-dollar club.

Dr. D.N. Ranasinghe is from University of Colombo School of Computing Sri Lanka. His research interests include Models of computation, GPU cluster computing, Natural heuristics for combinatorial optimization, Scalable fault-tolerant distributed algorithms, Opportunistic networks performance models, Distributed systems and Blockchain. He did his Ph.D. in the area of Network Performance Modelling at Cardiff University, UK, 1994. He was the Chair at IEEE-CS Sri Lanka Chapter (2010–2012), Vice Chairman at IEEE Sri Lanka Section (2004–2005) and Chair at ICTER conference, Colombo, 2014.

Dr Kasun De Zoysa is from University of Colombo School of Computing Sri Lanka. His research interests include Cryptocurrency, Multi-party document protection, Certification infrastructures, Security tokens, Web application security and privacy, Security in wireless ad-hoc and sensor networks, and Digital forensics. He obtained his Ph.D. in the area of Secure Electronic Payments based on Smart Cards from Stockholm University, Sweden, 2004. He has contributed over 20 research and development projects funded by various international funding agencies.