Distributed and Parallel Databases (2024) 42:403–445
https://doi.org/10.1007/s10619-024-07441-9

A blockchain datastore for scalable IoT workloads using data decaying

Panagiotis Drakatos[1] · Constantinos Costa[1,3] · Andreas Konstantinidis[1,2] · Panos K. Chrysanthis[1,3] · Demetrios Zeinalipour-Yazti[1]

Accepted: 1 April 2024 / Published online: 10 May 2024

Abstract
The Internet of Things (IoT) revolution has introduced sensor-rich devices to an ever growing landscape of smart environments. A key component in the IoT scenarios of the future is the requirement to utilize a shared database that allows all participants to operate collaboratively, transparently, immutably, correctly and with performance guarantees. Blockchain databases have been proposed by the community to alleviate these challenges, however existing blockchain architectures suffer from performance issues. In this paper we introduce Triabase, a novel permissioned blockchain system architecture that applies data decaying concepts to cope with scalability issues in regards to blockchain consensus and storage efficiency. For blockchain consensus, we propose the Proof of Federated Learning (PoFL) algorithm which exploits data decaying models as Proof-of-Work. For storage efficiency, we exploit federated learning to construct data postdiction machine learning models to minimize the storage of bulky data on the blockchain. We present a detailed explanation of our system architecture as well as the implementation in the Hyperledger fabric framework. We use our implementation to carry out an experimental evaluation with telco big data at scale showing that our framework exposes desirable qualities, namely efficient consensus at the blockchain layer while optimizing storage efficiency.

Keywords Blockchain · Data decaying · Storage and retrieval

1 Introduction

Internet of Things (IoT) refers to a large number of physical devices being connected to the Internet that are able to see, hear, think, perform tasks as well as communicate with each other using open protocols [1–4]. IoT devices are connected to Cloud and Edge computing appliances through massively parallel I/O channels (e.g., 5G, Wi-Fi

Extended author information available on the last page of the article

6) with millisecond latency offering new opportunities in industrial optimization, human health, and well-being as well as safety. This will procreate tremendous opportunities for IoT applications between multiple parties, such as collaborative multitasking techniques [5], machine learning [6], cooperative benchmarking [7], and augmented reality technology [8].

A key component in the IoT scenarios of the future is the requirement to utilize a shared database that allows all participants to operate collaboratively with more functionality. The shared database can bridge the actual gap between the data generated from the IoT applications [9] and the rate that these are processed and analyzed in real-time. The objective is to enable users execute updates and queries on the collaborative database while preserving a consistent view among all users maintaining the system consistency and transparency. Moreover, it is essentially common to be compromised by malicious outsources. To mitigate the problem described, an innovative design of a shared database with high performance is required for all the participants, in order to collaborate among each other with trust. Blockchain databases have been proposed by the community to alleviate these challenges, however existing blockchain architectures suffer from performance issues measured in terms of throughput and latency. In this situation, the transactions are basically executed in a sequential manner and this, in conjunction with confidentiality issues, does not leave much space for scaling.

It is imperative to devise a database architecture that can withstand billions of transactions per second, as opposed to thousands transactions per second that is currently the case for typical blockchains due to the expensive verification cost. For example, the popular Bitcoin network typically supports transaction ingestion rates of 7 s/TX, Hyperledger Fabric: 3000 TX/s and Bitcoin Satoshi Vision (SV): 9000 TX/s. In an IoT environment however, the ingestion rate is usually much higher and even more distributed, which calls for new architecture types we discuss in this work.

In order to motivate our description, we now explain a Web 3.0 scenario in the scope of *Telco Big Data (TBD)* [10]. A telecommunication company (telco) is traditionally only perceived as the entity that provides telecommunication services, such as telephony and data communication access to users. However, the radio and backbone infrastructure of such entities spanning densely most urban spaces and widely most rural areas, provides nowadays a unique opportunity to collect immense amounts of data that capture a variety of natural phenomena on an ongoing basis, e.g., traffic, commerce, mobility patterns and user service experience [10–12]. The ability to perform analytics on the generated big data within tolerable elapsed time and share it with key TBD enablers (e.g., municipalities, public services, startups, authorities, and companies), elevates the role of telcos in the realm of future smart cities from pure network access providers to information providers. Consider a TBD scenario in which telcos aim to share network health data from cell towers (e.g., signal strength, call drops, bandwidth measurements) with public authorities for monitoring and compliance (e.g., EMF-compliance). Huawei alone reports 5 TBs/day for 10 M clients (i.e., 2 PB/year) for Shenzhen, China, for a respective telco big data scenario. From an architectural perspective the challenge is how to transparently and immutably store the collected massive velocity data at the edge

of each telecommunication network in order to facilitate efficient and scalable data sharing and access. Storing big data in a centralized way is not a preferable choice, because it doesn't fulfill any of these requirements.

In this paper we propose *Triabase*[1] (inspired from Greek "Tria", meaning "three"), being a database architecture designed for the Web 3.0 era. This new era envisions a more decentralized and open Web with greater utility for its users, beyond the original Semantic Web vision being trust-less and permission-less and entailing Machine Learning, IoT and Artificial Intelligence. Triabase is a blockchain datastore system that carries out machine learning on IoT feeds at the edge, abstracts machine learning in primitive blocks that are subsequently stored and retrieved from the blockchain. In a permissioned blockchain the distributed ledger is not publicly accessible and we use this formulation for to ease the uptake without hindering the uptake of permission-less counterpart trust solutions in the future. In Triabase, we have two types of nodes those that store the entire shared database, and the others that use the database for their own operations, such as sending query and update requests to the blockchain shared ledger. We expect the blockchain nodes to be synchronized under the decentralized blockchain network. The clients that use the blockchain only for database operations store only the appropriate block header in contrast with the full nodes that store the entire blockchain ledger. Triabase is organized in a tiered architecture (see Fig. 1) that comprises of: (i) a Storage layer, which includes a local document store and blockchain node used for distributed data retrieval; (ii) a Data Postdiction layer, which abstracts locally-ingested data into machine learning models using federated learning; and (iii) an Application layer that includes APIs and access methods to initiate the search and retrieval functions at the application layer.

Our proposed Triabase architecture, has a number of provisionings to cope with the network bottleneck for the bulk of conventional machine learning models available in different sectors. Typical models found on Vertex.ai might be up to several GB capturing a range of applications from Computer Vision, Generative AI, MLOps and general Data Science. Our proposed Triabase system, is agnostic of the model type and size, as it essentially abstracts raw data into data postdiction models. This provides generality as it is not bound to specific types of ML models. To cope with large scale models, we utilize a combination of techniques enumerated below:

(i) we provide the possibility to transcode ML models down to lighter versions using fp16 (floating point 16) editions, which are typically also deployed in scenarios of tinyML and tensorflow lite on mobile devices. In this case, an original model size of 20 GB can be reduced in half or more down to 4–5 GB;
(ii) we optimize SQL queries by implementing batching principles that allow to minimize the communication cost in the execution of continuous queries on velocity data emerging from IoT scenarios. By minimizing the number of federated learning parameters that are communicated during the execution of a query, we can dramatically reduce the network load;

[1] Triabase. https://triabase.cs.ucy.ac.cy/.

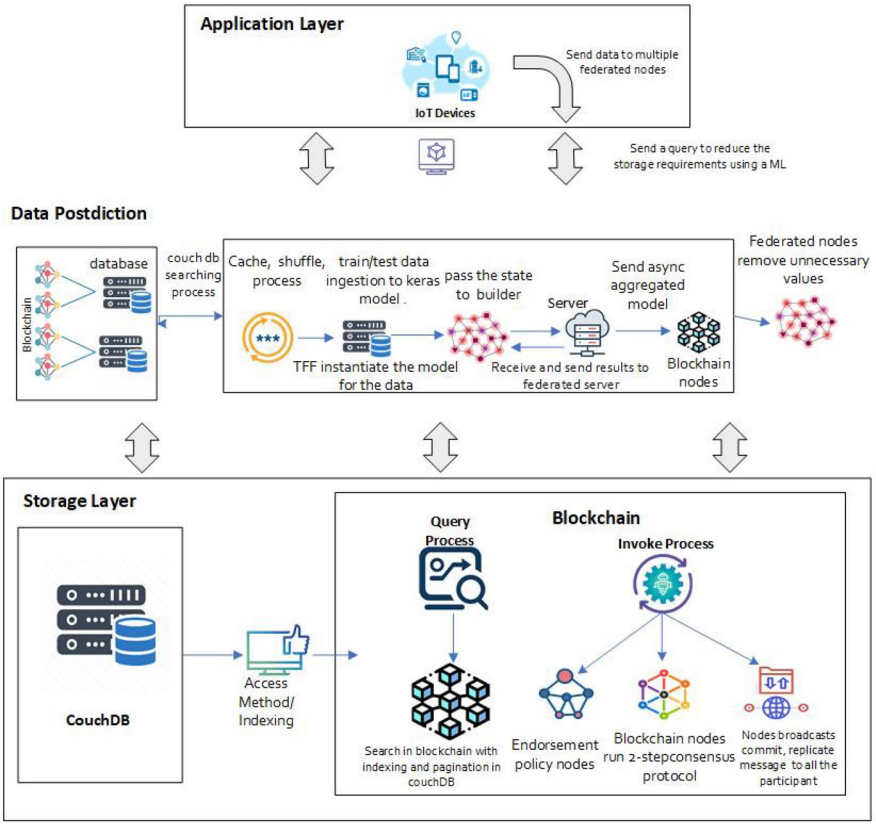

Fig. 1 The Triabase architecture layers

(iii) we implement sharding, which is a popular on-chain scalability method that focuses on scattering the blockchain network into more controllable, smaller units known as shards. The shards would then be executed concurrently by the network and process a fraction of the group's transaction load; and

(iv) We use a scalable consensus algorithm that provides greater scalability and transaction throughput. Byzantine Fault Tolerance (BFT) consensus techniques have been one of the most reliable tools for dealing with the Byzantine Generals Problem. BFT generally refers to a distributed system characteristic that suggests the necessity for continual consensus, despite multiple antagonistic participants in the network.

For extremely large models, like proprietary GPT-4 (OpenAI) or BART (Google), designated data centers with high performance HPC with NVLink, SmartNIC and RDMA might be necessary, but these remain outside the scope of this work.

This paper builds on our previous work in [13, 14], in which we presented the preliminary design and results of our Triabase architecture. In this paper

we propose several new improvements, particularly a structured and tiered architecture along with an extensive description of implementing the architecture in the Hyperledger Fabric framework (including relevant algorithms and techniques). Additionally, all our propositions are evaluated using real telco data in a prototype architecture we have developed. The overall contributions of our work are summarized as follows:

- We introduce Triabase, a platform for a permissioned blockchain datastore that employs data decaying principles to ingest massive amounts of IoT data swiftly;
- We propose a new consensus empowering collaborative mechanism namely PoFL to share parameters over distributed multiple parties to reduce the risk of data leakage and to protect federated nodes from being tampered;
- Triabase integrates the fabric open-source platform to provide a more realistic blockchain assessment using Telco Big Data.

The remainder of this paper is structured as follows. Background and related work is included in Sect. 2. An overview of the *Triabase* architecture is presented in Sect. 3, where we discuss the specific internal techniques of each layer in our architecture. Section 4 presents our prototype architecture and its user interfaces. In Sect. 5, we describe our experimental methodology, the datasets, and evaluation metrics while Sect. 6 presents our experimental results. Finally, we summarize our conclusions and future work in Sect. 7.

2 Background and related work

In this section we overview the background and related work with a focus on: (i) federated learning; (ii) blockchain data management; and (iii) data decaying, both of which are instrumental in the design of our architecture.

2.1 Federated learning

Federated learning [15] is a machine learning approach that protects privacy by training models on several devices using local data samples without needing to send the whole model to the aggregators but instead only an updated version. Federated learning presents a number of difficult challenges, including coordinating member actions, adjudicating participant rewards, and aggregating models. The majority of systems now in use have a centralized approach, requiring coordination from a reliable central authority. Such a strategy has a number of drawbacks, such as assault susceptibility, lack of trust, and challenges in estimating incentives [16].

To properly protect the privacy of companies and customers, several federated learning issues must be addressed. The authors of [17] categorize existing system models into three classes: decoupled, coupled, and overlapped, according to how the federated learning and blockchain functions are integrated. Then, they compare the advantages and disadvantages of these three system models, especially focusing

on the challenges issues on BlockFed, and investigate corresponding solutions. Finally, they identify and discuss the future directions, including open problems in BlockFed. In another survey, authors of [18] explained the rationale behind MEC (Mobile Edge Computing) and discussed how Federated Learning may be used as an enabling technology for group model training at mobile edge networks. The foundations of DNN model training, Federated Learning, and system architecture for Federated Learning at scale are then discussed. They also provide thorough assessments, analyses, and comparisons of various implementation strategies for new implementation difficulties in Federated Learning. Costs associated with communication, resource allocation, data privacy, and data security are among the problems. Additionally, the authors talk about how Federated Learning may be used for privacy-preserving mobile edge network optimization. Finally, they talk about problems and potential future study areas. In addition in article [19] the authors of this article build FedIoT platform that includes the FedDetect algorithm for detecting anomalous data on-device and a system architecture for federated learning on IoT devices. Furthermore, the authors are building FedDetect learning framework, which boosts performance by employing a cross-round learning rate scheduler and a local adaptive optimizer (such as Adam). They analyze the model and system performance of the FedIoT platform and the FedDetect algorithm. The results show that federated learning is effective in identifying a greater variety of attack types that happened at numerous devices. According to the system efficiency study, end-to-end training time and memory costs are reasonable and show promise for IoT devices with limited resources.

The authors of [20], remediate this problem by introducing the concept of proof-of-learning in ML. Inspired by research on both proof-of-work and verified computations, they observe how a seminal training algorithm, stochastic gradient descent, accumulates secret information due to its stochasticity. This produces a natural construction for a proof-of-learning which demonstrates that a party has expended the compute require to obtain a set of model parameters correctly. In particular, their analyses and experiments show that an adversary seeking to illegitimately manufacture a proof-of-learning needs to perform at least as much work as is needed for gradient descent itself. They also instantiate a concrete proof-of-learning mechanism in both of the scenarios described above. In model ownership resolution, it protects the intellectual property of models released publicly. However, the authors lack the novelty of how the distributed process happens and how the nodes reach agreement or decide to commit blocks and how we can protect the privacy of the data. Furthermore, they are not targeting IoT devices and finally, there is a condition to adjust a lot of parameters.

Since federated learning is advocated as a solution to the issue of privacy data protection in machine learning, we must make sure that the training model in federated learning does not divulge users' personal data [21]. In a dispersed context, the quantity of data on each mobile device is insufficient, while a big amount of data is needed to train a model with high performance in classical machine learning [22]. On the other side, centralizing data collection might result in significant costs. For that reason, federated learning mandates that each device utilize local data to train the local model, which is subsequently aggregated into a global model on the

server. In the federal environment, there are many edge devices, and the data stored on these devices may not be independent and identically distributed (Non-IID).

2.2 Blockchain data management

The main usage of the blockchain architecture is to keep records on an immutable chain of blocks, so later on, nodes agree on the shared state across a network of untrusted participants. Thus, it forms the blockchain platform that can be viewed as a distributed (transaction-log or) database system. The blocks are agreed by the majority of validators according to the consensus protocols that tolerate Byzantine faults. The most well-known platforms include Capera [23], Hyperledger [24], Monoxide [25]. This design does not require a centralized server and operates in untrusted environments of arbitrary nodes. The state of the art and technical emphasis on the most recent developments in the underpinnings of blockchain systems are first presented in this book [26]. It addresses hot topics in blockchains from a theoretical perspective, including cryptographic primitives, consensus, formalization of blockchain properties, game theory applied to blockchains, and economic issues. It is a collaborative work between experts in cryptography, distributed systems, formal languages, and economics.

The authors of [23] introduce a system named Caper, a permission blockchain architecture based on an acyclic graph and on three consensus protocols to support internal and all cross-application transactions. Moreover, [27] introduces a novel framework, called vChain, which is able to improve the storage and computing costs of the user and employs verifiable queries to ensure the system integrity. The design of a privacy-preserving contact tracing framework to ensure the integrity of the tracing procedure has not been sufficiently studied and remains a challenge. In paper [28], the authors propose P2B-Trace, a privacy-preserving contact tracing initiative based on blockchain and privacy-preserving principles are a future direction of our proposed architecture.

Artificial intelligence along with the integration of blockchain technology is a great promise to solve various resource optimization problems. For instance, the merit of the two technologies is proposed in [29] providing a secure resource sharing scheme by developing a caching mechanism with the usage of DRL (Deep Reinforcement Learning). Reyna et al. [30] introduced how blockchain may potentially improve the IoT environments and how blockchain can protect from IoT security problems. However, AI algorithms, which are vulnerable to security threats depend much on centralization approaches, a fact that has a negative impact on improving efficiency, because it consumes a large number of communication resources.

Moreover, a considerable interest in the blockchain field is the scalability and performance characteristics of blockchain networks. Algorand [31] and RandHound [32] achieve high scalability by randomly selecting a subset of validators to participate in the consensus, while they maintain and guarantee the same security level with other blockchain infrastructure. Other works [33] use directed acyclic graphs instead of a blockchain structure and they ensure that the average amount

of time for each transaction is reduced. Blockbench [34] was the first to look for permissioned blockchain in the context of benchmarking. They present an approach for comparing the performance of different platforms including Ethereum Parity, and Hyperledger Fabric by using a set of micro and macro benchmarks. Furthermore, [24] introduces the architecture of fabcoin which presents the performance of bitcoin in Fabric. The research [35, 36] presents Adrestus which describes what techniques cryptocurrencies should adopt to build a scalable cryptocurrency with enhanced security.

Ghost protocol [37] is a well-prominent work paper that leverages existing problems on PoW algorithms, in order to prevent malicious attackers to create forks in the network by following selfish mining attacks. Moreover, some other recommendations come next to improve the scalability of blockchain. Bitcoin-NG [38] is a distributed fault tolerant protocol designed to scale the blockchain architecture, which claimed the same trust model as Bitcoin. Although Bitcoin-NG increases the overall throughput, it is still vulnerable to these kinds of attacks [39, 40]. However, it goes beyond the state of the art and can be seen as an enhancement of the existing models, improving the performance and focusing on the achievement of better security, scalability, and robustness.

2.3 Compacting data

There are a variety of techniques to compact data, ranging from compression algorithms [11] to data synopsis and data decaying ideas, described in this section.

2.3.1 Data decaying

This refers to the progressive loss of detail in information as data ages with time until it has completely disappeared. Kersten refers to the existence of data fungus in [41] with a decaying operator coined *"Evict Grouped Individuals (EGI)"*. The given EGI operator performs biased random decaying, resembling the rotting process in nature (e.g., in fruits with fungus). In our previous work [12], we used the *First-In-First-Out (FIFO)* data fungus, i.e., *"Evict Oldest Individuals"*, which retains full resolution for recent data but abstracts older data into compact aggregation models. Both EGI and FIFO do not retain full resolution for important instances that occurred in the past. Consequently, data would have been rotted and purged either randomly or based on its timestamp. We call this the *long-term dependency* problem. In [12], we chose a radically new decaying technique that could be termed as *LSTM data fungus*, which is explicitly designed to avoid the *long-term dependency* problem. Particularly, the *TBD-DP* operator replaces the data with abstract LSTM models, which capture the essence of the past, i.e., both recent data and important old data is retained at the highest possible resolution. There are a variety of amnesia functions, namely FIFO amnesia, UNIFORM amnesia, SPATIAL amnesia and query-based amnesia that differ in the predicate used for amnesia function.

2.3.2 Compressing incremental archives

Scientific simulation floating point data [42–45], spatiotemporal climate data [43], text document collections [42], and data streams, are all frequently use domain-specific compression algorithms. The trade-off between compression ratio and decompression times for incremental archive data has also been studied using differential compression algorithms in a number of research investigations [46, 47]. However, none of these earlier studies have ever suggested a method for addressing data decay in distributed systems that are special to telecommunications companies.

2.3.3 Data synopsis

This is the procedure of picking a subset of data pieces at random from a sizable dataset. Using probabilities and statistics, sophisticated approaches like Bernoulli and Poisson sampling select data items. Stratified sampling was suggested by Chaudhuri et al. [48] when the likelihood of the selection was biased. Zeng et al. [49] implemented G-OLA, a model that generalizes online aggregation in order to accommodate general OLAP queries using delta maintenance techniques, in order to deal with the huge data sampling problem. In particular, BlinkDB [50] uses dynamic sampling methods to let users select the error bounds and query response times. A system called SciBORQ [51] enables users to select the level of the query result's quality based on a variety of intriguing data samples known as impressions.

3 The Triabase architecture

In this section, we present the tiered architecture of Triabase, which comprises of a Storage Layer, Processing/Indexing Layer and the Application Layer.

3.1 Storage layer

We introduce the proposed Storage Layer of Triabase, and discuss its two internal routines, namely: (i) *Proof of Federated Learning (PoFL) routine*, which trains in a distributed manner a global model for the ingested data; and (ii) *Blockchain Consensus* routine, which commits this generated model data on permissioned blockchain datastore. The core functionality of our proposition is illustrated at a high level in Algorithm 1. The first routine of Triabase is the *PoFL*, which utilizes a convolution network loss function to train the local models across multiple decentralized edge nodes holding local data samples, without exchanging them. The final goal is to compute an average model and to converge fast with high learning accuracy. The second routine of Triabase is the blockchain process that is triggered after a respective leader election process takes place. The blockchain

process is responsible to collaboratively maintain the blockchain structure, endorse new transactions from blockchain nodes, and is partially responsible for the 2-step consensus protocol.

(A) Triabase storage: The Bitcoin protocol uses a PoW (Proof-of-Work) consensus mechanism to validate users' transactions in the blockchain. This is associated with an extremely high energy consumption bill, which is unnecessary in a private (permissioned) blockchain where contributing nodes are of higher trust. Yet, provisioning a consensus mechanism is still necessary in order to provide an incentive to participating nodes to contribute to the transaction verification process. To this end, in this work we propose such a consensus mechanism that relies on Federating Learning, as such, is coined *Proof of Federated Learning (PoFL)*.

Algorithm 1 Triabase: proof of federated learning blockchain consensus

Input: Blockchain nodes N, Time Epoch t, Model Weights (from Raw Data) W_t
Output: Blockchain TX identifier B_{id}

determine $l = leader(N, t)$ ▷ Leader / Orderer Election
determine $u = view\_number(N, t)$ ▷ Consensus Round
$T_{rx} \leftarrow init\_tx(t, u, W_t)$; ▷ Initialize transaction
while (!T_{rx}) **do** ▷ **Fabric Consensus Phase**
  if T_{rx}.PRE-PREPARE **then** ▷ Fabric stage PRE-PREPARE
    calculate f = l.difficulty(W_t,alpha) ▷ $alpha = bc_{\text{depth(t-1)}}$
    construct B_{id} = **l.build**(T_{rx}, W_t, u, f) ▷ **Triabase Block**
    for all $n_i \in N$ **do**
      send(PREPARE,t,l,f,B_{id}) ▷ lead by l
      n_i.receive(PREPARE,t,l,f,B_{id}) ▷ 2-step consensus initiated
    end for
  end if
  if T_{rx}.LEDGER-UPDATE **then** ▷ Fabric stage LEDGER-UPDATE
    for all $n_i \in N$ **do**
      l.send(COMMIT,t,l,f,B_{id}); ▷ lead by l
      n_i.receive(COMMIT,t,l,f,B_{id}); ▷ Commit block in Triabase
    end for
  end if
end while
return B_{id}

Particularly, our system ingests model weights (constructed from raw data *D* which will be described in the next section), and generates a global model *M* every *t* epochs with weights W_t in a distributed manner through federated learning. This procedure has two usages: (i) it contributes to the transaction verification process at the blockchain layer; and (ii) it helps in the reduction of space at the storage layer, as our framework stores now only W_t on disk as opposed to the raw data *D*. The storage layer is complemented with a local datastore for caching and handling of intermittent network connectivity, which however does not affect the overall philosophy of the system architecture where all data blocks have to eventually be committed to the blockchain layer. To improve performance in Triabase, we minimize the amount of data committed to the blockchain layer through data decaying principles, namely through the

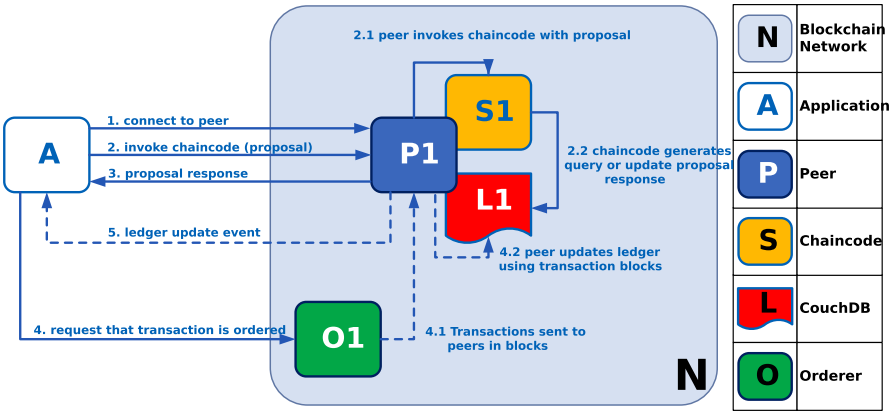

Fig. 2 Triabase blockchain consensus workflow

storage of a data postdiction model that allows for the retrieval of stored data through abstract models trained through federated learning.

In this section, we focus on the Triabase Storage Consensus algorithm (i.e., *Proof of Federated Learning (PoFL)*), which entails the first contribution of this work. Particularly, in Algorithm 1, we show the overall execution of the blockchain consensus routine. The process starts in lines 1–2 with a leader election routine followed by a view_number routine, both of which take as input the blockchain network N and the epoch t. The former yields the leader l while the later infers the fabric consensus round, which helps in the convergence of the consensus process and guarantees the liveness of the consensus protocol. Subsequently, in line 3 the transaction is bootstrapped and passes through two stages: the PRE-PREPARE stage (lines 5–12) and the LEDGER-UPDATE stage (lines 13–18). The PRE-PREPARE stage starts out by having the leader l computing the blockchain *difficulty*, which is derived based on the length of the blockchain ($alpha = bc_{\text{depth(t-1)}}$). Particularly, longer chains are expected to be more difficult compared to shorter chains. Based on the above a Triabase data block is constructed and broadcasted in the network for storage (i.e., lines 8–11). The LEDGER-UPDATE stage basically wraps up the communication by carrying out a final commit broadcast (lines 13–18).

Blockchains require to be versatile to different type of storage technologies. For this reason, our system architecture deploys pluggable local document stores. We have assessed two different types of NoSQL stores in our design, namely a CouchDB (default for Hyperledger Fabric) and LevelDB and provide experimental evidence for the utility of each of these storage layers and the impact they have on our overall system architecture.

(B) Triabase Consensus Workflow: The overall scheme of Triabase is shown in Fig. 2. The process starts with the local training of the model on user's data. After that, the communication process takes place where all users broadcast and upload the appropriately trained models to the blockchain nodes and store them as transactions to the distributed ledger. The blockchain node that was the winner

from the previous round (depends on the blockchain difficulty) is responsible for initiating a 2-step consensus protocol and construct the blocks with all the cached transactions that are not validated yet. In addition, the winner node is in charge of aggregating the local model of clients and producing a shared model by putting it as the first transaction in the block, so later on, the federated learning nodes can access it in the next round $r + 1$. Our PoFL consensus protocol contemplates that users who participate in the blockchain process get rewarded with training coins for their contribution in the whole algorithm (e.g, system usage coins contributed by participating parties). The coins of each user are awarded according to the performance in the training process. Particularly, federated nodes converging faster and achieving more accuracy are rewarded higher. The node that receives highest accuracy considering the difficulty of the block is recognized as the winner of round r. Furthermore, in every training round the coins will be adjusted to the users depending on their work.

Nevertheless, to secure our protocol and ensure that every user will obey the protocol, we introduce a new hierarchy of nodes that we called *peacemaker entity*. This entity is responsible to observe the correctness of the protocol followed by all the federating nodes. For example, users that refuse to cooperate with the protocol will get no payment for their work. Moreover, users that will try to get more rewards and try to counterfeit the correctness of the whole process will not be rewarded by the peacemaker entity. The peacemaker will then claim the adjusted coins as it's own reward for it's effort in the protocol correctness. Initial coins will be delivered to all participants as rewards after every epoch, it will also be available to claim after each block creation to those who prove correctness with the digital signatures. We set a minimum of 30 coins as the default setting, like Bitcoin used to have at the early stages generated for every block and spread it to validators.

(C) (C) Consensus Protocol: The two steps of the consensus protocol, summarized in Algorithm 1, include the execution of the PBFT [52, 53] algorithm and the notation of the detector. The orderer collects all the received transactions along with some endorsement proposals, constructs a new block, and initiates the 2-step phase by sending the proposal block to all blockchain nodes for verification.

In particular, the orderer broadcasts a pre-specified message to all fabric nodes to initiate the protocol. The message contains the proposed block B_{id}, the current epoch t, the current leader l based on the view number u (used for the PBFT [52] participation) and the block difficulty f that relies on the height of the blockchain. Then all fabric nodes $n \in N$ echo the same message until the majority of them receive at least a quorum of $3f + 1$ valid messages. Each blockchain node checks the validity of all transactions that exist in a block, by analyzing the endorsement policy from the assigned peers.

The detector is responsible for supervising that all the appropriate nodes comply with the endorsement policy and only the applicable peers join the process. The latter ensures that the client is not compromised and does not incorporate invalid results that may cause erroneous behavior to the Triabase blockchain.

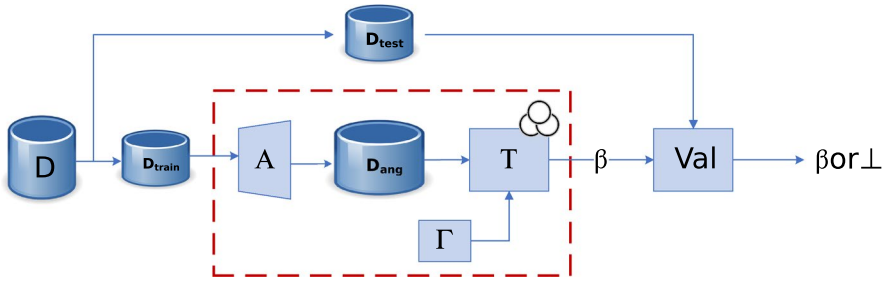

Fig. 3 Triabase converts raw data coming from TBD into dataset to be passed on federated nodes. If the parameters are approved, they could either be made public or used in a prediction service

(D) Ledger Update: After the fulfilment of the consensus protocol, the invoking process is called in which: (i) each client updates its copy of the ledger; and (ii) each client is notified about the ledger updates. In the proposed Triabase algorithm, the learning process is executed locally, i.e., trains machine learning models locally. Furthermore at the edge division, all local models are aggregated iteratively (in multiple rounds) to construct the final models, which are then stored in the blockchain (during the invoking process). We assume that all edge server nodes have enough computing and caching resources for completing complex calculations and maintaining the blockchain, in order to store the federated learning parameters collected from the users.

3.2 Data postdiction layer

The major objective of this layer is to reduce the query response time and keeping the storage capacity low by using data decaying concept.

The core idea of the layer is to deploy *Data Postdiction (DP)*, which is a technique that attempts to restore historical data from tuples that have been deleted in order to free up disk space [11, 12, 54]. Unlike data prediction, which aims to make a statement about the future value of some tuple, *data postdiction* aims to make a statement about the past value of some tuple, which does not exist anymore as it had to be deleted to free up disk space. Application send a query to compact models that can be stored and queried when necessary and with the notation of *DP* we recreate the past value of some tuple, which has been deleted to reduce the storage requirements, using a ML model from the compacted model. The DP operator has been modified to employ federated learning to transform IoT and telecommunications data into machine learning models that can be saved and retrieved on a blockchain network as needed. The DP operator utilizes an index to obtain the transactions found in the Blockchain after the successful completion of the federated learning and stores them to the temporal base index for quick retrieval.

In Fig. 3 we describe an ordinary ML training pipeline. Training set D_{train} and test set D_{test} are created from the data D that is streaming to Triabase. Cache values received then batch and shuffle data and prepare for processing. A training algorithm $\mathbb{A}$ that makes use of a regularizer Γ may be used to enhance the training data $\mathbb{T}$ after

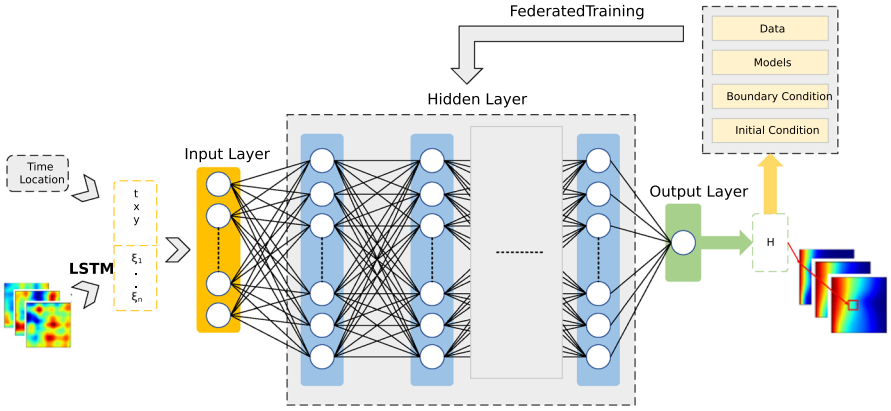

Fig. 4 Triabase reverse process indexing layer that show how the datasets given as input to the LSTM and produce the output layer to generate the model. The reverse operation is also happening to transform machine-learning models back to past values

which parameters are generated. The test set is used to validate the output parameters, which are then approved or refused (in which case an error is output). If the parameters β are approved, they could either be made public or used in a prediction service that the federated node has input/output access to (black-box model). The pipeline segments pass trained data with test data to keras model wrapper. TensorFlow Federated (TFF) is an open-source framework for machine learning and other computations on decentralized data. TFF can properly instantiate the model for the data that will actually be present on client devices are indicated by the dashed box defines the sequence for better performance. Federated nodes initialize the mode and pass the state to the federated process builder and send the results to federated server. Server collects results from all clients aggregates them and send the result back to federated nodes to process on next round. Federated nodes also remove the values that are not needed anymore to free up disk space. Server then sends the aggregated model to the Blockchain nodes in an asynchronous manner.

Additionally, we keep a global model M for a period p and its pointers, which lead to a list of transactions kept on the blockchain under the *Triabase* of the algorithm. With nodes that are the same size as disk blocks, the index is a B+ tree that minimizes the number of disk visits. The overall number of disk accesses for the majority of the activities are greatly decreased because the B+ tree's height is low. According to this, leaf nodes replicate all values related to non-leaf nodes. The B+ tree data structure is more difficult to use for Blockchain indexing since it contains both a key and a value attached to it. A pointer to the underlying data record is contained in this value. The payload is the result of the union of the key and the value. Each node's data in the B+ tree structure is stored in ascending order. Each of these keys contains two pointers that go to two further child nodes. There are fewer keys on the left side of the child node than there are now, whereas there are more keys on the right side. The maximum number of child nodes is $n + 1$ if a single node contains k keys. The DP operator

works in two conceptual phases:(i) offline phase, where it uses LSTM-based federated learning to build a tree of models *M* over time and space; and (ii) online phase, when it applies the tree *M* to retrieve data with a specific level of accuracy (Fig. 4).

Algorithm 2 shows the pseudocode for the *Retrieve Data algorithm*. The loading phase, in summary, loads the database sequence file in parallel such that federated node *Pi* gets approximately the *i*th $\frac{N}{p}$ byte chunk of the file. It's important to read sequences near the edges thoroughly, in order to improve the accuracy of future predictions. This phase guarantees that the database sequences are distributed evenly among the federated nodes. The query file is read in the same way, so each P_i gets around $\frac{m}{p}$ queries. The queries are then processed across p iterations in the following phase. Processor P_i checks all of its queries against D_j at any step s, where $j = (i + s)\ mod\ p$. A non-blocking request to obtain the database part for the next iteration is sent before the queries are executed. The communication is accomplished via the *Triabase* Get() one-sided communication primitive, which ensures that the distant CPU is not disturbed. P_i also maintains a separate running list of the highest results for each query in Q_i at each iteration phase. This list is printed after the program ends.

Algorithm 2 Retrieve data algorithm

▷ **Step 1: Loading**

Input: $D_i = P_i$ share of database (size O($\frac{N}{P}$))
Input: $Q_i = P_i$ share of ($\frac{m}{p}$) queries

▷ **Step 2: Blockchain Query processsing**
for all $s \leftarrow 0$ to $p - 1$ **do** ▷ **Retrieval Iteration**
  $j = (i + s) mod$ p
  if $s \geq 0$ **then** ▷ wait until D_j is received in D_{recv}:
    $w_{t,k} \leftarrow w_{t-1}$
    $D_{comp} \leftarrow$ copy D_{recv} ▷ Compute $\forall\ q \in Q_i$ against D_{comp}
    $j_{next} = (j + 1) mod$ p
    D_{recv} = Place non blocking request for $D_j next$
  end if
  for all $q_i \in Q$ **do** ▷ **For each local query**
    $res \leftarrow Update_{(q)}$ ▷ Find τ hits seen so far and update in every iteration
  end for
end for

▷ **Output: Reporting**
Output the final top τ hits for every local query

3.3 Application layer

An application that controls how a network system functions is part of the application layer. Users interact with the network, download data, and send data to other users of the network. They also utilize tools to access and share information at each other. Additionally, this layer is the highest level of our system, providing services directly to the underlying processes. The application layer contains the querying module and the user interface. In our example, the *Triabase* query module receives a data exploration question $Q(a, b, w)$ and uses the index to recover the data and respond to the query based on a, b, and w parameters. Finally, *Triabase* includes an RestFull API that hides the system's complexity while yet allowing access to all *Triabase* functionality.

It further allows smart devices coming from the TBD infrastructure to connect with one or more remote peers in the other layers and query information and get their results back as the whole data is saved in the blockchain and can be queried and verified very fast. In addition, users can query machine learning models and with the notation of *DP* model they can convert this model to raw data and take the result past for further processing.

At the application layer, the transfer of all operational data from mobile devices to edge servers required to create distributed models leads to a massive amount of communication burden and making the system susceptible to user privacy issues. In our proposed architecture, the learning process is carried out locally, allowing for the training of machine learning models on data from a range of users. The permissioned blockchain is maintained and the federated learning parameters collected from IoT devices are stored by the base station, which are computational and caching nodes. The base station aggregate the parameters in order to update the distributed model. Each BS implements the permissioned blockchain consensus process in order to maintain the consistency of the distributed model.

Due to the sensitive nature of the majority of the data and the amount of data to be processed, storing it on the *Triabase* limited storage space is a time-consuming and potentially dangerous operation. As a consequence, we use blockchain to get access to data, while the original data remains in the consumers' hands. When a new data provider joins, the blockchain records the data provider's unique identification (ID) together with other data attributes stored as a transaction. Each user's data account will be recorded in the form of transactions, which will be confirmed using the Merkle tree [55]. Each material distribution event is also logged as a transaction on the *Triabase*.

4 The *Triabase* prototype system

In this section, we describe our prototype system developed in the Hyperledger Fabric framework. We particularly overview the GUI and protocol of the framework as well as its evaluation and setup. Our prototype realizes the *Triabase* architecture using the *DP*, Blockchain and federated learning subsystems we described earlier.

4.1 Overview

Our server-side code[2] is written in Golang 13.8 node.js 10.23.0 and consists of around 8500 lines of code. In particular our server-code uses 5000 LOC its open source and you can fully retrieved from GitHub and runs over docker containers and Ubuntu Linux. The server side also includes CouchDB database and utilizes the *Triabase* package for drawing the docker images. A cross-platform, open-source runtime environment called NodeJS is used to create server-side web applications. The event-driven design of NodeJS also supports asynchronous I/O. NodeJS implements an event-driven, non-blocking I/O mechanism, which contributes to its efficiency and portability. NPM is a package module that aids in efficiently loading dependencies for javascript developers.

Our client-side code uses has around 2000 lines of code its written in python and can easily be integrated and handled with PyPi. It has a size of 2GB excel elements with datasets from Telco Big data. In order to run successfully run the code, you need to have Python 3.7–3.10, pip version 19.0 or higher for Linux and Windows. pip version 20.3 or higher for macOS. Because we use NVIDIA cuda cores in our testbed, the following NVIDIA software is required for GPU support: NVIDIA® GPU drivers version 450.80.02 or higher. CUDA® Toolkit 11.2. cuDNN SDK 8.1.0. (Optional) TensorRT to improve latency and throughput for inference.

4.2 Setup

The below commands bring up the REST server and execute the following from the project's app directory:

```
nvm use 12.19.0
npm install
rm -rf credstore
node cred-store.js org1 Admin
node --max-old-space-size=4096 server.js /
    none 10 false true 15
```

The last command can be adapted to suit the user's requirements as follows: *max-old-space-size* sets the server's max heap size (in bytes). The server (optionally) uses data compression to reduce the volume of the data sent to Layer 2. The options are *none*, *compress-json*, *compressed-json*, *jsonpack*, and *zipson*. The server uses

[2] Triabase. https://triabase.cs.ucy.ac.cy/.

paging in order to be able to handle big data volumes more efficiently. The page size is by default set to 10 MB. The value *false* deactivates console prints that can be used for debugging. Set to *true* to activate the debugging prints, which prints the average model submission latency of the N first model submission requests received.

4.3 Graphical user interface

Our system's Graphical User Interface gives users a basic interface via which they may inquire about the community's active users (the details of the protocol are presented in the next paragraph). *Triabase* provides the following distributed algorithms for storage and retrieval: (i) *Storage Algorithm of the* Triabase *Architecture*, and (ii) the *Retrieve Data algorithm*.

4.4 Query evaluation and processing

The Application Layer performs the basic functions of the interface between the Processing Layer and the Storage Layer. The usage of an intermediate level, inspired by the Hourglass Architecture internet-based is mainly intended to increase the interoperability of the Edge and Storage levels, sharing at least one function. Therefore, we have the maximization of the number of Federated Learning and Blockchain platforms/technologies that can be used in our *Triabase* architecture.

When scaling data across numerous nodes and dividing databases into separate partitions, CouchDB's architectural design allows for great flexibility. In order to establish an easily managed method for balancing read and write loads during a database deployment, CouchDB enables both horizontal partitioning and replication. The Application Layer acts as an intermediate layer that hides the complexity of the communication with the database blockchain network. This layer includes a REST server that communicates with the blockchain in behalf of its clients (the smart devices from the Edge Layer), who just use its simple endpoints instead. As of now, the supported endpoints offer model submission, updates and retrieval services, as well as basic metadata querying options, while the aspiration is to expand on a full-fledge datastore over the years. The ability to hide the complexity of communication with the storage level is another benefit of employing the intermediate level of application. This enables the option of manual data management of the Storage Layer to the authorized users of the system.

In the current version of the *Triabase* system at the application level there is a REST server, which implements two main functions. The first is the reception (from the Edge Layer) of the models data to be stored and their (conversion and) transmission in the form of transactions to the underlying Blockchain (in the Storage Layer). The second major function is to retrieve a model data from Blockchain. Specifically, after the successful execution of the relevant queries for their collection, the data (of the requested model) are returned to their original state and sent to the applicant. At this point, we note that the server offers exactly the same interface to both smart devices and authorized system users (the users don't provide a split/authentication mechanism), hence the introduction of the term: "applicant".

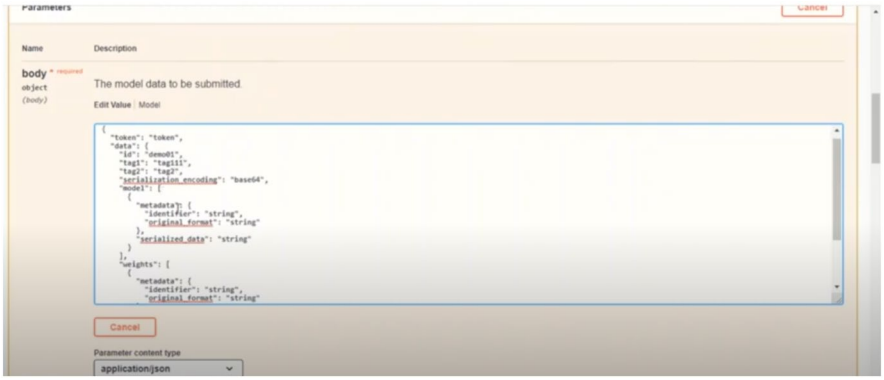

Fig. 5 *Triabase* provides an API through which ML models can be inserted to the blockchain in Base64

Finally, an interesting point to consider is the choice of utilizing the technique of tokens in the system. This is done to protect the security of the data of the storage level, by confirming the authorization to use the services of the Application Layer by the applicant. In particular, further to the information identification data of the model for storage/retrieval purposes, requests for services must include the applicant's token. The validity of this token is checked, and the execution of the request takes place only if it is confirmed (validity). Figure 5 shows the functionality of the model that stored and retrieved from the *Triabase* system.

In the current version of *Triabase*, we assume that authorized applicants have the ability to communicate directly with the permissioned blockchain of the Storage Layer, and store in the system ledger a string claimed to be the (valid) token. Although not an optimal solution, the system security is not compromised by the aforementioned approach, as in any case, only authorized users can contact Blockchain directly and successfully secure the token they want.

4.5 Query exploration interfaces

This subsection analyzes the various techniques and schemes selected during the interface component design phase, taking into consideration the requirements and limitations mentioned above.

First of all, like any interface that can be used by any programmer, the interface component must be easy to understand and use. It must also hide the complex communication with the system Blockchain, providing already implemented easy-to-use methods of interacting with it. Moving on to more technical requirements, the interface component must be able to handle the (possibly large and) different types of files of the serialized learning models that the *Triabase* system deals with. Moreover, it should be compatible with a wide range of other technologies that can be used by smart devices and IoT network systems. Finally, the interface component is called upon to ensure the integrity of the data as it is transferred from the application layer to the storage layer.

Fig. 6 The Triabase API definition is a file that describes all the provided functionality along with the necessary documentation

Triabase implements a full-fledge REST 2.0 API using Node.js and documented with Swagger. REST(Representational State Transfer) is the client–server architecture standard used in modern web applications. Figure 6 defines the available requests, as well as the response of each request. Among others, an important feature of a RESTful API is the provision of services through URL-based endpoints by a dedicated server. In the case of the interface component, the REST architecture was chosen for ease of use (simple, well-known concepts), moreover to increasing the number of smart devices capable of interacting with the application-level compatibility API (most devices support HTTP messaging). Additionally, regarding the implementation technology of the REST-full API of the component, given the limited options (SDKs) mentioned in the previous subsection, the Node.js ecosystem was selected. This selection was made for performance purposes only.

Let us mention the ways in which the interface component utilizes the techniques of digests and data compression. Regarding the use of digests, when receiving data in the endpoint model, the server of the interface component creates the corresponding pages (data) that are forwarded for storage in the system blockchain. For each page, the server calculates a digest, stored it in its database, and attaches it before sending it for storage. When, at a later stage, a request for retrieval of this data is received (endpoint model), the server retrieves the pages of the request model, ensures the integrity of the data of each and, restores the data. After that, it sends them to the user. Specifically, for the existing version of the interface component, the use of the MD5 fragmentation function was chosen, mainly for saving space (MD5 digest size = 128 bits). The purpose of the application of compression mechanisms is to reduce the volume of data circulating in the network of the Blockchain system and at the same time to increase the speed of completion of storage and recovery operations of models. Noting that the data files of the machine learning models may already be in fairly compact/compressed form.

4.6 Compression

This technique is mainly responsible for reducing the amount of storage space needed while exerting as little pressure as possible on query response times. It makes sense to adopt compression methods that result in high compression ratios while also ensuring quick decompression duration. In this paper, the GZIP compression is adopted because it provides fast compression and decompression times, a high compression ratio, and optimal compatibility with I/O stream libraries. Additionally, in order to supply the decay mechanisms for the following layer, we employ the *DP*. The *Triabase* indexes' (B+ trees) leaf pages are essentially the only thing the storage layer is in charge of; this is covered in the following layer.

With each new data snapshot received, the multi-resolution spatio-temporal index used by the Indexing Layer is increased on the rightmost path (i.e., every 30 min). Additionally, the component creates highlights-interesting event summaries-from the data kept in the child nodes and keeps them at the parent node. The internal node that covers the query's temporal window is accessible for each data exploration query, and its highlights are used to provide an answer. The querying module and the data exploration interfaces are implemented in the application layer. These interfaces accept visual or declarative data exploration questions and use the index to compile the necessary highlights and snapshots to respond to the query.

5 Experimental testbed and methodology

In this section, we describe our experimental methodology, which involves both a set of real micro-benchmarks for the *Triabase* system, utilized with real datasets from the Telco dataset, Smarty dataset and Marta dataset.

5.1 Datasets

We make use of the following three realistic datasets in our trace-driven experiments to simulate regular-scale, medium-scale, and large-scale distributed machine learning models:

– **Telco Dataset:** We [12, 54] utilize anonymized data from a real telecommunications company with 1192 genuine cell towers (i.e., 3660 cells from 2G, 3G, and LTE networks) spread across a 5896 km^2 area. The cells are linked to a cluster of computers through a gigabit network. For the performance of the tower, each cell tower keeps numerous UMTS/GSM network logs and passes the information to the base station controller (BSC) or the radio network controller (RNC) to be kept. In the enterprise, a CDR server generates call detail records (CDRs) for incoming and outgoing calls. The management server and third-party application can use SFTP to get a CDR from the CDR server after it has been generated. The Telco can then

query the CDRs for call/data details and check the carrier's outbound call/data fees. We utilize an anonymized dataset of telco traces comprising of 100 M network measurements records (NMS) and 3660 cells (CELL) coming from 2G, 3G and LTE antennas. The data traffic is created from about 300K objects and has a total size of 10GB. Our dataset includes 200 snapshots from the 5GB anonymized and uncompressed telco dataset that comprises of 1.7M CDR and 21 M NMS records. Our microbenchmark is performed atop an HDFS v2.5.2 filesystem.

- **Marta Dataset:** This [56] is an anonymized dataset of IoT network data comprising of 93.1MB network measurements records and 102 columns from 13 different files coming from sensors of an intelligent city. This dataset will showcase data collected via sensors within the system, specifically real-time data of trains buses and parking. It contributes to the development of practical solutions to issues that help improve the riding experience and boost ridership.
- **Smarty Dataset:** Sma-Rty [57] is an Italian-French startup specializing in AI and Machine Vision. Sma-Rty produces solutions for the integration of modern artificial intelligence technologies in real life through close collaboration with research institutes. In this context, the 5G Automotive Digital Twin (ADT) project intends to test and validate an innovative system for driving assistance and support based on 5G technology and Artificial Intelligence. The ADT system creates a digital depiction of the setting called a Digital Twin using traffic cameras and 5G infrastructure. This representation is used to replicate road user behavior in a virtual environment in order to predict potential risk situations in the real world and improve road safety. The behavioral prototypes of the identified entities and their visual features (for example, type, color, form, and speed) will be reconstructed beginning with the acquisition of video flows. After that, the ADT model gathers data from local and simulation units to deliver a proactive, non-invasive service for increasing road safety. The Torino City Lab experiments will thus focus on the validation of the digital twin model's functions as well as the accuracy of the rebuilt model.

We also considered other types of datasets/benchmarking efforts for our experimental methodology as follows: TPC and YCSB focus on data management workloads in conventional SQL and NoSQL environments and have little provisionings for both IoT scenaria and Blockchain scenaria. Also our effort was not to investigate these systems from a clearly data management standpoint, but rather from the standpoint of IoT data ingestion over a distributed blockchain layer that made the selection of benchmarking datasets a challenge. For this reason we carefully chose to feed our architecture with custom datasets from Telco (TBD), IoT (Marta) and AI/ML (Smarty). The TPC council only recently issued the TPCx-IoT bechmark, which is an iot-specific benchmark but has no specific provisionings for blockchain operation. On the other hand, there are also some Blockchain-specific benchmarking efforts underway (such as BlockBench [34]), but unfortunately these efforts have not ripened to allow generalizability for the types of IoT blockchains we consider in this work.

5.2 Metrics

The experimental evaluation described in this section focuses on the following metrics that aim to assess the performance qualities of the Triabase framework. We break the evaluation into two sections: (i) data ingestion, storage capacity, accuracy in NRMSE, and data retrieval experiments; and (ii) blockchain storage layer experiments, which respectively feature separate performance metrics.

For data ingestion and data retrieval experiments, as part of the decaying layer, we employ the following metrics:

- **Ingestion Time:** This measures the wall clock time to ingest each new snapshot and is measured in seconds (s).
- **Storage Capacity:** This measures the total space that machine learning data occupy together, as a percentage of storage required by the RAW method (no decaying, no compression).
- **Accuracy:** This measures the error of the machine learning data using the Normalized Root Mean Square Error (NRMSE). A lower NRMSE value indicates a higher accuracy in the recovered data.

$$NRMSE = \frac{\sqrt{\frac{1}{n}\sum_{t=1}^{n}(x_1, t - x_2, t)^2}}{y_{max} - y_{min}} \tag{1}$$

  which is the normalized difference between the actual data ($x1$, t) and the predicted data ($x2$, t), where t is a discrete time point and y_{max}, y_{min} are the maximum and minimum observed differences.
- **Retrieval Time:** This measures the wall clock time to recover a data block from the blockchain and is measured in seconds (s).

Fabric's major performance indicators are throughput and latency, which we investigate thoroughly in the subsection 6.2. In *Triabase* we say that the cost where transactions are passed the consensus and stored to the ledger is known as throughput. Latency is defined as the time it takes for an application to deliver a transaction proposal to a transaction commit. For blockchain latency, as part of the storage layer, we employ the following metrics:

- **Blockchain Duration and Throughput:** This measures the duration of the individual blockchain latency to finalize a Triabase transaction, measured in *ms (millisecond)* and *tps (transactions per second)*.

Algorithms: The proposed Triabase framework is compared with the following approaches:

- **RAW:** does not apply any decaying on the whole dataset.
- **COMPRESSION:** the decayed dataset is compressed with the *GZIP* library, which has been shown in [11] to offer the best balance between compression/

decompression speeds, compression ratios and compatibility with I/O stream libraries.
– **SAMPLING:** a sampling method that picks every second item in the input stream, yielding a 50% sample size.
– **RANDOM:** uniformly randomly select one record from the decayed dataset.

Note that RAW and RANDOM are the baseline approaches used to demonstrate the trade-off between the storage capacity and the NRMSE objectives. This project's primary objective is to identify performance bottlenecks, thus we built a system called *Triabase* that spans several clients and stresses the system by constantly making transactions. In addition, each client sends out proposal requests at the same time and collects endorsements. The transactions are sent asynchronously in order to meet the deadline without having to wait for commitments. The benchmark framework, on the other hand, calculates performance and latency. All organizations and their colleagues participate in multi-channel trials. While various combinations are feasible, we believe our strategy will put the system through its paces.

5.3 Testbed and workloads

Testbed: The DMSL VCenter IaaS cluster of computers, a private cloud, houses 5 IBM System x3550 M3 and HP Proliant DL 360 G7 rackables, each with a single socket (8 cores) or twin socket (16 cores) Intel(R) Xeon(R) CPU E5620 @ 2.40GHz Intel(R) Xeon(R) CPU E5620 @ 2.40GHz Intel(R) Xeon(R) CPU E5620 @ 2. On an IBM 3512, these hosts contain a total of 300GB of main RAM, 16TB of RAID-5 storage, and are connected through a Gigabit network. The cluster of computers is controlled by a VMWare vCenter Server 5.1, which is connected to the VMWare ESXi 5.0.0 hosts. Nodes for computing: The compute cluster, which is running on our VCenter IaaS, is made up of four Ubuntu 16.04 server images, each with 8GB of RAM and two virtual CPUs (both running at 2.40GHz). Fast local 10K RPM RAID-5 LSI- Logic SCSI drives formatted with VMFS 5.54 are used in the images (1MB block size).

Workloads: Our experimental evaluation has been conducted based on an a diverse mix of federated learning, Tensorflow, blockchain, data mining, and Machine Learning (ML) workloads. All aforementioned workloads are driven by a telco-specific domain task. We particularly formulated the following five tasks (T1-T5). More specifically the query types supported by *Triabase* include *standard queries*, where only the newest database version is queried, *full historical queries* on a particular predicate, *range historical queries* on all updates in a specific time range, *delta query* and *equality*.

– **T1. Standard queries:** The main objective is to query the *Triabase* blockchain as a traditional database, where the clients send a query result and they only care about the result in the newest version. We require to guarantee that only the records that are not expired could be selected.

- **T2. Full historical queries:** Another type of supported query in the *Triabase* blockchain is called a historical query. This type of query provides transparent history to database federated clients and grants them access to all of the data records. In a full historical query, the client wants to see all historical records that satisfy a specific predictor.
- **T3. Range historical queries:** Historical queries can also be executed with desired time ranges in mind. This can be achieved by applying additional conditions to the blockchain attributes (To be more specific, all records are paired with two extra attributes: VF (stands for 'valid from') and VT (stands for 'valid to'). For example, let's assume that we want to get a snapshot of the database for a random block b at the height h, we are able to collect the records with $VF \leq h$ and $VT \geq h$., e.g.,

```
SELECT upflux,downflux
FROM CDR
WHERE ts>=2015 AND ts<=2016;
```

- **T4. Delta query:** *Triabase* further guarantees transparency by supporting delta queries. Delta query gives the opportunity to clients to make more flexible queries and always be informed from the previous updates. More specifically, we plan to provide an interface for clients to query the changes made by the transactions committed at any particular block. For example, we assume a user u, then if we make a delta query for this user we will get as a result the records of his transactions that involved before and after the height of the block which we give as input.
- **T5. Equality:** This task aims to retrieve the download and upload bytes for a requested snapshot, e.g.,

```
SELECT upflux, downflux
FROM CDR
WHERE ts=201601221530;
```

6 Experimental evaluation results

This section presents the experimental evaluation of our proposed *Triabase* system. We start out with data decaying evaluation, followed by four sets of benchmarks. Then, we continue with blockchain control experiments with respect to the throughput and latency of our blockchain system, as well as, benchmarks on different databases. This is followed by processing learning experiments to measure the learning time, NRMSE, and percentage of raw for three different ML modes. In addition, we measure machine learning models that have been pre-trained from

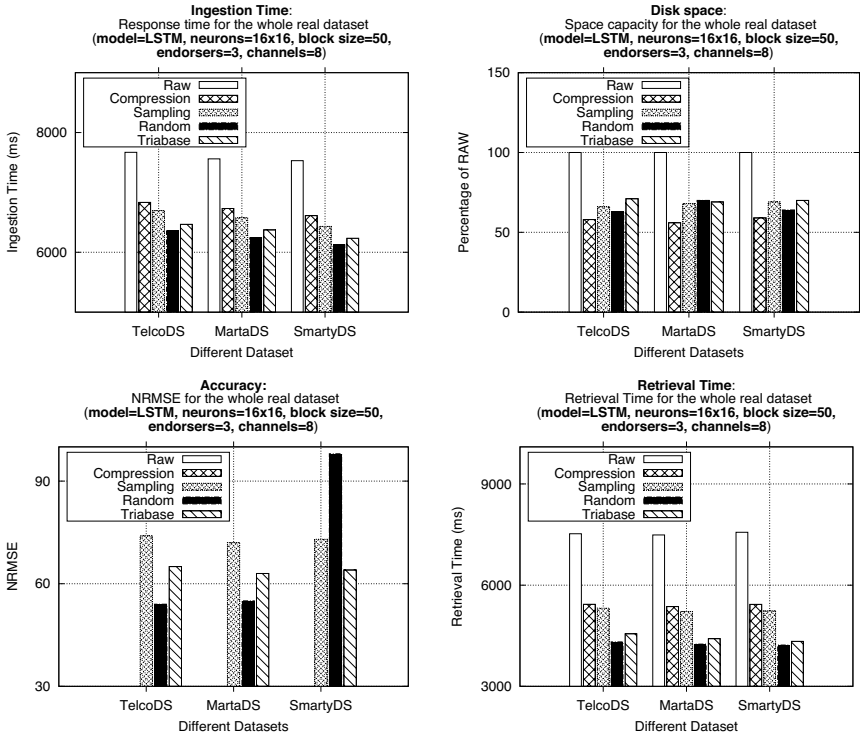

Fig. 7 Triabase performance evaluation: Triabase data ingestion, evaluation in terms of storage capacity S as a percentage to the RAW data (left) and accuracy in terms of NRMSE on the decayed set of data (right) in all datasets and retrieval evaluation for four evaluation metrics

different Machine Learning hubs. Typical examples of such hubs are the TensorFlow Hub and the Hugging Face.

6.1 Data decaying evaluation

In the first experiment, we evaluate the performance of the proposed *Triabase* system against all algorithms and over all datasets (Telco, Marta, SmaRty) introduced in Sect. 5.1, with respect to performance (as ingestion time of the model), space capacity (as a percentage to the RAW data), accuracy (in terms of NRMSE on the federated set of data) and retrieval time with the given datasets. We configure the Triabase framework according to the best configuration of blockchain network parameters and machine learning parameters that have been inferred through the control experiments of Sect. 6.2 and 6.3.

Figure 7 (top-left) demonstrates the data ingestion time for the three datasets in our evaluation. We observe that the highest ingestion time for all five methods is by the RAW method, which ingest the data in its raw representation. In contrast, the random method achieves the lowest time as it takes only a portion of

the models. Sampling and compression are semi-equal with only a difference of 10–20%. This is also reasonable, because first the decayed dataset is compressed with the *GZIP* library, to offer the best balance between compression-decompression speeds, thus we expect less data travel through the network and hence less retrieval time. *Triabase* demonstrates that the system has optimal ingestion time near random or sometimes semi-equal to it due to the learning time required by the federated process and the time required by the sequel for the commitment of the models as transactions on the Fabric blockchain store.

Figure 7 (top-right) demonstrates that the RAW occupies the most disk space because it stores all raw data. The sampling approach is following as the second worst case in terms of disk space. This happens because the sampling method picks every second item in the input stream, yielding a 50% sample size so we expect the disk space to be fair and not optimal. Random and *Triabase* are placed together with a difference of 2–5%. We expect *Triabase* to occupy this disk space because there is an extra overhead from the LSTM method that we can't avoid. The COMPRESSION approach, however, cannot be customized to achieve an even lower disk space occupancy. In comparison, the *Triabase* system can be configured, through its neurons, block size, endorser, and channels to accomplish a space occupancy that will fit the space budget of the application. This particular parameter will be investigated next in Sects. 6.2 and 6.3.

Figure 7 (bottom-left) demonstrates the trade-off between the space capacity S and the accuracy (NRMSE) objectives. The figure shows that the RAW approach obtained the worst possible $S = 100\%$ of the three datasets, and the lowest error $NRMSE = 0\%$. In contrast, the RANDOM (almost all data model in *Triabase*) approach obtained the best possible $S = 60\%$ disk space of the whole dataset because it takes a batch size of the whole dataset, and an error rate of $NRMSE = 30\%$ on the decayed dataset. The proposed *Triabase* system, however, provides around 5% and 10% worst space capacity S compared to COMPRESSION and SAMPLING approaches, respectively. This is due to the fact that an additional space required by the set of LSTM models is much more than the sample set of SAMPLING and the compressed decayed dataset of COMPRESSION. The SAMPLING outperforms the *Triabase* approach by 10–30%, on average. The COMPRESSION approach provides an optimal $NRMSE = 0\%$, since it does not apply any further prediction on the Blockchain model data but recovers it via decompression when requested.

Figure 7 (bottom-right) investigates ingestion time and we can observe that the RANDOM approach outperforms all the other approaches in all datasets. This happens because the RANDOM does not take all the dataset but instead takes a random batch size that is determined by the user. Hence, the retrieval time is less than all other approaches. After that, sampling and compression take place because compressed data can be retrieved more efficiently. Finally, the *Triabase* system takes the second lowest time because it needs not much time to calculate the LSTM models due to the extra optimization that we make.

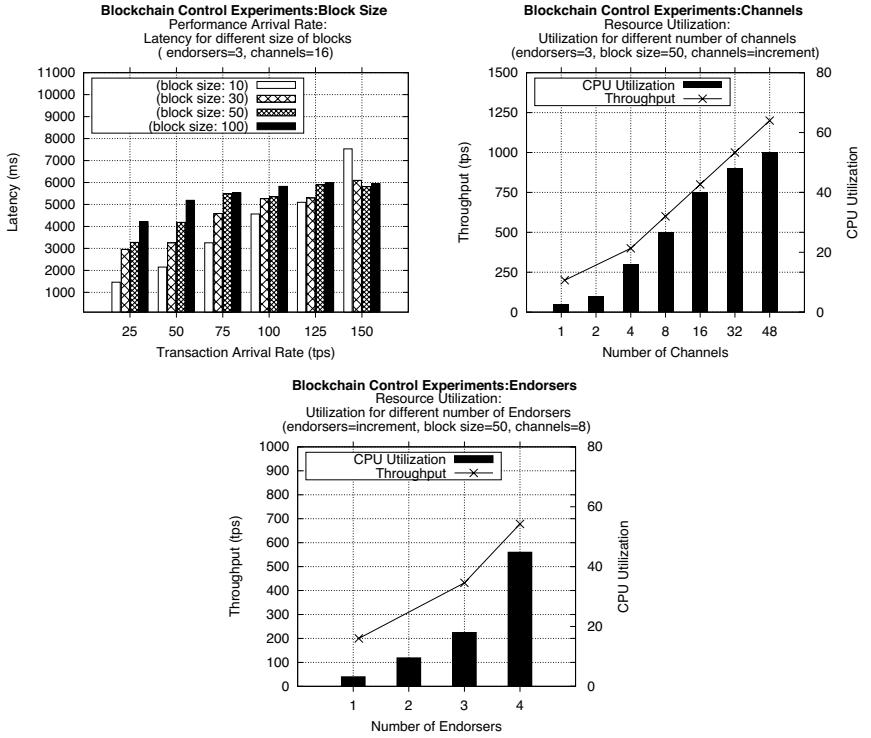

Fig. 8 Blockchain control experiments: the effect of various number of channels and endorsers on throughput

6.2 Blockchain control experiments

The storage layer of *Triabase* comprises of: (i) local data store used for caching and disconnected operation; and (ii) a fabric blockchain network. In this experiment we aim to evaluate the blockchain layer of the Triabase architecture, through a series of control experiments where various configuration parameters (blockchain block size, channels and endorsers) are assessed in isolation. In the next experiment of this section, we also present a microbenchmark where we assess the following incurred latency in the scope of two data store systems, namely CouchDB and LevelDB. Below is a breakdown of latencies incurred at the Blockchain layer:

- **Endorsement Latency:** The time it takes the client to collect all proposal answers as well as endorsements.
- **Broadcast Latency:** The time between when a client submits a request to an orderer and when the orderer acknowledges the request.
- **VSCC Latency:** The time it takes to check all of a block's endorsement signatures against the endorsement policy.

- **Ledger Update Latency:** The time it takes to check all of a block's endorsement signatures against the endorsement policy.
- **Commit Latency:** The time it takes for a node to prove that a transaction is valid and save it on the blockchain.
- **Ordering Latency:** The time needed for a transaction to complete its final ordering.

Finally, performance metrics for the response, ingestion time, and percentage of the total time are also included to present detail about how *Triabase* behaved both on the machine learning process and blockchain query time.

Figure 8 shows the impact of Block size, Channels, and Endorsement latency and throughput.

Block Size Evaluation: Figure 8 (top-left) demonstrates a liner improvement in latency while the transaction arrival rate increases until the congestion point of around 250tps (depends on block size). In addition, it shows that close to the congestion point there is a significant increase of latency, which is mainly due to the fact that the transactions are waiting in a queue to pass the validation phase and consequently delay the process. Moreover, the results show that latency is also affected by the block size, since when the block size is high, the latency is also high for low arrival rates. For example, when the arrival rate is 100 tps and the block size is increased from 30 to 50, the latency of the transactions is also increased from 4000 to 7000 ms. This happened because large block sizes increase the forging time of a block at the leader node, on average. For high arrival rates, however, and greater than the congestion point, the latency decreases. This is due to the fact that the time required to verify and store a block m is always less than the amount of time needed to verify and store b blocks. The major conclusion drawn from this experiment is that: (i) if the arrival rate of a request that represents a transaction is lesser than the congestion pivot, it is preferable for the application to use in most cases a lower block size to achieve low transaction latency. The throughput in those cases will be the same with the arrival rate. (ii) in cases where the arrival rate of a request is high and greater than the congestion point, it is preferable to use a large block size to achieve higher throughput and lower transaction latency.

Channel Evaluation: Figure 8 (top-right) demonstrates the *Triabase* throughput along with CPU metrics. The table below shows the number of channels and transaction arrival rate that were employed in this study. As indicated, all peers join all of the channels. Throughput increased as the number of channels increased. This is reasonable because more transactions happened parallel via the new channels as the throughput scales linearly. The negative fact of that is that as per channel increased the time to sync between nodes and channels increased dramatically because the network needs more hops to reach all endorsers and come to a consensus thus having all synced blocks on the network consumes enough time. Figure 8 (top-right) shows how resource use, such as CPU, increases.

Parameters	Values
Number of channels	8
Number of endorsers	3
Transaction complexity	1 KV write (1 − w) of size 20 bytes
SatateDB database	GoLeve1DB
Peer resources	32 vCPUs, 3 Gbps link
Endorsement Policy AND/OR	OR [AND(a, b, c), AND(a, b, c), AND(b, c, d), AND(a, c, d)]
Block size	30 transactions per block
Neurons	16x16
Model	LSTM

Endorser Evaluation: Endorsers are nodes that exhibit leadership behavior and are in charge of starting the consensus process. As we can see in Fig. 8 (bottom-left) with 4 endorsers, which is the default setting and most ideal, we are able to obtain 700 tps. Figure 8 (bottom-right) also shows that as the number of endorsers grows, the number of throughput scales linearly. It is also important to note that if the number of endorsers is increased excessively, throughput will not increase but rather eventually be destroyed because more time will be required to sync blocks across channels. As a result, we can anticipate an increase in latency and additional overhead, but this also depends on the block size. For instance, when the number of zones was increased from 2 to 15 we observe, the performance increased from 250 tps to 750 tps and 1200 tps (i.e., by 9.5 in the overloaded situation). This is due to the fact that each channel is self-contained and maintains its own blockchain. As a result, the validation process and the phase of updating the ledger of multiple blocks (one per channel) are executed with a parallel way, resulting in better CPU utilization and throughput.

Observation 1 It is preferable to dedicate at least one vCPU per channel in order to obtain good performance and reduced latency. To distribute vCPUs optimally, we must first evaluate the projected demand at each zone and then allocate sufficient vCPUs.

Observation 2 To maximize throughput and minimize delay, it is better to avoid heterogeneous peers, since their performance will be dominated by less powerful peers.

Microbenchmark: In this subsection we start out with a microbenchmark that evaluates the latencies incurred at the lower layers of the architecture measured in *ms (milliseconds)* as well as the generated throughput measures in *tps (transactions per second)* for two types of local key-value store systems: GoLevelDB and CouchDB.[3] LevelDB is a fast key-value storage library written at Google that

[3] CouchDB. https://couchdb.apache.org/.

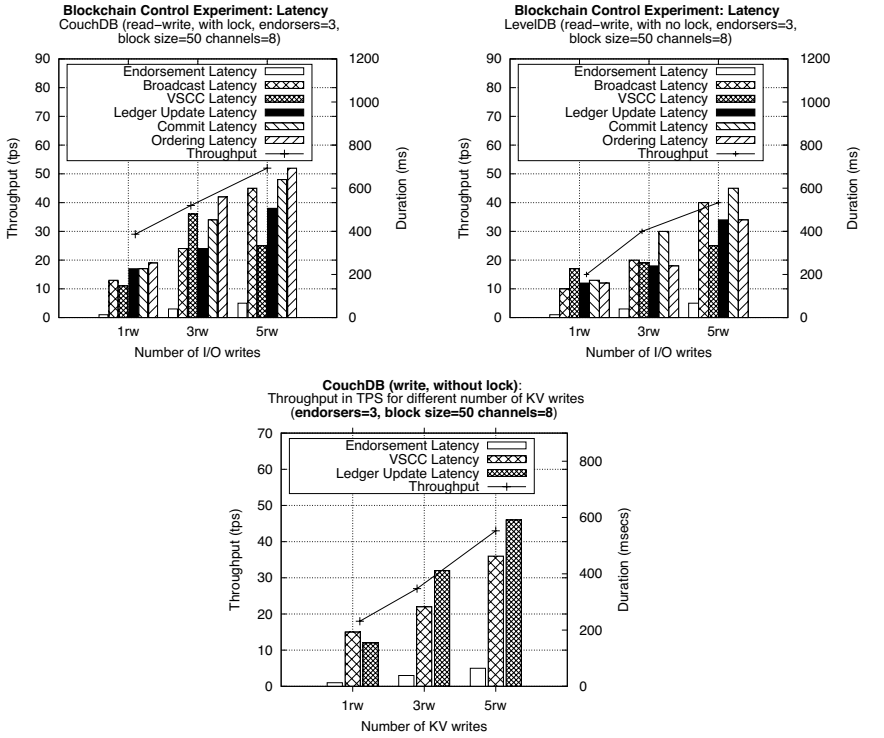

Fig. 9 Microbenchmark—key value store comparison: breakdown of latencies for two local data stores

provides an ordered mapping from string keys to string values. CouchDB is an open-source document-oriented NoSQL database, implemented in Erlang that uses JSON to store its data and MapReduce, and HTTP for an API. CouchDB is the default data store layer in the IBM Fabric blockchain network architecture we use to cope with temporary persistency and local caching.

Figure 9 demonstrates the total duration and throughput that each of the following latencies incurs on the system for three granularities of writes (i.e., 1, 3 and 5 Input/Output writes). The impact of different ledger databases (i.e., LevelDB and CouchDB) is also investigated, in terms of average throughput and latency for different transaction arrival rates. The results of Fig. 9 show that the transaction throughput with the LevelDB is greater than with the CouchDB. The maximum throughput measured with LevelDB was 450 tps while the couch database achieves around 400 tps. The primary reason for this is because LevelDB is a database that is contained in another database that processes transactions, while CouchDB relies on REST API calls, which pass over a secure HTTP tunnel and additionally has a delay of ledger updates and consequently result is in a lower throughput than the throughput of LevelDB.

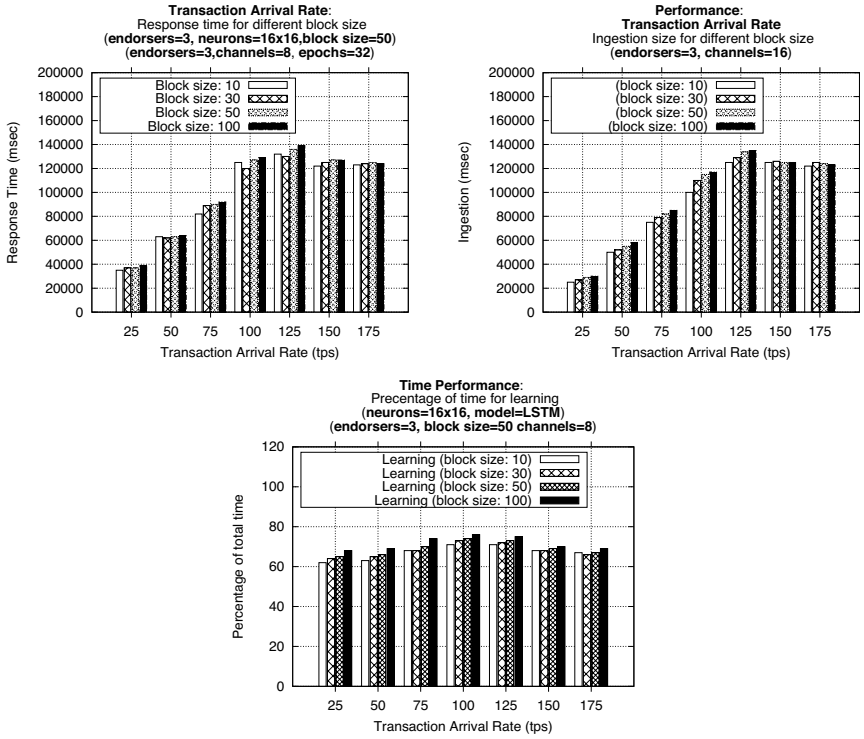

Fig. 10 Control experiment: the effect of various block sizes on latency, response and ingestion time

The results of Fig. 9 also show that as the CouchDB amount of writes per transaction rises, the latency of ledger updates increases. This is because CouchDB locking schemes cost more than those of LevelDB. Particularly, during the time of endorsement, the transaction acquires an exclusive lock to provide consistency of the chain code, which negatively affects the performance, as it performs three responsibilities for each transaction write set. That is, it firstly needs to retrieve the key with the use of a receive request and search if it finds it on the database. Secondly, it constructs an appropriate JSON scheme and lastly updates the DB by registering the put request. This results in an extra delay to the methods of the blockchain ledger phase.

In summary, the conclusion of this experiment has suggested that in order to achieve better performance in the fabric open-source network, GoLevelDB should be the best option for the blockchain operations. CouchDB, on the other hand, is a better choice when the design principles for the application require fewer read/write numbers of keys to validate a transaction. Moreover, CouchDB with a special operation described in [58], will restrict the locking scheme latency and will empower the overall performance.

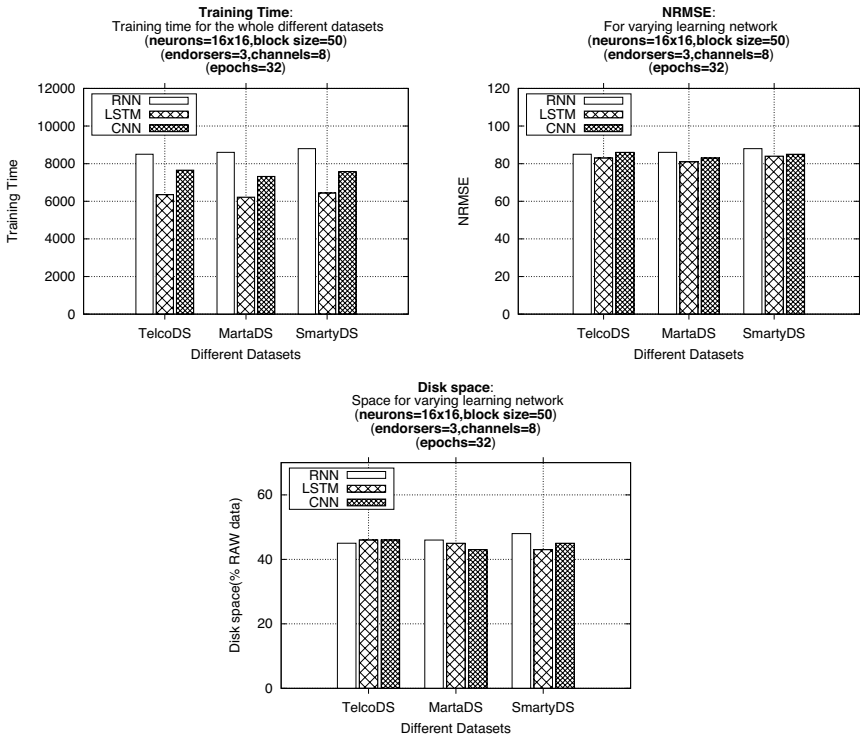

Fig. 11 Control experiment—learning models: examining the storage capacity S and NRMSE of the proposed Triabase approach while combined with various ML models

Experiment 10 was conducted to determine the size data page to be used in the system. More specifically, the performance of application and storage levels was examined using different page sizes. The results of the experiment are summarized in Fig. 10, which shows the throughput (MB/s) of the system during the submission/storage and retrieval of the models, respectively. We observe that, as expected, larger models have a smaller (submission) throughput. This is because they require longer process time, as they have larger number of pages (than smaller models) and therefore, require more transactions to be stored. In addition, we observe smaller page sizes resulting in higher throughput, which is due to the reduction in the amount of data exchanged between network peers.

However, the use of very small pages is also not recommended, as the transaction processing time (and not payload)—overhead—increases, and also reducing the throughput. Similarly in the case of data entry, we observe that the larger models have a lower (retrieval) throughput than the smaller ones. This is again due to the large number of transactions required to retrieve their multiple pages. Moreover, the results show that larger sizes (pages) have the best performance, unlike above (insertion), since the number of pages recovered is significantly smaller, thus reducing the overhead of data processing (and

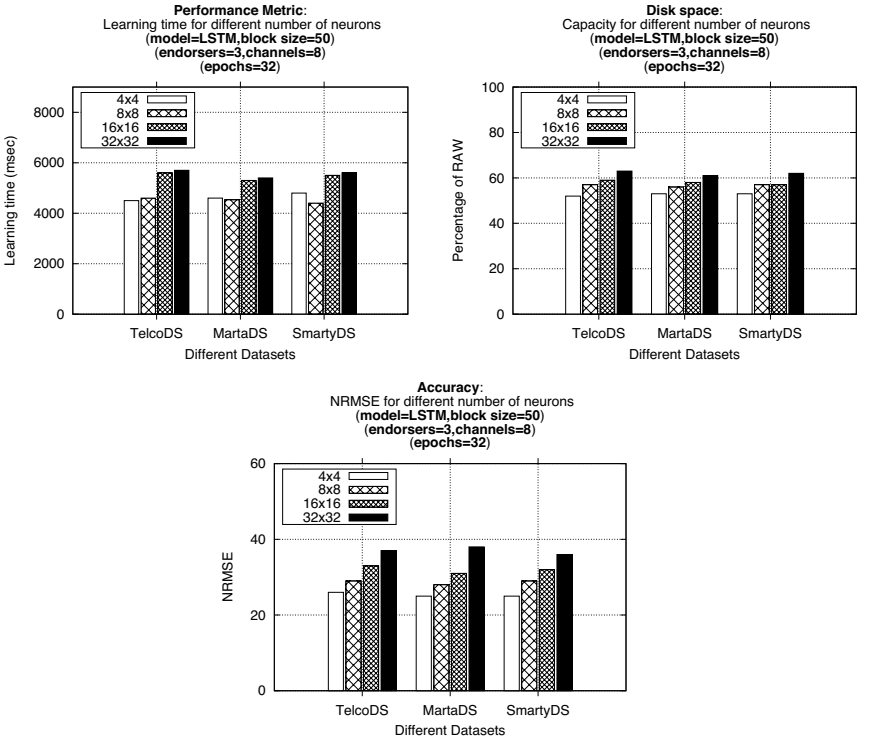

Fig. 12 Control experiments: training time, NRMSE and disk space for varying neurons and learning network configurations

increasing the throughput). This is mainly due to the fact that during the execution of retrieval transactions no data exchange takes place between the peers of the network (the consensus algorithm is not executed when reading data), which is emphasized by the fact that the submission throughput retrieval is higher than the retrieval throughput.

In addition, the Experiment 10 aims to study the system's performance during the management (insertion and retrieval) of large models. Test models and 10 MB as the data page size were used for this evaluation. The data entry and retrieval were done through the client simulator, while throughput (MB/s) was used again as the evaluation of metric. The results in Fig. 10, are quite positive, since they saw that *Triabase* can be used for larger models without problems. Also, we observe that when the size of the model exceeds 400 MB the throughput of the system reduced in both, the submission and the retrieval of data. In both cases, however, the submission and retrieval throughput is fixed at approximately 3.5 and 9.4 MB, respectively.

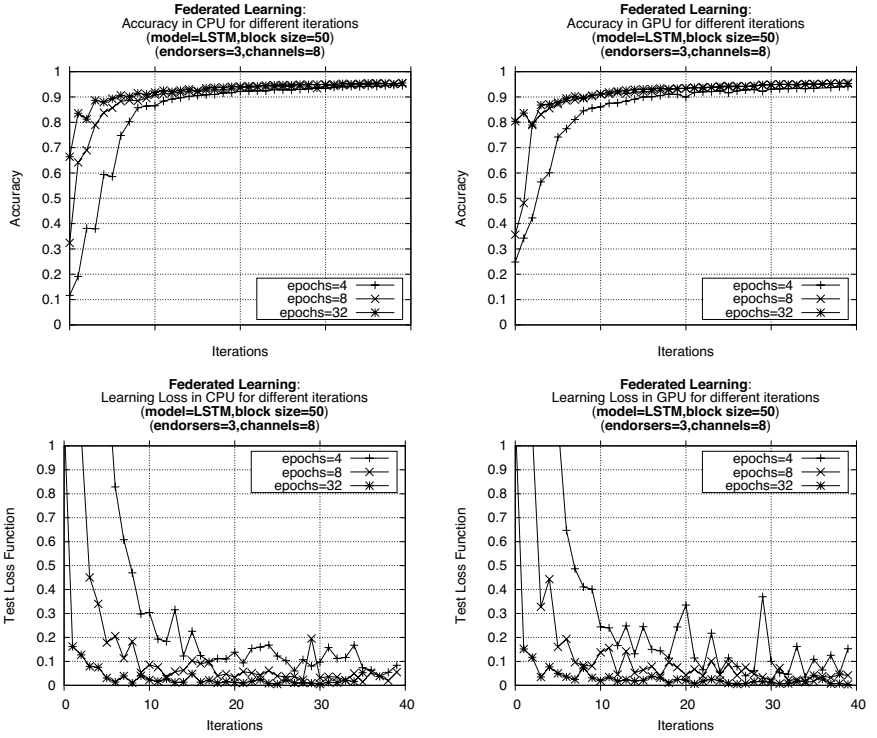

Fig. 13 Control experiments: fedarated training time, and loss for varying learning network configurations and different iterations

6.3 Machine learning control experiment

Figure 11 examines the performance of the *Triabase* in terms of training time, NRMSE, and percentage of raw when combined with three different ML models by performing the federated process, namely, the traditional Recurrent Neural Network (RNN), the Gated Recurrent Unit (GRU), and the Long Short Term Memory (LSTM) which is finally adopted by our proposed approach.

The results show that *Triabase* maintains a similar training time for different models for both Telco Marta and smarty datasets, with a slight decrease (about 5%) when the LSTM model is used. More specifically we observe that the worst-case scenario is when we use the Recurrent Neural Network (RNN) and this is happened due to gradient exploding and vanishing problems. Training RNN is a completely difficult task because it requires slow and complex training procedures. It finds difficulties in processing very lengthy sequences if the usage of Tanh or Relu as an activation feature. In terms of NRMSE, however, the Triabase and LSTM combination clearly outperforms the other two combinations providing around 1–2% less error, on average.

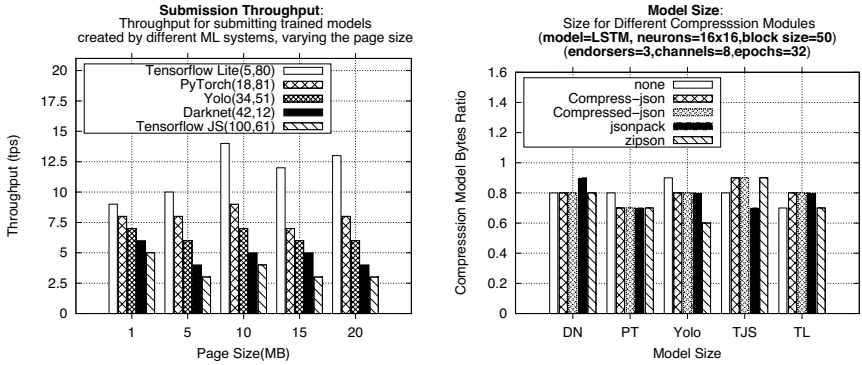

Fig. 14 Control experiments: submitting ML models on blockchain/fabric experiment

Furthermore, Fig. 12 examines how the number of neurons of the LSTM model influences the *Triabase* performance. The results support our previous observations on the scalability and efficiency of the proposed *Triabase* approach. The increase in numbers of neurons, slightly increases the required storage space of the *Triabase* system. This because the increase in the number of neurons results in bigger models that require more disk space to be stored. The additional required space, however, is almost negligible in comparison with the disk space needed to store the raw data before the federated process. In terms of NRMSE, the increase in the number of neurons does not influence the performance of the *Triabase* system, since NRMSE remains almost the same while in almost all datasets.

Figure 13 illustrates, we compare the proposed approach with respect to the state-of-the-art federated learning algorithm FedAvg [59]. This technique is also applied from Google in the keyboard app for better improving the user query suggestions [60].

Federated Setup: We use the CNN convolution layer and two dense layers. The first two convolutional layers have 32 and 64 filters respectively and they are responsible for setting the communication channels dynamically based on the width and the height of the image. The pool size is set dynamically (2,2) and the kernel size is 5. Moreover, convolution layers followed by a dropout [61] with a probability of 0.7. The second convolution layer has also a flatten operation. The last two dense layers are fully connected layers with 512 units activated by ReLu and a softmax output layer.

Algorithmic settings: In all experiments the algorithmic parameters were configured as follows: local mini-batch $B = 20$, the trained local epochs $E = 10$, the total number of clients $K = 500$ and the fraction of clients that performs computation at each round $C = 0.05$. The local training process for each client proceeds with the SGD optimizer with a learning rate $\eta = 0.001$ and no weight decay.

Figure 13 illustrates the performance of the proposed approach in terms of learning accuracy and learning loss, respectively, for various epochs over two different metrics. The results show that the proposed federated learning approach achieves high accuracy ($> 95\%$) and low learning loss ($< 10\%$) with a small set of iterations

for both CPU and GPU metrics. Moreover, the federated learning is performed faster when the GPU version is used and increases while the number of epochs increase. In particular, in the first 10 rounds, the training of the model converges faster and the accuracy of the model increases with the increase of the epoch. After 35 rounds, the accuracy is slightly reduced (by 2–4%) or remains the same, especially for the models trained with larger epoch values. This is due to the overfitting of the CNN model.

In addition from Fig. 13, we can observe that the learning loss is generally high at the beginning and it highly depends on the epoch value. For example, at round 5, the learning loss is around 0.5, which is relatively high when the epoch value is low. In contrast, when the epoch value is high (> 8) then the learning loss is reduced, which shows that the model converges. Moreover, the results show that when the epoch value is 32, the learning loss is reduced to almost zero, after round 10. There are also cases where the learning loss is high due to the overfitting but then it is reduced again to a close to zero value after some iterations (e.g., in round 38).

Regarding Fig. 14, the data used in this experimental evaluation are machine learning models, which are pre-trained on various Machine Learning hubs such as the TensorFlow Hub and the Hugging Face. The dataset also includes models used in Computer Vision. In particular, due to the valuable contribution of the sector in the field of IoT and given the most extensive use of machine learning techniques in the field of Computational Vision, this area is a typical example of the applicability of the *Triabase* system that we thoroughly discussed in previous Sections. Furthermore, the dataset models have been trained in object detection, using the well-known COCO dataset. In addition, it should be mentioned that the models implement different algorithms and/or architectures to achieve their task (object detection), such as YOLO, SSD Mobilenet v2.

6.3.1 Further optimization

To further improve the performance of the proposed system and to achieve a lower retrieval rate and ingestion time we also propose the following configurations:

- **Number of units in dense layer:** The dense layer is a layer where every neuron gets input from every other neuron in the layer below, making it "densely linked." Dense layers increase overall accuracy, and a reasonable starting point is 5–10 units or nodes per layer. Therefore, the number of neuron/units given will have an impact on the output form of the final dense layer.
- **Dropout:** A dropout layer should be included between each LSTM layer. By excluding randomly chosen neurons, such a layer lessens the sensitivity to particular weights of the individual neurons, preventing overfitting in training. 20% is a decent place to start, but the dropout rate should be maintained low (up to 50%). The ideal balance between avoiding model overfitting and maintaining model accuracy is generally agreed to be 20%.
- **Decay rate:** If no further weight update is planned, the weight decay may be added to the weight update rule that causes the weights to decline exponentially

to zero. The weights are multiplied after each update by a value slightly below 1, preventing them from becoming too large. This describes network regularization.
- **Activation function:** Technically, activation functions could be included into dense layers, but doing so would make it impossible to recover the density layer's decreased output.
- **Momentum:** Research has been done to combine the momentum hyperparameter with RNN and LSTM. Momentum is a special hyperparameter that enables the search to be guided by the accumulation of the gradients from previous steps rather than just the current step's gradient alone.
- **Parameters setup:** An effective strategy is to use the early stopping approach, define a high number of training epochs, and terminate training as soon as the model's performance on the validation dataset stops increasing by a predetermined threshold. 32 is generally recognized as a fair batch size default. We also experiment by using multiples of 32 like 64, 128 and 256 to find the most optimal use case.
- **Adaptive Setup:** Adaptive optimizer like Adam are advised to manage the complicated training dynamics of recurrent neural networks (which a simple gradient descent may not solve). by multiplying the total length of the sequence by the loss terms added along the way. In turn, it will be simpler to reuse the hyperparameters across tests since this will average out the loss throughout the batch. Gradient spikes have the potential to screw up training parameters. To avoid this, plot the gradient norm first (to determine its typical range) and then scale down any gradients that are outside of this range.

7 Conclusions and future work

In this paper, we introduce Triabase, a novel permissioned blockchain system architecture that applies data decaying concepts to cope with scalability issues in regards to blockchain consensus and storage efficiency. For blockchain consensus, we propose the PoFL algorithm which exploits data decaying models as Proof-of-Work. For storage efficiency, we exploit federated learning to construct data postdiction machine learning models to minimize the storage of bulky data on the blockchain. We have prototyped Triabase in Hyperledger Fabric and assess its performance using a variety of datasets from the IoT spectrum and Telco Big Data Spectrum showing that the proposition can achieve superior storage capacity and high throughput (i.e., ingestion and retrieval of data using the proposed data postdiction ideas.

In the future, we aim to expand the experimental evaluation with additional and more diverse machine learning models from platforms like Vertex AI,[4] which integrates processes for data engineering, data science, and machine learning engineering, allowing teams to work together using a single set of tools. We also aim to assess Triabase in a realistic TBD edge computing scenario and expand the

[4] Vertex AI. https://cloud.google.com/vertex-ai.

experimental evaluation with additional and standardized benchmarking frameworks when these become available. Finally, we aim to devise practical application scenarios of federated learning and sort out the current challenges and future research directions of data postdiction.

Author contributions A.B.C.E wrote the main manuscript text and A.B. prepared the experimental evaluation results. All authors reviewed the manuscript.

Funding The authors did not receive support from any organization for the submitted work.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose. The pre-last author is a Special Issue Coordinator of the Distributed and Parallel Databases journal. The last author is on the Editorial Board of the Distributed and Parallel Databases journal.

References

1. Yao, L., Sheng, Q.Z., Dustdar, S.: Web-based management of the Internet of Things. IEEE Internet Comput. **19**(4), 60–67 (2015)
2. Fuqaha, A.A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of Things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutor. **17**(4), 2347–2376 (2015)
3. Li, S., Xu, L.D., Zhao, S.: The Internet of Things: a survey. Inf. Syst. Front. **17**(2), 243–259 (2015)
4. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
5. Du, W., Atallah, M.J.: Privacy-preserving cooperative scientific computations. In: Proceedings of 14th IEEE Computer Security Foundations Workshop, pp. 273–282 (2001). https://doi.org/10.1109/CSFW.2001.930152
6. Billsus, D., Pazzani, M.J.: Learning Collaborative Information Filters. In: Proceedings of the 15th International Conference on Machine Learning. ICML '98, pp. 46–54. Morgan Kaufmann Publishers, San Francisco (1998)
7. Atallah, M., Bykova, M., Li, J., Frikken, K., Topkara, M.: Private collaborative forecasting and benchmarking. In: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society. WPES '04, pp. 103–114. Association for Computing Machinery, New York (2004). https://doi.org/10.1145/1029179.1029204
8. Li, J., Wang, C., Kang, X., Zhao, Q.: Camera localization for augmented reality and indoor positioning: a vision-based 3D feature database approach. Int. J. Digit. Earth **13**(6), 727–741 (2020). https://doi.org/10.1080/17538947.2018.1564379
9. Chen, S., Xu, H., Liu, D., Hu, B., Wang, H.: A vision of IoT: applications, challenges, and opportunities with china perspective. IEEE Internet Things J. **1**(4), 349–359 (2014). https://doi.org/10.1109/JIOT.2014.2337336
10. Costa, C., Zeinalipour-Yazti, D.: Telco big data research and open problems. In: Proceedings of the 35th IEEE International Conference on Data Engineering. ICDE'19, pp. 2056–2059. IEEE Computer Society, 8–12 April 2019, Macau SAR, China (2019). https://doi.org/10.1109/ICDE.2019.00238
11. Costa, C., Chatzimilioudis, G., Zeinalipour-Yazti, D., Mokbel, M.F.: Efficient Exploration of Telco big data with compression and decaying. In: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 1332–1343 (2017). https://doi.org/10.1109/ICDE.2017.175.2375-026X
12. Costa, C., Konstantinidis, A., Charalampous, A., Zeinalipour-Yazti, D., Mokbel, M.F.: Continuous decaying of telco big data with data postdiction. GeoInformatica **23**(4), 533–557 (2019)

13. Drakatos, P., Demetriou, E., Koumou, S., Konstantinidis, A., Zeinalipour-Yazti, D.: Triastore: A Web 3.0 blockchain datastore for massive IoT workloads. In: 2021 22nd IEEE International Conference on Mobile Data Management (MDM), pp. 187–192 (2021). https://doi.org/10.1109/MDM52706.2021.00038
14. Drakatos, P., Demetriou, E., Koumou, S., Konstantinidis, A., Zeinalipour-Yazti, D.: Towards a blockchain database for massive IoT workloads. In: 2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW), pp. 76–79 (2021). https://doi.org/10.1109/ICDEW53142.2021.00021
15. Li, L., Fan, Y., Tse, M., Lin, K.-Y.: A review of applications in federated learning. Comput. Ind. Eng. **149**, 106854 (2020). https://doi.org/10.1016/j.cie.2020.106854
16. Mammen, P.M.: Federated learning: opportunities and challenges. arXiv Preprint (2021). arXiv.arXiv:2101.05428 [cs]. https://doi.org/10.48550/arXiv.2101.05428. Accessed 17 Aug 2023
17. Zhu, J., Cao, J., Saxena, D., Jiang, S., Ferradi, H.: Blockchain-empowered federated learning: challenges, solutions, and future directions. ACM Comput. Surv. **55**(11), 240–124031 (2023). https://doi.org/10.1145/3570953
18. Lim, W.Y.B., Luong, N.C., Hoang, D.T., Jiao, Y., Liang, Y.-C., Yang, Q., Niyato, D., Miao, C.: Federated learning in mobile edge networks: a comprehensive survey. IEEE Commun. Surv. Tutor. **22**(3), 2031–2063 (2020). https://doi.org/10.1109/COMST.2020.2986024
19. Khan, L.U., Saad, W., Han, Z., Hossain, E., Hong, C.S.: Federated learning for Internet of Things: recent advances, taxonomy, and open challenges. IEEE Commun. Surv. Tutor. **23**(3), 1759–1799 (2021). https://doi.org/10.1109/COMST.2021.3090430
20. Bravo-Marquez, F., Reeves, S., Ugarte, M.: Proof-of-learning: a blockchain consensus mechanism based on machine learning competitions. In: 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON), pp. 119–124 (2019). https://doi.org/10.1109/DAPPCON.2019.00023
21. Truex, S., Baracaldo, N., Anwar, A., Steinke, T., Ludwig, H., Zhang, R., Zhou, Y.: A hybrid approach to privacy-preserving federated learning. In: Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security. AISec'19, pp. 1–11. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3338501.3357370. Accessed 17 Aug 2023
22. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: concept and applications. ACM Trans. Intell. Syst. Technol. **10**(2), 12–11219 (2019). https://doi.org/10.1145/3298981
23. Amiri, M.J., Agrawal, D., Abbadi, A.E.: CAPER: a cross-application permissioned blockchain. Proc. VLDB Endow. **12**(11), 1385–1398 (2019). https://doi.org/10.14778/3342263.3342275
24. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S.W., Yellick, J.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the 13th EuroSys Conference. EuroSys '18, pp. 1–15. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3190508.3190538
25. Wang, J., Wang, H.: Monoxide: scale out blockchains with asynchronous consensus zones. In: Proceedings of 16th USENIX Symp. Netw. Syst. Des. Implement. (NSDI), pp. 95–112 (2019)
26. Fernández Anta, A., Georgiou, C., Herlihy, M., Potop-Butucaru, M.: Principles of Blockchain Systems. Synthesis Lectures on Computer Science, vol. 9(2), pp. 1–213 Springer, Cham (2021). https://doi.org/10.2200/S01102ED1V01Y202105CSL014
27. Xu, C., Zhang, C., Xu, J.: vChain: enabling verifiable Boolean range queries over blockchain databases. In: Proceedings of the 2019 International Conference on Management of Data. SIGMOD, pp. 141–158 (2019). https://doi.org/10.1145/3299869.3300083
28. Peng, Z., Xu, C., Wang, H., Huang, J., Xu, J., Chu, X.: P^2B-Trace: privacy-preserving blockchain-based contact tracing to combat pandemics. In: Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data (2021). https://doi.org/10.1145/3448016.3459237
29. Dai, Y., Xu, D., Maharjan, S., Chen, Z., He, Q., Zhang, Y.: Blockchain and deep reinforcement learning empowered intelligent 5G beyond. IEEE Netw. **33**(3), 10–17 (2019). https://doi.org/10.1109/MNET.2019.1800376
30. Reyna, A., Martín, C., Chen, J., Soler, E., Díaz, M.: On blockchain and its integration with IoT. Challenges and opportunities. Future Gen. Comput. Syst. **88**, 173–190 (2018). https://doi.org/10.1016/j.future.2018.05.046

31. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. SOSP, pp. 51–68 (2017). https://doi.org/10.1145/3132747.3132757
32. Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 444–460 (2017). https://doi.org/10.1109/SP.2017.45
33. Li, C., Li, P., Zhou, D., Xu, W., Long, F., Yao, A.: Scaling Nakamoto consensus to thousands of transactions per second. arXiv Preprint (2018). arXiv:1805.03870 [cs]
34. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.-L.: Blockbench: a framework for analyzing private blockchains. In: Proceedings of the 2017 ACM International Conference on Management of Data. SIGMOD '17, pp. 1085–1100. Association for Computing Machinery, New York (2017). https://doi.org/10.1145/3035918.3064033
35. Drakatos, P., Koutrouli, E., Tsalgatidou, A.: Rapid blockchain scaling with efficient transaction assignment. In: 2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), pp. 1–9 (2021). https://doi.org/10.1109/SEEDA-CECNSM53056.2021.9566222
36. Drakatos, P., Koutrouli, E., Tsalgatidou, A.: Adrestus: secure, scalable blockchain technology in a decentralized ledger via zones. Blockchain Res. Appl. (2022). https://doi.org/10.1016/j.bcra.2022.100093
37. Sompolinsky, Y., Zohar, A.: Accelerating bitcoin's transaction processing. Fast money grows on trees, not chains. Citeseer (2013)
38. Eyal, I., Gencer, A.E., Sirer, E.G., Renesse, R.v.: Bitcoin-NG: a scalable blockchain protocol. In: Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16), pp. 45–59 (2016)
39. Apostolaki, M., Zohar, A., Vanbever, L.: Hijacking bitcoin: routing attacks on cryptocurrencies. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 375–392 (2017). https://doi.org/10.1109/SP.2017.29
40. Gervais, A., Ritzdorf, H., Karame, G.O., Capkun, S.: Tampering with the delivery of blocks and transactions in bitcoin. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. CCS '15, pp. 692–705. Association for Computing Machinery, Denver (2015). https://doi.org/10.1145/2810103.2813655. Accessed 6 June 2020
41. Kersten, M.L.: Big data space fungus. In: CIDR 2015, 7th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, 4–7 January 2015, Online Proceedings (2015)
42. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: Proceedings of the 18th International Conference on World Wide Web. WWW '09, pp. 401–410. Association for Computing Machinery, New York (2009). https://doi.org/10.1145/1526709.1526764. Accessed 26 Sept 2022
43. Soroush, E., Balazinska, M.: Time travel in a scientific array database. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE), pp. 98–109 (2013). https://doi.org/10.1109/ICDE.2013.6544817
44. Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.F.: Compressing the incompressible with ISABELA: in-situ reduction of spatio-temporal data. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011 Parallel Processing. Lecture Notes in Computer Science, pp. 366–379. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-23400-2_34
45. Bhattacherjee, S., Deshpande, A., Sussman, A.: PStore: an efficient storage framework for managing scientific data. In: Proceedings of the 26th International Conference on Scientific and Statistical Database Management. SSDBM '14, pp. 1–12. Association for Computing Machinery, New York (2014).https://doi.org/10.1145/2618243.2618268. Accessed 26 Sept 2022
46. You, L.L., Pollack, K.T., Long, D.D.E., Gopinath, K.: PRESIDIO: a framework for efficient archival data storage. ACM Trans. Stor. **7**(2), 6–1660 (2011). https://doi.org/10.1145/1970348.1970351
47. Bhattacherjee, S., Chavan, A., Huang, S., Deshpande, A., Parameswaran, A.: Principles of dataset versioning: exploring the recreation/storage tradeoff. In: Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases, vol. 8(12), pp. 1346–1357 (2015). https://doi.org/10.14778/2824032.2824035. Accessed 05 Apr 2021
48. Chaudhuri, S., Das, G., Narasayya, V.: Optimized stratified sampling for approximate query processing. ACM Trans. Database Syst. **32**(2), 9 (2007). https://doi.org/10.1145/1242524.1242526
49. Zeng, K., Agarwal, S., Dave, A., Armbrust, M., Stoica, I.: G-OLA: generalized on-line aggregation for interactive analysis on big data. In: Proceedings of the 2015 ACM SIGMOD International

Conference on Management of Data. SIGMOD '15, pp. 913–918. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2723372.2735381. Accessed 26 Sept 2022
50. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: BlinkDB: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM European Conference on Computer Systems. EuroSys '13, pp. 29–42. Association for Computing Machinery, New York (2013). https://doi.org/10.1145/2465351.2465355. Accessed 2022-09-26
51. Sidirourgos, L., Kersten, M., Boncz, P.: SciBORQ: scientific data management with bounds on runtime and quality. In: Proceedings of the biennial Conference on Innovative Data Systems Research (2011)
52. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation. OSDI '99, pp. 173–186. USENIX Association, Berkeley (1999)
53. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176
54. Costa, C., Charalampous, A., Konstantinidis, A., Zeinalipour-Yazti, D., Mokbel, M.F.: Decaying Telco big data with data postdiction. In: 2018 19th IEEE International Conference on Mobile Data Management (MDM), pp. 106–115 (2018)
55. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 839–858 (2016). https://doi.org/10.1109/SP.2016.55
56. MARTA hackathon—dataset by brentbrewington. https://data.world/brentbrewington/marta-hackathon Accessed 15 Dec 2022
57. Sma-RTy—SMArt systems & aRTificial intelligence. https://sma-rty.com/ Accessed 15 Dec 2022
58. Apache CouchDB: https://couchdb.apache.org/. Accessed 15 Dec 2022
59. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.y.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics, pp. 1273–1282. PMLR (2017)
60. Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., Beaufays, F.: Applied federated learning: improving Google keyboard query suggestions (2018). arXiv:1812.02903 [cs, stat]
61. Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of neural networks using DropConnect. In: International Conference on Machine Learning, pp. 1058–1066. PMLR (2013)

Authors and Affiliations

Panagiotis Drakatos[1] · Constantinos Costa[1,3] · Andreas Konstantinidis[1,2] · Panos K. Chrysanthis[1,3] · Demetrios Zeinalipour-Yazti[1]

✉ Demetrios Zeinalipour-Yazti
dzeina@ucy.ac.cy

Panagiotis Drakatos
pdraka01@ucy.ac.cy

Constantinos Costa
costa.c@cs.pitt.edu

Andreas Konstantinidis
com.ca@frederick.ac.cy

Panos K. Chrysanthis
panos@pitt.edu

[1] Department of Computer Science, University of Cyprus, 2109 Nicosia, Cyprus

[2] Department of Computer Science, Frederick University, 1036 Nicosia, Cyprus

[3] Department of Computer Science, University of Pittsburgh, Pittsburgh 15260, USA