# GriDB: Scaling Blockchain Database via Sharding and Off-Chain Cross-Shard Mechanism

Zicong Hong
Hong Kong Polytechnic University
zicong.hong@connect.polyu.hk

Song Guo
Hong Kong Polytechnic University,
PolyU Shenzhen Research Institute
song.guo@polyu.edu.hk

Enyuan Zhou
Hong Kong Polytechnic University
21038299r@connect.polyu.hk

Wuhui Chen
Sun Yat-sen University
chenwuh@mail.sysu.edu.cn

Huawei Huang
Sun Yat-sen University
huanghw28@mail.sysu.edu.cn

Albert Zomaya
The University of Sydney
albert.zomaya@sydney.edu.au

## ABSTRACT

Blockchain databases have attracted widespread attention but suffer from poor scalability due to underlying non-scalable blockchains. While blockchain sharding is necessary for a scalable blockchain database, it poses a new challenge named *on-chain cross-shard database services*. Each cross-shard database service (e.g., cross-shard queries or inter-shard load balancing) involves massive cross-shard data exchanges, while the existing cross-shard mechanisms need to process each cross-shard data exchange via the consensus of all nodes in the related shards (i.e., on-chain) to resist a Byzantine environment of blockchain, which eliminates sharding benefits.

To tackle the challenge, this paper presents GriDB, the first scalable blockchain database, by designing a novel *off-chain cross-shard mechanism* for efficient cross-shard database services. Borrowing the idea of off-chain payments, GriDB delegates massive cross-shard data exchange to a few nodes, each of which is randomly picked from a different shard. Considering the Byzantine environment, the untrusted delegates cooperate to generate succinct proof for cross-shard data exchanges, while the consensus is only responsible for the low-cost proof verification. However, different from payments, the database services' verification has more requirements (e.g., completeness, correctness, freshness, and availability); thus, we introduce several new *authenticated data structures* (ADS). Particularly, we utilize consensus to extend the threat model and reduce the complexity of traditional accumulator-based ADS for verifiable cross-shard queries with a rich set of relational operators. Moreover, we study the necessity of inter-shard load balancing for a scalable blockchain database and design an off-chain and live approach for both efficiency and availability during balancing. An evaluation of our prototype shows the performance of GriDB in terms of scalability in workloads with queries and updates.

**PVLDB Reference Format:**
Zicong Hong, Song Guo, Enyuan Zhou, Wuhui Chen, Huawei Huang, and Albert Zomaya. GriDB: Scaling Blockchain Database via Sharding and Off-Chain Cross-Shard Mechanism. PVLDB, 16(7): 1685 - 1698, 2023.
doi:10.14778/3587136.3587143


Proceedings of the VLDB Endowment, Vol. 16, No. 7 ISSN 2150-8097.
doi:10.14778/3587136.3587143

## 1 INTRODUCTION

Characterized by trustworthiness, transparency, and traceability, blockchain technologies have been integrated into many areas, such as cryptocurrency [37], supply chain [24], international trade [13], etc. In database management, blockchain technologies have attracted considerable interest in upgrading traditional databases to blockchain-empowered distributed databases [48], which forms an emerging research direction namely *blockchain databases*.

Compared with traditional distributed databases, blockchain databases transact and record data via blockchains and construct an abstract database layer supporting various query functionalities on top of blockchains, which endow the distributed databases with immutability and traceability [8, 14, 40, 41, 56, 59, 62]. For example, BlockchainDB provides shared tables as easy-to-use abstractions as well as a key/value interface to read/write data stored in the blockchain [8]. Pei *et al.* introduces a Merkle Semantic Trie-based index to support semantic query, range query and fuzzy query on the blockchain [40]. SEBDB adds relational data semantics into blockchain storage and thus supports SQL query [62] and FalconDB presents a blockchain database with SQL query with time window [41].

Due to the underlying non-scale-out blockchains, most existing blockchain databases suffer from poor scalability. For example, schemes in [41, 62] adopt Tendermint which achieves throughput of about 1000 transactions per second (TPS) but its network scale is less than 100. Schemes in [40, 59] adopt Ethereum aiming to support thousands of participants but only have tens of TPS. The poor scalability makes the blockchain databases hardly meet the quality of service required in large-scale business in practice.

Sharding is one of the most promising technologies for the blockchain scalability [5, 20, 28, 30, 42, 54, 57]. It divides the nodes into small groups called *shards*, which can handle transactions in parallel and alleviate the storage overhead for each node. In such an approach, the transaction throughput scales linearly with the number of nodes. To develop a scalable blockchain database, this paper is going to construct an abstract database layer on a sharding blockchain by distributing database data and the corresponding task of storing, querying, or updating to different blockchain shards. However, such a sharding for database storage and workload introduces a new requirement namely *cross-shard database services*, i.e., *data aggregation* for query and *workload balancing* for management.

As shown in Figure 1a, the data aggregation is caused by the sharding for storage. Particularly, the data related to a query may be stored by the blockchain nodes from multiple shards; thus, the

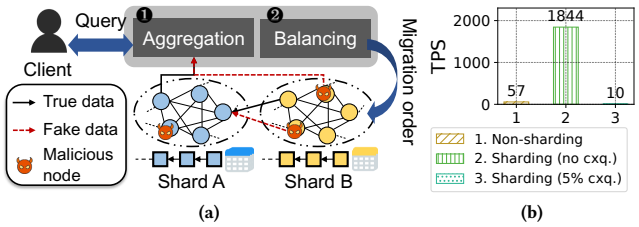



**Figure 1: (a) Illustration for sharding blockchain database, which requires two new functions, i.e., data aggregation for query and workload balancing for management. (b) Transaction throughput of non-sharding and sharding blockchain databases. (cxq. represents cross-shard query.)**

query requires the involvement of several shards. For example, if there are two tables stored in Shard A and B, respectively, then a SQL `JOIN` query combining these two tables involves both shards. Moreover, the workload balancing is caused by the sharding for workload. Particularly, due to the sharding, each shard is only responsible for the workload of query and update to its storage. The demand imbalanced and dynamic nature of applications results in workload imbalance among shards, which significantly degrades the performance of sharding blockchain database.

The Byzantine environment of blockchain databases makes the technologies of traditional distributed databases no longer applicable. Malicious nodes may collude with each other and violate the protocol in arbitrary manners. For example, in distributed databases [6], a query can be easily realized by requesting from one database node in every related shard. However, in a sharding blockchain database, the correctness, completeness, and freshness of cross-shard queries can hardly be guaranteed when accidentally requesting from a malicious node. Moreover, elastic workload balancing is the first-class feature in modern databases [50] achieved by load migration. However, unlike one-to-one crash-tolerant migration in most distributed databases, the sharding blockchain database requires a many-to-many migration across shards in which malicious nodes can intercept, tamper or forge the migrating table.

To resist the Byzantine faults for cross-shard database services, an intuitive idea is to process them through the cross-shard mechanism of blockchain sharding. In detail, the core of cross-shard database services is to transfer tables among shards despite Byzantine failures. The cross-shard mechanism guarantees that each data transfer (e.g., money transfer in the conventional blockchain) is agreed by the majority of honest nodes in all its related shards. Transferring a table among shards through the mechanism can guarantee that the table transfer will not be compromised (detailed in § 2.1 and § 5.1). However, such an idea is costly. On the one hand, each query or migration involves a massive set of semantics-related data (e.g., rows belonging to a table) in a blockchain database. On the other hand, all existing cross-shard mechanisms are on-chain (i.e., requiring the consensus of all the related shards). Therefore, all nodes in the related shards need to participate in the consensus on numerous data. As proved in Figure 1b, the existing on-chain cross-shard mechanisms cannot support even 5% cross-shard queries in a sharding blockchain database with 32 shards (detailed in § 8).

To this end, this paper focuses on relational blockchain database and proposes the first relational sharding blockchain database, named GriDB. In comparison with the previous blockchain databases, GriDB guarantees high scalability while providing support for relational database services in blockchain sharding. Motivated by the idea of off-chain payments and verifiable computing, GriDB enables an off-chain execution of the cross-shard database services by adopting *authentication data structure* (ADS) to delegate cross-shard communication-intensive tasks to a few nodes in a verifiable manner. We summarize our contributions as follows.

- GriDB introduces relational data semantics and query functionality into blockchain transactions to abstract a sharding blockchain as a distributed relational database. The clients can send requests to any untrusted blockchain nodes for storing, manipulating and retrieving data.
- To provide a query layer of abstraction on sharded data, we design a cooperative delegation-based approach with a constant complexity of data transfer among shards without sacrificing security. It delegates the tasks of data aggregation to a few nodes in different shards and constructs a succinct proof used to on-chain verify.
- To meet the dynamically skewed workloads and achieve inter-shard balancing, we propose an off-chain live migration that migrates the database service among shards with security, low cost, and minimum interruption.
- We develop a prototype for GriDB and conduct a comprehensive evaluation. The result shows that GriDB achieves a scalable throughput for SQL linearly increasing with the shard number compared with the non-sharding works.

## 2 PRELIMINARIES

### 2.1 Blockchain Sharding

Blockchain sharding achieves scalability by dividing the blockchain nodes into multiple shards, each of which is responsible for receiving, validating, and processing part of transactions. A blockchain sharding scheme is generally composed of *shard formation*, *intra-shard consensus* and *cross-shard mechanism* as follows.

1) At the beginning, each node is assigned to a shard. For example, in Elastico [30], a node generates an identity by solving a Proof-of-Work (PoW) puzzle to avoid Sybil attack and is assigned to a shard with an ID equal the last several bits of the identity.

2) In each round, the nodes within a shard run an intra-shard consensus to agree on the same block consisting of valid transactions. For example, some systems [28, 30, 57] adopt a Byzantine fault tolerant (BFT) protocol (e.g., PBFT [4] and collective signing BFT [27]) as intra-shard consensus to resist malicious nodes. An intra-shard consensus should satisfy *safety*, i.e., the honest nodes agree on the same valid block in each round, and *liveness*, i.e., the block in each round will eventually be committed or aborted.

3) Due to the sharding, some transactions may involve the state of more than one shard and thus are called *cross-shard* transactions. The core idea of most of the existing cross-shard mechanisms is to divide each cross-shard transaction into several sub-transactions, each of which is processed by a shard. Then, the shards handle them with the guarantee of ACID, i.e., atomicity, consistency, isolation, and durability, for every cross-shard transaction. In other words, if a

cross-shard transaction is committed by the cross-shard mechanism successfully, it means that the majority of honest nodes in every related shard receive the same transaction and agree that it is valid.

## 2.2 Authenticated data structures

**Verifiable set operation (VSO).** VSO [3, 39] is an ADS which enables clients to outsource set computation tasks (e.g., intersection and union) to an untrusted server. The server returns the result with an ADS, depending on which, one can verify the result without downloading the original data and re-calculating by itself. VSO consists of the following probabilistic polynomial-time algorithms:

- $(sk, pk) \leftarrow \text{genKey}(1^\lambda)$: Let $\lambda$ denote the security parameter. Based on a bilinear-map accumulator primitive [38], it begins by choosing randomly a value $s \in \mathbb{Z}_p$ and letting $sk = s$ be the secret key and $pk = (g^s, \cdots, g^{s^q})$ be the public key, where $g$ is the generator of a cycle multiplicative group $\mathbb{G}$ and $q$ is an upper-bound on the cardinality of sets.
- $acc(X) \leftarrow \text{setup}(X)$: For a set $X \subset \mathbb{Z}_p$, it computes the accumulation value of $X$ by $acc(X) = g^{\prod_{x \in X}(x+s)}$.
- $(X^*, \pi) \leftarrow \text{prove}(X^i, X^j, pk)$: Given two sets $X^i$ and $X^j$, it returns the intersection (or union) result $X^*$ and a proof $\pi$.
- $\{\text{accept}, \text{reject}\} \leftarrow \text{verify}(acc(X^i), acc(X^j), X^*, \pi, pk)$: On input the accumulation values $acc(X^i)$, $acc(X^j)$, a proof $\pi$, and the public key $pk$, it return *accept* if and only if $X^i \cup X^j = X^*$ for intersection (or $X^i \cap X^j = X^*$ for union).

The unforgeability for VSO has been proved to be held under the q-SBDH assumption [2] in bilinear groups. For more details about the proof, refer to [3, 39]. Additionally, the case of the intersection (or union) for an arbitrary number of sets is similar.

**Merkle tree.** Merkle tree [32] is an ADS which enables clients to verify the correctness of committed transactions. Its leaf nodes are composed of hash values of transactions, and non-leaf nodes are generated upward through hash operations until the root named *Merkle root* is generated. Each block header stores a Merkle root for its contained transactions; thus, the clients only need to synchronize the headers to verify all committed transactions. A blockchain node can provide the clients with any transaction and a Merkle proof consisting of the siblings path of the transaction in the tree. The clients can validate the transaction by constructing a Merkle root based on the proof and comparing it with the locally stored one.

# 3 SYSTEM MODEL

**System Components.** In GriDB, there are two types of entities:

1) The *database clients* are the service consumers of GriDB. They neither participate in the consensus nor store the whole content of blockchains locally because they are often lightweight devices.

2) The *blockchain nodes* are the consensus participants for the blockchain and are divided into a number of shards. Each node is responsible for verifying, processing and storing transactions of its located shard. GriDB is a layer-2 database framework constructed on top of existing blockchain sharding systems, and it is *sharding-agnostic*, which means the underlying system can adopt any sharding schemes (including shard formation, intra-shard consensus and cross-shard mechanism) from [22, 28, 54, 57]. However, the underlying blockchain sharding system's intra-shard consensus

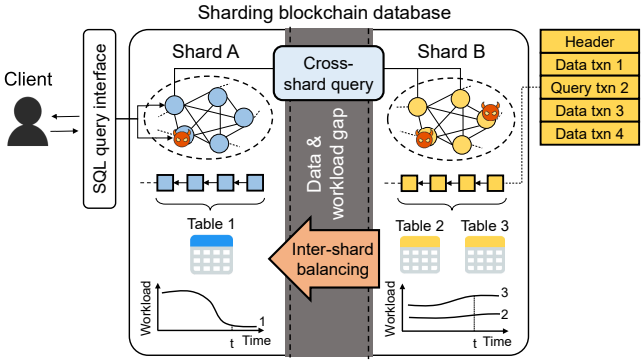



**Figure 2: System overview for GriDB.**

should satisfy both safety and liveness, and its cross-shard mechanism guarantees the ACID of each cross-shard transaction (see § 2.1). We emphasize that the cross-shard mechanism of the underlying blockchain sharding system is one of the important components of the off-chain cross-shard mechanism of GriDB, which will be described in the following sections. To avoid confusion, we will call the former *on-chain cross-shard mechanism*.

GriDB considers an outsourced database scenario [35, 58] in which the clients outsource their data management to the blockchain. The nodes host the client's databases and the clients send requests to the nodes to create, store, update and query their databases.

**Threat Model.** The threat model of GriDB relies on that of the underlying blockchain sharding composed of two kinds of blockchain nodes: *honest* and *malicious*. The honest nodes abide by all protocols in GriDB while malicious nodes may collude with each other and violate the protocols in arbitrary manners, such as denial of service, or tampering, forgery and interception of messages. Although there are malicious nodes in the shards, the sharding blockchains [28, 54, 57] can guarantee that each shard is trusted with high probability, i.e., the result published by any shards is trusted. Different from [8], GriDB does not require a strong assumption that each client trusts the nodes it connect.

**Transaction Model.** The requests of clients are processed in the form of blockchain transactions that are divided into two types: *data* and *query transaction*. The first one is used to update (such as insert, update, and delete) the database state and the second one is used to query the database state. Besides, in § 5.3, there are some *control transactions* used to support the database management such as database migration. The type division will not affect the compatibility for the underlying blockchain because there is a "data" field in the transactions of most blockchains, and GriDB places different data in the field for different types of transactions. The details of transactions will be described as follows.

# 4 GRIDB OVERVIEW

## 4.1 System Overview

As shown in Figure 2, GriDB considers a distributed relational database storing a number of tables. For scalability, the workload of each table is distributed to a shard (A fine-grained sharding for blockchain database through table partition will be described in

§ 6.4.) If a client decides to query or update a table, it can issue requests to any nodes in the shard responsible for the table. As described in the threat model, the nodes receiving the requests may be malicious, thus they are required to return a proof used for authentication. To generate a proof, based on the request received, a node first proposes a transaction which can be one of the following.

1) A data transaction is used to manipulate (such as insert, update, and delete) the data and includes an `INSERT`, `DELETE` or `UPDATE` SQL statement. In GriDB, the data is in the form of a relational model. The same type of data has a unified semantic description as a schema composed of several attributes. An insert statement inserts new data based on explicitly specified values or from the existing data via a nested subquery. Since blockchain is append-only, a delete statement is implemented by marking old data as invalid, i.e., cannot be queried, and an update one is implemented by a sequence of delete and insert operations to overwrite. Based on the data transactions recorded in the blockchain, each node in a shard maintains a tamper-proof copy of a relational database.

2) A query transaction is used to record query results to clients. A query statement begins with a keyword `SELECT` followed by a subset of column names, and then a keyword `FROM` followed by a table (or a `JOIN` sub-clause used to combine multiple tables in a later section.) Following these, a `WHERE` clause is followed by a sequence of predicates connected by logical operators (e.g., `AND`, `OR`, `NOT`) that restrict the rows used when computing the output. After processing the request, the node can put the query statement and result into the data field of a query transaction. To avoid occupying too much on-chain storage, the query results can be offloaded to an off-chain storage and the query transactions only store the hash of query results, according to which the clients can download the correct results from the off-chain storage based one the hash.

Next, the node broadcasts the proposed transaction to the network. If a data transaction is committed to the blockchain, the majority of honest nodes accept and execute the data transaction, which means the database state has been successfully updated. If a query transaction is committed, the majority of honest nodes agree on the query results. Finally, the client can authenticate the result returned by its connected node by validating if the transaction for its request is committed. This transaction validation for clients has been implemented in most blockchains, such as Simplified Payment Verification (SPV) in Bitcoin and Ethereum.

In addition to a Merkle tree storing transactions like traditional blockchains, to enable every node or client to know every table's location, every node maintains an additional Merkle tree. The tree stores the location of all tables in the form of *table name-shard id* pairs. It is updated in each epoch according to a global cross-shard control transaction including a new planning strategy for the following epoch (refer to § 6.2). Thus, depending on the tree, every node or client can find the correct shard storing the target table.

## 4.2 Challenges

Dividing the tables across different shards improves the blockchain database's scalability. However, it is not enough for a scalable blockchain database due to two following problems.

**Problem 1 (cross-shard query):** A client's query involving tables only in a single shard can be served by one shard because each node of the shard can validate and agree on the query result in an intra-shard consensus. However, a request to query the tables from different shards cannot be completed by a single shard and requires the cooperation of multiple shards. For example, in Figure 2, a query joining on Table 1 and 2 involves the data of Shard A and B thus cannot be completed by only one of them.

**Problem 2 (inter-shard workload imbalance):** It is hard to guarantee that the workload of every table is the same and static, thus some shards can be overloaded while some others remain idle. For example, in Figure 2, Shard A is responsible for Table 1 and Shard B is responsible for Table 2 and 3. At the beginning, the total workload in Shard A and B is similar. However, if the workload of Table 1 drops over time, Shard A becomes idle. To fully utilize the throughput, dynamically migrating the workload among shards and alleviating the effect of hotspots are crucial.

# 5 SYSTEM DESIGN

## 5.1 Strawman

For the two challenges above, we first describe a strawman sharding blockchain database only based on the on-chain cross-shard mechanism of the existing sharding systems as follows.

Consider a cross-shard query involving two tables, i.e., Table 1 and 2, located in Shard A and B, respectively. For such a cross-shard query, we present a *shard-cooperation approach* based on the on-chain cross-shard mechanism. Shard A first commits many cross-shard data transactions involving Shard A and B, including the data of Table 1 via the cross-shard mechanism. As described in § 2.1, the mechanism guarantees the transactions can be committed in Shard A and B. Then, Shard B can get Table 1 from the transactions, compute the query result, and commit a query transaction with the query result. However, when there are many cross-shard queries, the table transfer among shards will be frequent, resulting in system overloaded and network blocked.

For the inter-shard load balancing, the latest version of the table should be transferred from the source shard to the destination shard. If there is data being left out, the completeness of queries on the table cannot be guaranteed after migration. Thus, we present a *stop-restart migration approach* based on the on-chain cross-shard mechanism as follows. Shard A first stops processing new transactions for the table via an intra-shard consensus, which avoids the migrating table being modified during migration. Then, Shard A commits many cross-shard data transactions to reconstruct the latest version of the table in Shard B. After all transactions are committed, Shard A commits a cross-shard transaction to mark the end of the migration, and Shard B can restart the service of the table. However, when the migrating table is enormous, the approach incurs a high penalty due to the prolonged service interruption for the migrating table and the influence on other tables' throughput.

To solve these drawbacks, we introduce two key designs for our off-chain cross-shard mechanism in § 5.2 and § 5.3.

## 5.2 Cross-Shard Query Authentication

**Motivation.** Although the above shard-cooperation approach is safe, it is expensive since the table transfer among shards for each cross-shard query is between groups of nodes (to guarantee that there is a majority of honest nodes for each shard participating).

An intuitive idea to avoid this overhead is to pick one delegate from each shard. Then, for each cross-shard query, a delegate downloads the related tables from the other delegates and evaluates the query results. However, any malicious delegate can easily tamper with the query result by providing a fake or out-of-date table.

We aim to design an ADS to allow the delegates to prove the validity of cross-shard query results. The existing outsourced databases have designed some ADS for SQL [35, 58]. For example, for a `JOIN` query involving the same column of two tables, a node treats the columns of these two tables as two sets and constructs a proof for their intersection through VSO [61]. However, the existing ADS for SQL cannot be applied in our cross-shard query due to two difficulties. First, different from the outsourced database in which there is no sharding and a server stores the whole data copy, any delegate in GriDB only stores the tables of its located shard and downloads tables from the other untrusted delegates and thus cannot construct a valid proof by itself. Second, to support arbitrary verifiable SQL queries, besides VSO, the other outsourced databases need to adopt interval trees [61] or zero-knowledge proof [60], which costs tens of minutes for a query [60] due to high computation complexity.

Thus, we propose a *delegation-based approach* by integrating VSO with the intra-shard consensus to implement an efficient and secure ADS for arbitrary SQL query in GriDB. Its main idea is to divide each query into some algebra operators with different input data. Particularly, it validates the operators involving multiple shards' data through VSO and those involving single shard's data through the intra-shard consensus. The cross-shard query validity can finally be proved through a chain of trust, i.e., proving the validity of every operator from beginning to end. Such a manner makes the best use of the advantage of both the shard-cooperation approach (i.e., low computation) and the existing ADS for verifiable SQL (i.e., low communication) and bypasses their disadvantage.

**Design.** The overall cross-shard query procedure is given in Algorithm 1. For each cross-shard query, we identify the related shard of the table following `FROM` as *main shard* and the shards of the tables following `JOIN` as *sub shards*. A client can issue a cross-shard query request to any nodes in the main shard. Next, one node is chosen from each related shard (Line 2), which can be round-robin or randomly by a verifiable random function [15]. The malicious or low-response delegates can be replaced by a view change similar to PBFT. The delegated node in the main shard is called the *main node* denoted by $\mathbb{M}$ and those in the sub shards are called *sub nodes* denoted by $\mathbb{S}$. The main node downloads each involved table for the query from the sub node in the corresponding sub shards (Line 3). After downloading all involved tables, the main node evaluates the query result and generates a proof (Lines 4, 8-14).

To generate the proof, the main node first translates each SQL query into a relational algebra tree composed of algebra operators [45], e.g., the right part of Figure 3. Each node in the tree denotes a unary (or binary) algebra operator taking one (or two) inputs, applying a function, and outputting its result to the next operator. The edges represent data flow from bottom to top.

In the tree, we identify join (or union) operators involving tables in different shards as *cross-shard operators* and the others as *intra-shard operators*. Each intra-shard operator can be processed by the nodes of the corresponding shard based on their stored tables. In comparison, each cross-shard operator involves the data gap among shards, thus the main node needs to generate a proof. The proof is composed of the accumulation values (refer to § 2.2) of the corresponding columns in the tables to be joined or unioned, a VSO proof, and a position indicator for the intermediate result (or final result). The position indicator is a bitmap to indicate which rows are chosen in a table. For example, considering the SQL statement in Figure 3, we denote the oid columns of these two tables after processing the selection operators as $C^i$ and $C^j$, respectively. The generated proof $\Upsilon$ is $\langle acc(C^i), acc(C^j), \pi, [1,0,1], [1,0,0]\rangle$. The position indicators $[1,0,1]$ and $[1,0,0]$ mean that the first and third rows in Table 1 and the first row in Table 2 are chosen, respectively. A cross-shard query may include multiple cross-shard operators thus the main node will produce a list of proofs $\Upsilon$, each of which is for a cross-shard operator.

After the query result and the corresponding proof are generated, the main node proposes a cross-shard query transaction, including the result and the proofs and involving the related shards (Line 5). Then, to validate the cross-shard query, each related shard runs an intra-shard consensus on the transaction by evaluating each algebra operator for their stored tables and verifying the proofs related to the tables of the shard (Lines 15-20). For example, as shown in Figure 3-❸, during the consensus on the cross-shard query transaction, each node can validate and execute intra-shard operators based on the local data and validate and execute cross-shard operators based on the VSO proof. During the validation, they can optimistically assume that the accumulation values related to the other shards' tables are valid. Finally, if the cross-shard query transaction passes the validation of every related shard (Line 6), it

**Algorithm 1:** Cross-Shard Query Authentication

**Input:** query request $Q$ involving tables in a set of shards $S$
**Output:** query result $R$, verification object $VO$

1 Delegates $\mathbb{M}$ and $\mathbb{S}$ are selected from $S$
2 $\mathbb{M}$ downloads the related tables from $\mathbb{S}$
3 $\mathbb{M}$ evaluates query result $R$ and get proof $\Upsilon$ via genProof($Q$)
4 $\mathbb{M}$ proposes a cross-shard query transaction $txn$ involving $S$ and including $R$ and $\Upsilon$
5 **if** validateCx($S, txn$) == *True* **then**
6 $\quad VO \leftarrow$ the list of SPV proofs in $S$ for $txn$
7 **Function** genProof($Q$):
8 $\quad$ **for** cross-shard operator $op \in Q$ **do**
9 $\quad\quad$ Set $C^i$ and $C^j$ as the columns involved by $op$ and $pk$ as the public key
10 $\quad\quad$ $(C^*, \pi) \leftarrow \text{prove}(C^i, C^j, pk)$
11 $\quad\quad$ Get $bm^i$ and $bm^j$ based on $C^i$, $C^j$ and $C^*$
12 $\quad\quad$ Add $\langle acc(C^i), acc(C^j), \pi, bm^i, bm^j\rangle$ to $\Upsilon$
13 $\quad$ return $\Upsilon$
14 **Function** validateCx($S, txn$):
15 $\quad$ **for** shard $s \in S$ **do**
16 $\quad\quad$ **if** $\Upsilon$ *or* $R$ is invalid **then**
17 $\quad\quad\quad$ return *False*
18 $\quad\quad$ $txn$ is committed in the blockchain of $s$
19 $\quad$ return $True$

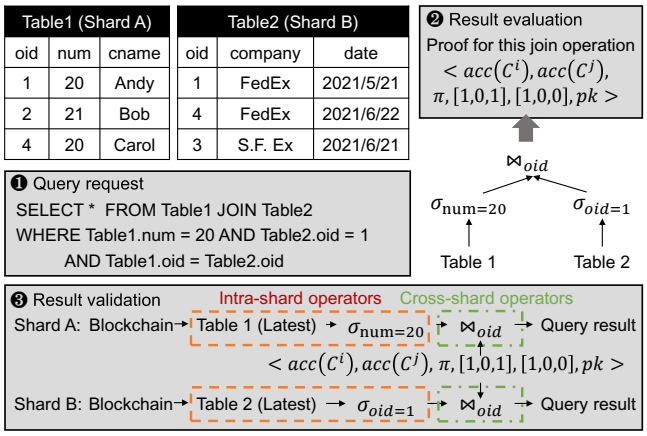



**Figure 3: Example for ADS proof generation of two tables distributed in Shard A and B, respectively. ($\sigma$ is an operator to select rows from a relation and $\bowtie$ is an operator to join tables based on a specified column.)**

will be committed in the blockchains of all related shards and the client can accept the query result included in the transaction via SPV (Line 7). Besides, if a malicious main node sends different copies of a transaction to shards, the client can detect the inconsistency by checking the Merkle proofs of the transaction via SPV (Line 7).

**Security Analysis.** The analysis relies on the intra-shard consensus of blockchain sharding thus we define $v$ as the fault threshold [34] of the adopted blockchain sharding in GriDB. For example, Rapidchain [57] tolerates up to $v = 1/2$ Byzantine faults, while the asynchronous or Omniledger [28] tolerates only up to $v = 1/3$ Byzantine faults. Next, we describe the formal definition [60, 61] of our cross-shard query's security as follows.

Definition 1. *A query is secure if any polynomial-time adversary's success probability is negligible in the following experiment:*

*For a query $q$, the adversary is picked as main node or sub node for the generation of query transaction including result $R$. The adversary succeeds if the query transaction is committed in all related shards and one of the following results is true: 1) $R$ includes a row which does not satisfy $q$ (**correctness**); 2) There exist a row which is not in $R$ but satisfies $q$ (**completeness**); 3) $R$ includes a row not from the latest tables generated by all the committed data transactions (**freshness**).*

Theorem 1. *Our proposed cross-shard query mechanism satisfies the security property as defined in Definition 1 if the proportion of malicious nodes in each shard is no more than the fault threshold $v$.*

Proof. We prove Theorem 1 in three cases, corresponding to how GriDB defends against the three different adversaries in Definition 1 for each cross-shard query in correctness, completeness, and freshness. We first describe the three cases: Case 1: This case means a tampered or fake row within the result is returned, which does not satisfy the query $q$. In this case, the tampered or fake row can pass the client's verification under the correctness in Definition 1. Case 2: This case means a row that satisfies $q$ is missing from $R$. In this case, the incomplete result can pass the verification of the client under the completeness in Definition 1. Case 3: This case means the result $R$ involves an old row that satisfies $q$ but is not from the latest tables. In this case, the old result can pass the client's verification under the freshness in Definition 1.

If any of the above three cases occur, it means the computation of at least one relational operator (intra-shard or cross-shard operator) for a committed query is invalid, i.e., the malicious nodes in a related shard tamper with the executing of intra-shard operators during intra-shard consensus, or the main node generates a wrong result in the executing of cross-shard operators. However, this contradicts two assumptions. The first one is that when the proportion of malicious nodes in each shard is no more than the fault threshold $v$, the safety of the intra-shard consensus holds [53]. Second, according to the unforgeability of VSO under the q-SDH assumption [3, 39], the ADS for set operations guarantees that the computation of each cross-shard operator in delegates is valid, and any invalid results can be detected by the intra-shard consensus. □

**Performance Analysis.** We analyze the time for a cross-shard query involving $m$ cross-shard operators as follows. Three steps occupy most of the time, i.e., the table transfer (Line 3), the proof generation (Line 4), and the confirmation latency of the query transaction (Line 6), which is also proved in § 8. Thus, the analysis is developed around these three steps. For the table transfer, the time cost is linear to the size of the related tables. We introduce several refinements to reduce this time cost and improve query efficiency in § 6.1. Next, according to [3, 39], the proof generation time for each set operation involving $N$ elements is $O(N \log^2 N \log\log N)$. Thus, the proof generation time is $O(mN \log^2 N \log\log N)$. Finally, the confirmation latency denotes the delay between the time that the query transaction is issued from the main node until the transaction is committed, which depends on the throughput, demand, and number of block confirmations of the blockchain.

## 5.3 Inter-Shard Load Balancing

**Motivation.** Observe that the drawback of the stop-restart approach in the strawman system results from the interruption for transaction processing during migration. Moreover, because the approach is on-chain, the migration occupies the transaction throughput of the shards involved, which interrupts the new transactions of the other tables. Thus, to avoid these drawbacks, we design an *off-chain live migration* approach for GriDB. Its main idea is to design an off-chain technique to minimize the number of on-chain transactions and a dual mode with cross-shard synchronization and concurrency control to minimize the impact of interruption to the migrating table during migration.

**Design.** Figure 4 illustrates the timeline of cross-shard migration and the messages exchanged between two shards. The life cycle of a table includes the following three modes.

*1) Normal Mode:* The normal mode for a table (called $\mathcal{T}$) is the period in which the data or query transactions of the table are processed normally by the shard it belongs to. The normal mode accounts for most of the time for the table.

*2) Init Mode:* When $\mathcal{T}$ is going to be migrated from the source shard (called $\mathcal{S}$) to the destination shard (called $\mathcal{D}$), the init mode starts. (We will discuss the trigger of table migration in § 6.2 which aims for load balancing and guarantees that there is a super majority of honest nodes in $\mathcal{S}$ knowing $\mathcal{T}$ and $\mathcal{D}$.) The nodes in $\mathcal{S}$ first

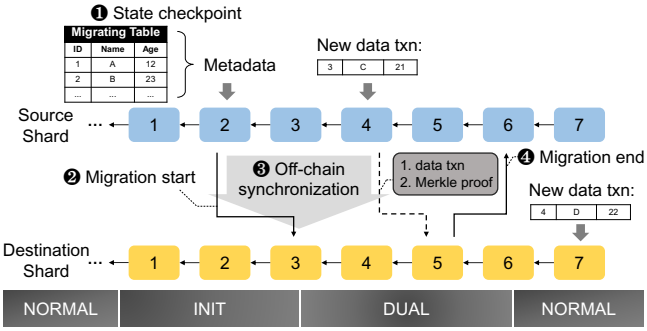



**Figure 4: Overview of off-chain live migration. A solid line with arrowhead represents a cross-shard transaction and a dotted line with arrowhead represents an off-chain cross-shard communication.**

construct the metadata for $\mathcal{T}$ via a hash function such as SHA (Figure 4-❶) and commit a cross-shard control transaction involving $\mathcal{S}$ and $\mathcal{D}$ (Figure 4-❷). The transaction includes the metadata and a block number, representing a checkpoint for $\mathcal{T}$ in this block number. When the control transaction is committed in both shards by the on-chain cross-shard mechanism, the init mode ends.

*3) Dual Mode:* In the dual mode, $\mathcal{S}$ begins to transmit $\mathcal{T}$ to $\mathcal{D}$. The transmission among shards is pluggable and can be implemented by one-to-one communication or gossip mechanisms (Figure 4-❸). The nodes in $\mathcal{D}$ only accept the table matching the metadata in the control transaction. Because the download of the whole table may cost a lot of time, we adopt a pre-copy scheme in which the nodes in $\mathcal{D}$ can pre-download $\mathcal{T}$ from $\mathcal{S}$ in the normal or init mode and validate it after the commitment of the control transaction.

To keep the service for $\mathcal{T}$ during the dual mode, $\mathcal{S}$ continues to process the newcoming data and query transactions related to $\mathcal{T}$. The new data transactions in $\mathcal{S}$ may change the content of $\mathcal{T}$, thus $\mathcal{D}$ should be notified. It can be realized by committing all new data transactions as cross-shard transactions, however, which slows the service of $\mathcal{T}$ due to the overhead of cross-shard mechanism and blocks the throughput of $\mathcal{S}$ and $\mathcal{D}$ when the demand is high. Thus, we adopt an off-chain cross-shard notification mechanism based on Merkle tree as follows. First, in GriDB, similar to the other sharding systems [22, 54], each node will be a light node for the other shards and store the block headers of all shards. It does not hurt the scalability, since the light nodes do not need to participate in the consensus and each header occupies little storage space and bandwidth. The notification is in the form of an off-chain message including a data transaction and its Merkle proof. Any nodes in $\mathcal{S}$ can notify $\mathcal{D}$ via gossip mechanism [25]. Based on the notification received, $\mathcal{D}$ gets the latest data transactions for $\mathcal{T}$. For example, as shown in Figure 4, a new data transaction for the migrating table arrives in $\mathcal{S}$ and is committed at the 4-th block. Any honest node in $\mathcal{S}$ can send the new transaction with its Merkle proof in the 4-th block to $\mathcal{D}$ for synchronization of $\mathcal{T}$ between $\mathcal{S}$ and $\mathcal{D}$.

After a node in $\mathcal{D}$ completes downloading, it proposes a cross-shard control transaction involving $\mathcal{D}$ and $\mathcal{S}$ or participates in the consensus on the one proposed by another node to show that it has downloaded the table successfully. Thus, the transaction can be committed if the majority of honest nodes in $\mathcal{D}$ confirm that they have downloaded the migrating table (Figure 4-❹). We assume that the off-chain notification arrives reliably and without latency here, which will be discussed later. Finally, the migration is completed and $\mathcal{D}$ has full ownership of the migrating table. It means that the later transactions (e.g., data/query transactions and migration requests) for the migrating table are processed by $\mathcal{D}$ only.

**Asynchronous Issues.** In the above, we ignore some problems resulting from the network latency or malicious nodes. Thus, we discuss them and provide some designs as follows.

*Problem 1:* In the init mode, due to the transaction latency existing in the blockchain, i.e., the delay between the time that a node sends a transaction to the network until the time that the transaction can be confirmed by all (honest) nodes, the metadata generated by different nodes may be in different versions. Thus, $\mathcal{S}$ may be unable to reach a consensus on the same control transaction. To synchronize the metadata among nodes in $\mathcal{S}$, GriDB sets a rule as follows. When a node begins to generate the metadata, it stops processing any new data transactions of $\mathcal{T}$ and disagree on blocks including these transactions during consensus until the init mode ends. Note that a node still accepts the newly committed blocks and updates its local database state and the corresponding metadata even if it disagrees them. Moreover, before a cross-shard control transaction is committed successfully, the nodes in $\mathcal{S}$ keep updating the metadata they generate based on the new block. If there is already the same control transaction proposed by other nodes waiting to be committed, the node can participate in its consensus.

*Problem 2:* In the dual mode, we adopt an off-chain notification mechanism to minimize the impact of interruption during migration. However, the off-chain communication among shards is not reliable thus the notifications may get lost. For the problem, we adopt the following designs. First, every new data transaction in the dual mode will be assigned an increasing sequence number before being committed in $\mathcal{S}$. Thus, if a node in $\mathcal{D}$ finds itself missing some transactions, it can directly request the corresponding notifications from the nodes in $\mathcal{S}$. Second, after the control transaction is committed (Figure 4-❹), $\mathcal{S}$ needs to commit a control transaction including the total number of new data transactions in the dual mode and sends the control transaction with a Merkle proof to $\mathcal{D}$. Besides, the nodes in $\mathcal{D}$ can actively ask the control transaction. Each node in $\mathcal{D}$ begins to process new transactions for $\mathcal{T}$ until it gets the total number of notifications and downloads all data transactions. Finally, $\mathcal{D}$ continues the service of $\mathcal{T}$ when the majority of honest nodes in $\mathcal{D}$ finish downloading.

**Security Analysis.** We first describe the formal definition of the security [10] for our off-chain live migration as follows.

Definition 2. *A migration is secure if achieving safety and liveness despite Byzantine failure. The safety requires serializable isolation, i.e., the migrating table's transactions run in serial order during migration, and durability, i.e., the committed transactions will not get lost after migration. The liveness indicates it eventually terminates.*

Theorem 2. *Our proposed off-chain live migration satisfies the security property as defined in Definition 2 if the proportion of malicious nodes in each shard is no more than the fault threshold $v$.*

Proof. During the migration, only one of $\mathcal{S}$ and $\mathcal{D}$ has full ownership and processes transactions for the migrating table. The intra-shard consensus guarantees that there is a serializable order for transactions among nodes in each shard, thus achieving serializable isolation. For durability, in the init mode, because the honest nodes update their metadata before the control transaction is committed, $\mathcal{D}$ can download all data for $\mathcal{T}$ that are committed before the dual mode begins. The durability for transactions that are committed in the dual mode can be guaranteed by the final control transaction, including their total number in $\mathcal{S}$. Furthermore, the liveness can be guaranteed since the end conditions for each mode depend on the commitment of cross-shard transactions whose liveness has been shown to hold under the threat model of super-majority honest in each shard in any sharding works [53]. □

If the malicious nodes in $\mathcal{S}$ construct a wrong metadata for $\mathcal{T}$, the intra-shard consensus guarantees that the invalid transaction including the wrong metadata would not be committed. The nodes in $\mathcal{D}$ can detect wrong tables and wrong notifications according to the on-chain metadata and Merkle root, respectively.

**Performance Analysis.** The time for each migration is at least the latency of two cross-shard control transactions, one of which denotes the beginning of dual mode and the other the end. In parallel with the first one, $\mathcal{T}$ is transferred between shards. Its time depends on the adopted communication methods, network status, and network scale. After the second one, to guarantee that all data transactions are received by $\mathcal{D}$, there is a control transaction in $\mathcal{S}$ and the nodes in $\mathcal{D}$ need to download the data transactions that they miss. The time of the step also depends on the network environment. In the worst case, if all data transactions during migration are missed by the majority of honest nodes, the nodes may need some time to download the transactions they missed. Besides, the service of $\mathcal{T}$ may be halted for a time due to the first and third designs for the asynchronous issues during migration.

# 6 DESIGN REFINEMENT

## 6.1 Cross-shard Query Efficiency

Although the delegation-based approach in § 5.2 reduces the complexity for table transfer in a cross-shard query from $O(SN^2)$ (derived from the strawman in § 5.1) to $O(S)$ where $N$ is the number of nodes in a shard and $S$ is the number of related shards for the query, transferring a huge table from the sub nodes to the main node still costs a lot. However, many rows are useless in practice. For example, for query #5 in § 8.2 involving tables with millions of rows, the size of its final result only have single-digit items. GriDB optimizes the transferring of table among delegates as follows.

We first optimize by applying the unary operators and binary operators involving tables in the same shard early in the tree of each cross-shard query. Particularly, each sub node processes all selection operators related to its table and transfers the processed table to the main node. For example, in Figure 3, the selection operation $\sigma_{num} = 1$ is moved to the bottom of the tree and processed by the sub node in Shard A, reducing the size of Table 1 to be transferred to the main node in Shard B. Because the execution order of the tree is bottom-up, the main node in Shard B downloads the part of tables including these temporary outputs, based on which it can continue to process the next operation. Moreover, some projection operators also can be applied early similar to the selection operators.

Next, we adopt bloom filter (BF) for each cross-shard operator to filter out unnecessary data before transferring tables. BF [46] is a space-efficient probabilistic data structure used to test whether an element belongs to a set or not. In GriDB, before downloading the tables for a cross-shard operator, the main node can build a BF for the target column in its table and send the filter to the sub nodes. The sub nodes use it to filter their own table before transferring tables to the main node. Thus, most useless data are filtered out before being transmitted, reducing communication overhead.

## 6.2 Load Balancing Scheduler

A critical problem to achieve inter-shard balancing is how to generate a good planning strategy to distribute the load to shards and how to apply the strategy in a distributed and safe manner in GriDB.

For a planning strategy, similar to distributed databases [9, 55], GriDB follows a widely-used greedy planning algorithm [50]. It iterates through the list of tables, starting with the one with the hottest demand. If the shard currently holding this table has a load exceeding the average demand, the algorithm migrates the table to the least load shard. The algorithm is easy to implement and has been proved to be efficient in many database scenarios [9, 50, 55].

To run the above algorithm in a decentralized manner, GriDB extends it by considering its execution in the existing sharding blockchains. Particularly, resharding phase is an important phase during the lifecycle of sharding blockchains [28, 30, 57]. A sharding blockchain proceeds in epochs, where each epoch consists of a resharding phase followed by multiple intra-consensus rounds. During the resharding phase of an epoch, a shard will be elected as the reference shard based on a round-robin rule. In GriDB, the reference shard can act as the load balancing scheduler in the resharding phase. The leader of every shard computes the demand of each table and reports it to the reference shard in a cross-shard control transaction via the cross-shard mechanism. After receiving the demand of every table, the leader of the reference shard proposes a cross-shard control transaction, involving all shards and including a new planning strategy in the following epoch, via the cross-shard mechanism. The cross-shard mechanism can guarantee that the new planning strategy is known by a majority of honest nodes in every shard. Based on this strategy, the tables can be migrated among shards using the approach in § 5.3.

**Security Discussion.** The greedy planning algorithm and the table demand computing can be deterministic, thus any node can check the validity of their results. This guarantees that only the cross-shard control transactions, including valid table demand or valid planning strategy, can be committed in all the related shards and the invalid ones will be aborted via the cross-shard mechanism.

## 6.3 Cross-shard Insertion/Deletion/Update.

In GriDB, a data transaction can include a SQL statement for an insert, delete or update operation with nested subqueries or a multiple-table delete/update operation [36]. If the nested subquery is cross-shard in the first case or the related tables belong to multiple shards in the second case, the data transaction involves multiple shards.

For the first case, a data transaction including the cross-shard subquery result (or its hash) can be processed by the delegation-based approach in § 5.2 and committed as a cross-shard transaction. The transaction involves both the shard for the inserted/deleted/updated table and the related shards for the nested subquery. For the second case, a multi-table deletion/update can be considered as deleting/updating the specified rows in multiple tables based on a query to these related tables. Thus, it can also be processed as a cross-shard data transaction including query results similar to the first case. Finally, because the cross-shard mechanism guarantees the atomicity of cross-shard transactions (see § 2.1), the cross-shard insertion, deletion, and update take effect in all related shards. Any invalid cross-shard data transactions (e.g., including wrong subquery results) will be aborted by the cross-shard mechanism.

### 6.4 Horizontal/Vertical Table Partition

For each table, besides storing the entire table in one shard as discussed in § 4.1, GriDB can be developed into a fine-grained sharding blockchain database through *horizontally* or *vertically* partitioning the table into partitions. The former allows the table to be partitioned into disjoint sets of rows and the latter disjoint sets of columns. Load balancing can benefit from this fine-grained sharding for blockchain database since the database workload can be more evenly distributed to the blockchain shards.

The partitions of each table are distributed to different shards, thus a table is stored in multiple shards. For a horizontally partitioned table, each query needs to commit the same query transaction to all the related shards of the query. For a vertically partitioned table, each of its partitions can be regarded as an individual table. If a query involves the columns within a partition or the partitions related to the same shard, the query involves one shard and can be processed as a query transaction in the shard. However, if the query involves multiple columns of several partitions from different shards, it needs to be committed as a cross-shard query transaction.

## 7 DISCUSSION

### 7.1 Permissioned and Permissionless Setting

GriDB can be applied in both permissioned and permissionless scenarios, relying on the underlying blockchain sharding system. For a permissioned scenario, only a set of known, identified, but untrusted nodes can serve as blockchain nodes similar to the permissioned blockchain databases [8, 41]. For a permissionless scenario, the blockchain database is public and open, and anyone can become a blockchain node without a specific identity. To resist Sybil attacks caused by the permissionless setting, GriDB can use a PoW-based identity generation as described in § 2.1, which is widely adopted by the permissionless blockchain sharding [28, 57]. Moreover, to compensate for the consensus overhead of blockchain nodes and avoid the Verifier Dilemma [31], GriDB will explicitly charge fee for each transaction and reward the blockchain nodes [47]. We leave an incentive mechanism design for GriDB as our future work.

### 7.2 General Join

The cross-shard query authentication in § 5.2 works for equality join, because the cryptography primitive adopted in GriDB supports set intersection only. For a general join case such as non-equijoin (i.e., join operation using comparison operator like >, <, >=, <= with conditions), we can resort to cryptographic technologies with more general verifiable computing capacity, e.g., Trusted Execution Environment (TEE) and Succinct Arguments of Knowledge (SNARK), which will be left as our future works.

## 8 EXPERIMENTAL EVALUATION

**Implementation.** We implement a prototype of GriDB in Go [16] based on Ethereum [12] and Harmony [18]. We adopt a BFT consensus with BLS multi-signature [19] as the intra-shard consensus and a library named ate-pairing [21] for the VSO. The on-chain cross-shard mechanism of GriDB is similar to that of Monoxide [54]. Particularly, to commit a cross-shard transaction, each of its related shards needs to validate and commit it. Only if the transaction is committed in the blockchains of all its related shards, it is regarded as being committed successfully. This can be checked based on a list of Merkle proofs, each corresponding to a related shard. Besides, by checking the transaction hash included in every Merkle proof, it can be guaranteed that every related shard commits the same transaction. The optimization designs in § 6 are also implemented. To implement a MySQL interface to GriDB, we adopt a storage-agnostic SQL engine with in-memory table implementation [7].

**Setup.** The testbed is composed of 16 machines, each of which has an Intel E5-2680V4 CPU and 64 GB of RAM, and a 10 Gbps network link. Similar to [28, 57], to simulate geographically-distributed nodes, we set the bandwidth of all connections between nodes to 20 Mbps and impose a latency of 100 ms on the links in our testbed.

**Baseline.** For comparison, we implement a non-sharding blockchain database. This type of blockchain database does not need to consider the challenge of cross-shard query and inter-shard balancing because each node stores and processes the whole database. For a fair comparison, this blockchain database also adopts the signature-based BFT consensus adopted by GriDB as its underlying consensus. The basic idea of the non-sharding blockchain database is similar to that of the existing works such as FalconDB and SEDBD [41, 62] except that they adopt the other variants of BFT consensus and support some other functionalities (such as indexes). Moreover, we implement an on-chain sharding blockchain database including shard-cooperation cross-shard query and stop-restart inter-shard migration based on our strawman system in § 5.1.

**Workloads.** We evaluate the performance of GriDB using TPC-H [52] which is widely used by the database community. It consists of 8 tables for each dataset and 22 types of SQL queries. Our experiments are run on a database with 16 TPC-H datasets which are uniformly split across shards. Besides, we add data transactions, each of which insert, delete or update a new row, for the workload of each dataset. To simulate the cross-shard query, there is a proportion of query transactions involving tables in different shards and the proportion is called *cross-shard ratio*. To simulate the workload imbalance, similar to [9, 55], we set two imbalanced settings. For low imbalance, we adopt a Zipfian distribution where two-thirds of the accesses go to one-third of the datasets. For high imbalance, 40% of transactions follow the Zipfian in low imbalance, and the other transactions target 4 datasets initially on the first shard.

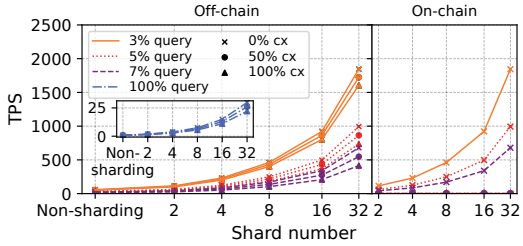



**Figure 5: Transaction throughput for GriDB, the on-chain sharding blockchain database, and the non-sharding blockchain database (cx means cross-shard ratio.)**

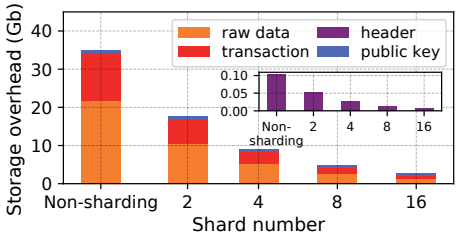



**Figure 6: Storage overhead per node for GriDB and the non-sharding blockchain database.**

## 8.1 Overall Performance

To evaluate the scalability, we measure the transaction throughput in TPS for the non-sharding blockchain database and GriDB with varying percentages of query transactions and cross-shard ratios. We deploy 30 nodes for each shard. Figure 5 shows that the measured TPS of GriDB increases linearly with the number of shards and decreases when there are more cross-shard query transactions in the workload. It is because the data transactions only involve one shard, and the verification is simple. However, a query transaction is computationally-intensive (it requires 0.17 ∼ 2.38 seconds even in a local database as discussed in § 8.2) and needs the delegation-based procedure for cross-shard verification, thus, committing query transactions costs more. Moreover, a query involving more shards causes more table transfers and more complex proof generation among the delegated nodes, which will be further studied in § 8.2. In comparison with GriDB, the on-chain sharding blockchain database has a similar throughput when there are no cross-shard queries. However, its throughput drops to nearly 0 when 50% or 100% of queries are cross-shard. It is because, for the on-chain one, the table transfer among shards caused by cross-shard queries can result in serious network blocked.

To evaluate the performance of GriDB for cross-shard data transactions, we pack cross-shard queries (used to delete the cross-shard query results) into cross-shard data transactions. According to Figure 5, GriDB's throughput for cross-shard data transactions is similar to that for cross-shard query transactions. It is because, as described in § 4.1, GriDB implements a delete statement by marking old data as invalid. Except for reaching consensus on cross-shard query results like a cross-shard query transaction, a cross-shard data transaction needs to include the information of marking the results as invalid, and each node needs to delete the results from its in-memory tables. However, these additional overheads are negligible. Thus, the expense of cross-shard data transactions and cross-shard query transactions are similar.

We also evaluate the storage overhead per node after loading all tables in the non-sharding blockchain database and GriDB with varying shard numbers. The results are given in Figure 6. Because each row is committed in the form of a data transaction and the data transactions are packed into blocks, loading the tables will introduce the block-related data including transaction-related and header-related data. From Figure 6, we can observe that, first, as the number of shards increases, the storage overhead for each node is reduced. Second, the transaction-related data cost half storage compared with the raw data. Third, compared with the other data, the storage of headers can be ignored. Forth, because the largest table in the evaluation consists of 6 million rows, the public key size of verifiable set operation (VSO) is about 0.76 GB. We regard the storage overhead caused by the public key as acceptable in the case of tables with millions of rows since it is considerably less than the recommended storage space of most blockchain nodes (such as 2 TB in Ethereum [11]) nowadays. Additionally, the storage of the on-chain sharding database is the same as that of GriDB.

**Table 1: Comparison of server and client times for evaluating queries using different approaches (The results for vSQL and SNARKs are provided in [60].)**

| | vSQL | | SNARKs | | GriDB | | MySQL |
|---|---|---|---|---|---|---|---|
| Query | Server | Client | Server | Client | Server | Client | |
| #19 | 4892s | 162ms | 196000s | 6ms | 41.14s | 221ms | 2.38s |
| #6 | 3851s | 129ms | 19000s | 6ms | 4.93s | 221ms | 1.44s |
| #5 | 5069s | 398ms | 615000s | 110ms | 490.33s | 221ms | 1.95s |
| #2 | 2346s | 508ms | 58000s | 40ms | 56.86s | 222ms | 0.17s |

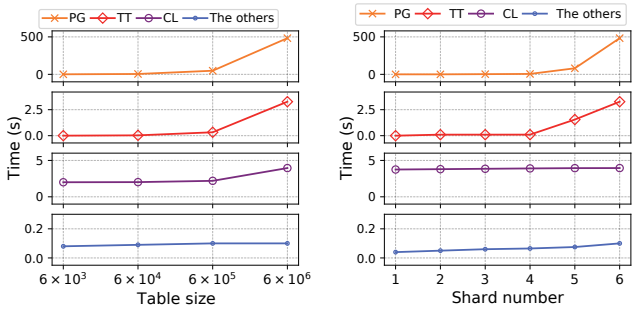
## 8.2 Performance of Cross-shard Query

We evaluate the performance of cross-shard queries. For comparison, we adopt two approaches providing the same functionality as our cross-shard query. These approaches are motivated by two previous works, i.e., vSQL [60] and libsnark [49], which can support arbitrary SQL queries based on interactive proof and SNARKs, respectively. Depending on either of these two works, any nodes can directly provide the result of a cross-shard query and a proof to the clients without consensus. We also evaluate the performance of the local computation for SQL in our nodes, based on MySQL. The server time is the time required for the server to evaluate the query and produce a valid proof and the client time is the time for the client to verify the proof. In GriDB, the server time is the duration from Line 2 to Line 6 in Algorithm 1, and the client time is the duration of Line 7 in Algorithm 1.

As a representative example, we pick the query #19, #6, #5, #2 in TPC-H and the results are given in Table 1. These queries include most SQL types, e.g., join, range, min and nested query. According to Table 1, the server time of GriDB is orders of magnitude less than that of vSQL and SNARKs while the client time is similar. For the server time, it is because our cross-shard query only constructs the expensive ADS for a few cross-shard operators while the security of the other operations depends on the intra-shard consensus. For

**Table 2: Time of each step for queries in GriDB (CL: Confirmation latency, PG: Proof generation, TT: Table transfer.)**

| Query | CL | PG | TT | The others |
|---|---|---|---|---|
| #19 | 4.38s | 36.74s | 4.04ms | 10ms |
| #6 | 3.44s | 1.44s | 0s | 5ms |
| #5 | 3.95s | 483.01s | 3.2s | 100ms |
| #2 | 2.17s | 54.46s | 139.57ms | 80ms |




(a) Table size (b) Number of the related shards

**Figure 7: Performance for query #5 with different table size and number of related shards in GriDB.**

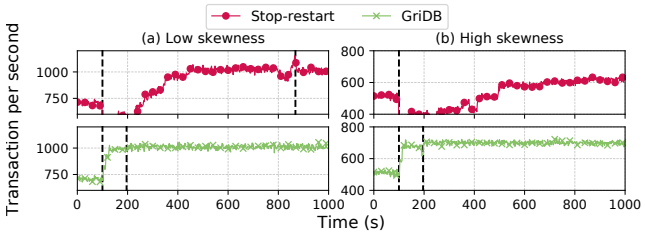
the client time, it is because the clients of GriDB only need to check whether their query transactions are confirmed or not via SPV. Note that the evaluation is based on the worst case, which means the tables for each cross-shard query are all located in different shards.

The time cost of each step for the queries in GriDB is summarized in Table 2. The results shows that the three steps occupy most of the time, matching the performance analysis in § 5.2. Furthermore, from Table 2, we have the following observations. First, according to the result of MySQL in Table 1, query #19 is the most complex one and the nodes spend more time on validating it during the intra-shard consensus, thus its confirmation latency is the most. Then, the proof generation and table transfer of query #5 is the most, because the query needs to join six tables, which results in six cross-shard operators in the worst case. Finally, the time cost of query #6 is the least, because it is a simple 3-dimensional range query followed by an aggregation for a single table.

We also evaluate the performance of the cross-shard query of GriDB with varying table size and number of related shards. We scale the number of rows in the largest participating table in query #5 from $6 \times 10^3$ to $6 \times 10^6$ and distribute its participating tables to $1 \sim 6$ shards. Figure 7a and Figure 7b show that the time cost is significantly reduced when the participating tables are smaller or there are fewer related shards. It is because the complexity of proof generation depends on the participating table size and the number of cross-shard operators, matching the analysis in § 5.2.

## 8.3 Performance of Inter-shard Balancing

We evaluate the throughput during migration via the off-chain live migration in GriDB and the stop-restart approach in the on-chain sharding blockchain database with various skewed workloads and the results are given in Figure 8. The process includes 48 migrations.

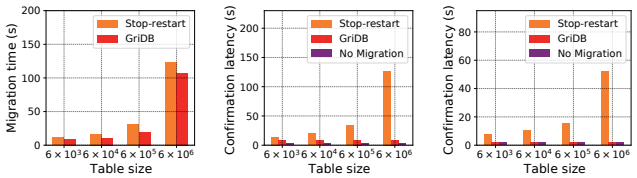



**(a) Fluctuation of throughput during migration.**

| Migration time (s), TPS | Low skewness | High skewness |
|---|---|---|
| Stop-restart | 770, 981 | 10678, 623 |
| GriDB | 96, 1012 | 96, 700 |

**(b) Statistics on migration time and throughput.**

**Figure 8: Transaction throughput during inter-shard migration with varying skewness.**




**(a) Migration time** **(b) Latency of txs for the migrating table** **(c) Latency of txs for the other tables**

**Figure 9: Inter-shard migration for tables with varying size.**

After migration, the throughput increases by 1.40× for the low skewness and 1.37× for the high skewness. It shows the load balancing among shards is helpful for the performance of sharding blockchain database. The off-chain live migration can shorten the migration time by nearly 87% compared with the stop-restart approach for the low skewness and 99% for the high skewness. Furthermore, the performance degradation in GriDB is minimal during migration. It is because, in GriDB, the off-chain manner significantly reduces the number of on-chain transactions, avoiding the massive overhead for consensus, and the dual mode minimizes service interruption during migration using the cross-shard off-chain notification.

Figure 9 plots the impact of the table size on the migration time, the confirmation latency of transactions for the migrating table and the other tables in the shards involved. Figure 9a shows that it costs more time to migrate a bigger table for both approaches. However, the migration time in GriDB is less than that in the stop-restart approach because there are only two on-chain transactions in GriDB, and a bigger table only requires more transmission time rather than more consensus rounds like the stop-restart approach. According to Figure 9b and Figure 9c, in GriDB, the confirmation latency for the tables during the migration is similar to that during normal mode (i.e., "No Migration" in the figures). Furthermore, the latency of transactions in the migrating table during migration is more than the latency during normal mode. It is because, in the dual mode, they are required to notify the destination shard.

# 9 RELATED WORK

## 9.1 Blockchain Database

We summarize the recent works that support various queries (e.g., SQL, key-value query, semantic query) on the blockchain as follows.

In the conventional blockchain, a query requires nodes to traverse the whole blockchain to guarantee a complete search. To avoid the expensive traversing, Pei *et al.* design a Merkle Semantic Trie for efficient semantic query [40]. SEBDB [62] builds three indices for efficient SQL queries, which can be complementary to our work. For instance, one can add these indices into each node in GriDB to improve the query efficiency. Motivated by outsourced databases [58], some works provide verifiable queries with blockchain clients, enabling clients to verify query results from untrusted nodes. SEBDB [62] designs a Merkle B-tree-based ADS for verifiable range query. Zhang *et al.* support verifiable Boolean range query [56, 59] while Falcon [41] supports verifiable SQL [61] for blockchain databases. However, they are constructed on top of the non-scale-out blockchains suffering from poor scalability.

The work most related to us is BlockchainDB [8], a key-value database on top of a sharding blockchain. In BlockchainDB, each transaction includes a key-value pair and the database layer provides clients with the interfaces of get, put, and verify operations. However, due to the simplicity of the key-value data model, every shard is isolated and every operation can be served by a single shard. In comparison, GriDB considers a relational sharding blockchain database, thus having richer transactional semantics than BlockchainDB. It brings the challenge of cross-shard queries as discussed in § 4.2. Moreover, the data management for sharding, i.e., the inter-shard balancing, has not been considered in BlockchainDB. Besides, GriDB's off-chain live migration can be applied to BlockchainDB by considering a key-value database as a particular case of a relational database with two-column tables.

## 9.2 Sharding

Elastico [30] is the first decentralized sharding blockchain. Every shard is responsible for validating a set of transactions via PBFT. A final shard verifies all the transactions received from shards and pack them into a global block. However, it only realizes verification sharding and each node needs to store all blocks. Omniledger [28] is the first blockchain achieving full sharding by a client-driven cross-shard mechanism. Another client-driven sharding system named Chainspace [1] is presented to support sharding for smart contracts. The client-driven cross-shard mechanisms put extra burden on typically lightweight user nodes and are vulnerable to denial-of-service attacks. To further improve the performance, researchers propose shard-driven mechanisms, e.g., RapidChain [57] and Monoxide [54]. Additionally, some works [5, 22, 51] share a similar idea in which some nodes store the blockchain state of multiple shards and thus can efficiently validate and execute cross-shard transactions.

Most of the existing sharding blockchains focus on transfer transactions (i.e., user-to-user transfers of digital funds) and smart contracts. Thus, their challenge of cross-shard transactions results from cross-shard payments (i.e., the transfer between accounts in different shards) or cross-shard smart contract calls (i.e., a smart contract that utilizes smart contracts at different shards). In comparison, in GriDB, the challenge result from *cross-shard queries* (i.e., queries to tables from different shards), *cross-shard data operations* (i.e., insert, delete or update operations on tables from different shards), and *inter-shard load balancing* caused by workload imbalance among shards. As discussed in § 1, these cross-shard database transactions make the on-chain cross-shard mechanism of the existing blockchain sharding systems suffer from extremely poor performance.

## 9.3 Off-chain

Off-chain protocols aim for blockchain scalability and build on top of the existing blockchains without changing trust assumptions [17]. The main idea is to avoid processing every transaction via the consensus and instead utilize the consensus only to undertake critical tasks (e.g., settlement and dispute resolution)[1]. There are two types of off-chain protocols. The first one is *channels* in which the involved parties can update their balance unanimously and privately by exchanging authenticated transitions off-chain [23, 29, 33, 44]. The second one is *commit-chains* which deploys a centralized but untrusted party to collect transactions on a child-blockchain and periodically update them to the parent-blockchain [26, 43].

GriDB shares the same idea with the existing off-chain protocols but uses a different approach (i.e., ADS) and targets a different problem (i.e., the high expense of cross-shard mechanism in the sharding blockchain database). To the best of our knowledge, GriDB is the first work to present an off-chain cross-shard mechanism to ease massive cross-shard data exchange, which may motivate the design of other sharding blockchain databases in the future.

# 10 CONCLUSION

We present GriDB, a sharding blockchain database that achieves a few thousand transactions per second on about one thousand nodes in a Byzantine environment while supporting the functionalities of data insert/update, relational queries, and database management. The off-chain cross-shard mechanism, including delegation-based cross-shard query and off-chain live migration, is the key contribution in GriDB. They offer a database layer of abstraction on top of the existing sharding blockchain and hide the complexity of the data and workload partition in the underlying sharding blockchain from the clients. GriDB also includes some database key components, including query optimization, and load scheduler. We plan to study other sharding strategies in future work, such as functional partitioning for sharding blockchain databases.

# ACKNOWLEDGMENTS

This research was supported by fundings from the Key-Area Research and Development Program of Guangdong Province under grant No. 2021B0101400003, Hong Kong RGC Research Impact Fund (RIF) with the Project No. R5060-19, General Research Fund (GRF) with the Project No. 152221/19E, 152203/20E, 152244/21E, and 152169/22E, and the National Natural Science Foundation of China (NSFC) 61872310, 62172453 and 62272496, Shenzhen Science and Technology Innovation Commission (JCYJ20200109142008673), the Major Key Project of PCL (PCL2021A06), and the Pearl River Talent Recruitment Program (2019QN01X130). We thank all anonymous reviewers who helped improve the paper.

[1]Note that the word "off-chain" in BlockchainDB [8] means it enables a node of a shard to verify the data from the other shards, which is different from our paper.

# REFERENCES


[1] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. 2017. Chainspace: A Sharded Smart Contracts Platform. *CoRR* abs/1708.03778 (2017). arXiv:1708.03778 http://arxiv.org/abs/1708.03778

[2] Dan Boneh and Xavier Boyen. 2008. Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *J. Cryptol.* 21, 2 (2008), 149–177.

[3] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. 2014. Verifiable Set Operations over Outsourced Databases. In *Public-Key Cryptography – PKC 2014*, Hugo Krawczyk (Ed.). Springer Berlin Heidelberg, 113–130.

[4] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI 99)*. USENIX Association.

[5] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards Scaling Blockchain Systems via Sharding. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 123–140. https://doi.org/10.1145/3299869.3319889

[6] Azure SQL Database. 2022. Scaling out with Azure SQL Database. Retrieved March 20, 2023 from https://docs.microsoft.com/en-us/azure/azure-sql/database/elastic-scale-introduction

[7] DoltHub. 2023. go-mysql-server. Retrieved March 20, 2023 from https://github.com/dolthub/go-mysql-server

[8] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. 2019. BlockchainDB: A Shared Database on Blockchains. *Proc. VLDB Endow.* 12, 11 (jul 2019), 1597–1609. https://doi.org/10.14778/3342263.3342636

[9] Aaron J. Elmore, Vaibhav Arora, Rebecca Taft, Andrew Pavlo, Divyakant Agrawal, and Amr El Abbadi. 2015. Squall: Fine-Grained Live Reconfiguration for Partitioned Main Memory Databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. 299–313. https://doi.org/10.1145/2723372.2723726

[10] Aaron J. Elmore, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. 2011. Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (Athens, Greece) *(SIGMOD '11)*. Association for Computing Machinery, New York, NY, USA, 301–312. https://doi.org/10.1145/1989323.1989356

[11] Ethereum. 2022. Hardware requirements for Go-Ethereum. Retrieved March 20, 2023 from https://geth.ethereum.org/docs/getting-started/hardware-requirements

[12] Ethereum. 2023. Go Ethereum. Retrieved March 20, 2023 from https://github.com/ethereum/go-ethereum

[13] Emmanuelle Ganne. 2018. *Can Blockchain revolutionize international trade?* World Trade Organization Geneva.

[14] Zerui Ge, Dumitrel Loghin, Beng Chin Ooi, Pingcheng Ruan, and Tianwen Wang. 2022. Hybrid Blockchain Database Systems: Design and Performance. *Proc. VLDB Endow.* 15, 5 (2022), 1092–1104. https://doi.org/10.14778/3510397.3510406

[15] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 51–68. https://doi.org/10.1145/3132747.3132757

[16] Google. 2023. The Go Programming Language. Retrieved March 20, 2023 from https://golang.org/

[17] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2019. SoK: Layer-Two Blockchain Protocols. Cryptology ePrint Archive, Paper 2019/360. https://eprint.iacr.org/2019/360 https://eprint.iacr.org/2019/360.

[18] Harmony. 2023. Harmony. Retrieved March 20, 2023 from https://github.com/harmony-one/harmony

[19] Harmony. 2023. Harmony consensus protocol design. Retrieved March 20, 2023 from https://github.com/harmony-one/harmony/tree/main/consensus

[20] Jelle Hellings and Mohammad Sadoghi. 2021. ByShard: Sharding in a Byzantine Environment. *Proc. VLDB Endow.* 14, 11 (2021), 2230–2243. https://doi.org/10.14778/3476249.3476275

[21] Herumi. 2020. High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves. Retrieved March 20, 2023 from https://github.com/herumi/ate-pairing

[22] Zicong Hong, Song Guo, Peng Li, and Wuhui Chen. 2021. Pyramid: A Layered Sharding Blockchain System. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488747

[23] Zicong Hong, Song Guo, Rui Zhang, Peng Li, Yufen Zhan, and Wuhui Chen. 2022. Cycle: Sustainable Off-Chain Payment Channel Network with Asynchronous Rebalancing. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 41–53. https://doi.org/10.1109/DSN53405.2022.00017

[24] IBM. 2020. Blockchain for supply chain solutions. Retrieved March 20, 2023 from https://www.ibm.com/blockchain/industries/supply-chain

[25] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. 2000. Randomized rumor spreading. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*. 565–574.

[26] Rami Khalil, Alexei Zamyatin, Guillaume Felley, Pedro Moreno-Sanchez, and Arthur Gervais. 2018. Commit-chains: Secure, scalable off-chain payments. Retrieved March 20, 2023 from https://eprint.iacr.org/2018/642.pdf

[27] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Austin, TX, USA) *(SEC'16)*. USENIX Association, USA, 279–296.

[28] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*. 583–598. https://doi.org/10.1109/SP.2018.000-5

[29] Lightning. 2021. The Lightning Network. Retrieved March 20, 2023 from https://lightning.network/

[30] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A Secure Sharding Protocol For Open Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 17–30. https://doi.org/10.1145/2976749.2978389

[31] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. 2015. Demystifying Incentives in the Consensus Computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 706–719. https://doi.org/10.1145/2810103.2813659

[32] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*. Springer, 369–378.

[33] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. 2019. Sprites and State Channels: Payment Networks that Go Faster Than Lightning. In *Financial Cryptography and Data Security*. Springer International Publishing, 508–526.

[34] Atsuki Momose and Ling Ren. 2021. Multi-Threshold Byzantine Fault Tolerance. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 1686–1699. https://doi.org/10.1145/3460120.3484554

[35] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. 2006. Authentication and Integrity in Outsourced Databases. *ACM Trans. Storage* 2, 2 (2006), 107–138. https://doi.org/10.1145/1149976.1149977

[36] MySQL. 2023. MySQL 8.0 Reference. Retrieved March 20, 2023 from https://dev.mysql.com/doc/refman/8.0/en/sql-data-manipulation-statements.html

[37] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. Retrieved March 20, 2023 from https://bitcoin.org/bitcoin.pdf

[38] Lan Nguyen. 2005. Accumulators from Bilinear Pairings and Applications. In *Proceedings of the 2005 International Conference on Topics in Cryptology* (San Francisco, CA) *(CT-RSA'05)*. Springer-Verlag, Berlin, Heidelberg, 275–292. https://doi.org/10.1007/978-3-540-30574-3_19

[39] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. 2011. Optimal Verification of Operations on Dynamic Sets. In *Advances in Cryptology – CRYPTO 2011*, Phillip Rogaway (Ed.). Springer Berlin Heidelberg, 91–110.

[40] Q. Pei, E. Zhou, Y. Xiao, D. Zhang, and D. Zhao. 2020. An Efficient Query Scheme for Hybrid Storage Blockchains Based on Merkle Semantic Trie. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*. 51–60. https://doi.org/10.1109/SRDS51746.2020.00013

[41] Yanqing Peng, Min Du, Feifei Li, Raymond Cheng, and Dawn Song. 2020. FalconDB: Blockchain-Based Collaborative Database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 637–652. https://doi.org/10.1145/3318464.3380594

[42] George Pîrlea, Amrit Kumar, and Ilya Sergey. 2021. Practical Smart Contract Sharding with Ownership and Commutativity Analysis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 1327–1341. https://doi.org/10.1145/3453483.3454112

[43] Joseph Poon and Vitalik Buterin. 2017. Plasma: Scalable autonomous smart contracts. Retrieved March 20, 2023 from https://www.plasma.io/plasma.pdf

[44] Raiden. 2021. The Raiden Network. Retrieved March 20, 2023 from https://raiden.network/

[45] Raghu Ramakrishnan and Johannes Gehrke. 2000. *Database Management Systems* (2nd ed.). McGraw-Hill, Inc.

[46] Sukriti Ramesh, Odysseas Papapetrou, and Wolf Siberski. 2009. Optimizing Distributed Joins with Bloom Filters. In *Distributed Computing and Internet*

*Technology*. Springer Berlin Heidelberg, 145–156.

[47] Pingcheng Ruan, Gang Chen, Tien Tuan Anh Dinh, Qian Lin, Beng Chin Ooi, and Meihui Zhang. 2019. Fine-Grained, Secure and Efficient Data Provenance on Blockchain Systems. *Proc. VLDB Endow.* 12, 9 (may 2019), 975–988. https://doi.org/10.14778/3329772.3329775

[48] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen, Qian Lin, and Beng Chin Ooi. 2021. Blockchains vs. Distributed Databases: Dichotomy and Fusion. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1504–1517. https://doi.org/10.1145/3448016.3452789

[49] SCIPRLab. 2020. libsnark: a C++ library for zkSNARK proofs. Retrieved March 20, 2023 from https://github.com/scipr-lab/libsnark

[50] Rebecca Taft, Essam Mansour, Marco Serafini, Jennie Duggan, Aaron J. Elmore, Ashraf Aboulnaga, Andrew Pavlo, and Michael Stonebraker. 2014. E-Store: Fine-Grained Elastic Partitioning for Distributed Transaction Processing Systems. *Proc. VLDB Endow.* 8, 3 (2014), 245–256. https://doi.org/10.14778/2735508.2735514

[51] Yuechen Tao, Bo Li, Jingjie Jiang, Hok Chu Ng, Cong Wang, and Baochun Li. 2020. On Sharding Open Blockchains with Smart Contracts. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 1357–1368. https://doi.org/10.1109/ICDE48307.2020.00121

[52] TPC. 2023. TPC-H Benchmark. Retrieved March 20, 2023 from http://www.tpc.org/tpch/

[53] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. 2019. SoK: Sharding on Blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (Zurich, Switzerland) *(AFT '19)*. Association for Computing Machinery, New York, NY, USA, 41–61. https://doi.org/10.1145/3318041.3355457

[54] Jiaping Wang and Hao Wang. 2019. Monoxide: Scale out Blockchains with Asynchronous Consensus Zones. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 95–112. https://www.usenix.org/conference/nsdi19/presentation/wang-jiaping

[55] Xingda Wei, Sijie Shen, Rong Chen, and Haibo Chen. 2017. Replication-Driven Live Reconfiguration for Fast Distributed Transaction Processing. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA, USA) *(USENIX ATC '17)*. USENIX Association, USA, 335–347.

[56] Cheng Xu, Ce Zhang, and Jianliang Xu. 2019. VChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 141–158. https://doi.org/10.1145/3299869.3300083

[57] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling Blockchain via Full Sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 931–948. https://doi.org/10.1145/3243734.3243853

[58] Bo Zhang, Boxiang Dong, and Wendy Hui Wang. 2021. Integrity Authentication for SQL Query Evaluation on Outsourced Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 33, 4 (2021), 1601–1618. https://doi.org/10.1109/TKDE.2019.2947061

[59] Ce Zhang, Cheng Xu, Jianliang Xu, Yuzhe Tang, and Byron Choi. 2019. GEM2-Tree: A Gas-Efficient Structure for Authenticated Range Queries in Blockchain. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 842–853. https://doi.org/10.1109/ICDE.2019.00080

[60] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In *2017 IEEE Symposium on Security and Privacy (SP)*. 863–880. https://doi.org/10.1109/SP.2017.43

[61] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2015. IntegriDB: Verifiable SQL for Outsourced Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 1480–1491. https://doi.org/10.1145/2810103.2813711

[62] Yanchao Zhu, Zhao Zhang, Cheqing Jin, Aoying Zhou, and Ying Yan. 2019. SEBDB: Semantics Empowered BlockChain DataBase. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 1820–1831. https://doi.org/10.1109/ICDE.2019.00198