Transparent Sharding

Deepal Tennakoon
University of Sydney
dten6395@uni.sydney.edu.au

Vincent Gramoli
University of Sydney
vincent.gramoli@sydney.edu.au

Abstract

Sharding is a well known technique to scale distributed systems horizontally. With the recent advent of blockchains, which typically run in open networks in the presence of malicious participants, new forms of sharding techniques have arisen. While the database sharding was typically seemless for the client of the system, blockchain sharding allows clients to select the location of their data, or the shardchain where their contract executes.

A critical requirement in this new adversarial environment is for clients to be able to consult the current sharding state without being fooled by malicious participants, a property we refer to as transparency.

In this chapter, we survey classic sharding techniques inherited from the database literature and more recent sharding techniques inherited from the blockchain literature. Finally, we focus on a recent technique that builds upon these techniques and exploits the smart contract logic to adjust sharding on demand and the transparency of the blockchain to let clients consult the current sharding state securely.

1 Introduction

Sharding is a term originally tossed in the context of massively multiplayer online games, in which parallel worlds use the same database source but evolve different database runtime dedicated to different players. One explanation for the term "shard" stems from the game Ultima Online whose fictional story mentioned shattering a crystal into shards, holding copies of a world continent, such that these copies evolve in parallel [15].

Due to the growing amount of transactions, sharding became popular to scale databases horizontally [6, 8]. The sharding technique consists of replicating a database structure across multiple machines while splitting its dataset into chunks, each maintained by a distinct set of machines, called a *shard*. In particular by assigning different machines to the maintenance of separate rows of a table, sharding lowers the size of the database index at each machine. This allows to speed up the information retrieval by searching into a smaller index. By adding resources, sharding helps increasing the performance of database services. In particular, sharding can be used to dynamically adjust the provisioning of resources based on the demand, a notion often referred to as *elasticity* [23, 22, 24, 2].

Due to the scalability limitations of classic blockchains [20, 29], sharding has naturally been applied to blockchains [19, 14, 32] in the hope of scaling blockchains horizontally. However, the context of blockchains differs significantly from the traditional context in which databases operate: instead of running in the closed networks of datacenters, blockchains typically run in open networks where users are incentivized to steal digital

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

assets. The arbitrary (byzantine) failure model that can mimic a coalition of malicious users is thus a central problem of blockchains and differs radically from the crash failure or isolated arbitrary failure models present in datacenters. As an example, Figure 1 illustrates two different contexts in which the techniques used for database sharding and blockchain sharding can be implemented.

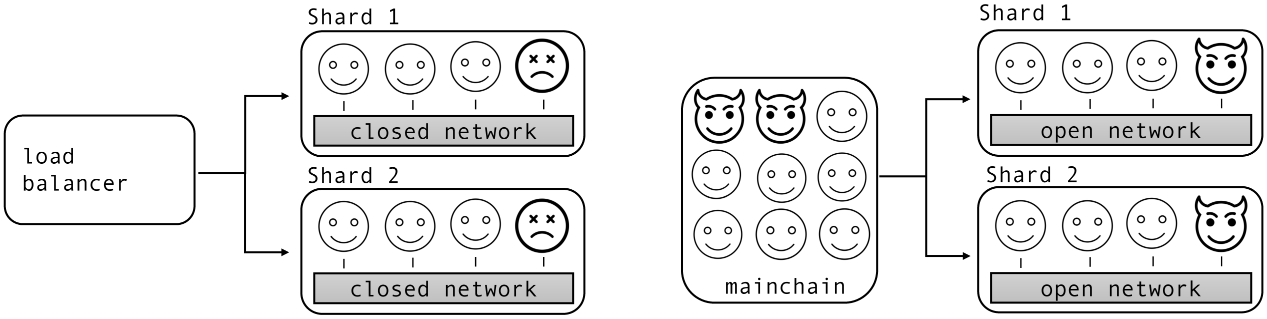

Figure 1: The database sharding (left) is typically using a load balancer to direct requests to the right shards running in closed networks whereas the blockchain sharding (right) typically uses different chains, like a mainchain and multiple shardchains, that run in an open networks where malicious participants can collude to attack the system.

Due to the growing demand for decentralized applications (DApps), the blockchain sharding techniques are evolving from static sharding techniques [25] to more dynamic sharding techniques [1]. These dynamic sharding techniques promise the elasticity of database sharding [23, 22, 24, 2]: they aim at adjusting the resource provisioning based on the demand. In some cases, the user could exploit this dynamism by deploying its popular DApp in a separate blockchain shard to avoid the congestion in other shards. In some other cases, one could migrate less demanded DApps in the same shard to decommission resources.

Unfortunately, most dynamic blockchain sharding solutions focus on creating or modifying shards [16], without considering how to access the existing sharding state. In other words the shard state is not *transparent*. If a user requests to adjust the amount of resources in a shard, then it typically ignores whether this adjustment was successful. The problem of offering transparency is not easy precisely because of the open networks in which the blockchains operate. A user could for example easily be fooled by a malicious participants that misinforms it of the success of its request, while the malicious participant never relayed it to the rest of the blockchain participants.

In this chapter, we survey existing sharding techniques and describe a new design that could leverage the inherent transparency of the blockchain to offer dynamic sharding that can be securely monitored, despite the presence of byzantine failures.

The rest of the chapter is organized as follows. In Section 2 we present the preliminaries. In Section 3, we present the database sharding techniques and in Section 4, we survey the blockchain sharding techniques. We present transparent sharding in Section 5. Finally, we present the related work in Section 6 and conclude in Section 6.

2 Preliminaries

We consider a distributed system of n *nodes* that exchange messages via a network. The failure model is different depending on the type of networks. The network can be *closed*, in which case nodes need to be authorized to join the network or *open*, in which case any nodes can join the network. Examples of closed and open networks are a local area network within a single datacenter and the Internet, respectively. It is typically more frequent to observe arbitrary failures in an open network than in a closed network, simply because one does not control the nodes joining the open network or their motivations.

In general, we consider that nodes communicate via point-to-point reliable channel. Hence if a node sends a message to another node and none of them fail, then the message is eventually received. Note that point-to-point channels can be implemented using secure channel. At times, we consider that every message takes less than some amount of time to be delivered, in which case we say that the network is *synchronous*. However, it is hard to predict the time a message will take in an open network due to various reasons (e.g., congestion, failures, disasters) that are out of the control of any administrator. Hence we often relax the synchrony assumption and assume that the network is *partially synchronous* [11] in that the bound exists but is not known by the protocol.

We assume the presence of up to f nodes that can fail among the n nodes of the group (we consider the group as either all nodes or just the nodes taking part in a shard). And we consider two types of failures: *crash* failures after which the faulty node stops acting and does no longer send messages and *byzantine* failures, when the faulty node behaves arbitrarily by not necessarily following the protocol. Interestingly, since byzantine nodes can store and send any arbitrary data to other nodes, it is important to provide a secure way for *correct* (i.e., non-faulty) nodes to retrieve the correct information. The property by which the system offers a node the possibility to retrieve correct information about the system is called *transparency*.

Blockchains are distributed systems [20]. Upon receiving client transactions, blockchain nodes first validate and broadcast them to the blockchain network. Then, all correct nodes in the blockchain agree upon the order to execute client transactions. This is known as reaching consensus on the order of transactions. Subsequently, all correct nodes execute transactions according to the order determined by consensus, reaching the same final state. After execution, the blockchain nodes store the transactions in blocks and each block points to the previous block forming a chain of blocks (i.e., an immutable ledger), hence the name "blockchain". Finally, the client queries the blockchain to retrieve the result of the execution from the blockchain nodes. Just like databases, blockchains store the results of transaction executions, however, blockchains differ by the openness of the network in which they typically operate and the crytpography they require to avoid trusting a central entity.

3 Database Sharding

Sharding comes originally from the database literature where it proved instrumental for optimizing performance. It offers a natural way to adjust resource provisioning to serve the demand.

3.1 Storage Database Sharding

Google has been influential in the design of novel sharding techniques to scale its storage systems, in particular with BigTable [6] and Spanner [8].

BigTable [6] offers eventual consistency to NoSQL data by replicating the data on clusters of machines to offer horizontal scalability. It is optimized to offer low latency single-value reads and writes. The client requests a front-end server that redirects the request to a bigtable cluster, whose nodes, called *tablet servers* handle subsets of requests. One can improve the performance by adding tablet servers to the cluster. Bigtable stores data in tables, each being a key-value map, that is sharded into blocks of contiguous rows but tablet servers simply points to the data without hosting them, hence rebalancing tablets from one node to another is fast.

Spanner [8] is an SQL distributed database that scales horizontally. Rows are organized lexicographically by primary keys. Spanner replicates data across multiple zones. The data is sharded by ranges of rows across the multiple replicas of a zone. The replicas execute the Paxos consensus protocol to guarantee that consistent data are sufficiently replicated to tolerate potential crash failures before being committed. In case of failures, the data are migrated across machines to balance the load.

3.2 Dynamic Database Sharding

Dynamic sharding consists of modifying the sharding state at runtime. It is key to provide database elasticity.

Slicer [2] is a dynamic sharding solution for datacenters that proved effective to allocate resources to web-service frontends and increase the efficiency of web caches. Slicer derives a key from a request and exploits a centralized slicing service that monitors load and task availability to map key ranges or slices to tasks. Slicer combines with the Google Stubby's RPC system to balance tasks across datacenters. Slicer minimizes resource usage by 63% compared to a static sharding technique. Slicer uses a centralized algorithm rather than a client-based consistent hashing.

Accordion [22] offers the possibility to scale out and scale in a distributed database system by provisioning more or less servers as the load varies. Usually, databases are horizontally partitioned such that each partition is owned by exactly one server. The partitioning algorithm is key to the performance of the system but the placement of partitions is generally static. Accordion makes this mapping of partition to servers dynamic. E-Store [24] is an elastic technique for on-line transaction processing that exploits a two-tier horizontal partitioning technique that migrates data when load imbalance is detected to address the problem of the workloads being skewed towards the same resources.

Basil [23] presents a vertical database sharding approach for ACID transactions, maintaining a sharded key-value store in a byzantine fault-tolerant setting. This facilitates the execution of transactions concurrently in different shards. Basil requires the clients (i.e., the transaction senders) to decide to commit or abort the transactions based on the votes of replicas in a shard. Consequently, the client relays the outcome to the application. By ensuring byzantine isolation, whereby correct clients observe a state of the database produced by correct clients alone, and byzantine independence, whereby byzantine participants cannot collude to abort a client's transaction, Basil provides safety and liveness. However, the dynamism of Basil is limited as it does not explain how to change the number of shards at runtime. It also requires $5f + 1$ replicas per shard to ensure the aforementioned properties.

4 Blockchain Sharding

As blockchain typically operate in open networks, the blockchain sharding solutions must typically cope with byzantine failures. This is why multiple probabilistic techniques able to rotate the shard membership in an unpredictable way have become popular.

4.1 Deterministic sharding

Deterministic sharding [5, 7, 10] consists of assigning transactions to shards deterministically. The advantage of such an approach is that the current shard state is inherently transparent as anyone can infer the shard responsible for each transaction by simply computing a local deterministic function.

SharPer [5] creates shards deterministically based on geographical distribution. Nodes that are located close to each other are assigned to the same shard. Red Belly Blockchain [10] shards only the verification without sharding the consensus nodes. The motivation stems from the fact that the verification of cryptographic signatures is CPU intensive. Instead of having all nodes verifying all transactions, Red Belly Blockchain assigns deterministically each transaction, based on its hash, to two subsets of nodes: its $f + 1$ primary verifiers and its f secondary verifiers. The secondary verifiers wait for some time for the primary verifiers to verify the transaction. If the primary verifiers are too slow or unresponsive, then the secondary verifiers start verifying the transaction. A node detects whether a transaction is properly signed once it receives the same response from $f + 1$ distinct verifiers.

The drawback of deterministic sharding is that the outcome of the function is predictable, which makes the system vulnerable to attacks. In particular, an attacker can exploit this information in order to bribe the nodes that are responsible of a shard. If the attacker manages to bribe a sufficient portion of a shard then it can prevent the members of this shard from reaching consensus, potentially leading the system to an inconsistent state. Such inconsistent states are called "forks" and are exploited by various attacks [21, 12] to double spend. SSChain [7] allows nodes to freely join a shard deterministically. To avoid shard-takeovers SSChain follows a two-chained approach: a root chain that verifies the blocks coming from each shard before committing them, and a shardchain that agrees upon blocks to send to the root chain. The root chain is able to make an accurate verification of shard blocks by keeping the full state of the blockchain.

4.2 Probabilistic sharding

To cope with the predictability of deterministic sharding, probabilistic sharding protocols were proposed [14, 19, 32, 13]. Probabilistic sharding relies on a *mainchain* also known as beacon chain, final committee, or main committee, that performs administrative tasks like creating new shards and synchronizing the states of multiple shards. The mainchain creates each shard, also called *shardchain*, probabilistically which maintains separate state, transactions, blocks and a chain. Each shard verifies a unique subset of transactions and executes consensus separately on those transactions. Unlike deterministic sharding, the probabilistic creation of shards mitigate risks of shard-takeovers. For this purpose, probabilistic sharding mechanisms typically select a shard size and a number of shards to guarantee with high probability that the shard members can reach a consistent state through consensus. In particular, when the network is open, one cannot predict the time a message takes to be delivered, hence the sharding mechanisms must ensure that less than 1/3 of the shard members are byzantine nodes with high probability [11]. In probabilistic sharding to mitigate bribery from slowly-adaptive adversaries, shards are changed within a specific time period known as an epoch. This is to avoid a shard-takeover potentially causing double spending.

OmniLedger [14] is a permissionless sharded blockchain that creates shards probabilisitically based on the RANDHOUND protocol and a VRF. A shard remains active in a time period known as an epoch. OmniLedger assumes synchrony for shard creation and partial synchrony in a shard epoch. OmniLedger handles cross-shard transactions using an atomic commit-abort protocol run by clients sending transactions. However, clients can censor cross-shard transactions as they are tasked with creating cross-shard transactions. OmniLedger has performance enhancements such as concurrent processing of non-conflicting transactions in a shard as well as using state blocks as checkpoints to reduce the size of the downloaded blockchain when syncing. The fault tolerance of clients is not mentioned. Assuming synchrony for shard creation is not realistic for real-world cases.

RapidChain [32] assumes synchrony within an epoch but assumes partial synchrony in all other parts of the protocol. RapidChain's probabilistic shard creation involves a reference committee using proof-of-work (PoW) coupled with randomization to assign nodes to shards in a way that minimizes the probability of $f > n/3$ where f is the number of byzantine nodes and n is the number of shard nodes.

In contrast, Monoxide [27] assumes asynchrony and creates zones (i.e., shards) by assigning random identifiers to miners which assign those miners to zones. Each zone processes transactions, keeps state and executes consensus separately. Within a zone Monoxide uses PoW to agree on the order of transactions, hence making the consensus probabilistic. To mitigate adversaries centralizing their mining power to one zone to take over a zone, Monoxide [27] introduces a novel proof-of-work scheme known as Chu-ko-nu mining. This scheme allows a miner to create a block in any zone by solving a PoW puzzle, which evenly distributes the mining power across all zones, preventing it from being gathered to a single zone. Cross zone transactions are processed in an asynchronous and lock-free manner that allows the zones to concurrently process transactions. However, the number of zones in Monoxide is not adjustable at runtime to the best of our knowledge.

The next major release of Ethereum known as Ethereum 2.0 is said to contain a probabilistic sharding mechanism consisting of a fixed set of 64 shard chains and a single beacon chain [13]. Nodes require to escrow a deposit to Eth2.0 before assigning them to shardchains probabilistically using a random beacon. Eth2.0 requires a minimum of 111 nodes to be in a shard [28] to lower the probability of having $2/3$ adversarial nodes in a shard to 2^{-40}. Each shard runs a series of 64 Casper FFG consensus instances per epoch, after which a new block containing the shard states is appended to the beacon chain.

4.3 Probabilistic transaction sharding

Probabilistic transaction sharding creates shards probabilistically and inherits all characteristics of probabilistic sharding except it only shards transactions. In other words it assigns transactions to a subset of nodes (i.e., shards). The state, chain and blocks are not sharded.

Elastico [19] is a permissionless byzantine fault tolerant blockchain that partitions the network into shards that only process a subset of the entire set of transactions. Elastico assumes partial synchrony and achieves linear scalability. Since Elastico is permissionless, Sybil resistance is achieved by establishing identities using a PoW puzzle, public key and IP addresses. An adversary's capability to create multiple identities is limited since they require to solve a PoW puzzle. Elastico consists of a final committee and multiple committees each running its own BFT consensus. Based on the generated node identities and a random beacon generated by the final committee, nodes are assigned to shard committees per epoch. The final committee receives all agreed values from each committee at the end of an epoch and reaches consensus using a BFT consensus. Note that epochs in Elastico are based on an adjustable value N such that N is the number of blocks. To tolerate adaptive adversaries, Elastico rotates entire committees after an epoch, in contrast to the gradual or constant number of nodes rotated in other probabilistic approaches [13, 32]. As a result, Elastico nodes, despite being assigned to shards, require to store the entire state of all shards.

Zilliqa [25] exploits PoW and a random beacon to select nodes into a "DS committee", which is Zilliqa's mainchain. Every newly elected node in the DS committee churns out the oldest node making sure that at all times the most recently elected n nodes are in the DS committee. Nodes wanting to join shards use PoW and a random beacon generated by the DS committee to solve a puzzle and derive a nonce which is submitted to the DS committee. By reaching consensus on nonces, the DS committee assigns nodes to shards probabilistically where each shard processes a subset of transactions.

4.4 Probabilistic state sharding

Probabilistic state sharding creates shards probabilistically by only sharding states. It inherits all other characteristics of probabilistic sharding.

Al-Bassam et al. [3] presents ChainSpace that assigns smart contract objects to a set of nodes randomly based on $\Psi(o) = id(o) \mod K$ where K represents the constant number of shards and $id(o)$ is the $\mathrm{SHA256}$ hash of the object. Since smart contract objects are assigned to separate shards, each shard keeps a separate state corresponding to the smart contract objects. ChainSpace assumes asynchronous communications and uses BFT-Smart for consensus.

In the NEAR protocol[1], the set of nodes that have the highest stake in an epoch are randomly assigned to shards probabilistically. Each shard keeps a separate state. A node can be a member of one or many shards. When a node is a member of multiple shards it keeps the state of all those shards.

5 Transparent dynamic sharding

As recent sharding techniques offer blockchain users the ability to control the locations of their data or to select the shard in which they execute their contract, it has become crucial for sharding to be transparent.

5.1 Dynamic blockchain sharding

Dynamic blockchain sharding (DBS) [1] is a blockchain sharding protocol made transparent by exploiting the blockchain transparency itself. It shares commonalities with traditional sharding from the database literature [22, 24, 2, 23] that offer elasticity: new shards can be created and existing shards can be closed at runtime, hence

[1] `https://near.org/papers/economics-in-sharded-blockchain/`

adjusting potentially the provisioning of resources based on the demand. It differs from traditional sharding from the database literature by tolerating byzantine participants. It runs the Democratic Byzantine Fault Tolerant consensus protocol [9] so that any shard adjustment is decided by all correct nodes unanimously, despite partial synchrony and the presence of up to $f < n/4$ byzantine nodes, and it rotates shards probabilistically to cope with slowly-adaptive adversaries, similar to probabilistic sharding approaches [14, 19, 32, 13].

5.2 Transparency

To achieve transparency, DBS differs from other techniques by exploiting the smart contract logic to adjust the sharding state. Initially, when the blockchain starts, it is equiped with a built-in smart contract that exposes to client users the functions to create a new shard by spawning potentially more computational resources, to close an existing shard by decommissioning computational resources, and to adjust the size of the existing shards at runtime.

As each function invocation to adjust the shards consists of a blockchain transaction request that gets securely stored in the distributed ledger (like any other blockchain transactions), a node simply needs to consult the current state of the mainchain to derive the most current sharding state. This state indicates the amount of shards that exist, the number of nodes in each shard, the locations (e.g., static IP addresses, domain names) of these nodes. Note that if the client needs to retrieve the state of the shard (e.g., its DApps, past transactions), then the client would need to download the corresponding shardchain as well.

When a client wants to download the mainchain, it first contacts the nodes running the mainchain. Note that it is reasonable to assume that a client can retrieve the nodes of the mainchain, otherwise this client would not be able to use the service (classic blockchains like Bitcoin [20] and Ethereum [29] use hard-coded DNS seed for clients to retrieve blockchain nodes). The client then asks a copy of the blockchains to the mainchain nodes. Upon confirmation of the current state of the mainchain by $f + 1$ distinct mainchain nodes, then it knows that this mainchain state is correct. This is because f is the maximum number of byzantine nodes in the system by assumption, so there cannot be $f + 1$ malicious nodes responding to the client with a fake state. Note that we could hardcode directly the domain names of the nodes hosting the shards but every adjustment to the shard would require a lengthy DNS reconfiguration.

5.3 Mainchain and shardchains

DBS lets existing users of the initial blockchain, called the mainchain, become a user of a shard, called a shardchain, by depositing assets into the mainchain while invoking the shard creation function. These deposited assets remains frozen in the mainnet but can be used to transact in the shardchain for the lifetime of the shardchain. When the shardchain is closed, the balances are reconciled and the remaining deposits are returned to their users. DBS was shown instrumental to accelerate the performance of the blockchain almost linearly with the number of shards in good executions. In case of unexpected network delays, DBS may not succeed in adjusting the sharding during the first attempt, then it retries after allocating more time for the second attempt and so forth. As the network is partially synchronous, there is a point in the execution where DBS has allocated a sufficient amount of time for the sharding adjustment to succeed.

Once the client has successfully downloaded the mainchain, as explained in Section 5.2, then it is easy to reconstruct the current sharding state. The client can inspect the history of transactions stored in the mainchain and retrieves one by one the smart contract function invocations that created, closed and altered the shards. By replaying these transactions, the client can derive the current sharding state, by retrieving exactly the resources (computational nodes involved in running the shards) and the users (the users that deposited assets to access each shardchain).

6 Related work

There exist several surveys on blockchain sharding [26, 31, 30]. In this section, we discuss similarities and disparities of these surveys with our blockchain sharding survery.

In [26], the authors provide a systemic and comprehensive review of blockchain sharding methods. First, they introduce the basic concepts of blockchain sharding such as identity establishment, randomness generation for shard creation, intra-shard consensus, cross-shard transactions, epochs, and shard committee reconfiguration. Then they discuss the key characteristics of state-of-the art sharding methods and they summarized in a table based on shard creation method, network model, intra-shard consensus, inter-shard consensus, safety and performance. Our survey in contrast classifies database sharding and then blockchain sharding methods based on common characteristics.

Yu et al. [31] present a systemic survey of blockchain sharding techniques for permissionless blockchains. However they do not cover sharding blockchain works such as Red Belly Blockchain [10], NEAR [2] and Zilliqa [25]. This survey, similar to Sok [26] summarizes the surveyed blockchain sharding techniques according to their key characteristics but in contrast does not discuss database sharding approaches and its lead up to blockchain sharding.

Xi et al. [30] perform a comprehensive survey on blockchain sharding that includes Monoxide [27], Elastico [19], OmniLedger [14], RapidChain [32], ChainSpace [4], Ethereum2.0 [13] and TEEEChain [17]. They identify the following characteristics of each sharded blockchain: the network model (e.g. synchronous, partially-synchronous), node allocation method into shards (e.g. PoW, random beacon, deterministic, etc), transaction model (e.g. UTXO, account-based), intra-shard consensus algorithm, threat model, cross-shard transaction processing techniques and performance. Similar to our work, Xi et al. [30] classify blockchain sharding into three categories. Namely, network sharding, transaction sharding and state sharding. However, they do not explicitly assign sharded blockchains to these three categories.

In [18], the authors offer a systematic analysis of existing sharded blockchain systems. They decompose the blockchains that benefit from sharding into functional components, classify these systems, and analyze their components. They present a layered decomposition similar to the layers 0, 1, 2 of Xi et al. [30] but called them network, consensus and application layers. In their context, sharding consists of splitting the work so that each shard generates its own independent chains completely disconnected from other chains. As a result, they consider solutions that partially order transactions. This is a distinction with some of the sharding techniques we consider, like verification sharding, that totally order all transactions [10].

Interestingly, the large body of work on blockchain sharding explains how shards can be created or modified despite the presence of malicious participants, however, they do not explain how one can retrieve the current sharding state in a secure way. By contrast, we consider transparency as an important property to allow users to retrieve the current state of the blockchain sharding despite the presence of malicious participants.

7 Conclusion

In this chapter, we surveyed sharding techniques both in the database context and in the blockchain context. Although the blockchain sharding techniques are inspired by the database context, they raise an interesting challenge related to the openness of the network in which they execute. In this novel context, an interesting problem is the one of transparency where the users can retrieve the correct sharding state despite the presence of malicious coalitions. We presented a recent solution to this problem that lets users adjust the shards dynamically and consult the current sharding state securely through transparency.

[2] https://near.org/papers/economics-in-sharded-blockchain/

Acknowledgements

This research is supported under Australian Research Council Future Fellowship funding scheme (project number 180100496) entitled "The Red Belly Blockchain: A Scalable Blockchain for Internet of Things".

References

[1] Deepal tennakoon and vincent gramoli. In *Proceedings of the Fifth International Symposium on Foundations and Applications of Blockchain (FAB)*, 2022.

[2] Atul Adya, Daniel Myers, Jon Howell, Jeremy Elson, Colin Meek, Vishesh Khemani, Stefan Fulger, Pan Gu, Lakshminath Bhuvanagiri, Jason Hunter, Roberto Peon, Larry Kai, Alexander Shraer, Arif Merchant, and Kfir Lev-Ari. Slicer: Auto-sharding for datacenter applications. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, page 739–753, 2016.

[3] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778*, 2017.

[4] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. In *25th Annual Network and Distributed System Security Symposium NDSS*. The Internet Society, 2018.

[5] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. *SharPer: Sharding Permissioned Blockchains Over Network Clusters*, page 76–88. Association for Computing Machinery, New York, NY, USA, 2021.

[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2), 2008.

[7] Huan Chen and Yijie Wang. Sschain: A full sharding protocol for public blockchain without data migration overhead. *Pervasive and Mobile Computing*, 59:101055, 2019.

[8] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, 31(3), 2013.

[9] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: efficient leaderless Byzantine consensus and its application to blockchains. In *Proc. 17th IEEE Int. Symp. Netw. Comp. and Appl (NCA)*, pages 1–8, 2018.

[10] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Red Belly: a secure, fair and scalable open blockchain. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21)*, May 2021.

[11] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[12] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The Attack of the Clones against Proof-of-Authority. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'20)*, Feb 2020.

[13] The eth2 upgrades. Accessed: 2022-03-26.

[14] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. Cryptology ePrint Archive, Report 2017/406, 2017.

[15] Ralph Koster. Database "sharding" came from uo?, 2009. Accessed:2022-04-23 - `https://www.raphkoster.com/2009/01/08/database-sharding-came-from-uo/`.

[16] Jae Kwon and Ethan Buchman. Cosmos White Paper. Accessed:2022-05-30 - `https://v1.cosmos.network/resources/whitepaper`

[17] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter R. Pietzuch, and Emin Gün Sirer. Teechain: Scalable blockchain payments using trusted execution environments. Technical Report 1707.05454, arXiv, 2017.

[18] Yizhong Liua, Jianwei Liua, Marcos Antonio Vaz Sallesb, Zongyang Zhanga, Tong Lia, Bin Hua, and Rongxing Luc Fritz Hengleinb. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. Technical Report 2102.13364, arXiv, 2021.

[19] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 17–30, 2016.

[20] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008.

[21] C. Natoli and V. Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *47th IEEE/IFIP Int. Conf. Dependable Syst. and Netw. (DSN)*, Jun 2017.

[22] Marco Serafini, Essam Mansour, Ashraf Aboulnaga, Kenneth Salem, Taha Rafiq, and Umar Farooq Minhas. Accordion: Elastic scalability for database systems supporting distributed transactions. *Proc. VLDB Endow.*, 7(12), 2014.

[23] Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, and Natacha Crooks. Basil: Breaking up bft with acid (transactions). In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 1–17, New York, NY, USA, 2021. Association for Computing Machinery.

[24] Rebecca Taft, Essam Mansour, Marco Serafini, Jennie Duggan, Aaron J. Elmore, Ashraf Aboulnaga, Andrew Pavlo, and Michael Stonebraker. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proc. VLDB Endow.*, 8(3), 2014.

[25] The ZILLIQA Team. The zilliqa technical whitepaper. Technical report, Zilliqa, 2017. Accessed February 2022.

[26] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, page 41–61, 2019.

[27] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 95–112. USENIX Association, February 2019.

[28] SJ Wels. Guaranteed-tx: The exploration of a guaranteed cross-shard transaction execution protocol for ethereum 2.0. Master's thesis, University of Twente, 2019.

[29] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.

[30] Jinwen Xi, Shihong Zou, Guoai Xu, Yanhui Guo, Yueming Lu, Jiuyun Xu, Xuanwen Zhang, and Francesco Gringoli. A comprehensive survey on sharding in blockchains. *Mob. Inf. Syst.*, jan 2021.

[31] Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J. Andrew Zhang, and Ren Ping Liu. Survey: Sharding in blockchains. *IEEE Access*, 8:14155–14181, 2020.

[32] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. RapidChain: Scaling blockchain via full sharding. In *ACM CCS*, pages 931–948, 2018.