Future Generation Computer Systems 119 (2021) 154–165

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Tikiri—Towards a lightweight blockchain for IoT

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

[a] Old Dominion University, Norfolk, VA, USA
[b] University of Texas at El Paso, El Paso, TX, USA
[c] University of Colombo School of Computing, Sri Lanka

ARTICLE INFO

Article history:
Received 10 April 2020
Received in revised form 9 January 2021
Accepted 13 February 2021
Available online 16 February 2021

Keywords:
Blockchain
Smart contact
Edge computing
IoT
Big data

ABSTRACT

Internet of Things (IoT) platforms have been deployed in several domains to enhance efficiency of business process and improve productivity. Most IoT platforms comprise of heterogeneous software and hardware components which can potentially introduce security and privacy challenges. Blockchain technology has been proposed as one of the solutions to realize IoT security by leveraging the (a) Immutable ledger, (b) Decentralized architecture and (c) Strong cryptography primitives. However, integrating blockchain platforms with IoT based applications presents several challenges due to lack of (a) acceptable performance on resource-constrained devices, (b) high transaction throughput, (c) keyword-based search and retrieve, (d) transaction back pressure operations, and (e) real-time response. In this paper, we propose a lightweight blockchain platform, "Tikiri", for resource-constrained IoT devices. Tikiri uses Apache Kafka for the consensus and proposes new blockchain architecture to handle real-time transaction execution on the blockchain. Tikiri is characterized by functional programming and actor-based smart contract platform that realizes concurrent execution of transactions in the blockchain. Tikiri realizes a lightweight and scalable blockchain that can provides performance on the resource-constrained IoT devices.

1. Introduction

Internet of Things (IoT) technology leverages the cyber infrastructure and the physical world effectively to impact transportation, medical, consumer electronics sectors. Adoption of IoT in both civilian and tactical applications has been exponentially rising, which is the fundamental reason behind innovating many IoT solutions [1–3]. However, the exponential growth results in increased connectivity and complex computing infrastructure that is more vulnerable to cyber-attacks. Although the networked and pervasively connected IoT devices allow enabling greater situational awareness, it also expands the attack surface where vulnerabilities in those devices could damage the entire ecosystem by propagating through other vulnerable devices. Since many of the devices remotely controlled, the data along with commands generally travel over an untrusted medium which is highly prone to malicious alteration. Hence, it is critical to devise robust authentication and data provenance mechanisms for resource-constrained IoT networks. Given the centralized nature of IoT deployments, it raises concerns of single-point-failure and targeted cyber-attacks. Thus, it is important to depart from such centralized paradigm and build a truly decentralized and inter-operable security solution that can enable trustworthy and reliable data transfer among the connected IoT devices.

One approach that is being researched is the use of blockchain technology to improve the integrity issues with IoT [4,5]. Blockchain can serve as an effective IoT security solution because of following reasons: (1) tamper-resistant distributed ledger could enable untrusted participants to record important transactions; and (2) the high-availability yet resilient feature of the blockchain infrastructure will deter system failures and other cyber-attacks. Since blockchain exhibits unique transparency yet immutability feature on the data transactions, it could inherently establish trust between the IoT devices and back-end servers.

Currently, there are many different blockchain platforms that exist, such as: Bitcoin [6], Ethereum [7], BigchainDB [8], Hyperledger [9], Rapidchain [10], Mystiko [11]. Although public blockchains such as Bitcoin and Ethereum have been rigorously used for crypto-currency applications, they have been evolving to accommodate other e-commerce use cases. There have been several permissioned blockchains developed, including Hyperledger Suite by IBM, which could be controlled and deployed inside the organization. This has a different transaction storage model compared to Ethereum, however both offer the smart contract execution features. Furthermore, several blockchain solutions, such as

* Corresponding author.
E-mail addresses: cmedawer@odu.edu (E. Bandara), dktosh@utep.edu (D. Tosh), 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.future.2021.02.006
0167-739X/

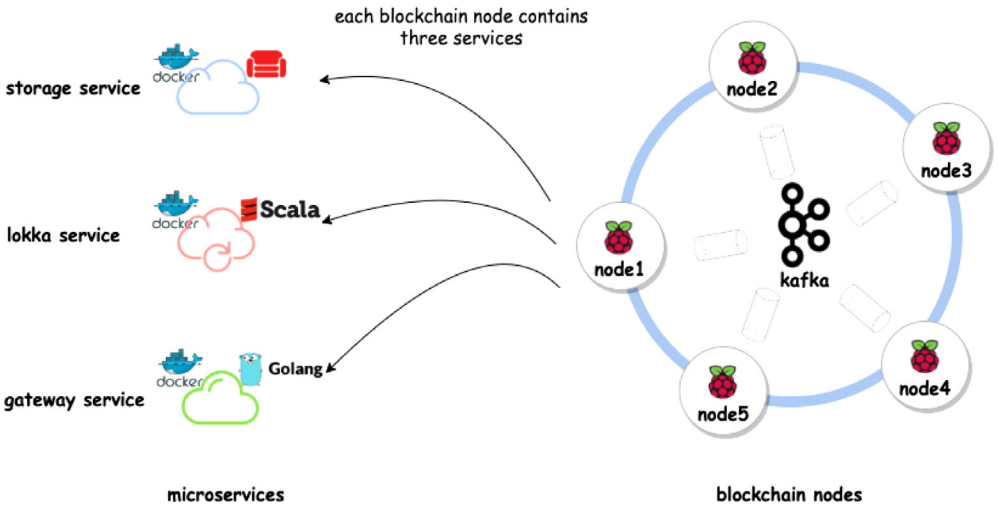

Fig. 1. Tikiri blockchain microservice-based architecture. Each blockchain node contains three services — storage service, Lokka service and gateway service.

Mystiko and BigchainDB, have been created especially for big data applications. Most of the blockchains support a unique programming paradigm called "Smart Contract" for interacting with the distributed ledger, which lies between the client and blockchain. This software layer takes clients' requests in the form of scripts written in the smart contract and interacts with the blockchain to maintain state, enabling governance, performing authentication etc. Although most of the smart contract platforms have similar functionalities, their working and implementation details are different. For example, Ethereum supports Solidity [12], Kadena supports Pact [13], RChain supports Rholang [14], Hyperledger Fabric supports Chaincode [9] and Mystiko supports Aplos [11,15] as their smart contract platform.

However, when integrating the existing blockchain platforms with resource-constrained IoT device-based applications, the following challenges are unavoidable: (1) existing blockchain platforms and their consensus algorithms are computationally heavy and difficult to run on resource-constrained IoT devices; (2) most blockchain implementations are based on monolithic architecture where all the functionalities (such as consensus, storage, client interaction) are implemented on single service; (3) current blockchain systems do not meet high transaction throughput demands of IoT applications; (4) real-time transaction processing is not supported; (5) lack of rich querying (full-text search) APIs; and (6) high overhead for full node data replication among resource-constrained IoT devices in terms of bandwidth and storage.

In this paper, we introduce a new lightweight and scalable blockchain platform, "Tikiri" for resource-constrained IoT devices. The Tikiri platform is built based on Hyperledger Fabric and Mystiko blockchains. The consensus protocol is designed on top of Apache Kafka [16]. Its smart contracts are written with a functional programming language [17] and Actor [18,19] based Aplos smart contracts. To run on lightweight resource-constrained environments, we built the blockchain with Golang [20,21] and Scala [22,23] languages-based on Microservices [24] architecture.

The performance of Tikiri blockchain is evaluated with a Raspberry-Pi based test-bed. The performance evaluation guarantees the success of Tikiri blockchain towards resource-constrained IoT environments. Following are the main contributions from Tikiri.

1. Real-time transaction performance blockchain architecture is introduced to avoid the overhead of order-execute architecture in the blockchain.
2. Leveraging actor-based smart contracts and functional programming to achieve concurrent transactions on the blockchain.
3. Reduction of performance overheads, including communication and computation, by opting for a sharding-based data replication approach without using full-node data replication.
4. Increased scalability and transaction throughput of the blockchain by using Apache Kafka based consensus.
5. Microservices based distributed system architecture introduced to build a lightweight blockchain.
6. Backpressure operations on IoT applications handled with Reactive Streaming based methodology.
7. The full-text search capability of the Tikiri blockchain has been facilitated by building search indices of the blockchain data (blocks, transactions, blockchain assets) on a lightweight distributed database.

The paper is organized as follows. Section 2 discusses the architecture of the Tikiri blockchain. Section 3 explains the details of the Tikiri blockchain implementation. Section 4 presents the performance evaluation of the Tikiri blockchain in comparison with Hyperledger Fabric platform. Section 5 offers insights into the related work. Finally, Section 6 concludes the Tikiri platform with future directions.

2. Tikiri architecture

2.1. Overview

Today, even the most advanced blockchain systems are built as monolithic systems [11]. All features of the blockchain in some cases are controlled from programs that are hosted by a single service [9,11]. Example services include handling consensus, maintaining the decentralized ledger, broadcasting transactions, checking double spends [6], etc. This could be modularized in a more distributed way to improve efficiency. In a monolithic system approach, everything is built in the same programming language to ensure easy integration. Since only one service is available, it is difficult to scale [24]. To improve scalability, we built Tikiri using a Microservice architecture, providing a better way to distribute the processing. All functionalities are implemented as Microservices, allowing architects to appropriately partition different services to the appropriate hardware. Fig. 1 shows the architecture of Tikiri where each node in the blockchain contains the following services/components:

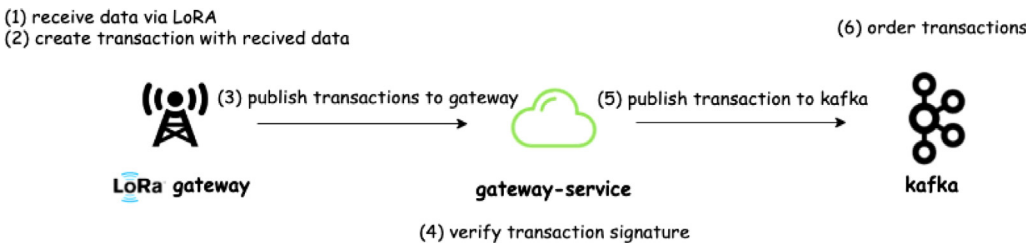

Fig. 2. Tikiri blockchain gateway service. Blockchain client (e.g LoRa gateway) connected to gateway service and publishes transactions. Then gateway service publishes them to kafka.

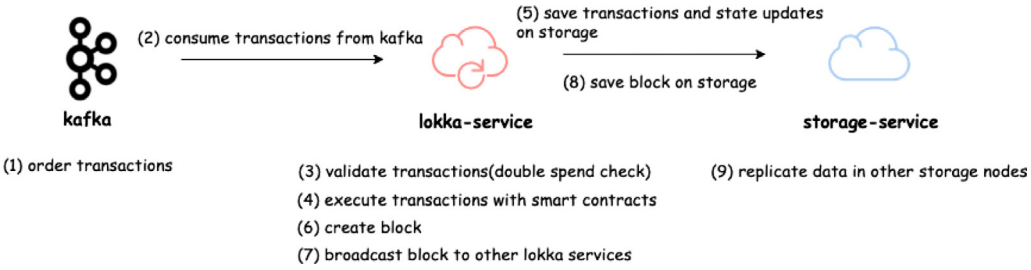

Fig. 3. Tikiri blockchain Lokka service architecture. Lokka services consume transactions via kafka, execute them and create blocks. Multiple Lokka services can work in parallel.

1. Gateway service (API service where clients interact with blockchain)
2. Lokka service (Smart contract service implemented with actor-based concurrency controlling)
3. Storage service (Distributed database storage service used to store transactions, blocks and assets)
4. Kafka Message broker (Distributed publisher–subscriber message broker used to order the messages and transmit the messages).

2.2. Gateway service

Gateway is the client API of the Tikiri blockchain. Blockchain clients (e.g. LoRa [25] gateway) connect to this service and publish their transactions. Once the transactions are received, it publishes the transactions into the Kafka message broker. Then the Kafka message broker orders the published transactions. Gateway service exposes HTTP based synchronous endpoint and Kafka message broker based asynchronous endpoint to the client, Fig. 2.

Each node in the cluster has a gateway service. The clients on each node publish transactions to their gateway service as JSON encoded objects (JSON message contains smart contract name and arguments) via the gateway API.

2.3. Lokka service

The Lokka service is the functional programming and actor-based smart contract service [15] in Tikiri blockchain. All blockchain-based software programs and the messages that pass between them are written as Aplos smart contracts [15] in Lokka service. Lokka consumes transactions from Kafka message broker as a batch (Kafka message broker is the consensus platform in Tikiri). The batch size can be configured based on the number of transactions or time according to the performance requirements of the IoT network. After that, it executes the batch transactions via smart contracts. The structure of a transaction is described in Fig. 4. Unlike other blockchain smart contracts, the Aplos smart contracts can execute transactions concurrently. Once transactions are executed, the state updates will be stored and distributed to other blockchain nodes via storage service, (Fig. 3).

```
{
    "execer": "<transaction executing user>",
    "uid": "<transaction id>",
    "actor": "<smart actor name>",
    "message": "<actor message>",
    "timestamp": "<transaction timestamp>",
    "digsig": "<digital signature>"
}
```

Fig. 4. Tikiri blockchain transactions structure.

```
{
    "lokka": "<block creator id>",
    "id": "<block id>",
    "header": {
        "merkle_root": "<merkel root of transactions>",
        "trans_count": "<transaction count in block>",
        "pre_hash": "<previous block hash>",
        "hash": "<block hash>",
        "lokko": "<digital signatures of lokka nodes>"
    },
    "trans": "<transaction set>",
    "timestamp": "<block create timestamp>"
}
```

Fig. 5. Tikiri blockchain block structure.

Tikiri blockchain is built with a Validate-Execute-Group blockchain architecture [11,15]. Unlike Order-Execute [9] architecture, Validate-Execute-Group architecture can validate and execute transactions concurrently when peers submit transactions to the network. Transactions will be executed only in one peer. The client does not need to wait until a block is created to validate and execute the transactions. After transactions are executed, Lokka service will generate a block with the transactions within the batch, Fig. 5. There will be multiple Lokka services in the blockchain network (each blockchain node can have their own Lokka service). All Lokka services are connected via Kafka message broker as a consumer group. There is a partitioned Kafka topic called "Lokka", all the Lokka services connected to this topic as Kafka consumer group. Then Kafka handles the message partitioning and message broadcasting between Lokka services, guaranteeing total order (provide total order by sending a message only to one consumer by topic partitioning [16], Fig. 6).

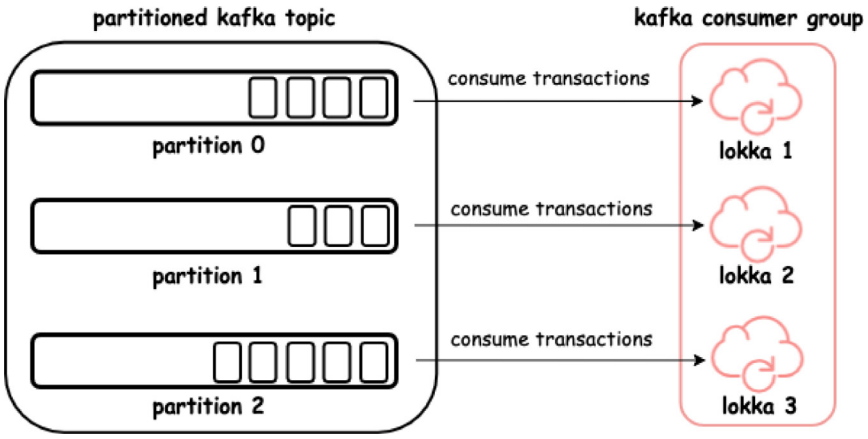

Fig. 6. Lokka services connected as Kafka consumer group. Kafka will handle message broadcasting between multiple Lokka services with guaranteeing total order.

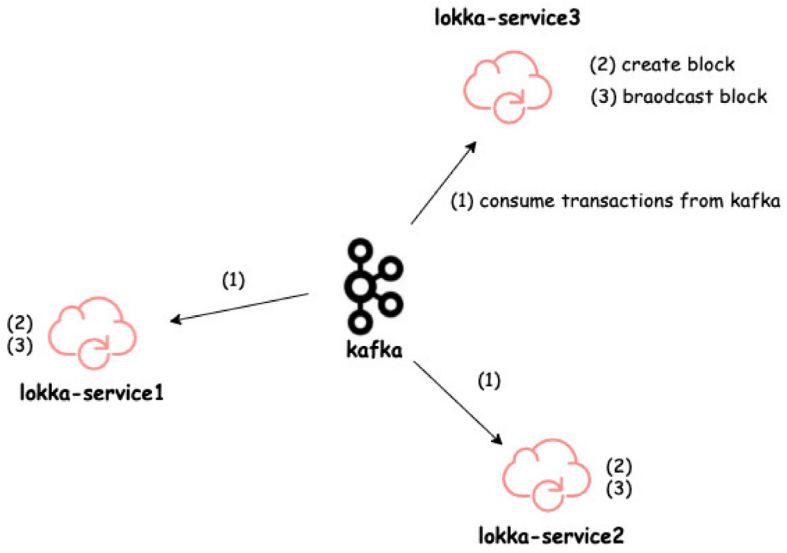

Fig. 7. Execute transactions and create blocks parallelly by Multiple Lokka services in Tikiri blockchain.

Each Lokka service consumes transaction only once and executes them. As shown in Fig. 7, they can work independently and create blocks concurrently. In this way when adding new Lokka services to the network it will linearly increase the transaction throughput and scalability of the blockchain.

When creating a block, Lokka service generates the block hash and adds the transaction list to the block. The structure of the block is described in Fig. 5. Similar to transactions, this block is also stored in storage service and distributed to other nodes in the blockchain. Once the block is created it needs to be approved by other Lokka services in the network. The block approval process done via federated consensus [26] is implemented between Lokka services. When block approving, the newly created block information (e.g block ID) is broadcast to other Lokka services in the blockchain network via Kafka Message broker. Other Lokka services take the block from their respective storage service and validate the transactions in the block. If all transactions in the block are valid, it gives a vote for the block (mark block as valid or invalid). To handle voting, Lokka service digitally signs the block hash and add the signature into the block header. When the majority of Lokka services vote for a block, that block is considered as a valid block. The block ordering will happen with the majority vote from the federated consensus which is considered an endorsement policy defined among the Lokka services. This policy defines how many Lokka services need to vote in order to approve a block.

2.4. Storage

Storage is the place where the blocks, transactions and assets are stored in Tikiri. We have used a lightweight distributed database as the storage service. Each node in the blockchain runs its own distributed database service/node. As shown in Fig. 8, these services are connected in a ring cluster. Once Lokka service executes transactions or creates a block, it will write the state updates, transactions, and blocks into its storage service instance (distributed database node). Then the saved data will be distributed to other nodes via the underlying distributed database.

Tikiri blockchain does not use full node data replication like Bitcoin or other blockchain platforms. Instead, Tikiri adopts a sharding-based data replication mechanism. In Tikiri blockchain, data replications are handled by the underlying distributed database. When one 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. Although sharding-based data replication handles the storage limitations in IoT applications to some extent, some IoT applications generate a large stream of data continuously (e.g. healthcare, sensor), which

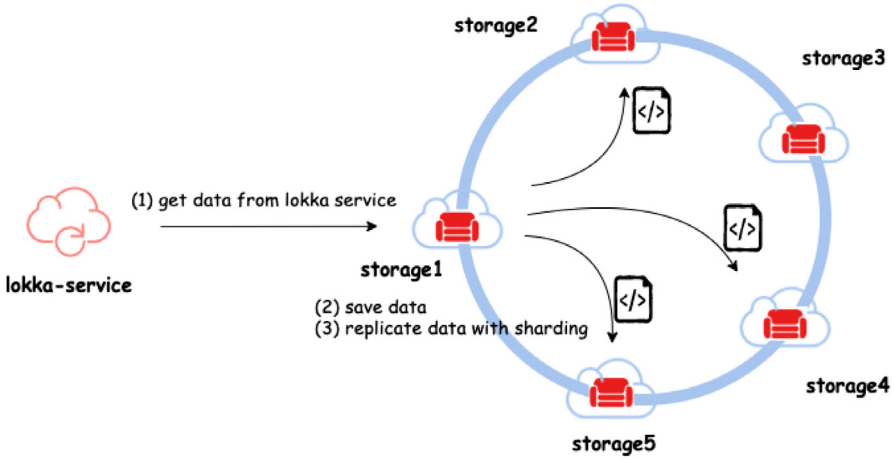

Fig. 8. Tikiri blockchain storage service. Storage service in each blockchain node connected as a ring cluster and use sharding based data replication.

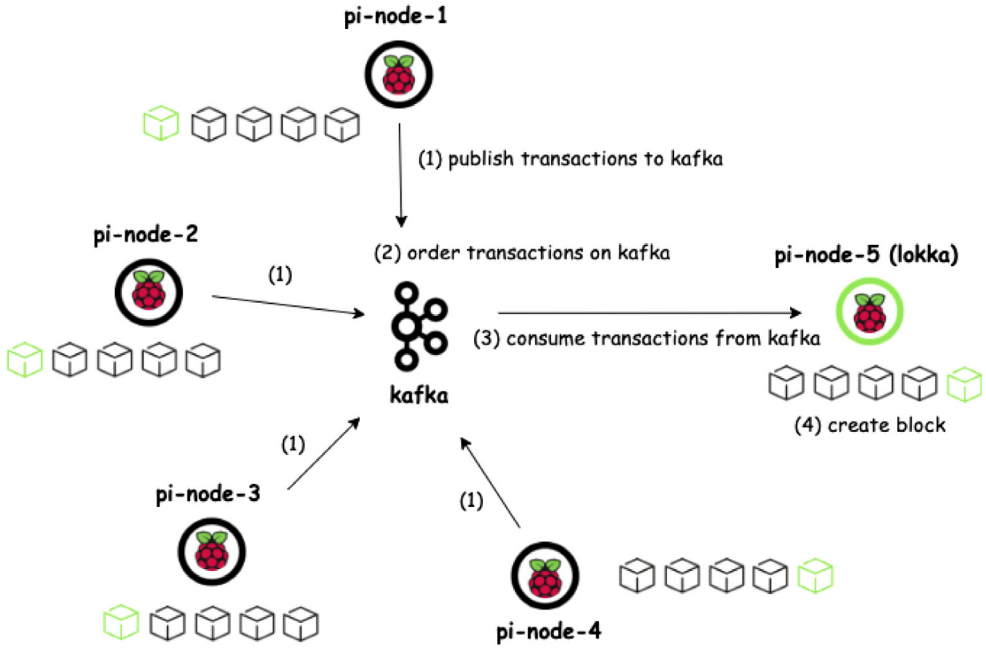

Fig. 9. Tikiri blockchain transaction flow. Different blockchain nodes publish transactions to Kafka. Lokka service consumes ordered transactions from Kafka, executes them and creates blocks.

requires special attention. To cope with this kind of situation, Tikiri blockchain has the ability to integrate off-chain data storage that will efficiently handle the large-scale streaming data generated in those IoT applications. In this case, the sensitive data, such as health records and sensor device identities etc., will be stored in off-chain storage and proof of the data will be kept in the blockchain.

By using a distributed database to handle the data replication among the nodes, additional gossip communication and broadcasts at the application level to distribute the data is not necessary. All the data distribution and gossip communication are handled via the underlying distributed database. Using this approach reduces the application level functions of Tikiri and builds a lightweight blockchain for resource-constrained IoT devices.

2.5. Kafka message broker

Kafka message broker (distributed publisher–subscriber service) is used as the consensus platform of Tikiri blockchain. All transactions published by the clients will be stored and ordered in a Kafka message broker. Lokka services take the ordered transaction batch from a Kafka message broker, execute them, and creates blocks. There are two main communication use cases of a Kafka message broker in Tikiri. In the first scenario, the client publishes transaction messages to the gateway service via Kafka message broker. There is a separate Kafka message broker channel (e.g. Kafka topic) for each Gateway service. Clients publish transaction messages to these Kafka message broker channels, (Fig. 9). By using the Kafka message broker for client-to-Gateway service communication, we are able to handle the client requests asynchronously by using a reactive streaming-based approach. It helps to handle back pressure [27] operations on IoT applications and high transaction load.

The second scenario involves communication between Lokka services. When a Lokka service generates a block and saves it in the storage, the block ID is broadcast to other Lokka services via Kafka message broker. As shown in Fig. 3 all communication between Lokka happens through the message broker.

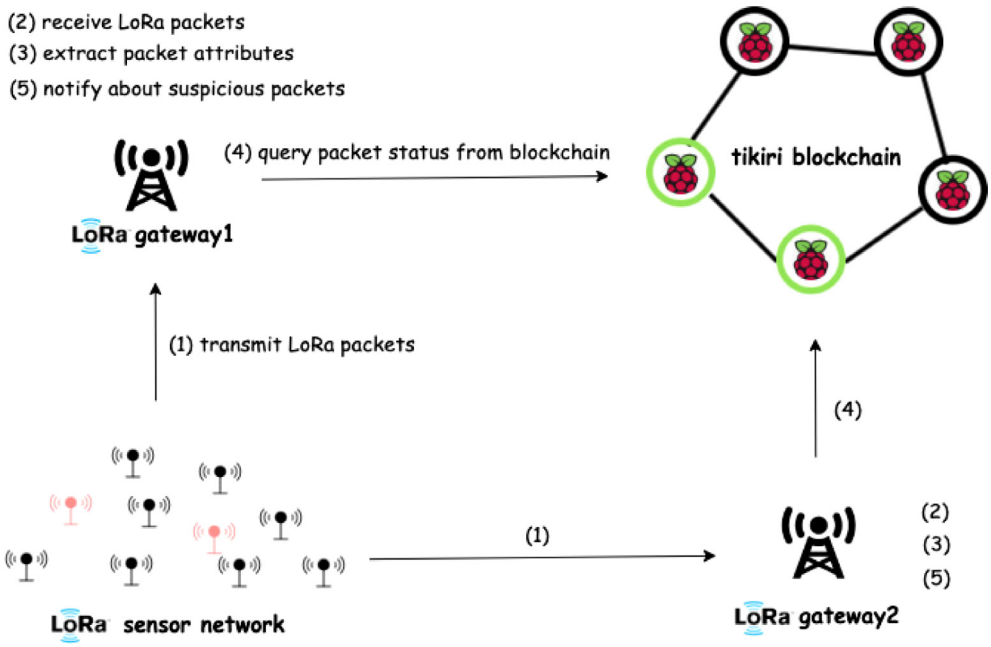

Fig. 10. Tikiri blockchain architecture of sensor device identity monitoring platform in power plant. Different LoRa gateway's can be connected to different blockchain nodes.

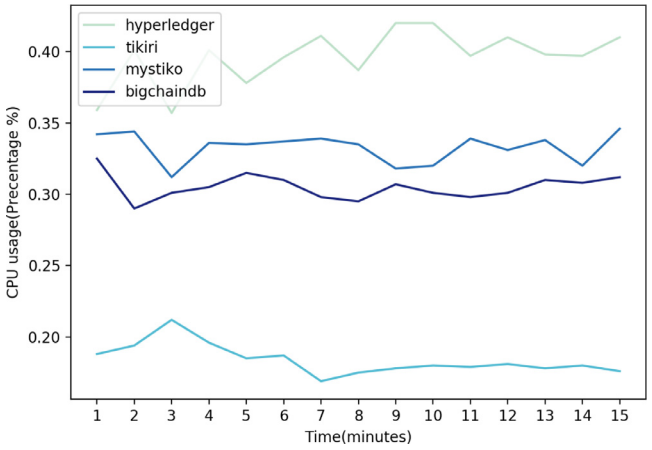

Fig. 11. CPU usage comparison of Tikiri, Hyperledger Fabric, Mystiko and BigchainDB. The statistics recorded with docker stat command.

3. Tikiri implementation

3.1. Overview

Tikiri blockchain is implemented on a Raspberry-Pi based test-bed. The architecture of the Tikiri blockchain implementation is described in Fig. 9. In this implementation, we used CouchDB [28] as the underlying storage service. Each node in the blockchain has a CouchDB instance. The Gateway service is implemented with Golang to run with less overhead on resource-constrained Raspberry-Pi nodes. Lokka service is implemented with the Scala functional programming language. Transaction streaming, block creation, and broadcasting implementation are achieved with Akka-streams [29] by using a reactive programming approach [30]. The reactive streaming model enables high transaction throughput on the Tikiri blockchain. In Tikiris production implementation, we have used Apache Kafka as the message broker. We run 4 Kafka broker nodes with 3 Zookeeper nodes with all services on Tikiri blockchain containerized [31] and deployed using Lightweight Kubernetes (K3S) container orchestration systems [32,33]. K3S manages the termination and recovery after the sudden failure of the microservices in each of the blockchain nodes.

3.2. Use case

The first prototype of Tikiri blockchain is built as an IoT sensor device identity monitoring platform in electric power plants. Tikiri blockchain is deployed in a Raspberry-Pi based test-bed. The power plant contains various IoT sensor devices that communicate over the LoRa protocol [25]. The sensor device identity information is stored in Tikiri blockchain. The identity includes `Device Name`, `Device Address`, `Communication Port`, `Communication Protocol`. There is a `Sensor` smart contract which corresponds to the create and search functions for the sensor device identities from Tikiri blockchain. Initially, sensor device identities (which are deployed in the power plant) need to be added (registered) to the blockchain. This is done via the web-based admin interface. When registering devices, the admin interface invokes `createSensor` function on sensor smart contract. The registered devices can be searched via executing `searchSensor` functions on Sensor smart contacts.

This blockchain infrastructure is used to detect unauthorized sensor devices in the power plant. All sensor device identities are registered on the blockchain when deployed into the power plant. Sensor devices transmit data via LoRa protocol. When sensor devices transmit the data, the transmitting LoRa packets are sent to the LoRa gateway/Network server running on a Raspberry-Pi node. LoRa packets include `Device Address`, `Frame Port`, `Frame Counter`, `Message Type` etc. Based on the packet information, LoRa gateway/Network server searches for the sensor device corresponding with this packet from the blockchain. `Frame Counter` fields are used to check for deduplication of transactions. The LoRa gateway/Network server acts as a blockchain client 10 with multiple Lora gateways/Network servers connected to different nodes in the Tikiri blockchain. When searching, it executes `searchSensor` smart

contract function with LoRa packet attributes. If the sensor device which corresponds to the LoRa packet exists on the blockchain, it is recognized as a valid packet (generated from the verified sensor). If not, the device does not exist and it is recognized as an invalid packet generated from an unauthorized sensor.

4. Tikiri performance evaluation

A performance evaluation of Tikiri is completed and discussed comparing Hyperledger Fabric, Mystiko and BigchainDB blockchains. To obtain the results, Tikiri blockchain is deployed on a Raspberry-Pi 4 cluster with 4 GB RAM. Hyperledger Fabric, Mystiko and BigchainDB blockchains deployed on AWS cluster (AWS 2xlarge instance with 16GB RAM and 8 CPUs). Tikiri blockchain is run with Kafka cluster (4 Kafka broker nodes and 3 Zookeeper nodes) and CouchDB state database. The smart contracts on the Tikiri blockchain are implemented with Scala functional programming and Akka actors. Hyperledger Fabric is set up to run with a Kafka-based consensus and LevelDB [9] state database. HLF Kafka cluster utilized 3 Orderer nodes, 4 Kafka brokers with 3 Zookeeper nodes. The smart contracts on HLF implemented with golang [20]. Mystiko blockchain is set up to run with Kafka-based consensus (Utilized 4 Kafka broker nodes and 3 Zookeeper nodes) and Apache Cassandra [34] state database. The Smart contracts of Mytiko blockchain implemented with Scala functional programming and Akka actors. BigchainDB blockchain is set up to run with Tendermint consensus [26] and MongoDB [35] as the state database. The evaluation tests performance for a varying number of blockchain peers (1 to 7 peers) and records the following results:

1. CPU usage
2. Memory usage
3. Performance of Invoke transactions
4. Performance of Query transactions
5. Performance of transaction scalability
6. Transaction execution time.

4.1. CPU usage

To evaluate CPU usage, the CPU usage (as a percentage of total CPU) is recorded for Tikiri, Hyperledger Fabric, BigchainDB, and Mystiko, blockchain systems to test the lightweight implementation of Tikiri blockchain. All blockchains support running services on top of docker. Docker stats and cadvisor [36] are used to obtain the CPU usage statistics. Fig. 11, shows the results of the CPU usage for Tikiri blockchain as lower than Hyperledger Fabric, Mystiko and BigchainDB. The lightweight blockchain architecture, isolated operation of Lokka service with Kafka consumer group, functional programming and actor based smart contracts are the main reasons there is less CPU usage on Tikiri blockchain. Hyperledger Fabric runs more docker contains in the blockchain network than the other blockchain platforms (e.g peer services, orderer services, multiple chaincode smart contract services, state database containers etc.). Because of its large infrastructure, it requires a higher CPU usage than other blockchain platforms. Mystiko and BigchainDB blockchains are mainly targeted for cloud-based scalable applications (e.g big data applications). Mystiko uses Apache Cassandra as the state database and BigchanDB uses MongoDB as the state database. Because of the resource requirements from these services, they consume considerably higher CPU usage than Tikiri.

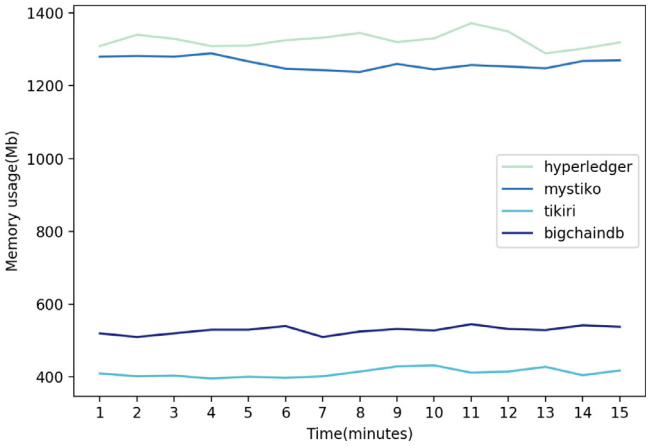

Fig. 12. Memory usage comparison of Tikiri, Hyperledger Fabric, Mystiko and BigchainDB. The statistics recorded with docker stat command.

4.2. Memory usage

Next, the memory usage is evaluated for Tikiri, Hyperledger Fabric, BigchainDB, and Mystiko blockchains. Similar to the CPU usage tests, memory usage statistics are observed by using docker stats and cadvisor [36]. As shown in Fig. 12, the memory usage of Tikiri is compared with Hyperledger Fabric, Mystiko and BigchainDB. BigchainDB and Tikiri both performed well using similar amounts of memory, but memory usage on Tikiri blockchain was less than the other blockchains tested. The lightweight blockchain architecture and Microservice based blockchain implementation are the main reasons for less memory usage on Tikiri blockchain. Mystiko blockchain uses Apache Cassandra distributed database as the state database which requires a large memory footprint with the in-memory Memtables that facilitate fast writes. Due to this reason, Mystiko blockchain consumes higher memory throughput. Similarly to CPU usage, Hyperledger Fabric consumes more memory usage from the many docker container services in the blockchain network (e.g peer services, orderer services, multiple chaincode smart contract services, state database containers etc.).

4.3. Performance of invoke transactions

The Invoke transactions update the asset status and create transaction records in the blockchain ledger. Here, we record the total number of “Invoke transactions” executed by a single peer. We allowed concurrent “invoke transactions” by each peer to record the total number of committed transactions. As shown in Fig. 13, we compare the “invoke transaction” performance of Tikiri with other blockchain platforms like Hyperledger Fabric, Mystiko, and BigchainDB. It is observed that the transaction throughput of Tikiri is higher than Hyperledger Fabric, but less than Mystiko and BigchainDB. Mystiko and BigchainDB are big data-friendly scalable blockchain storages. They are targeted for cloud-based applications and are built to gain higher invoke transaction throughput than Tikiri. Hyperledger Fabric comes with Multi-Version Concurrency Control (MVCC [37]) based on Execute-Order-Validate blockchain architecture [9]. Tikiri provides novel blockchain architecture (Validate-Execute-Group) to support real-time transactions. Instead of imperative style smart contracts, Tikiri provides functional programming and actor based concurrent transaction enabled smart contract platform. Due to these reasons, Tikiri provides a higher invoke transaction throughput than Hyperledger Fabric.

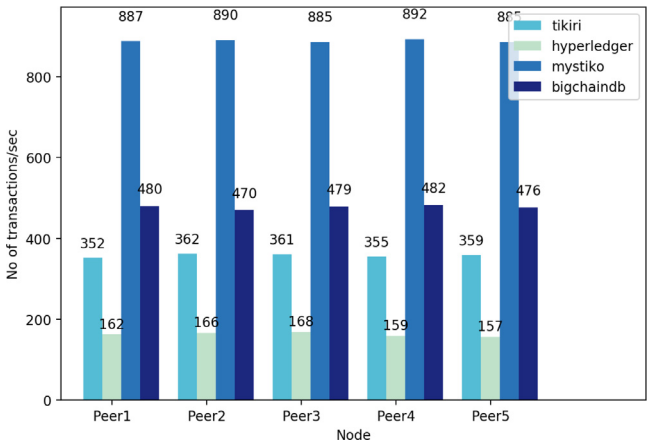

Fig. 13. Invoke transaction throughput comparison of Tikiri, Hyperledger Fabric, Mystiko and BigchainDB.

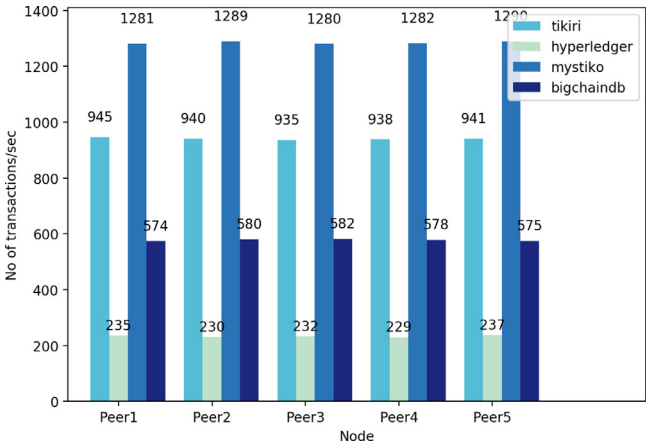

Fig. 14. Comparison of "Query Transaction" Throughput in Tikiri, Hyperledger Fabric, Mystiko, and BigchainDB.

4.4. Performance of query transactions

The Query transactions only read the status of the assets from the ledger. They do not necessarily create new transaction records and not update any asset status in the ledger. In this experiment, we record the total number of "Query transactions" executed by a sole blockchain peer. Concurrent Query transactions are executed in all the blockchain peers and then we record the total number of completed transactions. As shown in Fig. 14, the Query transaction performance of Tikiri is compared with Hyperledger Fabric, Mystiko and BigchainDB. Mystiko serves its query API via Apache Lucene index-based search API. BigchainDB facilitates the query API with MongoDB. Tikiri facilitates its search API with CouchDB. Tikiri's query transaction throughput is less than Mystiko but higher than HLF and BigchainDB. Hyperledger Fabric and BigchainDB blockchains do not support concurrent transaction handling and backpressure operation handling on blockchain and this results in less query transaction throughput. Concurrent transaction execution of smart contracts, Akka streams based backpressure handling, and CouchDB based data storage are shown to improve query transaction throughput results in Tikiri. Unlike invoke transactions, query transactions do not update the ledger status and do not create transaction entries in the ledger, therefore query transactions result in higher throughput compared to invoke transactions as shown in Fig. 14 and Fig. 13.

4.5. Performance of transaction scalability

Transaction scalability result is measured against the number of Invoke transactions that can be executed per second and the number of blockchain peers in the network. Transactions executed concurrently in each blockchain peer, and the recorded the number of executed transactions to show how well each system can scale up. Fig. 15 shows the transaction scalability comparison of Tikiri, Hyperledger Fabric, Mystiko and BigchainDB. Tikiri has higher transaction scalability than Hyperledger Fabric but less scalability than Mystiko and BigchianDB. Mystiko and BigchainDB blockchains come with scalable, real-time transaction enabled blockchain architectures. They produce higher transaction scalability than other blockchain-based systems. Hyperledger does not support real-time transactions processing blockchain architecture, it uses Multi-Version Concurrency Control (MVCC [37]) based Execute-Order-Validate architecture for transaction processing. Due to this, Hyperledger Fabric is seen to perform less

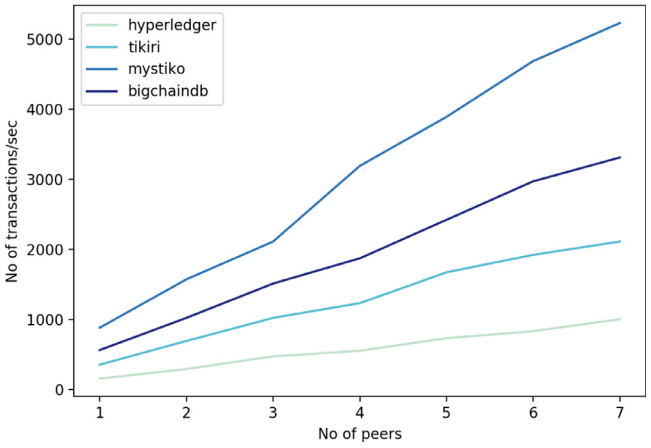

Fig. 15. Transaction scalability comparison of Tikiri, Hyperledger Fabric, Mystiko and BigchainDB.

transactions per second than other blockchain platforms. When adding a node to the cluster, Tikiri nearly linearly increases the transaction throughput. Fig. 16 shows how transaction execution rate varies when having a different number of blockchain peers in the Tikiri. When the number of peers increases, the rate of executed transactions increases relatively. In Tikiri, Lokka services work concurrently with a Kafka consumer group. Block creation and voting happen independently. These points contribute to the linear increase in the transaction when adding Lokka nodes to the cluster.

4.6. Transaction execution time

In this evaluation the time to execute different sets of transactions (100, 500, 1000, 2000, 3000, 5000, 7000, 8000, 10 000 transactions) in Tikiri blockchain is compared. Transaction execution time includes the double-spend checking time, ledger update time, and data replication time. Fig. 18 shows how the transaction execution time varies in different transaction sets. Fig. 17 shows the number of executed transactions and submitted transactions in a single blockchain peer. There is a back pressure operation [27] between the rates of submitted transactions and executed transactions. A reactive streaming-based approach with Apache Kafka is used to handle these backpressure operations in the Tikiri blockchain.

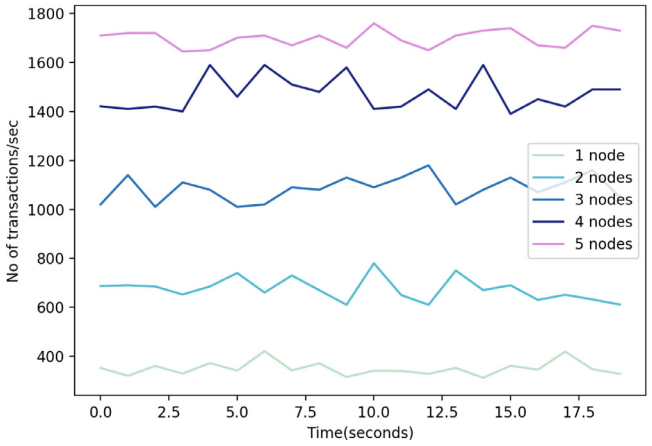

Fig. 16. Transaction execution rate comparison with number of nodes in Tikiri blockchain.

Fig. 17. Transaction execution rate and transaction submission rate in a Tikiri single blockchain node.

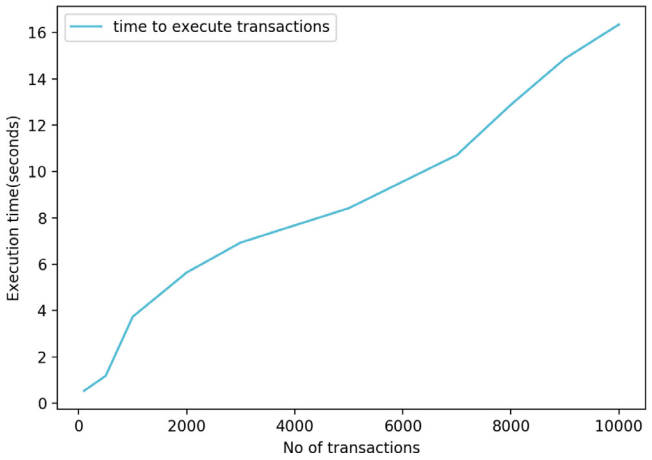

Fig. 18. Transaction execution time with different transactions set in Tikiri blockchain.

5. Related work

Recent research has been conducted integrating blockchain for resource-constrained IoT environments and scalable big data environments. In this section, we outline the main features and architecture of these research projects.

Lightchain [38] is a lightweight blockchain built for power-constrained Industrial IoT (IIoT) use cases providing a resource-efficient solution. It focuses on improving by reducing computational power, storage, and network resources usage. The consensus is handled by the protocol Synergistic Multiple Proof 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. A lightweight data structure, LightBlock, is leveraged to streamline broadcast content.

LSB — Lightweight Scalable Blockchain [39] is a lightweight blockchain storage platform that is optimized based on IoT needs. 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 IoT needs in the blockchain, LSB incorporates a Lightweight Consensus (LC) algorithm. The consensus mechanism aims to achieve distributed trust through optimized throughput management and traffic separation between data and regular transactions.

ELIB [40] proposed an efficient lightweight integrated blockchain model to meet IoT application requirements. It uses a smart-home environment to illustrate the purpose of ELIB for IoT applications. ELIB proposed three optimizations to support blockchain with resource-constrained IoT application requirements, (1) Lightweight Consensus algorithm, (2) Certificate-less Cryptography (CC), and (3) Distributed Throughput Management (DTM) scheme. The cluster heads have limited access to create new blocks within the consensus period. The CC method helps reduction of computational overheads. Finally, the DTM scheme allows modification of specific system variables so that varying network load can be accommodated in the blockchain system without drastic reduction of throughput.

FogBus [41] provides a lightweight blockchain framework for IoT systems when integrated with edge and fog computing infrastructures. The proposed framework aims to achieve secure and authenticated data transfer among fog nodes, IoT devices, and cloud data centers. The FogBus framework includes IoT devices (sensors and actuators), Fog Gateway Nodes (FGN — the entry points of distributed computing infrastructure), Fog Computational Nodes (FCN — handle the back-end processing of IoT applications, data sharing, secure storage), and Cloud data centers (extend resources to execute backend IoT applications). FogBus blockchain nodes follow Master–Worker mode. The master nodes in the network create blocks from received data and broadcast to worker nodes to verify. To prevent the brute force attacks on private key generation, the master node in the FogBus generates a dynamic public/private key-pair for each block. The FogBus blockchain is implemented with JVM based language in order to be easily installed across various platforms (cross-platform).

Data Allocation for IoT [42] work proposed a context-aware mechanism for on-chain data allocation in IoT blockchain. Such data allocation is decided by calculating Rating of Allocation (RoA) with fuzzy logic. Context parameters like (a) Network, (b) Data and (c) Quality have been used to calculate the RoA score. The proposed system contains a four-tier abstraction, (1) "IoT tier" (2) "data controller tier", (3) "fog tier", (4) "cloud tier". When saving the data, the "data control tier" decides weather the data needs to stored in on-chain (blockchain in fog/cloud) or off-chain (cloud database) by calculating the Rating of Allocation (RoA). They have

Table 1
Blockchain platform comparison.

Blockchain	Type (Public/Private)	Weight (Heavy/Mid/Light)	Consensus	Scalability	Smart contract	Full text search	Concurrent transactions	Sharding
Tikiri	Private	Light	Kafka/ZAB	High	Yes	Yes	Yes	Yes
Lightchain [38]	Public	Light	SMP	Mid	No	No	No	Yes
LSB [39]	Private	Light	LC	High	No	No	No	No
ELIB [40]	Private	Light	LC with Cluster Heads	Mid	No	No	No	Yes
FogBus [41]	Private	Mid	PoW	Mid	No	No	No	No
DAMIoT [42]	Private	Mid	PoW	Mid	N/A	N/A	N/A	N/A
BigchainDB [8]	Both	Heavy	Tendermint	High	Yes	Yes	No	Yes
HbasechainDB [43]	Private	Heavy	ZAB	High	No	Yes	No	Yes
Mystiko [11]	Private	Mid	Paxos	High	Yes	Yes	Yes	Yes
Hyperledger [9]	Private	Mid	Kafka/ZAB	Mid	Yes	No	No	Yes
Chain [44]	Public	Heavy	Federated	Mid	No	No	Yes	Yes
RapidChain [10]	Public	Heavy	Federated	Mid	No	No	No	Yes
RSCoin [45]	Private	Heavy	2PC variant	Mid	Yes	No	No	Yes

evaluated the data allocation mechanism of blockchain-based cloud and fog architecture by running healthcare application on top of FogBus [41].

BigchainDB [8] is built as an enterprise blockchain database using MongoDB [35]. MongoDB provides a library set that handles the consensus protocol. The later version has included features such as immutability, decentralized control, digital asset movement making it a scalable blockchain database. The term blockchain pipelining is used to describe the scalability features. Pipelining is successful because it does not validate blocks when they are added to the network, instead, blocks are queued and eventually validated via voting by participating nodes. Majority of nodes vote decides the next valid block. It has been shown that this strategy results in high transaction throughput.

HBasechainDB [43] is a system much like BigchainDB, but instead of MongoDB for the back end database, the Apache HBase uses Hadoop. HbasechainDB also uses a pipelining principal along with a federated consensus protocol. HBase along with Hadoop has shown that it is linearly scalable, but it offers many functions that analyze data present on the blockchain.

Mystiko [11] is a enterprise blockchain system targeted for highly scalable and concurrent applications (e.g Big data, IoT). It designed with utilizing Paxos-based federated consensus [26,34,46]. Mystiko uses distributed publisher/subscriber message broker system [16] and reactive streams based approach [29] to handle the back-pressure [27] operations in high transaction throughput enabled applications. Mystiko uses Apache Lucene index-based API [47,48] to enable the full-text search capability on blockchain data. The Mystiko-ML service capables of performing the data analytic and machine learning operations from the on-chain and off-chain blockchain data. Mytiko enables following performance improvements on existing Blockchains: (1) Order-Execute architecture; (2) full node data replication; and (3) imperative-style smart contracts.

Hyperledger Fabric [9] is an open standard, general, permissioned blockchain system. It uses a modular design giving great flexibility to developers allowing for the consensus method to be configured based on the need of the use-case with examples using `Redundant Byzantine Fault Tolerance`, `Apache Kafka`, `Proof of Elapsed Time` and `Sumeragi`. Fabric utilizes `Chaincode` smart contract [7] platform. Chaincode as well as most of the supporting components are primarily written in Golang [20]. The private communication channels in Fabric facilitates with using the Apache Kafka-based communication between the blockchain nodes.

Chain [44] is private blockchain system using a federated consensus. The majority of the node votes will write a block to the ledger. This system is capable of concurrently processing blocks making it a unique blockchain implementation. Nodes only keep partial states of the ledger referred to as a sharding-based approach to ledger storage. Shards of the ledger allow the nodes to concurrently validate their portions of the network, giving a more parallel result in scalability.

RapidChain [10] claims to be the first sharding-based public blockchain system with a protocol that is Byzantine fault tolerant [49,50]. Using partitioned ledger data, it distributes new blocks that need to be validated to multiple `committer` nodes (sharding). A special technique referred to as cross-shard transaction verification is utilized that creates a more efficient system. RapidChain also reduces noise produced by gossip networks by partitioning the network. Early results show that RapidChain can confirm and process more than 7300 transactions per second.

RSCoin [45] is another example of a sharding-based private blockchain system. It was developed to accommodate the high throughput demands of existing centralized monetary systems. It uses both a centralized monetary supply and distributed ledger. Trusted authorities called `mintettes` validate transaction. The centralized aspects in return result in a highly scalable system compared to the more decentralized solutions. A very simple/fast protocol is used to solve the double-spending problem and 2-phase commit to ensure the integrity of the system. With this approach, RSCoin can handle up to 2,000 transactions per second.

The comparison summary of these blockchain platforms and Tikiri platform presented on Table 1. It compares various Blockchain type (public and private), Weight level, Consensus, Scalability level, Smart contract support, Full-text search support, Concurrent transaction support, and Sharding details.

6. Conclusions and future work

With Tikiri, a lightweight blockchain that exhibits high transaction throughput and high scalability is developed. Novel blockchain architecture and Kafka based consensus to achieve real-time transactions on the blockchain is presented. The Microservice-based architecture makes the Tikiri blockchain a lightweight implementation. The functional programming and actor based smart contract platform in Tikiri supports concurrent execution of transactions. The full-text search capability is added to Tikiri by indexing transactions and blocks on distributed database storage. Container orchestration system enables easy deployment and easy scalability of Tikiri. With these features, Tikiri is a lightweight and scalable blockchain storage system for resource-constrained IoT devices. This paper has shown and described the scalability, transaction throughput, CPU/Memory usage features of Tikiri with empirical evaluations. Tikiri is integrated into a real-world electric power grid sensor device identity monitoring application. This deployment demonstrates the practical use of Tikiri as an ideal blockchain system for IoT environments.

For future developments, plans include the incorporation of the following features in Tikiri: (a) Support for MQTT-based communication between the sensor network and blockchain; (b) Integration with a distributed database (Apache Cassandra) to store transaction data; (c) Integration with off-chain storage to guarantee the privacy of the data and address the storage issues on the blockchain nodes; (d) Development of a sharding-based consensus that handles the network partitions and failures handling scenarios where the voting nodes that are not reachable.

CRediT authorship contribution statement

Eranga Bandara: Methodology, Software, Implementation, Writing - original draft. **Deepak Tosh:** Investigation, Writing - review & editing, Supervision. **Peter Foytik:** Writing, Reviewing, Editing. **Sachin Shetty:** Supervision, Investigation, Writing - review & 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) (Federal Grant #DE-FE0031744).

References

[1] N. Suri, M. Tortonesi, J. Michaelis, P. Budulas, G. Benincasa, S. Russell, C. Stefanelli, R. Winkler, Analyzing the applicability of Internet of Things to the battlefield environment, in: Intl. Conference on Military Communications and Information Systems (ICMCIS), 2016, pp. 1–8.
[2] A. Kott, A. Swami, B.J. West, The internet of battle things, 2017, CoRR abs/1712.08980, http://arxiv.org/abs/1712.08980.
[3] D. Miorandi, S. Sicari, F.D. Pellegrini, I. Chlamtac, Internet of things: Vision, applications and research challenges, Ad Hoc Netw. 10 (7) (2012) 1497–1516, http://dx.doi.org/10.1016/j.adhoc.2012.02.016.
[4] S. Huckle, R. Bhattacharya, M. White, N. Beloff, Internet of things, blockchain and shared economy applications, Procedia Comput. Sci. 98 (2016) 461–466.
[5] K. Christidis, M. Devetsikiotis, Blockchains and smart contracts for the internet of things, IEEE Access 4 (2016) 2292–2303.
[6] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2008.
[7] V. Buterin, et al., A next-generation smart contract and decentralized application platform, 2014, white paper.
[8] 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.
[9] 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.
[10] M. Zamani, M. Movahedi, M. Raykova, Rapidchain: A fast blockchain protocol via full sharding, IACR Cryptology ePrint Archive 2018 (2018) 460.
[11] 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.
[12] C. Dannen, Introducing Ethereum and Solidity, Vol. 1, Springer, 2017.
[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] E. Bandara, W.K. NG, K. De Zoysa, N. Ranasinghe, Aplos: Smart contracts made smart, 2019, BlockSys'2019.
[16] 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.
[17] J. Hughes, Why functional programming matters, Comput. J. 32 (2) (1989) 98–107.
[18] C. Hewitt, Actor model of computation: scalable robust information systems, 2010, arXiv preprint arXiv:1008.1459.
[19] M. Gupta, Akka Essentials, Packt Publishing Ltd, 2012.
[20] 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.
[21] F.S. Shoumik, M.I.M.M. Talukder, A.I. Jami, N.W. Protik, M.M. Hoque, Scalable micro-service based approach to FHIR server with golang and NoSQL, in: 2017 20th International Conference of Computer and Information Technology (ICCIT), IEEE, 2017, pp. 1–6.
[22] The Scala Programming Language, https://www.scala-lang.org/, (Accessed on 06/21/2020).
[23] 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.
[24] J. Thönes, Microservices, IEEE Softw. 32 (1) (2015) 116.
[25] A. Augustin, J. Yi, T. Clausen, W. Townsley, A study of LoRa: Long range & low power networks for the internet of things, Sensors 16 (9) (2016) 1466.
[26] J. Kwon, Tendermint: Consensus without mining, Draft v. 0.6, fall 1 (2014) 11.
[27] 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.
[28] J. Anderson, J. Lehnardt, N. Slater, CouchDB: The Definitive Guide: Time to Relax, O'Reilly Media, Inc., 2010.
[29] A.L. Davis, Akka streams, in: Reactive Streams in Java, Springer, 2019, pp. 57–70.
[30] Z. Wan, P. Hudak, Functional reactive programming from first principles, in: Acm Sigplan Notices, Vol. 35, (5) ACM, 2000, pp. 242–252.
[31] D. Merkel, Docker: lightweight linux containers for consistent development and deployment, Linux J. 2014 (239) (2014) 2.
[32] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Borg, omega, and kubernetes, Queue 14 (1) (2016) 70–93.
[33] H. Fathoni, C.-T. Yang, C.-H. Chang, C.-Y. Huang, Performance comparison of lightweight kubernetes in edge devices, in: International Symposium on Pervasive Systems, Algorithms and Networks, Springer, 2019, pp. 304–309.
[34] A. Lakshman, P. Malik, Cassandra: a decentralized structured storage system, Oper. Syst. Rev. 44 (2) (2010) 35–40.
[35] 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.
[36] E. Casalicchio, V. Perciballi, Measuring docker performance: What a mess!!!, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, 2017, pp. 11–16.
[37] 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.
[38] 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.
[39] 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.
[40] S.N. Mohanty, K. Ramya, S.S. Rani, D. Gupta, K. Shankar, S. Lakshmanaprabu, A. Khanna, An efficient Lightweight integrated Blockchain (ELIB) model for IoT security and privacy, Future Gener. Comput. Syst. 102 (2020) 1027–1037.
[41] S. Tuli, R. Mahmud, S. Tuli, R. Buyya, Fogbus: A blockchain-based lightweight framework for edge and fog computing, J. Syst. Softw. (2019).
[42] W. Yánez, R. Mahmud, R. Bahsoon, Y. Zhang, R. Buyya, Data allocation mechanism for internet-of-things systems with blockchain, IEEE Internet Things J. 7 (4) (2020) 3509–3522.
[43] M. Subhankar Sahoo, P.K. Baruah, Hbasechaindb – a scalable blockchain framework on hadoop ecosystem, 2018, pp. 18–29.
[44] C. Cachin, M. Vukolić, Blockchain consensus protocols in the wild, 2017, arXiv preprint arXiv:1707.01873.
[45] G. Danezis, S. Meiklejohn, Centrally banked cryptocurrencies, 2015, arXiv preprint arXiv:1505.06895.
[46] L. Lamport, The part-time parliament, ACM Trans. Comput. Syst. 16 (2) (1998) 133–169.
[47] C. Gormley, Z. Tong, Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine, O'Reilly Media, Inc., 2015.

[48] A. Białecki, R. Muir, G. Ingersoll, L. Imagination, Apache lucene 4, in: SIGIR 2012 Workshop on Open Source Information Retrieval, 2012, p. 17.
[49] L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401.
[50] M. Castro, B. Liskov, et al., Practical byzantine fault tolerance, in: OSDI, Vol. 99, 1999, pp. 173–186.

Eranga Bandra is 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 involved with research and developments in Distributed Systems, Functional Programming, Big Data, Actor based systems and DevOps.

Deepak Tosh is an assistant professor in Computer Science at the University of Texas at El Paso. Prior to that he was a Cybersecurity Researcher at Norfolk State University, Norfolk, Virginia. His research is centered around data provenance mechanisms, distributed system security, Blockchain, cyber-threat information sharing, cyber-insurance, and Internet of Battlefield Things (IoBT). Specifically, he is interested in distributed consensus models in Blockchain technology, cyber-resiliency in battlefield environment, and various practical issues in cloud computing security. He received his Ph.D. in Computer Science and Engineering (CSE) from University of Nevada, Reno. He has published more than 30 conference and journals papers, book chapters at reputed venues.

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 defense, and cyber security. 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 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.