Optimal Sharding for Scalable Blockchains with Deconstructed SMR

Jianting Zhang
Purdue University
zhan4674@purdue.edu

Zhongtang Luo
Purdue University
luo401@purdue.edu

Raghavendra Ramesh
Supra Research
r.ramesh@supraoracles.com

Aniket Kate
Purdue University
Supra Research
aniket@purdue.edu

ABSTRACT

Sharding enhances blockchain scalability by dividing nodes into multiple shards to handle transactions in parallel. However, a size-security dilemma where every shard must be large enough to ensure its security constrains the efficacy of individual shards and the degree of sharding. Most existing solutions therefore rely on either weakening the adversary or making stronger network assumptions.

This paper presents Arete, an optimally scalable blockchain sharding protocol designed to resolve the dilemma based on an observation that if individual shards can tolerate a higher fraction of Byzantine faults, we can securely create smaller shards in a larger quantity. The key idea of Arete, therefore, is to improve the security resilience of shards by dividing the blockchain's State Machine Replication (SMR) process. Like modern blockchains, Arete first decouples SMR in three steps: transaction dissemination, ordering, and execution. However, for Arete, a single ordering shard performs the ordering task while multiple processing shards perform the dissemination and execution of blocks. As processing shards do not run consensus, each of those tolerates up to half compromised nodes. Moreover, the SMR process in the ordering shard is extremely lightweight as it only operates on the block digests. Second, Arete considers safety and liveness against Byzantine failures separately to improve the safety threshold further while tolerating temporary liveness violations in a controlled manner. Apart from creating more optimal-size shards, such a deconstructed SMR scheme empowers us to devise a novel certify-order-execute architecture to fully parallelize transaction handling, thereby significantly improving the performance. We implement Arete and evaluate it on the AWS environment by running up to 500 nodes. Our results demonstrate that Arete outperforms representative sharding protocols in scalability, throughput, and cross-shard latency without compromising on intra-shard latency.

PVLDB Reference Format:
Jianting Zhang, Zhongtang Luo, Raghavendra Ramesh, and Aniket Kate.
Optimal Sharding for Scalable Blockchains with Deconstructed SMR.
PVLDB, 18(7): 2198 - 2211, 2025.
doi:10.14778/3734839.3734855

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/EtherCS/arete.

Proceedings of the VLDB Endowment, Vol. 18, No. 7 ISSN 2150-8097.
doi:10.14778/3734839.3734855

1 INTRODUCTION

Scalability is one of the major issues facing blockchain technology today. Sharding is proposed to address the scalability issue [5, 20, 22, 32, 38, 44, 72, 74]. By dividing nodes into multiple groups (i.e., *shards*) and enabling distinct shards to handle transactions with an *independent, intra-shard* consensus or state machine replication (SMR) process, blockchain sharding can achieve an increased transaction throughput as more nodes join the system. Due to the decreasing size of the consensus group, one primary concern of sharding is how to form secure shards, where the ratio of Byzantine nodes of a shard cannot exceed the security threshold f that an intra-shard consensus can tolerate. For instance, it is well-known that the classical non-synchronous Byzantine Fault-Tolerant (BFT) consensus/SMR can only tolerate up to 1/3 corruptions, i.e., $f < 1/3$ [24].

The Size-security dilemma. Unfortunately, the sharding protocol faces a size-security dilemma that constrains the shard size and the overall number of shards the system can create. To elaborate, in blockchain sharding, having smaller shards means more shards [22, 40, 52, 72]. When dividing nodes into shards, a small shard size enables more shards to be created, but increases the probability of forming an insecure shard with higher than f fraction of malicious nodes. Larger shards are more secure but only allow fewer shards to be created. The trade-off between shard size and the probability of forming secure enough shards is fundamental to securely scaling the blockchain systems via sharding. For instance, to achieve a 30-bit statistical security[1] in a system with $n = 1000$ nodes and up to $s = 1/4$ fraction of Byzantine nodes, the sharding system must sample 486 nodes to form a shard with $f < 1/3$ security threshold (Table 1). As only two shards can be created with the shard size $m = 486$ and $n = 1000$, the system scales poorly. Moreover, a large shard suffers from performance degradation when running most consensus protocols due to quadratic communication overheads [15]. Therefore, an exciting and promising research direction in blockchain sharding is to securely reduce the shard size so that more and smaller shards can be created to enhance the efficacy and scalability of sharding.

Existing solutions and limitations. To reduce the shard size while ensuring the security of the system, many existing solutions make stronger assumptions [20, 33, 38, 72]. For instance, RapidChain [72] assumes the network to be bounded synchronous, adopts synchronous consensus protocols [1, 14], and enhances the security threshold for every shard to $f < 1/2$. An enhanced security threshold enables shards to be smaller. However, for any low latency and high-throughput blockchain system over the Internet, the bounder-synchronous assumption can be unrealistic. Some recent

[1] A 30-bit security level here means that a shard-forming probability with a Byzantine node ratio exceeding the security threshold is $\leq 2^{-30}$.

works [22, 40, 52] advocate reducing the shard size by allowing more than f Byzantine nodes to be sampled into a shard during the shard formation stage and resolve/recover it once the security-violated shard is detected. However, the recovery process is costly, and if security violations happen frequently, the sharding system can be inactive for a long duration while recovering security-violated shards. For instance, when GearBox [22] can achieve the shard size $m = 72$, the probability of a shard violating liveness reaches up to 65.78%. GearBox recovers a liveness-violated shard by injecting more nodes into the shard, but this recovery process takes several hours (or even a couple of weeks [25]), delaying shard availability until nodes synchronize the states. Such frequent security violations significantly diminish the system's appeal in practice.

1.1 Arete Overview and Contributions

This work presents Arete, a highly scalable sharding protocol allowing the creation of lightweight shards securely. The key idea is to *reduce the shard size by securely increasing the security threshold f* without making any new assumptions or reducing the adversarial capacity. To this end, Arete consists of two building blocks.

(1) Ordering-processing sharding scheme. Previous sharding protocols employ every shard to process (i.e., disperse and execute) and order transactions. Under partial synchrony/asynchrony, this upper-bounds the security threshold f of shards to 1/3 as every shard needs to run consensus. We observe that, while every shard should perform its data dissemination and execution, there is no need to ask every shard to order its transactions by itself; the ordering task though requires a supermajority of honest nodes as compared to transaction processing tasks, it can be extremely lightweight by replacing transactions/blocks by signed-hash digests.

Specifically, the blockchain process can be divided into three repeatedly performed tasks: data dissemination/availability, ordering, and execution. The data availability task realizes that every honest node can reconstruct an intact block if the block is finalized. The ordering task realizes that all honest nodes must output the same transaction order. The execution task realizes that the SMR has the correct output state after executing the ordered transactions. Assuming the network is not synchronous, while the ordering task requires $f < 1/3$ to tolerate equivocations performed by Byzantine nodes (e.g., voting for two conflicting blocks), the data availability and execution tasks allow $f < 1/2$ as there is no equivocation problem for them. Moreover, the ordering task only requires slight resources in communication and computation (such as broadcasting and handling metadata of a block), but the data availability and execution tasks are always resource-intensive. For instance, the data availability and execution tasks require extensive bandwidth, computation, and storage to disperse and execute intact transactions, which have also been shown to be the bottleneck of SMR in recent works [19, 37, 60, 61].

Based on the above key insights, we propose a novel ordering-processing sharding architecture to increase the security threshold f of shards, allowing smaller shard sizes. The ordering-processing sharding scheme completely decouples data dissemination, ordering, and execution of SMR, and shards these tasks based on their required security thresholds and resources. To elaborate, there are two types of shards in Arete: a single *ordering shard* and multiple *processing shards*. The ordering shard runs a BFT consensus protocol to determine a global transaction order, tolerating up to 1/3 Byzantine nodes. Processing shards are responsible for dispersing intact transactions and executing the globally ordered transactions, each tolerating up to 1/2 Byzantine nodes. A higher threshold enables us to form more processing shards with smaller sizes, eventually enhancing the scalability of our shared system. Beneficial from the ordering-processing sharding scheme, Arete adopts a *certify-order-execute (COE)* architecture to efficiently finalize transactions (§ 5). The COE architecture consists of three stages corresponding to the three tasks in the SMR. Briefly, each processing shard first certifies transactions by ensuring at least one honest node has intact transaction data (performing data availability task). The ordering shard then runs consensus to establish a global order for these certified transactions (performing ordering task). Finally, processing shards execute and finalize these globally ordered transactions (performing execution task). It is worth noting that we are not the first to explore decoupling technology for blockchains, and many previous works (partially) decouple SMR to achieve high performance. However, they either fail to be scalable to accommodate (even) hundreds of nodes, or have to make stronger trust assumptions. Arete is the first decoupling architecture that is highly scalable with no extra trust assumptions. We provide a detailed comparison between Arete and related works in § 8. Notably, our ordering-processing sharding and the COE architecture bring five useful features:

- **F1: Improved fault tolerance**. Without running a consensus, processing shards tolerate up to half Byzantine nodes. A higher threshold allows the creation of more lightweight shards. For instance, with $n = 1000$, Arete can create 13 shards while Omniledger [38] can only create two shards.
- **F2: Asynchronous deterministic processing**. Without a consensus task, processing shards operate completely asynchronously without making any timing assumptions and requiring randomization to avoid the FLP impossibility [26].
- **F3: Lightweight ordering**. Nodes in the ordering shard neither receive transactions nor compute states for transactions. When only needed to handle lightweight signed digest, the ordering shard is scalable to service more processing shards. Our theoretical analysis shows the system needs *at least* 283 processing shards before the ordering shard handles data volume similar to a processing shard [75].
- **F4: Shard autonomy**. Processing shards disseminate and execute transactions independently without interfering with each other. Each of them can autonomously utilize their resources as the ordering shard orders varying numbers of transactions for them based on their speeds in processing transactions.
- **F5: Lock-free execution**. Cross-shard transactions are notorious in sharding systems as they involve data managed by multiple shards and introduce a cost cross-shard protocol (e.g., the lock-based two-phase commit). In Arete, with a global order in the ordering shard, processing shards can consistently execute cross-shard transactions without locking relevant states. This significantly enhances the practicality of Arete, particularly in scenarios where popular smart contracts (e.g., cryptocurrency exchanges [17] and Uniswap [64]) are frequently accessed.

(2) Safety-liveness separation. Next, motivated by GearBox [22], Arete separates the safety and liveness of processing shards to further increase the safety threshold. Informally, safety indicates honest nodes store the same ledger prefix, and liveness indicates the shard is available to handle transactions. In Arete, the security of a processing shard is defined as a tuple (f_S, f_L), in which: (i) f_S represents the safety threshold, ensuring safety as long as no more than f_S Byzantine nodes exist in the shard; (ii) f_L represents the liveness threshold, ensuring liveness as long as no more than f_L Byzantine nodes exist in the shard. The safety-liveness separation empowers us to adjust f_S and f_L to offer a higher f_S fraction. Specifically, the security thresholds in processing shards are set to $f_S > f_L$. When forming processing shards, Arete allows no more than f_S (but can exceed f_L) Byzantine nodes in a processing shard, which can reduce the size of processing shards due to a larger f_S. In this case, a processing shard ensures safety statistically but could temporarily violate liveness. The temporary existence of liveness violation is admissible because, in a real-world system, temporary unavailability can be caused by benign reasons, such as routine maintenance and upgrades. Remarkably, Arete can still guarantee the liveness of the system eventually even though liveness-violated processing shards are formed. This is because our ordering shard can detect and recover liveness-violated processing shards, making all processing shards eventually available to handle transactions.

The safety-liveness separation, however, inevitably forms liveness-violated shards, and the system needs to recover these shards. Fortunately, we notice that recovering shards would become less costly if it happens rarely. To evaluate the cost of shard recovery, we define a new property, called *$\mathcal{P}$-probabilistic liveness* (§ 2.3), for the sharding system that separates safety and liveness (e.g., GearBox [22]). Informally, it represents the probability of creating a new shard that guarantees liveness. In a practical scenario, a high $\mathcal{P}$-probabilistic liveness is acceptable. For instance, Amazon Web Service claims that 0.9999-probabilistic availability (i.e., 0.9999-probabilistic liveness as claimed in [75]) is deemed acceptable for a commercial corporation [9]. Therefore, Arete significantly reduces the cost of shard recovery caused by the safety-liveness separation. Notably, since processing shards can tolerate $f < 1/2$ corruptions, with safety-liveness separation, Arete only requires $f_S + f_L < 1$ under a partially synchronous/asynchronous model (see § 2.2 for more details), whereas the safety-liveness separation used in GearBox necessitates $f_S + 2f_L < 1$. As shown in Table 1, although both Arete and GearBox can achieve the shard size $m = 72$ by adjusting $f_S = 0.57$, Arete ensures 0.9999-probabilistic liveness while GearBox only achieves 0.3422-probabilistic liveness.

In summary, we make the following contributions:

- We propose Arete, a highly scalable blockchain sharding protocol allowing the system to create optimal-size shards to horizontally scale transaction handling (§ 4). Compared to existing solutions, Arete enhances the blockchain scalability without weakening the resilience to Byzantine failure or making synchrony assumptions, while maintaining low shard recovery costs.
- We propose a new COE architecture to efficiently finalize transactions (§ 5). Empowered by our sharding protocol, the COE architecture can fully harness the resources of a large number of nodes, bringing remarkable performance enhancements in terms of transaction throughput and cross-shard confirmation latency.
- We provide a full implementation of Arete and compare it with two representative sharding protocols: GearBox [22] that separates safety and liveness and RIVET [21] that separates the ordering shard from processing shards. We conducted experiments under a geo-distributed AWS environment (§ 7). The experiment results highlight the efficacy of our protocol. When ensuring a 0.9999-probabilistic liveness, Arete is allowed to create 20 processing shards with 500 nodes in total whereas GearBox can only create 7 shards, and RIVET can create 16 shards. Remarkably, Arete can achieve $180K$ raw transactions per second (4× improvement than GearBox and 1.4× improvement than RIVET), near intra-shard latency and 10× less cross-shard latency compared to GearBox, and better intra-shard and cross-shard latency than RIVET.

2 PRELIMINARIES

Throughout this paper, we assume there are n nodes (of which s proportion are Byzantine) in the system. We consider a partially synchronous network model [24], unless explicitly stated otherwise.

2.1 BFT SMR

A BFT SMR is a fundamental approach in distributed computing for building Byzantine fault-tolerant systems. It commits transactions for clients with the following security properties guarantee [45, 46]:

Definition 1 (Safety). *If sequences of transactions $(tx_1, \cdots, tx_j)$ and $(tx'_1, \cdots, tx'_{j'})$ are committed by two honest nodes, then $tx_i = tx'_i$ for all $i \leq min\{j, j'\}$, namely, any two honest nodes commit the same prefix ledger.*

Definition 2 (Liveness). *If a transaction tx is sent to at least one honest node, tx will be eventually committed by all honest nodes.*

The fault tolerance of an SMR is constrained by a security threshold f, indicating the SMR ensures safety and liveness if the ratio of Byzantine nodes in the system is up to f.

Implementation of SMR. An SMR can be implemented with three repeatedly performed tasks [19]:

- *Data dissemination:* Nodes disperse transactions received from clients to others. This task ensures that all nodes can eventually retrieve the transaction data consistently if the disperser is honest [49, Definition 3.2].
- *Ordering:* Nodes order the dispersed transactions and output the ordered transactions into their log (namely, a tamper-proof ledger in the blockchain context). The ordering task must ensure that every honest node agrees on the same order of transactions.
- *Execution:* After transactions are ordered, nodes execute them with an execution engine (e.g., EVM [27] and MoveVM [7]). The execution results must be consistent and externally verified.

Fault tolerance of each task. A BFT SMR must be designed to tolerate equivocations where Byzantine nodes vote for two conflicting blocks to break the safety property. The requirement of tolerating equivocations degrades the security threshold of SMR. For instance, it is well-known that the non-synchronous SMR can only tolerate at most a third of Byzantine nodes, i.e., $f < 1/3$ [24]. When $f < 1/3$ is the fault tolerance upper bound of an SMR coupling all three SMR tasks, a decoupled SMR can achieve enhanced fault tolerance [69].

To elaborate, assuming the network is not synchronous, while the ordering task can tolerate only a third of Byzantine nodes, the data dissemination and execution tasks can tolerate up to half of Byzantine nodes [16, 21, 30, 69]. The reason is that equivocations only exist in the ordering task. Intuitively, non-equivocation means that every node can only send the same message to different nodes in each round, in which at most one block can be committed in each round. Therefore, as long as there are more than 1/2 (honest) nodes following the protocol, the system can continuously commit blocks of transactions in the same order. We extend them to adapt to the safety-liveness separation condition below.

2.2 Safety-Liveness Separation in SMR

The classic SMR uses one security threshold f to indicate both safety and liveness fault tolerances simultaneously. Some recent SMR protocols [22, 46] consider fault tolerance thresholds for safety and liveness separately to achieve a higher threshold for one property while relaxing that of another. Specifically, two thresholds can be defined when separating safety and liveness in an SMR:

Definition 3 (Safety Threshold f_S). *The safety threshold f_S defines an upper bound of the ratio of Byzantine nodes that an SMR can tolerate while still ensuring safety.*

Definition 4 (Liveness Threshold f_L). *The liveness threshold f_L defines an upper bound of the ratio of Byzantine nodes that an SMR can tolerate while still ensuring liveness.*

When separating safety and liveness, each SMR task's fault tolerance can be further clarified. Specifically, f_S and f_L satisfy $f_S+f_L < 1$ in data dissemination and execution tasks, while $f_S + 2f_L < 1$ is required in the ordering task. One can prove these tolerance relations by setting $f_S = f_L = f$. We leave the rigorous proof to our full version [75]. Note that the definitions of safety and liveness differ slightly for different tasks in distinct literature. For instance, safety of the data dissemination is formally defined as commitment-binding in [49], which similarly specifies that honest nodes commit the same block in the same round. For simplicity, we broadly use safety (Definition 1) and liveness (Definition 2) for all SMR tasks.

2.3 Security Properties in Sharding Systems

The safety and liveness properties mentioned above are defined for a single blockchain/SMR. Sharding divides nodes into groups to maintain multiple blockchains. Followed by [74], we adopt the following security properties defined for a sharding system:

Definition 5 (Sharding Safety). *The sharding safety requires: (i) any two honest nodes of the same shard maintain the same prefix ledger, and (ii) any two honest nodes from two different shards have the same finalization sequence and operation (commit or abort) for all cross-shard transactions involving the two shards.*

Definition 6 (Sharding Liveness). *Transactions sent to honest nodes are eventually finalized by their relevant shards.*

Additionally, as discussed in § 1, a sharding system trading liveness threshold f_L for safety threshold f_S might create and need to recover liveness-violated shards. To evaluate the cost of shard recovery, we define $\mathcal{P}$-probabilistic liveness :

Definition 7 ($\mathcal{P}$-probabilistic liveness). *A blockchain sharding system possesses $\mathcal{P}$-probabilistic liveness if the probability of creating a new shard that guarantees liveness is at least $\mathcal{P}$.*

Intuitively, a higher $\mathcal{P}$-probabilistic liveness indicates a lower shard recovery cost as a newly created shard is more likely to guarantee liveness. For instance, a 0.9999-probabilistic liveness indicates when creating 10,000 shards, there is only 1 shard violating liveness in expectation. It is worth emphasizing that a $\mathcal{P}$-probabilistic liveness does not signify a system guarantee of sharding liveness under a $\mathcal{P}$ probability, as the system can recover temporary liveness-violated shards via the reconfiguration mechanism (see § 4.2 for more details). Therefore, a sharding system with a $\mathcal{P}$-probabilistic liveness can still guarantee the same level of security as previous sharding protocols [22, 38, 72].

3 BUILDING A SCALABLE SHARDING SYSTEM WITH OPTIMAL-SIZE SHARDS

A blockchain sharding system divides nodes into multiple shards to handle transactions in parallel. Intuitively, with more lightweight shards created, the sharding system has higher parallelism and becomes more scalable. This section will present how to enable the system to create more shards by reducing the shard size.

3.1 Calculate the Minimum Shard Size

Like prior works [22, 38, 72], we can employ the hypergeometric distribution to calculate the probability of forming an insecure shard with size m in a system with n nodes (where at most s fraction are Byzantine). Specifically, denote FAU as the event where the insecure shard contains more than f Byzantine nodes, with f representing the security threshold. The probability of FAU happening is:

$$Pr[FAU] = \sum_{x=\lceil mf \rceil}^{m} \frac{\binom{ns}{x}\binom{n-ns}{m-x}}{\binom{n}{m}}. \tag{1}$$

We hope to keep the probability of forming faulty shards negligible, such that a shard can guarantee its security properties under a high-security level. In particular, we define a security parameter λ and hope to satisfy the following formulation:

$$Pr[FAU]_{m=m^*} \leq 2^{-\lambda}. \tag{2}$$

Equation (2) can be used to evaluate the minimum shard size m^* that enables shards to ensure security properties with a high probability, i.e., λ-bit security. Specifically, given n, s, f, and λ, the minimum shard size m^* is the minimum m that satisfies Equation (2).

3.2 Existing Solutions and Our Key Insights

Equations (1)-(2) indicate the relationships among n, λ, s, f, and m^*. We focus on the impacts of s and f on m^* as distinct sharding protocols can have the same n and λ but assume varying s and f to reduce m^*. Shortly, we can conclude that a smaller s or a larger f allows the system to derive a smaller m^*. This insight has actually been noted by many representative sharding protocols. We compare their solutions for reducing shard size in Table 1. Here, k is the number of shards that can be created, and $Pr[f_L]$ denotes the probability that no more than an f_L fraction of Byzantine nodes are assigned to a shard. To elaborate further, $Pr[f_L]$ indicates the probability that the liveness of a newly created shard is guaranteed, which equals $Pr[f_L]$-probabilistic liveness as defined in Definition 7.

Table 1: Comparison of sharding protocols on the total Byzantine ratio s, safety threshold f_S, liveness threshold f_L, minimum shard size m^*, shard number k, ensured $Pr[f_L]$-probabilistic liveness, and network assumption. Assume $n = 1000$ nodes in the system and a $\lambda = 30$ security parameter.

Protocol	s	f_S	f_L	m^*	k	$Pr[f_L]$	network assumption
Omniledger [38]	25%	33%	33%	486	2	$1 - 2^{-30}$	partial sync.
RapidChain [72]	33%	49%	49%	247	4	$1 - 2^{-30}$	sync.
AHL [20]	30%	49%	49%	182	5	$1 - 2^{-30}$	partial sync.
RIVET [21]	33%	49%	49%	247	4	$1 - 2^{-30}$	partial sync.
GearBox [22]	30%	39%	30%	477	2	0.5768	partial sync.
	25%	35%	32%	403	2	0.9999	
	25%	57%	21%	72	13	0.3422	
Arete (ours)	30%	54%	45%	125	8	0.9999	partial sync.
	25%	57%	42%	72	13	0.9999	

To reduce shard size, many protocols [20, 38, 72] make stronger assumptions. Omniledger [38] assumes a smaller s to reduce m^*. RapidChain [72] assumes a synchronous network by which f can be increased to 49% via a synchronous BFT protocol. AHL [20] relies on trusted hardware in the intra-shard consensus such that the consensus leader cannot equivocate, and thus can increase f to 49%. GearBox [22] separates the safety and liveness of a shard and trades liveness threshold f_L for safety threshold f_S to get a larger f_S. To elaborate, GearBox replaces f of Equation (1) with a larger f_S to get a smaller m^*, by which a newly created shard statistically ensures the safety but may violate the liveness due to $f_S > f_L$. Thus, GearBox only provides a $\mathcal{P}$-probabilistic liveness. All the above sharding protocols employ every shard to perform all tasks of an SMR. The overall fault tolerance of their shards f is limited by the lower-bound fault tolerance required by the ordering task, e.g., $f < 1/3$ in [22, 38] that do not assume a synchrony network or rely on trusted hardware. Recall from § 2.2 that a shard performing the ordering task with $f < 1/3$ threshold necessitates $f_S + 2f_L < 1$. Due to this limitation, GearBox only increases f_S slightly if the system aims to provide a high $\mathcal{P}$-probabilistic liveness. For instance, to achieve 0.9999-probabilistic liveness with $s = 25\%$, f_S can be set to at most 35%, allowing only two shards to be created even with the safety-liveness separation mechanism.

To free shards from the limitation of the ordering fault tolerance, RIVET [21] proposes a reference-worker sharding scheme. In RIVET, the whole sharding system implements one SMR, and only one reference shard runs the ordering task. The worker shards exclusively disseminate and execute transactions and thus can tolerate 49% Byzantine nodes. Our Arete follows such a sharding scheme but employs new mechanisms to further reduce shard size and enable efficient transaction handling (see more comparison in § 8).

Key insights in creating optimal-size shards. This paper aims to reduce the shard size to create more lightweight shards without making stronger assumptions. To create optimal-size shards, a key insight is to enable a larger fault tolerance threshold of shards. This can be done by freeing a shard from the limitation of the ordering fault tolerance and separating safety and liveness (but needed to avoid a high cost of shard recovery as in GearBox).

4 ARETE PROTOCOL

4.1 System Setting and Trust Assumption

Transaction model. We target the account transaction model because it is more powerful to support different functionalities with smart contracts. An intra-shard transaction is defined as a transaction that involves states/data from one shard. In contrast, a cross-shard transaction involves states from several shards. In this paper, we focus on the *one-shot* cross-shard transactions, such as cross-shard transfer and atomic swap transactions, which account for the majority in a realistic network [55]. A one-shot transaction can be divided into two intra-shard transactions for relevant shards to execute independently without exchanging involved states. Supporting *multi-shot* cross-shard transactions is still an open problem for all existing sharding protocols and is of independent interest.

Threat model. There are n nodes (of which s proportion are Byzantine) in the system. A Byzantine (malicious) node can behave arbitrarily to deviate from the sharding protocol. However, Byzantine nodes are computationally bounded and cannot break standard cryptographic constructions. Based on these assumptions, we say a shard guarantees safety (or liveness) statistically if the probability of forming the shard with no more than f_S (of f_L) ratio of Byzantine nodes satisfies a λ-bit security level (i.e., the probability is more than $1 - 2^{-\lambda}$). Additionally, like all sharding protocols, our protocol operates in *epochs*, which can be defined by a fixed number of blocks (e.g., 1000 blocks). At the beginning of each epoch, nodes reorganize shards through a reconfiguration mechanism to prevent adaptive attacks [53]. As GearBox [22], the reconfiguration mechanism is also used to recover liveness-violated processing shards (§ 4.2).

Network model. As our ordering shard runs a partially synchronous BFT protocol (i.e., Hotstuff [70]), we assume a partially synchronous network model, where there is an unknown global stabilization time (GST) after which all message delays between honest nodes are bounded by a known Δ [24].

4.2 Generating Optimal-size Shards with Arete

Arete is designed to achieve optimal shard size and is considered an optimal sharding protocol. By "optimal", we mean that this is the minimum shard size a sharding can achieve under the same system parameters, trust and network assumptions. The key idea is to allow a larger safety threshold f_S. To achieve this, Arete consists of two building blocks: ordering-processing sharding scheme and safety-liveness separation. Figure 1 gives an overview of Arete.

(1) Ordering-processing sharding scheme. In Arete, there are two types of shards: *one* ordering shard and *multiple* processing shards. An ordering shard performs the ordering task of SMR. It runs a BFT consensus protocol to establish a *global* transaction order for the system. Processing shards perform the data dissemination and execution tasks of SMR. They work as the ledger maintainer and transaction executor.

On a high level, our sharding protocol *shards ledger maintenance and transaction execution but not consensus*. By decoupling transaction processing (i.e., dispersing and executing) from ordering, Arete enables processing shards to be free from consensus. When there is no equivocation, each processing shard can tolerate up to $f < 1/2$ ratio of Byzantine nodes even under a non-synchronous network model [16, 69]. The improved fault tolerance threshold allows a

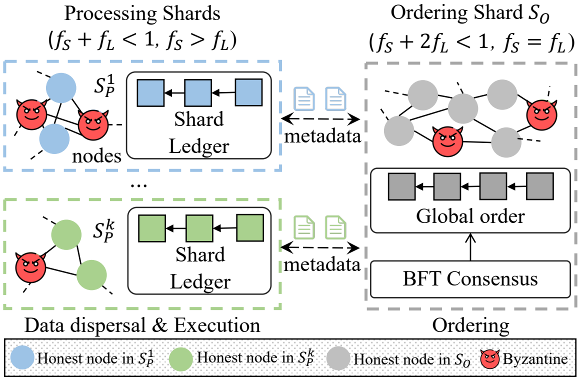


Figure 1: Arete overview: the system is divided into one ordering shard and k processing shards $\{S_P^1 \cdots S_P^k\}$. The ordering shard runs a BFT consensus to globally order transactions, tolerating up to $f_S = f_L < 1/3$ Byzantine nodes. A processing shard performs the data dissemination and execution tasks, tolerating up to $f_S \geq 1/2$ Byzantine nodes.

smaller shard size and the creation of more shards as we analyzed in § 3. Note that the ordering shard running the consensus protocol is designed to statistically guarantee safety and liveness and thus still requires a security threshold $f = f_S = f_L < 1/3$.

(2) Safety-liveness separation. Arete considers safety and liveness against Byzantine nodes separately in processing shards. Like GearBox, Arete guarantees the safety property of processing shards statistically, while allowing a processing shard to violate liveness for a while. Specifically, when forming processing shards, Arete uses the safety threshold f_S to compute the minimum shard size, ensuring that the proportion of malicious nodes in each processing shard does not exceed f_S. By appropriately increasing the safety threshold f_S and decreasing the liveness threshold f_L, Arete enables a newly created processing shard with optimal size to guarantee safety statistically. Unlike GearBox, Arete is designed to ensure a high $\mathcal{P}$-probabilistic liveness (thus a low shard recovery cost). Specifically, recall from § 2.2 that the safety-liveness separation enables our processing shards that only perform data dissemination and execution tasks to have $f_S + f_L < 1$. This is a significant improvement compared to the previous safety-liveness separation scheme [22, 52] which necessitate $f_S + 2f_L < 1$. As a result, when setting $f_S = 57\%$ to reduce the shard size to $m = 72$, Arete allows $f_L = 42\%$ (due to $f_S + f_L < 1$) and a relatively high 0.9999-probabilistic liveness (due to a high $f_L = 42\%$), as shown in Table 1.

Shards bootstrap. The process of bootstrapping Arete closely follows that of previous sharding protocols using the random assignment mechanism [22, 38]. We consider a permissionless setting where nodes are allowed to participate in the protocol with an identity obtained from permissionless Sybil-attack-resistant foundations (e.g., Proof-of-Stake). We assume that the identities of all nodes are public (e.g., via publishing identities on an existing blockchain or publicly available websites). Given the total number of nodes n, the security parameter λ, the assumed total ratio of Byzantine nodes s, and the required liveness threshold f_L, nodes can form shards with the shard size m calculated through Equation (1) and (2). Note that the shard sizes of the ordering shard (denoted by $m^\#$) and processing shards (denoted by m^*) are different, where $m^\# > m^*$. The ordering shard must be large enough to guarantee both safety and liveness properties statistically, where $f = f_S = f_L = 1/3$. In contrast, a processing shard can have fewer nodes, ensuring safety statistically but only providing 0.9999-probabilistic liveness. The formation of the ordering shard and processing shards is independent. Specifically, every node calculates a hash by concatenating its identity and randomness [22, 38, 44]. Nodes first form the ordering shard based on the ranking of their calculated hash values (e.g., with the $m^\#$ largest nodes forming an $m^\#$-size ordering shard). After that, the resulting hash values of the left nodes are mapped to a range $[0, 1)$, which is partitioned into $k = n/m^*$ regions to identify processing shards to which nodes belong. Note that the bootstrapping procedure in our work follows the random assignment mechanism as in previous works, and thus it can form (both ordering and processing) shards randomly.

Shards recover. Despite the potential for Arete to generate liveness-violated processing shards due to its safety-liveness separation, the shard reconfiguration mechanism comes into play to recover these liveness-violated processing shards. Our reconfiguration mechanism follows the methodology employed by GearBox and therefore inherently implies its effectiveness and security. Specifically, if the ordering shard receives messages (see § 5 for more details about the messages) from less than $f_S \cdot |S_P^{sid}| + 1$ nodes of a processing shard S_P^{sid} during an epoch, it can assign more nodes to S_P^{sid}, setting a higher liveness threshold (thus resulting in a lower safety threshold and a larger shard size). This process is iterated until the ordering shard receives messages from enough nodes of the processing shard, but when entering a new epoch, the liveness-recovered processing shard will be trimmed to the originally optimal shard size. During the recovery process, the newly assigned nodes need to synchronize the history data of the processing shard. However, Arete makes sure this recovery procedure rarely happens, e.g., 0.01% probability as we guarantee 0.9999-probabilistic liveness. To see how this high-probabilistic liveness reduces recovery costs, consider Ethereum state synchronization. Currently, a node can synchronize 450 GB of Ethereum states in around 12 hours [50]. The expected recovery cost for Arete, with 0.9999-probabilistic liveness, is $12 \times 0.0001 = 0.0012$ hours. In contrast, GearBox, which guarantees 0.3422-probabilistic liveness (Table 1), requires $12 \times 0.6578 = 7.8936$ hours for recovery, resulting in 6578 times higher recovery costs than Arete.

5 THE COE ARCHITECTURE

5.1 Overview

Similar to recent high-performance blockchain protocols [19, 23, 30, 61], the COE architecture in Arete handles transactions in three decoupled stages: disseminating transactions in the CERTIFY stage, ordering transactions in the ORDER stage, and executing transactions in the EXECUTE stage. However, unlike these protocols, Arete allows multiple processing shards to disseminate (and execute) transactions in parallel while a single ordering shard orders transactions.

Figure 2 shows the overview of the COE architecture. To elaborate, each processing shard disseminates transactions within the shard itself, forming *certificates* of these disseminated transactions

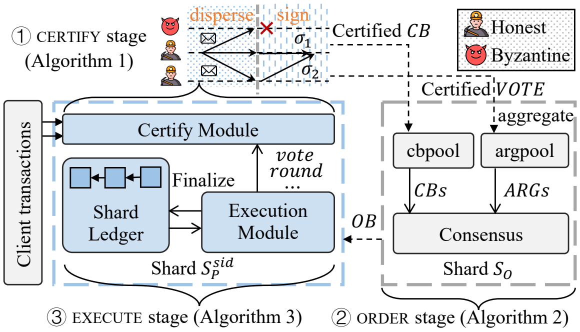


Figure 2: The COE architecture overview: ① The CERTIFY stage: each processing shard disseminates and generates certified messages, performing the data dissemination task. ② The ORDER stage: the ordering shard performs the ordering task to establish a global order. ③ The EXECUTE stage: each processing shard executes and finalizes the ordered transactions.

(the CERTIFY stage ①). The certificates consist of a quorum of signatures from nodes in the processing shard, ensuring that at least one honest node possesses intact transactions. The ordering shard runs a consensus protocol to order these certified transactions, ensuring every honest node has a consistent order for the relevant transactions (the ORDER stage ②). The ordered transactions are then executed by processing shards consistently (the EXECUTE stage ③).

To understand how the COE architecture works efficiently, notice that intact transactions are disseminated only during the CERTIFY stage, while the ORDER stage only needs to order extremely lightweight metadata (i.e., signed-hash digests), and EXECUTE stage is performed locally without involving transaction dissemination. This is because the certificates generated in the CERTIFY stage guarantee data (transactions) availability such that 1) nodes in the ordering shard can verify the validity of ordered transactions without possessing them, and 2) nodes in processing shard can retrieve the intact transactions for execution with the certified metadata. We elaborate on each stage below.

5.2 Architecture Specification

In Arete, nodes perform data dissemination, ordering, and execution tasks in the CERTIFY, ORDER, and EXECUTE stages respectively.

(1) CERTIFY stage (Algorithm 1). In this stage, nodes in every processing shard disseminate intact transactions within the shard and generate many lightweight certificates that are then forwarded to the ordering shard for ordering. Specifically, the CERTIFY stage is performed in a leaderless manner. Each node N_j^{sid} in a processing shard S_P^{sid} can create a new *execution block* $EB = \langle itxs, ctxs, sid, j\rangle$ with intra-shard transactions *itxs* and cross-shard transactions *ctxs* directly received from clients (such a direct assignment can be achieved easily based on the addresses of a transaction's sender and receiver), and then broadcast *EB* attached with its signature σ_j (lines 1-4). Once a node receives *EB*, it helps to certify that it has received intact transaction data of *EB* (lines 6-9). Specifically, it signs a CERTIFIED message *ct* that contains the digest of EB_{dst}, the metadata of cross-shard transactions in M_{ctxs}, and the shard ID *sid*,

Algorithm 1 CERTIFY stage for Node N_j^{sid} in shard S_P^{sid}

```
▷ certify new transactions
1: upon certify(transactions $txs$) do
2:     ▷ separate $txs$ into intra-shard transactions $itxs$ and cross-shard transactions $ctxs$ based on their involved processing shards
3:     $EB \leftarrow \langle itxs, ctxs, sid, j\rangle$
4:     disseminate $\langle \text{CERTIFY}, EB\rangle_{\sigma_j}$ to $S_P^{sid}$
5:     $certified_{EB} \leftarrow \{\}$
6: upon receiving $\langle \text{CERTIFY}, EB\rangle_{\sigma_k}$ do
7:     $EB_{dst} \leftarrow$ compute $EB$'s digest
8:     $M_{ctxs} \leftarrow \{\text{getMetadata}(ctx)\ |\ \forall ctx \in EB.ctxs\}$
9:     send $ct = \langle \text{CERTIFIED}, EB_{dst}, M_{ctxs}, sid\rangle_{\sigma_j}$ to Node $k$
10: upon receiving a CERTIFIED message $ct$ do
11:     $certified_{EB} \leftarrow certified_{EB} \cup \{ct\}$
12:     if $|certified_{EB}| \geq f_S \cdot |S_P^{sid}|$ then
13:         $CB \leftarrow \langle EB_{dst}, M_{ctxs}, sid, \sigma_j \cup \{certified_{EB}.\sigma\}\rangle$
14:         send $CB$ to the ordering shard
▷ certify the ordered cross-shard transactions
15: upon receiving a EXEVOTE message $ev$ from its execution module do
16:     disseminate $\langle \text{CVOTE}, ev.vote, ev.round, sid, j\rangle_{\sigma_j}$ to $S_P^{sid}$
17:     $voted \leftarrow \{\}$
18: upon receiving $\langle \text{CVOTE}, vote, round, sid, k\rangle_{\sigma_k}$ do
19:     if $vote$ is correct then
20:         send $\langle \text{CVOTED}, vote, round, sid\rangle_{\sigma_j}$ to Node $k$
21: upon receiving a CVOTED message $vt$ do
22:     $voted \leftarrow voted \cup \{vt\}$
23:     if $|voted| \geq f_S \cdot |S_P^{sid}|$ then
24:         $VOTE \leftarrow \langle vote, round, sid, \sigma_j \cup \{voted.\sigma\}\rangle$
25:         send $VOTE$ to the ordering shard
```

and sends the signed *ct* back to the sender node. The creator of *EB* (i.e., N_j^{sid} in this example) can then use these signed messages to create a certificate block $CB = \langle EB_{dst}, M_{ctxs}, sid, \sigma_{set}^P\rangle$ that contains $f_S \cdot |S_P^{sid}| + 1$ signatures σ_{set}^P, and send *CB* to the ordering shard (lines 10-14). Since the quorum $f_S \cdot |S_P^{sid}| + 1$ of signatures ensures the data availability of transactions (see Lemma 1 below), the ordering shard now can use the more lightweight certificate blocks rather than execution blocks for ordering.

LEMMA 1 (DATA AVAILABILITY). *Given a processing shard with $|S_P^{sid}|$ nodes and a safety threshold f_S, any honest node can recover an intact execution block EB_1 if EB_1's associated certificate block contains at least $f_S \cdot |S_P^{sid}| + 1$ signatures from distinct nodes of S_P^{sid}.*

Proof: Since Arete ensures that there are no more than $f_S \cdot |S_P^{sid}|$ malicious nodes in each processing shard, the quorum (i.e., $f_S \cdot |S_P^{sid}| + 1$) of signatures ensures that at least one honest node has the intact execution block. This enables all nodes in S_P^{sid} to eventually obtain the block by synchronizing it from the honest node, ensuring data availability. □

Since blockchain sharding introduces cross-shard transactions, nodes in a processing shard also need to certify the *VOTE* results of cross-shard transactions (lines 15-25). A *VOTE* result is a data structure mapping cross-shard transactions to the locally-executed results (i.e., success or failure) and will be aggregated by the ordering shard to determine if its corresponding cross-shard transactions can be committed (see the ORDER stage below). Specifically, upon receiving a EXEVOTE message from its execution module, N_j^{sid} disseminates its *VOTE* result and collects signatures from other nodes in S_P^{sid}. Similarly, the node then sends the *VOTE* result to the ordering shard after it collects at least $f_S \cdot |S_P^{sid}| + 1$ signatures. The quorum of

Algorithm 2 ORDER stage for the ordering shard S_O

```
    Local data: voteRounds    ▷ map processing shard IDs to the latest consensus
    rounds of VOTE results received from processing shards
 1: upon receiving a CB from shard $S_P^{sid}$ do
 2:     if Verify($CB, CB.\sigma_{set}^{P}$) then
 3:         store CB into cbpool
    ▷ aggregate VOTE results for cross-shard transactions
 4: upon receiving a VOTE result do
 5:     if Verify($VOTE, VOTE.\sigma_{set}^{P}$) then
 6:         if voteRounds[VOTE.sid] + 1 = VOTE.round then
 7:             voteRounds[VOTE.sid] ← VOTE.round
 8:             ARG ← fetch aggregator of VOTE.round
 9:             ARG ← AggreateVote(ARG, VOTE.vote)
10:             if ARG is ready to be finalized then
11:                 store ARG into argpool
12:         elseIf voteRounds[VOTE.sid] < VOTE.round then
13:             synchronize missing votes from $S_P^{VOTE.sid}$
14:             update the relevant aggregators
    ▷ order new transactions
15: when creating a new ordering block OB do
16:     ARGs ← fetch aggregated votes from its argpool
17:     CBs ← fetch certificate blocks from its cbpool
18:     OB ← createBlock(newRound, ARGs, CBs)
19:     coordinate a consensus instance for OB
    ▷ inform processing shards
20: upon finalizing the new ordering block OB do
21:     send $OB = \langle ARGs, \hat{EB}_{dsts}, \hat{M}_{ctxs}, r, \sigma_{set}^{O} \rangle$ to processing shards

22: function AggreateVote(ARG, vote)
23:     for $\forall M_{ctx} \in vote.key()$ do
24:         if ARG.vote.contains($M_{ctx}$) then
25:             ARG.vote[$M_{ctx}$] ← ARG.vote[$M_{ctx}$] && vote[$M_{ctx}$]
26:         else
27:             ARG.vote[$M_{ctx}$] ← vote[$M_{ctx}$]
28:     return ARG
```

signatures ensures that at least one honest node certifies the correctness of the *VOTE* result. Therefore, anyone (in the ordering shard) can verify the correctness of a *VOTE* result without re-executing the corresponding transactions.

(2) ORDER stage (Algorithm 2). The ORDER stage then helps globally order transactions indicated by certificate blocks from processing shards. The COE architecture orders transactions round-by-round. In each consensus round r, the ordering shard S_O runs an instance of consensus to create and finalize an *ordering block OB*.

Since Arete is agnostic to the consensus protocol, we omit the procedure of consensus on finalizing the ordering block in Algorithm 2, which depends on the specific consensus protocol. For instance, when running Hotstuff [70] in S_O, a leader $\mathcal{L}$ is designated to create *OB* (lines 15-19). Specifically, $\mathcal{L}$ fetches and orders a set of certificate blocks from its certificate pool cbpool, where cbpool stores all valid certificate blocks received from processing shards (lines 1-3). Note that a certificate block is considered valid if it contains a quorum of (i.e., at least $f_S \cdot |S_P^{sid}| + 1$) signatures from its processing shard S_P^{sid}. Once the fetched certificate blocks *CBs* are ordered, the local orders of transactions inside *CBs* are retained unchanged and combined into a global order. Then $\mathcal{L}$ coordinates several stages of Hotstuff to finalize *OB*.

Arete also assigns the ordering shard to coordinate the finalization of cross-shard transactions. Arete adopts an *asynchronous cross-shard commit approach*, where the ordering shard can move to the next consensus round without waiting to receive *VOTE* results of the current round from all relevant processing shards, i.e., cross-shard transactions are finalized in a non-blocking way. To this end,

Algorithm 3 (Simplified) EXECUTE stage for Node N_j^{sid} in shard S_P^{sid}

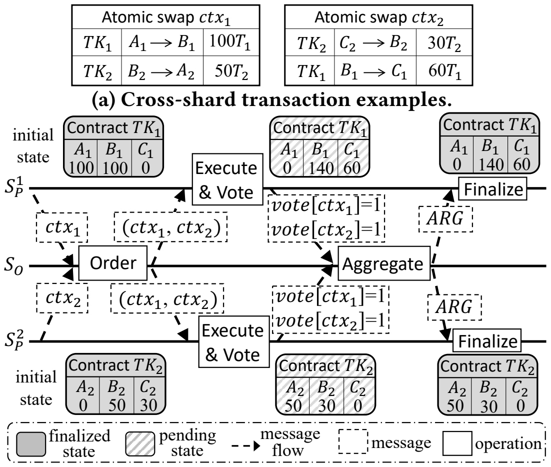
```
    ▷ execute ordered transactions, $OB = \langle ARGs, \hat{EB}_{dsts}, \hat{M}_{ctxs}, r, \sigma_{set}^{O} \rangle$
1: upon receiving a new ordering block OB do
2:     if OB.r > orderRound + 1 and Verify($OB, OB.\sigma_{set}^{O}$) then
3:         synchronize missing ordering blocks from $S_O$
4:         update orderRound with the synchronized ordering blocks
5:     elseIf OB.r == orderRound + 1 and Verify($OB, OB.\sigma_{set}^{O}$)
    ▷ Step 1: handle aggregated results ARGs
6:         finalize cross-shard transactions of ARGs for previous rounds
    ▷ Step 2: handle intra-shard transactions
7:         execute and finalize intra-shard transactions of $\hat{EB}_{dsts}$
    ▷ Step 3: handle cross-shard transactions
8:         execute and vote for cross-shard transactions of $\hat{M}_{ctxs}$
9:         generate EXEVOTE message and send it to the certification module
```

nodes trace the latest consensus round of the *VOTE* result from each processing shard, i.e., *voteRounds*. When receiving a certified *VOTE* result, nodes aggregate it into an aggregator *ARG* that is associated with a consensus round *ARG.round* (lines 4-14).

The aggregator *ARG* will serve as a reference to indicate if a cross-shard transaction can be committed. Briefly speaking, the aggregation process AggregateVote (lines 22-28) is to perform AND operation on the execution results (where 1 means success and 0 means failure) of cross-shard transactions from all relevant processing shards. If the final value corresponding to a cross-shard transaction in an aggregator *ARG* is 1, then the cross-shard transaction is executed successfully by all relevant processing shards; in contrast, 0 indicates at least one processing shard fails to execute the cross-shard transaction. Once a node receives all involved *VOTE* results for a previous consensus round *VOTE.round*, the corresponding aggregator *ARG* is ready to be finalized and stored in the node's aggregator pool argpool (Line 11). When creating a new ordering block *OB*, the leader $\mathcal{L}$ also fetches aggregators from argpool and orders them in the block based on their corresponding consensus rounds (Line 16). The new ordering block $OB = \langle ARGs, \hat{EB}_{dsts}, \hat{M}_{ctxs}, r, \sigma_{set}^{O} \rangle$ is then sent to processing shards for execution (Lines 20-21), where *ARGs* is a set of aggregations of execution results for cross-shard transactions, $\hat{EB}_{dsts}$ is a list of execution block digests, $\hat{M}_{ctxs}$ is a list of cross-shard transactions metadata, and σ_{set}^{O} is a quorum (i.e., $2f \cdot |S_O| + 1$) of signatures.

(3) EXECUTE stage (Algorithm 3). In this stage, processing shards execute and finalize the ordered transactions implied by the ordering block. Thanks to the established global order, the process of transaction execution can be simple. Algorithm 3 illustrates a simplified workflow of this stage, and we leave the detailed implementation in our full version [75] due to the page limitation.

Before executing transactions in a received ordering block *OB*, node N_j^{sid} in S_P^{sid} traces its locally latest consensus round *orderRound* and uses a synchronizer to ensure not miss any ordering blocks it involves (lines 1-4). Finalizing transactions consists of three steps (lines 5-9). Abstractly, N_j^{sid} first finalizes the cross-shard transactions for previous rounds (in *ARGs*) because *ARGs* aggregates execution results from all relevant processing shards and its corresponding cross-shard transactions are ready for finalization (line 6). Then, N_j^{sid} executes the ordered intra-shard transactions in $\hat{EB}_{dsts}$ (line 7). Since intra-shard transactions only access the data managed



(a) Cross-shard transaction examples.

(b) Lock-free execution.

Figure 3: The workflow of handling cross-shard transactions in Arete: (a) Two atomic swap transactions ctx_1 and ctx_2, both are cross-shard and involve contract TK_1 managed by shard S_P^1 and TK_2 managed by S_P^2. (b) Arete handles cross-shard transactions ctx_1 and ctx_2 via a lock-free execution approach.

by S_P^{sid}, N_j^{sid} is able to finalize them immediately if they do not involve any uncommitted data (generated by unfinalized cross-shard transactions). Finally, N_j^{sid} executes the newly ordered cross-shard transactions of $\hat{M}_{ctxs}$ (lines 8-9). Different from intra-shard transactions, the newly cross-shard transactions cannot be finalized in this round as they rely on the execution results from other relevant processing shards. Instead, N_j^{sid} votes for its local execution results and forward them to its certification module to certify the *VOTE* result (lines 15-25 in Algorithm 1).

Example: Arete utilizes the deterministic information provided by the ordering shard to realize a lock-free execution. Figure 3 illustrates an example of handling two common cross-shard transactions ctx_1 and ctx_2, both of which involve token swapping between two token smart contracts TK_1 (managed by shard S_P^1) and TK_2 (managed by shard S_P^2). Specifically, in ctx_1, account A swaps $100T_1$ with account B for $50T_2$, and in ctx_2, account B swaps $30T_2$ with account C for $60T_1$ (Figure 3a), where T_1 and T_2 represent different tokens. First, nodes in S_P^1 and S_P^2 send ctx_1 and ctx_2 to the ordering shard S_O via their certificate blocks, respectively. S_O then orders the transactions in an ordering block and forwards the block to S_P^1 and S_P^2. With the transaction order, S_P^1 and S_P^2 consistently execute ctx_1 and ctx_2 without locking relevant states. Each processing shard then sends *VOTE* messages to S_O, indicating whether a cross-shard transaction is executed successfully. S_O aggregates the *VOTE* messages from S_P^1 and S_P^2 and sends an aggregator message *ARG* to them, indicating whether a cross-shard transaction can be committed. With *ARG*, S_P^1 and S_P^2 eventually commit ctx_1 and ctx_2, ensuring consistency and atomicity. During this process, ctx_1 and ctx_2 do not lock the states and can be executed within a consensus instance.

Compared to two-phase commit approaches, Arete's lock-free execution is more promising in a real-world system where some hotspot contracts [64] are frequently accessed. However, a lock-free execution can lead to a *cascading abortion* problem where aborting a cross-shard transaction ctx will make all following transactions conflicting with ctx aborted. While several existing solutions [29, 31] can be integrated into Arete to alleviate this issue, there are also new mitigation strategies specifically tailored for our blockchain sharding scenario. For instance, processing shards can avoid certifying new transactions that involve uncommitted states before the corresponding cross-shard transactions are finalized. In fact, the trade-off between aborts (from lock-free execution) and waits (from lock-based execution) has been systematically quantified by previous work [31], showing that the performance obtained from the lock-free execution is better than that obtained from the lock-based execution under the scenario with hotspots.

6 ANALYSIS

This section analyzes sharding safety and sharding liveness of Arete. We focus on the security guarantee within each epoch, while the security proof across epochs can be referenced in GearBox [22], as our reconfiguration mechanism follows their scheme.

6.1 Sharding Safety Analysis

Since the ordering shard in Arete runs a BFT consensus to maintain its ledger, the inherent BFT consensus guarantees that nodes in the ordering shard maintain the same prefix ledger. Besides, cross-shard transactions only modify the states of processing shards but not the ordering shard. We therefore only discuss processing shards.

Lemma 2. *For every two honest nodes N_i^{sid} and N_j^{sid} with local ledgers $\mathfrak{L}_i{}^{sid}$ and $\mathfrak{L}_j{}^{sid}$ in processing shard S_P^{sid}, Arete guarantees: either $\mathfrak{L}_i{}^{sid} \subseteq \mathfrak{L}_j{}^{sid}$ or $\mathfrak{L}_j{}^{sid} \subseteq \mathfrak{L}_i{}^{sid}$.*

Proof: For the sake of contradiction, assume there is a consensus round r_c where $\mathfrak{L}_i{}^{sid}$ finalized a list of execution blocks with the digests EB_{dsts}, and $\mathfrak{L}_j{}^{sid}$ finalized another list of execution blocks with the digests EB'_{dsts} ($EB_{dsts} \neq EB'_{dsts}$). However, since EB_{dsts} and EB'_{dsts} are the content of ordering blocks, it means the ordering shard finalizes two ordering blocks in r_c, violating the safety of the ordering shard, leading to a contradiction. □

Lemma 3 (Cross-shard Atomicity). *For any cross-shard transaction ctx that is finalized by relevant processing shards, all relevant processing shards either commit or abort ctx.*

Proof: Without loss of generality, we assume ctx involves two processing shards S_P^m and S_P^n. For the sake of contradiction, assume S_P^m commits ctx while S_P^n aborts ctx. Recall from the COE architecture that the ordering shard picks all aggregators into its ordering blocks. Thus, S_P^m receives an aggregator ARG where $ARG.vote[M_{ctx}] = 1$ while S_P^n receives an aggregator ARG' where $ARG'.vote[M_{ctx}] = 0$, or there is another cross-shard transaction ctx' conflicting with ctx such that $ARG.vote[M_{ctx'}] = 1$ and $ARG'.vote[M_{ctx'}] = 0$. Both of them mean the ordering shard commits two different ordering blocks in the same round. However, the BFT consensus ensures the safety of the ordering shard, leading to a contradiction. □

Lemma 4 (Cross-shard Consistency). *For any two cross-shard transactions ctx_1 and ctx_2 that are finalized by relevant processing shards, either ctx_1 is finalized before ctx_2 or ctx_2 is finalized before ctx_1 in all relevant processing shards.*

Proof: Without loss of generality, we assume *ctx1* and *ctx2* involve two processing shards S_P^m and S_P^n. For the sake of contradiction, assume S_P^m finalizes *ctx1* before *ctx2* while S_P^n finalizes *ctx2* before *ctx1*. Recall from the EXECUTE stage that cross-shard transactions are finalized in the order of the *VOTE* results of the aggregators. The above finalization results indicate that S_P^m and S_P^n receive different aggregators from the ordering shard, which violates the safety of the ordering shard and leads to a contradiction. □

Lemma 2 proves Arete satisfies the condition(i) of Definition 5, and Lemmas 3 and 4 prove Arete satisfies the condition(ii) of Definition 5. Therefore, we have:

Theorem 1. *Arete guarantees the sharding safety.*

6.2 Sharding Liveness Analysis

Lemma 5. *Transactions sent to honest nodes are eventually ordered by the ordering shard.*

Proof: Recall from § 4.2 that our shard reconfiguration mechanism ensures at least $f_S \cdot |S_P^{sid}| + 1$ nodes in each processing shard S_P^{sid} eventually send their certificate blocks to the ordering shard. Since at most f_S fraction of nodes is Byzantine, the above condition ensures at least one honest node can certify new transactions to the ordering shard. Therefore, every transaction sent to honest nodes can be packed into a certificate block that will be eventually ordered by the ordering shard. □

Lemma 6. *Honest nodes can obtain intact transactions for execution if these transactions have been ordered by the ordering shard.*

Proof: If transactions are ordered by the ordering shard, it means their associated certificate blocks are valid (i.e., containing a quorum of signatures). From Lemma 1, we know that a valid certificate block enables nodes to retrieve all intact transactions corresponding with the block. Therefore, any honest node can eventually obtain these intact transactions. □

Lemmas 5 and 6 jointly ensure each transaction sent to honest nodes can be handled by relevant processing shards and finalized eventually. Therefore, we have:

Theorem 2. *Arete guarantees the sharding liveness.*

7 EVALUATION

7.1 Implementation

We implement a prototype for Arete in Rust, which uses Tokio for asynchronous network, ed25519-dalek for elliptic curve-based signature, and RocksDB for persistent storage. The consensus protocol of the ordering shard adopts a variation of Hotstuff [28]. For the communications within each shard, we use TCP to realize reliable channels. For communications between the ordering shard and processing shards, we allow nodes to connect one or more nodes from the other shard randomly. Our implementation involves around 6.5K LOC, and the source codes are public with the testing scripts [73].

We compare Arete to two representative sharding protocols: 1) GearBox [22], a SOTA sharding protocol focusing on reducing the shard size; 2) RIVET [21], a sharding scheme that resembles Arete, but uses a leader-based approach for transaction dissemination and does not separate safety and liveness to reduce the shard size (see Section 8 for more comparisons). Specifically, GearBox separates safety f_S and liveness f_L in each shard, but necessitates $f_S + 2f_L < 1$. Each shard runs an intra-shard consensus protocol to commit intra-shard transactions. To handle cross-shard transactions, GearBox adopts a two-phase commit protocol, which locks the involved states of a transaction until all relevant shards finish finalizing it. In GearBox, a specific control shard coordinates the two-phase commit protocol. RIVET deploys a leader node in each processing shard to produce blocks, which are then finalized by a single ordering shard via an intra-shard consensus protocol. Unlike GearBox, RIVET adopts an optimistic cross-shard consensus. Specifically, the ordering shard first orders the cross-shard transactions, and then all relevant processing shards execute them without locking the involved states. As both GearBox and RIVET do not provide their code implementations, for a fair comparison, we implement and evaluate GearBox and RIVET based on our codebase.

Table 2: Comparisons under different numbers of nodes (s = 15%, λ = 20), shown in the format [GearBox, RIVET, Arete]

	The total number of nodes n					
	50	100	200	300	400	500
$m^{\#}$	[21, 21, 21]	[42, 42, 42]	[63, 63, 63]	[72, 72, 72]	[78, 78, 78]	[81, 81, 81]
m^*	[20, 15, 13]	[38, 23, 18]	[49, 27, 20]	[57, 29, 22]	[60, 31, 24]	[63, 31, 24]
f_L (%)	[32, 49, 41]	[31, 49, 41]	[31, 49, 42]	[31, 49, 42]	[31, 49, 43]	[31, 49, 43]
k	[2, 3, 3]	[2, 4, 5]	[4, 7, 10]	[5, 10, 13]	[6, 12, 16]	[7, 16, 20]

7.2 Experiment Setup

We run our evaluation in AWS, using c5a.8xlarge EC2 instances spread across 8 regions in the world (3 in Europe, 3 in America, and 2 in Asia). Each instance provides 32 CPUs, 64 GB RAM, and 10 Gbps of bandwidth and runs Linux Ubuntu server 20.04. We deploy one client per node in processing shards to submit transactions at a fixed rate. Each transaction has a size of 512 Bytes. In the following sections, each measurement is the average of 2 runs where shards have been running for 10 minutes to obtain a more precise result.

In the following experiments, we set the total ratio of Byzantine nodes s = 15%, the security parameter λ = 20. This allows us to create more shards to evaluate the scalability while avoiding excessive monetary expenses on EC2. For the following measurements, we set the ratio of cross-shard transactions 20% by default but also evaluate its impact on the performance (Figure 7 and Figure 8). It is worth emphasizing that many solutions [35, 39, 51, 62, 76] that are proposed to reduce the cross-shard ratio can be applied to Arete. However, we consider them an orthogonal research topic.

7.3 Scalability

We first evaluate the scalability by running different numbers of nodes n. When calculating the shard size, we make sure: (i) the ordering shard (or control shard in GearBox) always satisfies safety and liveness; (ii) the processing shards always satisfy safety, but only guarantee 0.9999-probabilistic liveness. TABLE 2 gives the configurations for this experiment, where $m^{\#}$ represents the size of the ordering (or control) shard, m^* represents the size of the processing shards, f_L represents the liveness threshold of processing shards, and k represents the number of processing shards[2].

[2] For illustration purposes, we use the term "shard" to exclusively represent the shard running consensus in GearBox and the processing shard in RIVET and Arete.

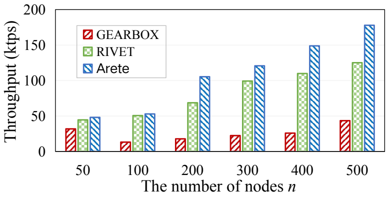


Figure 4: Throughput-Nodes

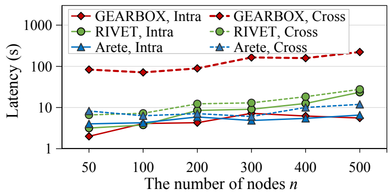


Figure 5: End-to-end latency-Nodes

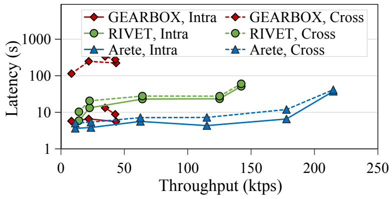


Figure 6: Throughput-Latency

Figure 4 shows the throughput under n and configurations. All compared protocols can scale the system where transaction throughput increases with the increasing number of nodes, except for the conditions between $n = 50$ and $n = 100$, where GearBox achieves higher throughput under $n = 50$ than under $n = 100$. This is because GearBox has the same number of (i.e., $k = 2$) shards but smaller shard sizes when $n = 50$ than when $n = 100$. This shows a larger shard size will compromise transaction throughput as it introduces higher communication overheads. Additionally, a smaller shard size enables the system to create more shards and achieve higher throughput. Thus, Arete achieves the best throughput with the same number of nodes, followed by RIVET and then GearBox. This shows that Arete can achieve better scalability than the compared protocols. Specifically, when $n = 500$, Arete has $k = 20$ processing shards to handle transactions in parallel and can achieve about 180K transactions per second (TPS), which is 4× improvement compared to GearBox and 1.4× improvement compared to RIVET.

Figure 5 shows end-to-end intra-shard and cross-shard latency under different n, where we start from when the client sends a transaction to when the transaction is finalized. When n increases, the latency of both intra-shard and cross-shard transactions increases since the shard size becomes larger. Compared to GearBox, Arete achieves near intra-shard confirmation latency, which varies from approximately 4s to 6s with n increases. For the cross-shard confirmation latency, Arete performs much better than GearBox due to our lock-free execution. To elaborate, the cross-shard confirmation latency of Arete varies from approximately 8s to 12s with n increases whereas GearBox requires 80s to 223s to finalize a cross-shard transaction due to its adopted two-phase commit protocol. Arete reduces the cross-shard confirmation latency by over 10 times compared to GearBox. Similarly, RIVET achieves lower cross-shard confirmation latency than GearBox thanks to its optimistic cross-shard consensus. However, due to its leader-based block production, RIVET disseminates transactions more slowly and experiences higher confirmation latency than Arete.

7.4 Performance

We then evaluate the performance in terms of transaction throughput, end-to-end intra-shard confirmation latency, and end-to-end cross-shard confirmation latency. In the following experiments, we run 500 nodes in total, and the other configurations regarding $m^{\#}$, m^{*}, f_L, and k are illustrated in TABLE 2 (in the column $n = 500$).

Fault-free performance. We first run the fault-free experiment under different workloads of the system. We use throughput-latency characteristics to depict the performance of the system. The characteristics can evaluate the capacity of a system to handle transactions. To elaborate, before the workload reaches saturated, the confirmation latency changes slightly while transaction throughput can increase noticeably as the workload enlarges; in contrast, after the workload is saturated, transaction throughput will become steady while the confirmation latency will increase noticeably as the workload enlarges. Therefore, the steady transaction throughput is used to evaluate the optimal performance of a system. Figure 6 shows the throughput-latency characteristics of compared protocols. Under $n = 500$, Arete can achieve an optimal performance of about 180K transaction throughput at an intra-shard latency below 7 seconds and a cross-shard latency below 12 seconds. In contrast, the optimal transaction throughput of GearBox is around 45K while its cross-shard latency is much higher than its intra-shard latency. Furthermore, the optimal transaction throughput of RIVET is around 125K at an intra-shard latency of around 23 seconds and a cross-shard latency of around 27 seconds. Intuitively, the high throughput of Arete benefits from smaller shard sizes and more shards. Furthermore, a larger liveness threshold f_L also enables Arete to move more quickly to the next phase since the quorum of moving to the next phase (1-f_L) decreases, and a leaderless approach amortizes the data dissemination overhead. This validates the benefits of reducing shard sizes and enhancing the transaction process with the proposed COE architecture.

Performance under cross-shard transaction ratios. We then compare the performance of distinct protocols under different ratios of cross-shard transactions (CTXs). Specifically, we set the cross-shard ratio to 20%, 40%, 60%, and 80%. A higher cross-shard ratio introduces more overheads to the system as the ordering shard (or the control shard in GearBox) needs to handle more data relevant to CTXs. Figure 7 and Figure 8 respectively show the throughput and latency under varying cross-shard ratios. We find that as the cross-shard ratio increases, the throughput of all protocols decreases, and their confirmation latency increases. However, Arete outperforms both GearBox and RIVET regardless of the cross-shard ratios.

Performance under crash faults. We finally evaluate the performance under crash faults. We conduct experiments with 0%, 10%, 20%, and 30% ratios of crash nodes in each shard respectively. A crash node neither responds to clients nor participates in the sharding protocol. Figure 9 shows the transaction throughput under different crashed ratios. We observe that crash nodes can compromise the throughput of Arete more than that of GearBox and RIVET. Specifically, the throughput of Arete drops from 180K to 88K as the ratio of crash nodes increases. In contrast, the throughput of GearBox drops slightly (approximately 45K to 38K), and the throughput of RIVET drops approximately from 125K to 86K. This indicates that nodes in Arete can contribute their bandwidth to the transaction

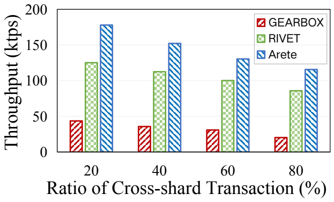


Figure 7: TPS-CTX ratios

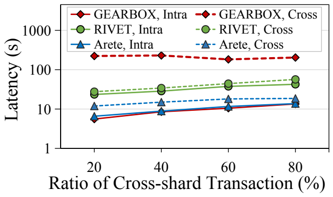


Figure 8: Latency-CTX ratios

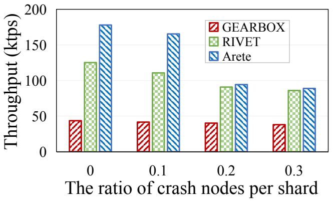


Figure 9: TPS-Crashes

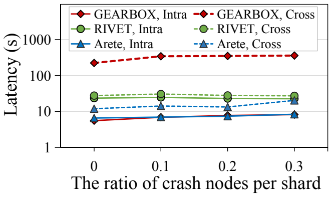


Figure 10: Latency-Crashes

throughput more effectively than both GearBox and RIVET as we deploy a leaderless approach to disseminate transactions. Figure 10 shows the latency of protocols under different crashed ratios. We observe that with more crash faults in the system, both intra-shard and cross-shard confirmation latencies of all protocols increase. Intuitively, due to fewer active nodes in each shard, a shard requires a larger ratio of nodes' signatures to move to the next stage. In this case, any slight network delay between two nodes may significantly slow down the process of finalizing transactions.

8 RELATED WORK

Blockchain sharding. Sharding protocols [2, 3, 5, 18, 20, 21, 32, 34, 38, 44, 55, 65, 68, 72] are proposed to scale blockchains. While a recent line of orthogonal sharding protocols [34–36, 43, 76] focuses on handling cross-shard transactions, this paper focuses on resolving the size-security dilemma. Many works [21, 22, 40, 52, 68] are designed to create smaller shards. CoChain [40] and DL-Chain [41] allow the creation of a large number of small corrupt shards and propose a CoC framework to finalize consensus results. Nevertheless, CoC introduces extra overheads, as handling transactions requires one more consensus instance, and each shard needs to serve multiple CoC committees. Reticulum [68] proposes a two-layer sharding protocol, where multiple process shards that propose blocks form a control shard to finalize the proposed blocks. By separating safety and liveness in process shards and resorting to the corresponding control shard for block finalization, Reticulum allows for more lightweight process shards to be created and enhances the scalability. We highlight several key differences between Reticulum and Arete. First, Reticulum relies on the synchrony network assumption for its security, while Arete operates in the partial synchrony network. Second, the process shard in Reticulum adopts a leader-based scheme to propose blocks, where only one block is proposed each time, and the leader could potentially become the bottleneck; in contrast, the processing shard in Arete adopts a leaderless scheme to produce blocks, fully utilizing nodes' resources. Last but not least, Arete employs a single ordering shard to establish a global order, facilitating the finalization of cross-shard transactions, while cross-shard transaction handling cannot benefit from Reticulum's design. RIVET [21] proposed a reference-worker sharding scheme that assigns a single reference shard for ordering, resembling our scheme but with notable distinctions. In Arete, we decouple and pipeline all SMR tasks, allowing shards to perform these tasks asynchronously and in parallel. In contrast, RIVET exclusively decouples the ordering task from the data dissemination and execution, where data dissemination and execution remain coupled and sequential. Moreover, RIVET employs an execute-order-commit architecture to handle intra-shard transactions, which not only necessitates a leader in each worker shard to coordinate the execution stage, but could also lead to a high transaction abortion due to potentially inconsistent order. In contrast, our processing shards perform in a leaderless way, and the global order established by the ordering shard can avoid the order inconsistency issue.

Order-execution decoupling. Many non-sharding blockchains decouple execution from ordering to enhance performance. Works [6, 48, 56] support parallel transaction execution before ordering, but experience a high abortion rate. The subsequent works [4, 42, 57, 58, 67] refine the abortion rate by applying deterministic ordering or reordering algorithms. Many other works adopt a consensus/ordering-free decoupling architecture for transaction execution. However, they either rely on a single trusted server ([54]) or solely support a simple transfer application [12, 47, 63]. Moreover, all the above (partially) decoupling architectures still couple the data dissemination of SMR with ordering/execution, which has been pointed out as a main bottleneck in a high-performance SMR [11, 19, 30, 71].

Disperse-order-execution decoupling. Recent works [23, 30, 66] show an incredible performance by fully decoupling SMR. However, they have limited scalability as a large network brings heavy communication overheads to nodes. The DAG-based BFT protocols [8, 10, 13, 19, 37, 59–61] also fully separate three SMR tasks. They scale the system by introducing worker nodes that must trust the primary node they associate with, introducing a stronger trust assumption. In contrast, Arete adopts a horizontal sharding architecture without bringing new trust assumptions. Furthermore, many DAG-based BFT protocols [19, 61] depend on reliable broadcast to disseminate transactions/blocks in the CERTIFY stage, as they rely on the creation of non-equivocated blocks to complete the ordering task. In contrast, Arete, like [23, 30], is free from reliable broadcast as its ordering task can be completed by the ordering shard independently of block creation.

9 CONCLUSION

This work proposed Arete, an optimal sharding protocol achieving scalable blockchains with deconstructed SMR. The extensive experiments show that Arete outperforms SOTA sharding protocols in scalability, throughput, and latency.

ACKNOWLEDGMENTS

We thank our anonymous reviewers for helpful suggestions. We also thank Ling Ren and Sourav Das for their insightful discussion about their RIVET project. This work was supported in part by NIFA award number 2021-67021-34252, the National Science Foundation (NSF) under grant CNS1846316.

REFERENCES

[1] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2020. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 106–118.

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

[3] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. On sharding permissioned blockchains. In *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 282–285.

[4] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. Parblockchain: Leveraging transaction parallelism in permissioned blockchain systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1337–1347.

[5] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. Sharper: Sharding permissioned blockchains over network clusters. In *Proceedings of the 2021 International Conference on Management of Data*. 76–88.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*. 1–15.

[7] Aptos. [n.d.]. Aptos Move VM. https://aptos.dev/en/network/blockchain/move. Accessed: 2024.

[8] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. 2024. Shoal++: High Throughput DAG BFT Can Be Fast! *arXiv preprint arXiv:2405.20488* (2024).

[9] AWS. [n.d.]. four nines scenario. https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/s-99.99-scenario.html. Accessed: 2024.

[10] Kushal Babel, Andrey Chursin, George Danezis, Lefteris Kokoris-Kogias, and Alberto Sonnino. 2023. Mysticeti: Low-Latency DAG Consensus with Fast Commit Path. *arXiv preprint arXiv:2310.14821* (2023).

[11] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 585–602.

[12] Mathieu Baudet, George Danezis, and Alberto Sonnino. 2020. Fastpay: High-performance byzantine fault tolerant settlement. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 163–177.

[13] Same Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, et al. 2023. Sui lutris: A blockchain combining broadcast and consensus. *arXiv preprint arXiv:2310.18042* (2023).

[14] Benjamin Y Chan and Elaine Shi. 2020. Streamlet: Textbook streamlined blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 1–11.

[15] Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. 2024. Byzantine consensus is Θ (n^{2}): the Dolev-Reischuk bound is tight even in partial synchrony! *Distributed Comput.* 37, 2 (2024), 89–119.

[16] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. 2012. On the (limited) power of non-equivocation. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*. 301–308.

[17] Coinbase. [n.d.]. Coinbase cryptocurrency exchange. https://www.coinbase.com/. Accessed: 2024.

[18] Tyler Crain, Christopher Natoli, and Vincent Gramoli. 2021. Red belly: A secure, fair and scalable open blockchain. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 466–483.

[19] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 34–50.

[20] 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*. 123–140.

[21] Sourav Das, Vinith Krishnan, and Ling Ren. 2020. Efficient cross-shard transaction execution in sharded blockchains. *arXiv preprint arXiv:2007.14521* (2020).

[22] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. 2022. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 683–696.

[23] Sisi Duan, Haibin Zhang, Xiao Sui, Baohan Huang, Changchun Mu, Gang Di, and Xiaoyun Wang. 2024. Dashing and Star: Byzantine fault tolerance with weak certificates. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 250–264.

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

[25] Ethereum. [n.d.]. Ethereum Synchronization Modes. https://ethereum.org/en/developers/docs/nodes-and-clients/#sync-modes. Accessed: 2024.

[26] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

[27] Ethereum Fundation. [n.d.]. Ethereum Virtual Machine (EVM). https://ethereum.org/en/developers/docs/evm/. Accessed: 2024.

[28] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2022. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *International conference on financial cryptography and data security*. Springer, 296–315.

[29] Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Dahlia Malkhi, Yu Xia, and Runtian Zhou. 2023. Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 232–244.

[30] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. 2024. Motorway: Seamless high speed BFT. *arXiv preprint arXiv:2401.10369* (2024).

[31] Zhihan Guo, Kan Wu, Cong Yan, and Xiangyao Yu. 2021. Releasing locks as early as you can: Reducing contention of hotspots by violating two-phase locking. In *Proceedings of the 2021 International Conference on Management of Data*. 658–670.

[32] Jelle Hellings and Mohammad Sadoghi. 2021. Byshard: Sharding in a byzantine environment. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2230–2243.

[33] 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*. IEEE, 1–10.

[34] Zicong Hong, Song Guo, Enyuan Zhou, Jianting Zhang, Wuhui Chen, Jinwen Liang, Jie Zhang, and Albert Zomaya. 2023. Prophet: Conflict-free sharding blockchain via byzantine-tolerant deterministic ordering. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 1–10.

[35] Huawei Huang, Xiaowen Peng, Jianzhou Zhan, Shenyang Zhang, Yue Lin, Zibin Zheng, and Song Guo. 2022. Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 1968–1977.

[36] Shan Jiang, Jiannong Cao, Cheung Leong Tung, Yuqin Wang, and Shan Wang. 2024. Sharon: Secure and efficient cross-shard transaction processing via shard rotation. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2418–2427.

[37] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 165–175.

[38] 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)*. IEEE, 583–598.

[39] Michał Król, Onur Ascigil, Sergi Rene, Alberto Sonnino, Mustafa Al-Bassam, and Etienne Rivière. 2021. Shard scheduler: Object placement and migration in sharded account-based blockchains. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 43–56.

[40] Mingzhe Li, You Lin, Jin Zhang, and Wei Wang. 2023. CoChain: High concurrency blockchain sharding via consensus on consensus. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 1–10.

[41] You Lin, Mingzhe Li, Qingsong Wei, Yong Liu, Siow Mong Rick Goh, and Jin Zhang. 2024. DL-Chain: Scalable and Stable Blockchain Sharding with High Concurrency via Dual-Layer Consensus. *arXiv preprint arXiv:2407.06882* (2024).

[42] Jian Liu, Peilun Li, Raymond Cheng, N Asokan, and Dawn Song. 2021. Parallel and asynchronous smart contract execution. *IEEE Transactions on Parallel and Distributed Systems* 33, 5 (2021), 1097–1108.

[43] Yizhong Liu, Andi Liu, Yuan Lu, Zhuocheng Pan, Yinuo Li, Jianwei Liu, Song Bian, and Mauro Conti. 2024. Kronos: A secure and generic sharding blockchain consensus with optimized overhead. *Cryptology ePrint Archive* (2024).

[44] 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*. 17–30.

[45] Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2019. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 1041–1053.

[46] Atsuki Momose and Ling Ren. 2021. Multi-threshold byzantine fault tolerance. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1686–1699.

[47] Pezhman Nasirifard, Ruben Mayer, and Hans-Arno Jacobsen. 2023. OrderlessChain: A CRDT-based BFT Coordination-free Blockchain Without Global Order of Transactions. In *Proceedings of the 24th International Middleware Conference*. 137–150.

[48] Senthil Nathan, Chander Govindarajan, Adarsh Saraf, Manish Sethi, and Praveen Jayachandran. 2019. Blockchain meets database: design and implementation of a blockchain relational database. *Proceedings of the VLDB Endowment* 12, 11 (2019),

1539–1552.

[49] Kamilla Nazirkhanova, Joachim Neu, and David Tse. 2022. Information dispersal with provable retrievability for rollups. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*. 180–197.

[50] NetherMind. [n.d.]. Ethereum Sync Mode. https://docs.nethermind.io/fundamentals/sync/. Accessed: 2025.

[51] Lan N Nguyen, Truc DT Nguyen, Thang N Dinh, and My T Thai. 2019. Optchain: optimal transactions placement for scalable blockchain sharding. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 525–535.

[52] Mustafa Safa Ozdayi, Yue Guo, and Mahdi Zamani. 2022. Instachain: Breaking the sharding limits via adjustable quorums. *Cryptology ePrint Archive* (2022).

[53] Rafael Pass and Elaine Shi. 2017. Hybrid Consensus: Efficient Consensus in the Permissionless Model. In *31 International Symposium on Distributed Computing*. 6.

[54] Zeshun Peng, Yanfeng Zhang, Qian Xu, Haixu Liu, Yuxiao Gao, Xiaohua Li, and Ge Yu. 2022. Neuchain: a fast permissioned blockchain system with deterministic ordering. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2585–2598.

[55] 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*. 1327–1341.

[56] Ji Qi, Xusheng Chen, Yunpeng Jiang, Jianyu Jiang, Tianxiang Shen, Shixiong Zhao, Sen Wang, Gong Zhang, Li Chen, Man Ho Au, et al. 2021. Bidl: A high-throughput, low-latency permissioned blockchain framework for datacenter networks. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 18–34.

[57] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2020. A transactional perspective on execute-order-validate blockchains. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 543–557.

[58] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. 2019. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *Proceedings of the 2019 International Conference on Management of Data*. 105–122.

[59] Nibesh Shrestha, Aniket Kate, and Kartik Nayak. 2024. Sailfish: Towards Improving Latency of DAG-based BFT. *Cryptology ePrint Archive* (2024).

[60] Alexander Spiegelman, Balaji Aurn, Rati Gelashvili, and Zekun Li. 2023. Shoal: Improving DAG-BFT Latency And Robustness. *arXiv preprint arXiv:2306.03058* (2023).

[61] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2705–2718.

[62] 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)*. IEEE, 1357–1368.

[63] Andrei Tonkikh, Pavel Ponomarev, Petr Kuznetsov, and Yvonne-Anne Pignolet. 2023. Cryptoconcurrency:(almost) consensusless asset transfer with shared accounts. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1556–1570.

[64] Uniswap.io. [n.d.]. uniswap. https://docs.uniswap.io/. Accessed: 2024.

[65] 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)*. 95–112.

[66] Xin Wang, Haochen Wang, Haibin Zhang, and Sisi Duan. 2024. Pando: Extremely Scalable BFT Based on Committee Sampling. *Cryptology ePrint Archive* (2024).

[67] Karl Wüst, Sinisa Matetic, Silvan Egli, Kari Kostiainen, and Srdjan Capkun. 2020. ACE: Asynchronous and concurrent execution of complex smart contracts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 587–600.

[68] Yibin Xu, Jingyi Zheng, Boris Düdder, Tijs Slaats, and Yongluan Zhou. 2024. A Two-Layer Blockchain Sharding Protocol Leveraging Safety and Liveness for Enhanced Performance. In *31st Annual Network and Distributed System Security Symposium*. The Internet Society.

[69] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. 2003. Separating agreement from execution for Byzantine fault tolerant services. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 253–267.

[70] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 347–356.

[71] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. 2020. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 90–105.

[72] 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*. 931–948.

[73] Jianting Zhang. 2024. The implementation of Arete. https://github.com/EtherCS/arete/.

[74] Jianting Zhang, Wuhui Chen, Sifu Luo, Tiantian Gong, Zicong Hong, and Aniket Kate. 2024. Front-running Attack in Sharded Blockchains and Fair Cross-shard Consensus. In *31st Annual Network and Distributed System Security Symposium*. The Internet Society.

[75] Jianting Zhang, Zhongtang Luo, Raghavendra Ramesh, and Aniket Kate. 2024. Optimal Sharding for Scalable Blockchains with Deconstructed SMR. *arXiv preprint arXiv:2406.08252* (2024).

[76] Yuanzhe Zhang, Shirui Pan, and Jiangshan Yu. 2023. Txallo: Dynamic transaction allocation in sharded blockchain systems. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 721–733.