Dagbase: A Decentralized Database Platform Using DAG-Based Consensus

Yepeng Ding
The University of Tokyo
Tokyo, Japan
youhoutei@satolab.itc.u-tokyo.ac.jp

Hiroyuki Sato
The University of Tokyo
Tokyo, Japan
schuko@satolab.itc.u-tokyo.ac.jp

***Abstract*—As the infrastructure to provide support for distributed database management systems, the distributed database platform is very important to unify the management of data distributed in intricate environments. However, a traditional distributed database platform with centralized entities faces diverse and serious threats when the central entity is compromised. Consensus mechanisms in distributed ledger technology (DLT) can enhance the capability of defending threats by decentralizing the platform, but the efficiency and cost of consensus mechanisms in classic blockchain techniques are notable issues. In this paper, we propose a novel decentralized database platform, named Dagbase, with the support of an efficient and cost-effective consensus mechanism that uses the directed acyclic graph (DAG) as the structure. Dagbase decentralizes the management and distributes data to prevent threats in untrustworthy environments, which gains benefits from recent DLT. The performance of near-native data reading and high-efficiency data writing is ensured by a layered architecture and DAG-based consensus. Furthermore, we ensure flexibility by decoupling the consensus mechanism from the architecture. Dagbase is also easy-to-use and can be integrated with mainstream database products seamlessly on account of great interoperability. The implementation demonstrates our work and the security and performance analysis are enforced for evaluation.**

***Index Terms*—distributed database, decentralization, decentralized system, database platform, distributed ledger technology, directed acyclic graph**

I. Introduction

Nowadays, the utilization of data plays a pivot role in information and communications technologies (ICT) such as web services and the Internet of Things (IoT). To manage data in an organized manner, the database is indispensable and usually integrated into the database management system (DBMS). For large-scale data management, it is imperative to design a distributed database platform as an infrastructure to support DBMS and unify control.

However, there are many sorts of threats to a distributed database platform on account of intricate and untrustworthy environments. It is quite a challenge to ensure security for a distributed database platform, especially for a platform with centralized enforcement or a central agency [1]. Typical threats can be decomposed to confidentiality, integrity, and availability [2]. For instance, information thefts try to hack the databases to obtain data over illegal authentication and authorization [3], which violates the data confidentiality. Besides, many attackers aim to tamper databases through unauthorized channels, which brings great damage to data integrity [4]. If the platform enforces a centralized database operation mechanism, attackers can make a successful attack only by compromising this centralized enforcement. Furthermore, the whole platform can be taken over and manipulated by attackers if the central agency has the privileges of administrative management, which is a serious threat to system integrity. Additionally, some attacks are launched to obstruct the platform by overwhelming the target central agency that provides services to the external, which violates the system availability.

Distributed ledger technology (DLT) provides a set of solutions with intrinsic capabilities of defending threats of a distributed database platform to replicating, synchronizing data and reaching consensus among multiple entities, which has become crucial in many ICT solutions since the rapid development of blockchain techniques alongside the rise of cryptocurrencies. Although DLT can be regarded as a kind of database with distinguishing features such as decentralization and immutability, it faces many critical issues while being applied to scenarios requiring efficiency such as the web service. One of the main issues is the high-cost and low-efficiency consensus algorithm in classic blockchain techniques such as proof-of-work (PoW) [5], proof-of-stake (PoS) as the improvement of PoW for public blockchain, practical Byzantine fault tolerance (PBFT) [6], Raft [7] for private blockchain. Hence, it is impractical to adopt the blockchain as a replacement for the traditional distributed database directly.

Recently, some researchers focus on another branch of DLT that is based on the directed acyclic graph (DAG) to enhance the efficiency and cost-effectiveness of consensus algorithms such as tangle [8], Hashgraph [9]. Furthermore, these DAG-based techniques are free from mining and incentive mechanisms to save unnecessary computing costs and improve a lot in scalability, which provides the possibility to construct a decentralized database platform with efficiency, cost-effectiveness and security benefits of DLT.

Problem statement.

- A distributed database platform must protect data from serious potential threats associating with vulnerabilities caused by centralized entities including the enforcement

This research was partially supported by KAKENHI (Grants-in-Aid for Scientific Research) (C) 19K11958 and (B) 19H04098.

and the agency. It is an arduous work to build such a platform with centralized entities.
- High efficiency is required. Besides security, the platform must ensure the performance at best efforts. It is a significant challenge to make a balance between security and performance.
- As an infrastructure, the functionality must be considered in the design of the platform such as auditability, resiliency, and interoperability.

Our contribution.

- We propose Dagbase, a decentralized database platform, which circumvents issues caused by vulnerabilities of centralized entities, especially integrity issues. Dagbase can adapt to untrustworthy environments on account of characteristics inheriting from DLT such as great fault-tolerance and tamper-proof capability.
- We enhance the efficiency and constrain the cost by designing the layered architecture with the feasible concurrency control and integrating the DAG-based consensus mechanism, which makes Dagbase have high performance and present linear scalability and great stability with the growth of the scale.
- We ensure the functionality of Dagbase by combining traditional database techniques and DLT. Dagbase is auditable and resilient while facing catastrophes. Furthermore, we ensure interoperability to support various database products seamlessly.

The remainder of this paper is organized as follows. Section II briefly introduces the related work. Section III presents our proposed scheme, followed by the implementation in Section IV. Section V shows the evaluation of our work. We make a discussion in Section VI. Section VII summarizes our work and concludes the paper.

II. Related Work

Although DLT has been widely applied to various fields both in academia and industry, very few of them focus on circumventing issues of a decentralized database platform by DLT.

There are numerous researches on assuring security and privacy of data storage and sharing using blockchain techniques in specific domains. For instance, a decentralized personal data management platform was proposed in [10], which focuses on protecting data privacy. For IoT applications, many researchers apply blockchain techniques to ensure data integrity such as [11] and [12]. Besides, the security of electronic health records as a notable issue in health informatics also benefits a lot from the application of the blockchain techniques such as [13]. These researches have addressed specific domain problems but none of them can be used as a feasible distributed database to provide general data services.

BigchainDB [14] is a solution to a decentralized system acting as a database with blockchain characteristics, which is complementary to both the traditional distributed database and the blockchain. The consensus mechanism of BigchainDB is based on Tendermint [15], a BFT consensus algorithm without mining. The efficiency was improved a lot compared to classic consensus algorithms in blockchain techniques. However, Tendermint can only be tuned in synchronous settings such a Byzantine eventual synchronous setting with $O(n^3)$ message complexity. With the increasing amount of nodes, the messages for exchange increase exponentially. Hence, it is still a notable problem while comparing to the practical distributed database.

One of the closest solutions to a distributed database platform was proposed in [16]. In this work, a decentralized database based on a layered blockchain-based architecture was implemented. The layered architecture consists of a fast first layer blockchain and a secure second layer blockchain. The first layer adopts a BFT consensus and a distributed hash table as the solution while the second layer is based on PoW consensus. The solution ensures strong data integrity but has many obvious defects including low efficiency, huge consensus cost, and delicate incentive mechanism.

III. Adversary Analysis

With the growth of the data value, the adversary can be very strong to threaten the security of the database platform by exploiting vulnerabilities with great resources. We mainly consider the abilities of the adversary without considering the concrete methods to reach the purpose.

1) The adversary can tamper all kinds of data in memory or on disk on any node. Logically, both data in the database and data for control are facing tampering threats.
2) The adversary can manipulate the behavior of any node. In this manner, the adversary can make the node behave in an unintended and impaired way.
3) The adversary can intercept and repackage any traffic between two nodes. The malicious traffic among nodes can be used to be against the normal nodes in the network.
4) The adversary can obstruct any node, which means the adversary is able to isolate any node from the network.

In Dagbase, all nodes are formally identical on account of the decentralized mechanism. Hence, it is reasonable to consider the equivalent difficulty of compromising each node in Dagbase for simplicity. However, we consider that the cost for adversaries to compromise the majority of nodes is higher than the value.

IV. Proposed Scheme

Firstly, we present the general scheme of Dagbase defining the participants and devices, which is an overview of the architecture. The component scheme illustrates the static structure of Dagbase from the internal view while the operation scheme presents a dynamic perspective.

A. General Scheme

The general scheme of Dagbase is shown in Fig. 1.

Participants of Dagbase include data, consumer and provider. Consumer is the user who interacts with data by

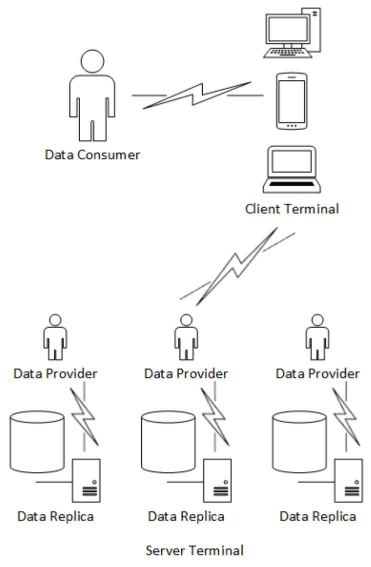

Fig. 1. General scheme of Dagbase.

conducting operations in a database. Provider provides services for consumers to facilitate the database interaction.

Devices of Dagbase can be divided into client terminals and server terminals. Client terminals are used by consumers to submit data operations. Server terminals, also called nodes, are controlled by providers to support a set of services including enforcing actual data operations from client terminals directly and making the response. Notably, the node is an abstract concept, which means a node can infer one or a set of physical or virtual machines.

B. Component Scheme

The component scheme of our work consists of three layers: persistence layer (PL), image layer (IL), and application layer (AL), which is shown in Fig. 2. PL stores data immutably by a DAG-based consensus algorithm. IL supports a traditional database that synchronizes with PL. AL provides a user interface and controls interactions between consumers and our platform. From the perspective of deployment, PL and IL are deployed on every node controlled by providers while AL is deployed on every client terminal accessed by consumers. The communication between IL and PL is local while remote communication is usually required between IL and AL.

1) Persistence Layer: This layer contains a DAG and acts as a distributed immutable data repository, which is deployed on every node. DAG is usually composed of a set of events marked as circles in Fig 2. Generally, an event contains the meta-information, one or more transactions, a timestamp, and two pointers. The meta-information is auxiliary for the consensus process. The timestamp identifies the created time point of the event. There are two pointers pointing to two different parents, which is adopted in many DAG-based consensus algorithms such as [9], [17] and [18].

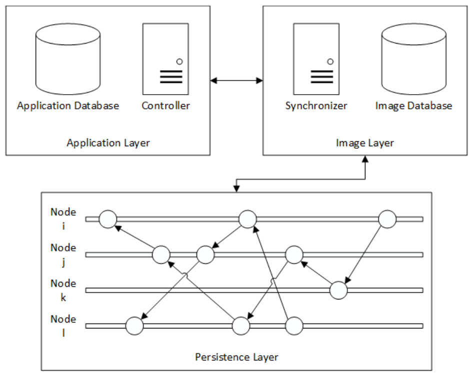

Fig. 2. Component scheme of Dagbase.

The transactions embedded in events are the information required the consensus among all nodes. It is notable that a transaction can also have a unique timestamp different from the event to identify the actual happening time point of the transaction. There are two kinds of transactions persisted in this layer: system transaction and database transaction. These two kinds of transactions are allowed to interleave persistence in this layer.

The system transaction can be regarded as the built-in transaction that is responsible for persisting system-level information. For instance, it is available to configure the system-level identity management mechanism in Dagbase by adding a type of system transaction to persist information of participants.

The database transaction contains database-oriented information such as database configuration and statements. For a database supporting SQL, the content in a transaction can be a statement like *CREATE DATABASE databasename*.

All nodes can reach consensus in a distributed environment by a DAG-based consensus algorithm integrated into PL. Nodes can make deterministic decisions on consensus by analyzing the local DAG.

2) Image Layer: This layer aims to ensure high performance for database operations by building and interacting with a traditional database, which is deployed with PL on every node. It is a middleware for making the swift response to requests from AL.

The image database acts as an image of the distributed ledger in PL and executes statements submitted from AL. Each image database keeps a consistent copy of database transactions in PL in a theoretical manner. In a practical situation, copies among image databases can be different. However, the inconsistency can be detected and eliminated to achieve final consistency by the DAG-based consensus mechanism.

The synchronizer is responsible for executing a set of programs, which encapsulates functions of interacting with PL for operations on transactions such as submitting, retrieving, processing transactions. It ensures that all executions are independent of the environment variables and can run in an unimpaired manner, which heavily depends on the implementation.

3) Application Layer: This layer is responsible for connecting consumers to Dagbase and interacting with IL to submit both system and database operations. AL does not need to and cannot access to PL directly. Identity information such as the private key of the consumer is stored in the application database to enforce identity verification with IL. Besides, AL also participates in consistency checking to facilitate the execution and synchronization of database operations.

For providers in Dagbase, AL acts as a content management system (CMS) to monitor and control the states of database services they are maintaining. The whole chain of service is traceable and immutable, which provides reliable management for providers.

C. Operation Scheme

We present four main procedures in the operation scheme including initialization procedure, reading procedure, writing procedure, and concurrency control procedure. Some built-in functions are encapsulated as the application programming interfaces (APIs) to facilitate the integration with database products and consensus algorithms.

1) Initialization Procedure: The initialization procedure is enforced on every node at the beginning, which is illustrated in Algorithm 1.

All nodes are set up with a set of the same initialization statements and the hash value of them in PL, which composes the genesis event by invoking the function *CreateEvent*. These initializing statements include statements for configuration and identity management such as a root account, which are executed in the image database of IL. Then all nodes must confirm the consistent initial world state S_{init} by validating the hash value of S_{init}. The confirmation procedure must pass the consistency checking with over the fraction of normal nodes. After making the confirmation, $\mathcal{G}_j$ is persisted in PL. If the confirmation fails, *initialization failure error* will be thrown out.

The initialization procedure is done when the majority of nodes have consistent initial information.

2) Writing Procedure: Assume a user with the identity u_i wants to execute a statement for writing data such as creating, updating or deleting a database, a table or data.

Firstly, a consistency checking procedure is triggered in AL, which sends the checking requests to a set of nodes to obtain the latest hash values stored in PLs of them. The consistency among these nodes can be determined by the comparison of the hash values. If these nodes are not consistent, then *Arbitration* function will be called to correct the consistency among them. Otherwise, the writing request is distributed to a node n_j selected from checked nodes. The writing procedure of node n_j is illustrated in Algorithm 2.

The statement associating with the latest hash value of the world state $\mathcal{H}(S_{AL})$ stored in AL, and the identity verification information u_i is submitted from AL to the node n_j. The synchronizer in IL can retrieve the latest hash value of the world state $\mathcal{H}(S_{PL})$ from PL and calculate a new hash value $\mathcal{H}(S_{IL})$ from the image database. A sub-procedure to match $\mathcal{H}(S_{AL})$ and $\mathcal{H}(S_{IL})$ with $\mathcal{H}(S_{PL})$ is launched.

If $\mathcal{H}(S_{AL})$ and $\mathcal{H}(S_{IL})$ exactly match with $\mathcal{H}(S_{PL})$, the authentication function will be invoked for access control. If the identity u_i passes the authentication validation, a new event will be created by the function *CreateEvent* to persist the writing statement and a new hash value $\mathcal{H}(S)$ based on the current world state S. Then, DAG $\mathcal{G}_j$ of n_j will be updated and sent for consensus amongst the other nodes. If the consensus succeeds, the writing statement will be executed in the image database and the result will be returned with the new hash value $\mathcal{H}(S)$. AL updates the application database with $\mathcal{H}(S)$.

If the matching fails, the function *RecoverDatabase* will be invoked asynchronously with *tampering detected error* thrown out. If the authentication fails, *authentication failure error* will be thrown out. If the consensus fails, *consensus failure error* will be thrown out. The errors prevent the execution of writing statements. It is also possible to identify which layer is compromised by comparing $\mathcal{H}(S_{AL})$ and $\mathcal{H}(S_{IL})$ with $\mathcal{H}(S_{PL})$ respectively.

3) Reading Procedure: Assume a user with the identity u_i wants to execute a statement for reading data such as showing or selecting the structure information or data. The same with the writing procedure, a consistency checking procedure is invoked. After successfully checking the consistency, the reading request is distributed to a node selected from checked nodes and the reading procedure is illustrated in Algorithm 3.

The statement associating with the latest hash value of the world state $\mathcal{H}(S_{AL})$ stored in AL and the identity verification information u_i are submitted from AL to IL. After retrieving $\mathcal{H}(S_{PL})$ and $\mathcal{H}(S_{IL})$, the matching sub-procedure starts.

If $\mathcal{H}(S_{AL})$ and $\mathcal{H}(S_{IL})$ exactly match with $\mathcal{H}(S_{PL})$, the identity verification procedure will be triggered. If the identity verification fails, *authentication failure error* will be thrown out.

If $\mathcal{H}(S_{AL})$ does not match with $\mathcal{H}(S_{PL})$, it is very possible that AL is compromised and *authentication failure error* will be thrown out.

If $\mathcal{H}(S_{IL})$ does not match with $\mathcal{H}(S_{PL})$, the function *RecoverDatabase* will be invoked asynchronously with *tampering detected error* thrown out. The procedure will end with executing the statement in the image database normally and returning the result if no errors are thrown out.

4) Concurrency Control Procedure: Dagbase can support one-user-one-node, multi-user-one-node, one-user-multi-node and multi-user-multi-node modes. One-node modes mean that one or more users can only access one node in the whole network invariantly. Multi-node modes mean that one or more users can select the arbitrary number of nodes for interactions freely.

A message queue MQ_{req} is used between AL and IL to record the requests containing statements of database operations submitted from AL. MQ_{req} is structured as a priority queue to always peek and consume the request with the latest timestamp. The peeking in MQ_{req} means the procedure that starts handling the latest request and marks the request as

Algorithm 1 Initialization Algorithm on Node n_i

```
Require: $(S_{init}, Size(N))$. $\exists s \in S_{init}$, where $s$ contains information of configuration and identity management. $Size(N)$ is
    the number of nodes. $\{S_{init}$ is a set of initial states.$\}$
Ensure: $(\mathcal{G}_i := \{E_0(n_i)\}, \mathcal{H}(S))$ or $error$
 1: $\mathcal{G}_i \leftarrow \emptyset$, $flag \leftarrow True$ $\{\mathcal{G}_i$ is the directed acyclic graph of node $i$.$\}$
 2: $E_0(n_i) \leftarrow$ call $CreateEvent(n_i)$ $\{E_0(n_i)$ is the first event of node $i$.$\}$
 3: $E_0(n_i) \leftarrow E_0(n_i) \uplus S_{init} \uplus \mathcal{H}(S_{init})$ $\{\mathcal{H}(S)$ is the hash value of $S$.$\}$
 4: randomly select an integer set $M$. $\{$For $\forall m \in M$, $0 < m \leq Size(N)$ and $m \neq i$. $Size(M) = \lfloor(1-\theta)Size(N)\rfloor + 1$,
    where $\theta$ is the threshold of the fraction of faulty nodes.$\}$
 5: repeat
 6:   $m \leftarrow Pop(M)$ $\{Pop(M)$ returns and removes an element of $M$.$\}$
 7:   if $E_0(n_i).\mathcal{H}(S_{init}) \neq E_0(n_m).\mathcal{H}(S_{init})$ then
 8:     $flag \leftarrow False$
 9:   end if
10: until $Size(M) = 0$
11: if $flag = True$ then
12:   $\mathcal{G}_i \leftarrow \mathcal{G}_i \cup E_0(n_i)$
13:
14:   return $(\mathcal{G}_i, \mathcal{H}(S_{init}))$
15: else
16:
17:   return $initialization\ failure\ error$
18: end if
```

Algorithm 2 Writing Algorithm on Node n_j

```
Require: $(u_i, statement, \mathcal{H}(S_{AL}))$ $\{u_i$ is the identity verification information of user $i$. $\mathcal{H}(S_{AL})$ is the latest hash value of
    the world state $S_{AL}$ stored in the application layer.$\}$
Ensure: $(result, \mathcal{H}(S))$ or $error$
 1: $\mathcal{H}(S_{PL}) \leftarrow$ call $RetrieveHPL()$
 2: $\mathcal{H}(S_{IL}) \leftarrow$ call $RetrieveHIL()$
 3: if $\mathcal{H}(S_{AL}) = \mathcal{H}(S_{PL})$ and $\mathcal{H}(S_{IL}) = \mathcal{H}(S_{PL})$ then
 4:   $V(u_i) \leftarrow$ call $Auth(u_i)$ $\{V(u_i)$ is the result of authentication of user $i$.$\}$
 5:   if $V(u_i) = True$ then
 6:     $E_k(n_j) \leftarrow$ call $CreateEvent(n_j)$
 7:     $\mathcal{H}(S) \leftarrow Rehash(S)$ $\{S$ is the latest world state with the update of a new statement.$\}$
 8:     $E_k(n_j) \leftarrow E_k(n_j) \uplus S \uplus \mathcal{H}(S)$
 9:     $\mathcal{G}_j \leftarrow \mathcal{G}_j \cup E_k(n_j)$
10:     $flag \leftarrow$ call $Consensus(\mathcal{G}_j)$ $\{Consensus(\mathcal{G})$ is the function to reach consensus on $\mathcal{G}$ amongst nodes.$\}$
11:     if $flag = True$ then
12:       $result \leftarrow Exec(statement)$
13:
14:       return $(result, \mathcal{H}(S))$
15:     else
16:
17:       return $consensus\ failure\ error$
18:     end if
19:   else
20:
21:     return $authentication\ failure\ error.$
22:   end if
23: else
24:   call async $RecoverDatabase()$
25:
26:   return $tampering\ detected\ error.$
27: end if
```

Algorithm 3 Reading Algorithm on One Node

```
Require: $(u_i, statement, \mathcal{H}(S_{AL}))$ {$u_i$ is the identity verification information of user $i$. $\mathcal{H}(S_{AL})$ is the latest hash value of the world state $S_{AL}$ stored in the application layer.}
Ensure: $result$ or $error$
1: $\mathcal{H}(S_{PL}) \leftarrow$ call $RetrieveHPL()$
2: $\mathcal{H}(S_{IL}) \leftarrow$ call $RetrieveHIL()$
3: if $\mathcal{H}(S_{AL}) = \mathcal{H}(S_{PL})$ and $\mathcal{H}(S_{IL}) = \mathcal{H}(S_{PL})$ then
4:   $V(u_i) \leftarrow$ call $Auth(u_i)$ {$V(u_i)$ is the result of authentication of user $i$.}
5:   if $V(u_i) = False$ then
6:
7:     return $authentication\ failure\ error.$
8:   end if
9: else
10:   if $\mathcal{H}(S_{AL}) \neq \mathcal{H}(S_{PL})$ then
11:
12:     return $authentication\ failure\ error.$
13:   end if
14:   if $\mathcal{H}(S_{IL}) \neq \mathcal{H}(S_{PL})$ then
15:     call async $RecoverDatabase()$
16:
17:     return $tampering\ detected\ error.$
18:   end if
19: end if
20: $result \leftarrow Exec(statement)$
21:
22: return $result$
```

Active. The consumption in MQ_{req} can be considered as the procedure that AL gets a corresponding response from IL with a finalized result r and pop the request out of MQ_{req}. If the statement embedded in the request is accepted by the network and executed in the image database successfully, $r = True$. Otherwise, $r = False$. There is a window W_{req} controls the rate of peeking, which improves the efficiency of handling requests by parallelization. In this manner, all requests from AL can get consumed eventually, which ensures fairness.

After parsing the request, the statement will be encapsulated as the transaction and put into another message queue MQ_{tx} that records the transaction sequence between the image database and the synchronizer. MQ_{tx} is also structured as a priority queue to align the transactions according to their timestamps. MQ_{tx} ensures that given a transaction containing a writing statement tx_j that is later than tx_i, tx_j will never get executed in the image database if tx_i is not consumed. The peeking in MQ_{tx} is the procedure that starts making the consensus of the latest transaction in the queue and marks the transaction as *Active*. The consumption in MQ_{tx} means the procedure that the transaction reaches consensus and the statement embedded in the transaction gets executed in the image database or the transaction is aborted. In the meanwhile, the transaction will be removed from MQ_{tx}. This can be regarded as a pessimistic lock for writing the database. For reading the database, it can be considered as a lock-free situation. Another window W_{tx} is used to control the number of parallel consensus processes by encapsulating multiple transactions into an event or multiple events separately.

Dagbase can ensure the order of transactions in different normal nodes. Formally, given two transactions tx_i^a and tx_j^a stored in node n_a with $\tau(tx_i^a) < \tau(tx_j^a)$ where $\tau(tx)$ denotes the timestamp of the transaction tx. Assume that there exists tx_k^b stored in node n_b with $\tau(tx_i^a) < \tau(tx_k^b) < \tau(tx_j^a)$ and tx_k^b starts the consensus before tx_i^a. In this situation, when the event containing tx_k^b reaches the node n_a or any other nodes participating the consensus of tx_i^a , the event can perceive that tx_i^a is in the message queue and $\tau(tx_i^a) < \tau(tx_k^b)$. A new event created in node n_a will carry this new information and return to node n_b eventually. When node n_b receives an event containing such an information, it will check the consensus status of tx_i^a. tx_k^b will get its turn when tx_i^a gets confirmed with the consensus and executed in the image database of node n_b. In other words, tx_k^b will get consumed after tx_i^a gets consumed in MQ_{tx} of node n_b.

V. Implementation

For the proof of concept, we present the implementation details of the prototype of Dagbase.

A. Database and World State

The application database in AL is a lightweight database LevelDB to store the identity information and the hash value of the current world state. We integrated the latest version of MySQL as the image database in IL. The world state is a set of MySQL statements associating with writing operations such as *CREATE*, *ALTER*, *INSERT INTO*, *UPDATE*, *DELETE*. We select MD5 as the hashing algorithm that hashes the concatenation of the new statement and the old world state. The asymmetric cryptography is used for identity verification, which is well supported by authentication plugins of MySQL.

B. DAG-based Consensus

We implemented a DAG-based consensus algorithm according to [9]. The transaction is regarded as an event that is created by calling the function *CreateEvent*. Every event has a self-parent pointer and an other-parent pointer, which constitutes a DAG. The function *Consensus* is to share the newly created event amongst all nodes by a gossip-about-gossip protocol. Every non-faulty node can know the same DAG and reach consensus efficiently. The consensus is asynchronous BFT and does not need both mining and incentive mechanisms.

C. Consistency Checking

Before the writing and reading procedures, consistency checking is imperative to ensure the correctness of each operation. In Dagbase, AL must collect checking responses from at least 5 arbitrary nodes to determine whether there exists inconsistency caused by faults or malicious behaviors.

If the inconsistency is detected, *Arbitration* function will be invoked to fix the inconsistency among these nodes. We implemented this function in AL to find a group of nodes with consistent hash values. This group has a higher trust level than the inconsistent group. Then *RecoverDatabase* function will be called on nodes in the inconsistent group.

D. Database Recovery

When the function *RecoverDatabase* is triggered, the synchronizer of IL will start rolling back the MySQL statements from the latest executed statement according to the logs. Every time after rolling back one statement, new hash value will be calculated and compared with the latest hash value in PL. Once a matching pair is found, the tampered point will be detected, and the database will recover successfully by executing statements persisted in PL from the tampered point to the latest.

This function is invoked asynchronously during the writing procedure and reading procedure when the tampering is detected. Hence, the recovery procedure will not affect the overall performance very much because the operation from AL can be migrated to other normal nodes.

E. Database Reconstruction

Dagbase can reconstruct the database from any world state at any time. When IL receives the reconstruction instruction, the function *Reconstruction* will be invoked to drop the database and execute the statements persisted in PL until some world state.

F. Communication

A routing table is shared among these three layers for peer discovery. The communications among AL and IL are secured by the transport layer security (TLS) protocol. IL and PL communicate with each other in the local environment of a node.

AL is set to select the nearest nodes with the highest trust level as the master nodes to interact with ILs on them. The trust level is determined by the frequency of inconsistency. A node has a high trust level with a low frequency of inconsistency and unavailability.

G. Deployment

AL is encapsulated as a software development kit (SDK), which can be integrated with the client applications developed by Java or JavaScript. A user interface was developed that integrates the SDK to connect to Dagbase. We use the containerization techniques for the scalable and consistent deployment of IL and PL, which makes the deployment of nodes very efficient.

VI. Evaluation

A. Security Analysis

1) Assumption: To analyze the security of Dagbase, we make some assumptions firstly.

a) Reachability:

$$\forall n_i, n_j \in N_{normal}, \Delta t_{n_i \to n_j} \leq \Delta t \tag{1}$$

N_{normal} denotes the set of normal nodes. $\Delta t_{n_i \to n_j}$ is the time interval for node n_i to reach n_j. Δt is a negligible time interval compared to processing procedures on a node.

We can also determine whether a node is not normal by *Reachability*:

$$n_a \in N_{normal}, \Delta t_{n_a \to n_b} > \Delta t \Rightarrow n_b \notin N_{normal} \tag{2}$$

b) Honesty:

$$\frac{|N_{faulty}|}{|N|} \leq \theta \tag{3}$$

N_{faulty} denotes the set of faulty nodes while $N = N_{normal} \cup N_{faulty}$. θ is the threshold according to the DAG-based consensus algorithm. In the prototype of Dagbase, $\theta \leq \frac{1}{3}$ is required to ensure BFT property.

The network composed by nodes presents macroscopic honesty.

2) Property:

a) Strong consistency: Dagbase ensures strong consistency by detecting the inconsistency with the probability greater than 99% during all procedures, which means that the probability of missing detecting the inconsistency is less than 1%.

In the initialization procedure, Dagbase requires checking the hash values of initial world states from $(\lfloor (1-\theta)|N| \rfloor + 1)$ nodes. According to the *Honesty* assumption, it is enough to initialize the node with consistent information.

For the data writing and data reading procedures, the consistency checking is sensitive enough to identify the inconsistency. Consistency checking procedure requires that AL collects hash values of the current world state persisted in PLs from at least 5 nodes. The probability of failing detecting the inconsistency by getting the untrustworthy responses from f faulty nodes can be calculated by (4).

$$\prod_{i=0}^{f-1} \frac{(|N_{faulty}| - i)}{(|N| - i)} \tag{4}$$

Consider the worst case that the fraction of faulty nodes reaches the threshold θ, which is the extreme situation the DAG-based consensus algorithm can endure. The case can be described by (5) according to (3).

$$\frac{|N_{faulty}|}{|N|} \to \theta \tag{5}$$

When $|N|$ is small with $|N| \in [4, 15)$, the number of faulty nodes $f < 5$. In this case, the probability of the detection failure is 0. In other words, any inconsistency can be detected successfully.

When $|N| \geq 15$, we can obtain (6) easily.

$$\frac{(|N_{faulty}| - i)}{(|N| - i)} \leq \frac{|N_{faulty}|}{|N|}, i \in [0, f) \tag{6}$$

With (4), (5) and (6), we can obtain (7).

$$\begin{gathered} \prod_{i=0}^{f-1} \frac{(|N_{faulty}| - i)}{(|N| - i)} < \prod_{i=0}^{f-1} \frac{|N_{faulty}|}{|N|} \\ \Longrightarrow \prod_{i=0}^{f-1} \frac{(|N_{faulty}| - i)}{(|N| - i)} < \theta^f \end{gathered} \tag{7}$$

In our default case, $f = 5$ and $\theta = \frac{1}{3}$. Even if the fraction of faulty nodes reaches the extreme situation, the probability of the detection failure is still less than 1% according to (7).

b) Conditional partition tolerance: Dagbase ensures weak partition tolerance by the DAG-based consensus mechanism which means that Dagbase is allowed to lose connections among a conditional number of nodes. Hence, Dagbase presents weak partition tolerance if normal nodes can compose a complete graph with normal nodes N_{normal} as vertices and effective connections $\mathcal{E}$ as edges. The number of edges $|E|$ of a complete graph with $|V|$ vertices can be calculated by (8).

$$|E| = \frac{|V|(|V| - 1)}{2} \tag{8}$$

Therefore, the number of the effective connections $|\mathcal{E}|$ must satisfy (9) to ensure the composition of the complete graph according to (8).

$$|\mathcal{E}|_{min} \geq \frac{|N_{normal}|(|N_{normal}| - 1)}{2} \tag{9}$$

According to (2) in *Reachability*, nodes cannot be reached are also marked as faulty nodes. Normal nodes can reach consensus and ensure the consistency and availability so long as *Honesty* holds, which means the fraction of partitioned nodes together with other faulty nodes is less than the threshold θ and a complete graph composed by normal nodes can be ensured.

c) Strong integrity: Dagbase can assure the strong system and data integrity by inheriting decentralization and immutability from DLT. The platform is decentralized, attackers must control most of the nodes to conduct data tampering over the platform. As for the database system, Dagbase integrates a DAG-based consensus mechanism, which makes the platform BFT. Even if attackers tamper the database systems on some nodes, the overall data integrity can be still satisfied. Therefore, all events persisted in PL cannot be tampered or destructed without violating *Honesty* assumption.

d) Availability: Dagbase adopts a decentralized management mechanism. Consumers obtain their services by interacting with the nearest available nodes with the highest trust level. If some nodes are crashed by attackers or internal exceptions, consumers can still obtain services by interacting with available nodes. The cost of disrupting the whole platform is huge due to the requirement of disrupting over θ nodes.

e) Concurrency correctness: Dagbase can ensure the concurrency correctness in one-node and multi-node modes. Assume there are three transactions tx_i^a, tx_k^b, tx_j^a satisfying $\tau(tx_i^a) < \tau(tx_k^b) < \tau(tx_j^a)$. Suppose a special case that transactions can be right in the order of timestamps in one node but appear to be the wrong order of timestamps in terms of the whole network when node n_a does not participate in the consensus process of tx_k^b and node n_b processes neither tx_i^a nor tx_j^a. But according to *Honesty* assumption, there must exist a node n_c that processes both tx_k^b and tx_i^a. Hence, the network can still perceive and arrange the right order eventually.

B. Performance Analysis

A practical distributed database requires high efficiency to ensure the support of some request-intensive scenarios such as the demands of web services. Especially, query usually occupies most requests. Dagbase can make the response as efficiently as a native database cluster. Besides, the cost of ensuring security properties is low compared to classic consortium blockchain techniques.

1) Theoretical Analysis: Dagbase can ensure the near-native data reading by the layered architecture. The time cost of reading data is nearly the same as the execution time on the database product deployed in IL. The response time of reading t_r can be factored as $t_r = t_{cc} + t_{hm} + t_{exec}$, where t_{cc}, t_{hm} and t_{exec} denote the time cost of consistency checking, hash value matching and execution respectively. According to the *Reachability* assumption, $t_{cc} << t_{exec}$. The hash value matching only needs negligible time cost for the judgement, which indicates $t_{hm} << t_{exec}$. Therefore, we can obtain that $t_r \approx t_{exec}$.

High-efficiency data writing is ensured by efficient DAG-based consensus mechanism. The response time of writing t_w

can be factored as $t_w = t_{cc} + t_{hm} + t_{ce} + t_{rehash} + t_{cons} + t_{exec}$, where t_{ce}, t_{rehash}, t_{cons} denote the time cost of *CreateEvent*, *Rehash* and *Consensus* respectively. *Consensus* and execution are the most time-consuming sub-procedures that heavily depends on the DAG-based consensus mechanism and the database product deployed in IL. Therefore, $t_w \approx t_{cons} + t_{exec}$.

Different from the classic consensus algorithms in blockchain techniques, we introduce DAG-based consensus algorithms in Dagbase that are more efficient and cost-effective [18]. Without the mining mechanism, we constrain the cost by eliminating the waste of computing resources.

2) Experiment: We conducted experiments to test the real performance of the prototype of Dagbase. For the proof of concept, all nodes are deployed on a powerful physical server (Intel Xeon Phi CPU 7250, 96GB) to make the communication cost negligible. Besides, we developed a MySQL cluster for comparison. MySQL services encapsulated in containers are deployed on the server with the same number and initial statements as the nodes of Dabase. A synchronizer was implemented as the central agency to ensure consistency among these MySQL services by checking states of all services during each operation.

We used iMac as the client terminal deployed with the automatic test program that submits requests to Dagbase and the MySQL cluster. The test program is integrated with the SDK to interact with nodes of Dagbase and the driver to interact with MySQL services of the cluster. The throughput is calculated by the time cost of 1000 operations including reading and writing respectively on each node or MySQL service. A SQL statement is regarded as a transaction. Hence, the throughput is measured in transactions per second (TPS). The result is shown in Fig. 3.

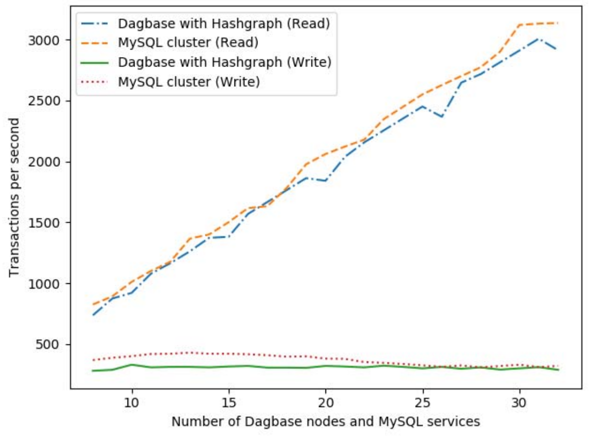

Fig. 3. TPS of writing and reading of Dagbase compared with the MySQL cluster.

From the result, we can obtain that the reading TPS of Dagbase and the cluster are almost the same. The major time cost is the actual execution time on the database. With the increase in the number of services, the cost of consistency checking of the synchronizer becomes higher and higher. In the meanwhile, the consistency checking mechanism of Dagbase is stable and is not affected by the number of nodes.

We also tested the TPS performance of Dagase with different consensus algorithms. Another version of Dagbase with PBFT was implemented for comparison. The result is shown in Fig. 4.

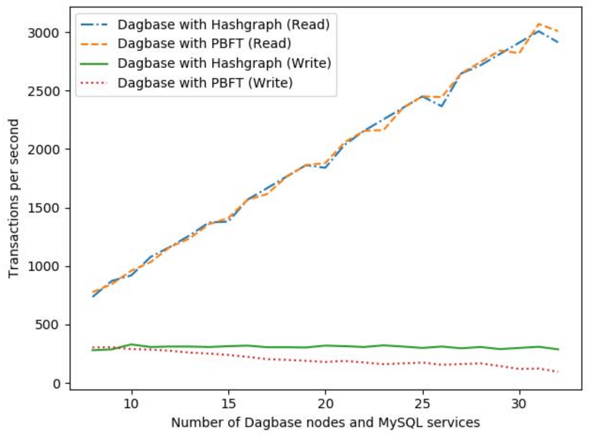

Fig. 4. TPS of writing and reading of Dagbase with Hashgraph compared with the PBFT version.

With the same architecture, the major difference lies in the writing procedure. We can obtain that the PBFT version faces a serious scalability issue while Hashgraph presents good scalability.

C. Functionality Analysis

1) Definition:

a) Consensus order: The consensus order sorts the timestamps of moments that transactions are accepted by the consensus network, which may be different from the actual order of submission timestamps of transactions.

b) Path: The path $\mathcal{P}$ in PL is a finite and world-state sequence $S_{init} S_1 \ldots S_{latest}$ starting with the initial world state S_{init}, ending with the latest world state S_{latest}, and organized by the consensus order.

c) Subpath: A subpath $\hat{\mathcal{P}}$ is a subsequence $S_i S_{i+1} \ldots S_j$ of the path $\mathcal{P}$.

d) Trace: A trace of a subpath $\hat{\mathcal{P}} = S_i S_{i+1} \ldots S_j$ is defined as $\mathcal{T}(\hat{\mathcal{P}}) = \mathcal{H}(S_i)\mathcal{H}(S_{i+1}) \ldots \mathcal{H}(S_j)$.

2) Property:

a) Auditability: Given any trace $\mathcal{T}(\hat{\mathcal{P}})$, an authorized identity can obtain corresponding subpath $\hat{\mathcal{P}}$.

Authorized identities can examine all writing statements submitted by them by interpreting traces persisted in PL because the world states in $\mathcal{P}$ contain all statements as transactions. The authorized identity is authenticated by Dagbase as a consumer or a provider. Furthermore, these records are tamper-proof on account of *strong data integrity* guaranteed by DLT.

b) Resiliency: $\forall S \in \mathcal{S}$, it is always true that $Post(S) = \{S' \in \mathcal{S} \mid S \longrightarrow S'\}$ and $Pre(S) = \{S' \in \mathcal{S} \mid S' \longrightarrow S\}$ are available. $\mathcal{S}$ denotes the universal set of world states. $Post(S)$ denotes the set of successors of a world state S while $Pre(S)$ denotes the set of predecessors of a world state S.

Dagbase can roll back the database to any world state while suffering catastrophes, which is ensured by *strong consistency* and *strong integrity*. The default setting is to roll back the database to the latest known world state and all predecessors of this world state can be recovered. The only boundary is that the fraction of faulty nodes needs to be less than the threshold required by the consensus mechanism.

c) Interoperability: Transactions in world states can contain statements from different database products including both SQL and NoSQL databases.

Providers can choose what kind of database products they want to deploy on their nodes. The only coupling of Dagbase and the database product is the log reading and API calling. Therefore, Dagbase can integrate all kinds of mainstream database products theoretically including SQL databases and NoSQL databases.

Furthermore, it does not matter whether the database product supports distributed features. The architecture of Dagbase can support standalone database products owing to the built-in concurrency control mechanism. In this manner, Dagbase can distribute the databases and decentralize the management of database products.

VII. Discussion

Although we have proposed and implemented Dagbase combining beneficial features from the traditional distributed database and DLT, there are still some issues.

Confidentiality is not specially addressed in Dagbase. In a practical database platform, unauthorized disclosure can not be neglected.

The data integrity is ensured by the DAG-based consensus algorithm which requires the fraction of honest nodes is greater than the threshold. If the number of faulty nodes is over the threshold, Dagbase can be compromised.

The current implementation of the consensus mechanism in Dagbase still has some flaws because the swirlds hashgraph has been protected by patents. We are implementing some new DAG-based consensus algorithms such as [18]. The main procedures in Dagbase are decoupled with the consensus mechanism, which ensures the flexibility to alternate better consensus solutions.

The performance of writing still needs to be optimized while facing high-frequency writing scenarios. For now, Dagbase supports the asynchronous writing procedure by writing into the local node synchronously and making consensus asynchronously. In this manner, $t_w \approx t_{exec}$ though dirty buffers may occur when multiple nodes write at the same time. The final order of consensus is still deterministic. Hence, dirty buffers can be consumed automatically after the consensus.

It is also significant to implement a sharding mechanism and control the replications of data according to safety levels. These extensions are promising to improve space utilization.

VIII. Conclusion

We have proposed Dagbase, a novel distributed database platform possessing decentralized characteristics of DLT and high efficiency with great functionality. Dagbase has been implemented for the proof of concept and tested to show the real performance. We still have some work to do such as optimizing the architecture, implementing a better DAG-based consensus algorithm and extending the functionality. It is very promising to make Dagbase a sophisticated decentralized database platform.

References

[1] E. Bertino and R. Sandhu, "Database security-concepts, approaches, and challenges," *IEEE Transactions on Dependable and secure computing*, no. 1, pp. 2–19, 2005.

[2] I. Basharat, F. Azam, and A. W. Muzaffar, "Database security and encryption: A survey study," *International Journal of Computer Applications*, vol. 47, no. 12, 2012.

[3] W. G. Halfond, J. Viegas, and A. Orso, "A classification of SQL-injection attacks and countermeasures," in *Proceedings of the IEEE International Symposium on Secure Software Engineering*, vol. 1. IEEE, 2006, pp. 13–15.

[4] C. Anley, "Advanced SQL injection in SQL server applications," 2002.

[5] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Annual International Cryptology Conference*. Springer, 1992, pp. 139–147.

[6] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[7] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX-ATC 14)*, 2014, pp. 305–319.

[8] S. Popov, "The tangle," *cit. on*, p. 131, 2016.

[9] L. Baird, "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance," *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep.*, 2016.

[10] G. Zyskind and O. Nathan, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*. IEEE, 2015, pp. 180–184.

[11] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, "Blockchain based data integrity service framework for IoT data," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 468–475.

[12] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards blockchain-based auditable storage and sharing of IoT data," in *Proceedings of the 2017 on Cloud Computing Security Workshop*. ACM, 2017, pp. 45–50.

[13] N. Nchinda, A. Cameron, K. Retzepi, and A. Lippman, "MedRec: A Network for Personal Information Distribution," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 637–641.

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

[15] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, 2016.

[16] L. Aniello, R. Baldoni, E. Gaetani, F. Lombardi, A. Margheri, and V. Sassone, "A prototype evaluation of a tamper-resistant high performance blockchain-based transaction log for a distributed database," in *2017 13th European Dependable Computing Conference (EDCC)*. IEEE, 2017, pp. 151–154.

[17] T.-Y. Chen, W.-N. Huang, P.-C. Kuo, H. Chung, and T.-W. Chao, "DEXON: A Highly Scalable, Decentralized DAG-Based Consensus Algorithm," *arXiv preprint arXiv:1811.07525*, 2018.

[18] F. Xiang, W. Huaimin, S. Peichang, O. Xue, and Z. Xunhui, "Jointgraph: A DAG-based efficient consensus algorithm for consortium blockchains," *Software: Practice and Experience*, 2019.