Improving Fault Tolerance in Blockchain Sharding using One-to-Many Block-to-Shard Mapping

Tirathraj Ramburn
Concordia University, Montreal, Canada
ramburntirathraj@gmail.com

Dhrubajyoti Goswami
Concordia University, Montreal, Canada
goswami@encs.concordia.ca

***Abstract*— The goal of sharding in a contemporary Blockchain system is to increase throughput linearly in proportion to the number of shards. This is achieved in practice by one-to-one mapping of each transaction block to a shard, with the assumption that each shard is 'perfect' and hence cannot fail. The notion of perfection is achieved by forming shards that have negligible failure probabilities. This contemporary approach to blockchain sharding has two drawbacks: (1) shards tend to be large in size to maintain low failure probability, which can negatively affect performance and throughput; (2) the 'perfect' shard assumption can easily be breached if any shard becomes faulty, which can fail an entire blockchain system because there is no fault-detection mechanism during transaction-processing (i.e., faulty blocks approved by faulty shards may only be detected after being appended to the blockchain). To overcome these drawbacks, this paper presents a multi-round consensus scheme which adopts one-to-many mapping of a transaction block to *k* shards, followed by a second consensus round among the *k* shard leaders (inter-shard consensus) in an epoch to validate and commit a transaction block with finality. In return, the following are achieved: (1) possibility of increased fault tolerance, despite using smaller shard sizes, because the collective failure probability of a group of *k* small shards can be much lower than the failure probability of an individual larger shard with the proper selection of the values of *k* and other parameters; (2) capability of faulty block detection with high probability during transaction processing; and (3) relaxation of the 'perfect' shard assumption so that the system can be tolerant to more faulty shards and still maintain safety. Detailed theoretical analyses are presented which demonstrate the benefits of such a multi-round block validation approach over contemporary approaches in terms of achieving better fault tolerance without compromising on throughput.**

Keywords*— *Blockchain, Sharding, Multi-Round Consensus, Faulty block detection, Throughput, Transaction mapping, Hierarchical blockchain, Byzantine Fault Tolerance.

I. INTRODUCTION

Sharding is a prevalent approach in blockchain systems to enhance scalability and throughput. Unlike in contemporary blockchain systems where all validators (i.e., nodes that participate in transaction processing to maintain a blockchain) process a single set of transactions (a transaction block or a block) in every epoch (round of processing), sharding introduces concurrency by splitting validators into different groups (shards) such that each shard processes a different set of transactions. This approach allows a system to increase its throughput linearly in proportion to the number of shards, where throughput is defined as the number of blocks processed per epoch time. Sharding is hence desirable because a system comprising of T shards can scale and increase its throughput by a factor of T, unlike contemporary systems such as Bitcoin [1] which maintains a constant throughput even when more resources are added.

However, trade-offs need to be made between fault tolerance and performance. Sharding usually lowers the collective fault tolerance of a blockchain system for performance gains. To curb this decrease in fault tolerance, sharded systems in general are built on strict assumptions and conservative parameters that may not be very practical in the real world.

One such major assumption in contemporary sharded systems is that shards are 'perfect' and always process transactions correctly during an epoch. These systems (especially public blockchains) generally use the random sampling technique to attempt to distribute faulty (Byzantine) nodes uniformly among equal-sized shards. An even distribution of faulty nodes is desired so that the majority of nodes in any shard is non-faulty with high probability. Random sampling, when unbiased, is nondeterministic and is the preferred way to form shards with very low (negligible) probability of failure [2], [3]. Shards having negligible failure probability can then be considered as 'perfect'. Section II further elaborates on random sampling.

To obtain 'perfect' shards with low failure probability, shards need to be large (e.g., OmniLedger uses shard size of around 700 [4]). However, the use of large shard sizes usually impacts performance negatively and increases transaction latency, especially in shards that use classical Byzantine Fault Tolerance (BFT) consensus which has quadratic communication cost [5], [6]. Use of larger shards also results in lesser number of shards in a system and hence less throughput, T.

In addition to large shard size, the 'perfect shard' assumption has other drawbacks. Firstly, in reality shards can be faulty. The presence of even a single faulty shard means that the system has failed, which is undesirable. Secondly, due to the assumption that shards will always produce correct results, contemporary systems do not have in place mechanisms for faulty shard/block detection during transaction processing (i.e., before a transaction block is appended to the blockchain). So, an invalid (faulty) transaction block validated by a faulty shard may get appended to the blockchain, sometimes with irreversible damages (e.g., loss of users' trust and devaluation of the entire blockchain ecosystem).

To overcome the previous drawbacks, this research proposes a second round of transaction block validation to commit a block with finality. This is achieved as follows: let S represent a large shard of size M that would have been formed in a contemporary sharded system. Instead, S is split into k smaller shards which together form a group. A transaction block is then assigned to

all the k smaller shards in a group. In other words, a one-to-k mapping of a transaction block to shards is applied, instead of a one-to-one mapping of a block to a shard in a contemporary system. The parameter k is also called the *mapping factor*. A first round of block validation is applied inside each individual shard to validate a transaction block (i.e., intra-shard consensus). However, unlike the contemporary systems, a second round of consensus is subsequently required among all the k shards in a group to validate and finally commit the block (i.e., inter-shard consensus among the shard leaders). This approach of one-to-many block-to-shard mapping coupled with the addition of a second round of consensus for block validation in an epoch enables the following: (1) capability of faulty block/shard detection during transaction processing with high probability, before a block is appended to blockchain; (2) smaller shard sizes giving better intra-shard performance; (3) smaller collective fault probability of the k shards in a group, with the proper selection of various parameters, as compared to individual fault probability of a large shard S in contemporary systems; and finally (4) relaxation of the 'perfect' shard assumption so that the system can maintain 'safety' [7] and be tolerant to faulty shards up to a certain threshold (Table 1).

The paper is organized as follows: Section II discusses the background and related works. Section III describes the multi-round block validation approach and two different ways to achieve it. Theoretical and use case analyses are presented which show the advantages of the multi-round approach. Section IV elaborates on some key performance metrics and continues discussion on the use cases to illustrate how a good mapping factor k can be chosen. Finally, Section V concludes the paper with a discussion on future works.

II. Background

A. Complete Random Sampling and Shard Formation

A shard can be considered as a sample taken from the total number of nodes, N in a blockchain system, where N represents the population. The shard failure probability can then be calculated using the cumulative Hypergeometric distribution [8]. Complete random sampling generally gives the best shard-formation results in public blockchains, as shards need to be unbiased. Random assignment of nodes to shards prevents adversaries from pre-determining to which shard they would be allocated, therefore making collusion of such malicious parties harder.

The Hypergeometric distribution in turn depends on four parameters: (1) total nodes N, (2) shard size M, (3) f (maximum allowable faults in a shard which is usually dictated by the network model and the consensus algorithm used), and (4) F (maximum allowable faults in a system). Let $prop_{Total}$ represent the proportion of faulty nodes in the whole system and $prop_{Shard}$ represent the proportion of faulty nodes inside a shard. Then, $prop_{Shard} = f/M$ and $prop_{Total} = F/N$. If $prop_{Shard} = prop_{Total}$, the cumulative hypergeometric distribution (1) yields a shard failure probability of around 0.5, which is large and undesirable. Recall that we should aim to minimize shard failure probability. If $prop_{Total} < prop_{Shard}$, the lower the value of $prop_{Total}$ is, the lower is the shard failure probability. The contrary is true as well where $prop_{Total} > prop_{Shard}$ yields a high probability of failure. Contemporary systems tend to use conservative values of F (minimize $prop_{Total}$ and maximize $prop_{Shard}$) and/or M (large shard sizes) to obtain negligible failure probabilities.

1) Correlation between Shard Size and Sampling

Let X be the random variable denoting the number of faulty nodes in a sampled shard. To determine the failure probability of a shard, we should calculate the probability of obtaining no less than f faulty nodes when randomly selecting a shard of size M without replacement from a population of N nodes that contains at most F faults [8]. This is precisely what the cumulative hypergeometric distribution can be used for as shown in (1).

$$P\left(X \geq \left\lfloor \frac{M}{3} \right\rfloor\right) = \sum_{x=\left\lfloor \frac{M}{3} \right\rfloor}^{M} \frac{\binom{F}{x}\binom{N-F}{M-x}}{\binom{N}{M}} \quad (1)$$

In (1), we assume that at least a third of a shard's nodes need to be faulty ($f \geq M/3$) for that shard to be considered faulty. *If F and N remain constant, the failure probability of a shard decreases as the shard gets bigger (i.e., as M increases). Small shards tend to be more performant (smaller number of message exchanges) but less fault-tolerant (smaller number of nodes to corrupt) while the reverse is true for large shards. We use (1) extensively to calculate shard failure probabilities in this paper, especially in Tables 2, 3 and in Section III D (Use Case Analysis).*

2) Correlation between BFT Consensus and Sampling

Validators inside a shard need to reach agreement (consensus) on a number of decisions such as block validity, security certificates during cross-shard transactions [8], and order of transaction blocks on the blockchain. While various consensus algorithms exist in the literature, contemporary blockchains use variants of the BFT consensus algorithm to agree on decisions. In this paper, in every consensus round, we use either classic synchronous BFT [6] or classic partially synchronous BFT, e.g., PBFT [5]. There are other variants of BFT which could also be used in each round of consensus and are discussed in the next subsection.

Partially synchronous BFT algorithms can withstand up to less than 1/3 faulty nodes in a shard (i.e., $X < M/3$ in (1)) [5] while the fault threshold of synchronous BFT algorithms [6], [9] is less than 1/2 (i.e., $X < M/2$ in (1)). Recall that $prop_{Total} < prop_{Shard}$ to minimise failure probability. Therefore, $prop_{Shard}$ in partially synchronous systems (e.g., [4] and [10]) is set to 1/3 while $prop_{Total}$ is generally set to a conservative value of 1/4. In synchronous systems such as [8], $prop_{Shard}$ is set to 1/2 while $prop_{Total}$ is usually set to 1/3. These thresholds are used in Sections III and IV to be consistent with other systems.

B. Single and Multi-Layered Consensus Protocols

HotStuff BFT by Yin et. al [11] achieves linear communication complexity at the expense of increased latency and is widely used in blockchains such as Facebook's LibraBFT [12]. HotStuff BFT achieves linearity by using threshold signatures [13] and introducing an additional round of message exchanges (called a pre-commit phase) between the prepare and commit phases found in classic BFT protocols. Note that HotStuff BFT and classic BFT protocols such as PBFT [5] are all single-layered consensus protocols.

Another class of consensus algorithms operate using a multi-layered structure. The goal of this reorganization of network structure is to improve node scalability and to collectively reduce the number of message exchanges to achieve sub-quadratic communication cost. The authors of [14] describe a partially synchronous 2-layer BFT protocol that reduces communication cost from $O(N^2)\ to\ O(1.9N^{\frac{4}{3}})$. They subsequently generalize this protocol to X-layers, such that linear communication complexity can be achieved when network layer depth X_{Max} is maximized. Similar to HotStuff BFT, the reduction in message complexity comes at the expense of increased latency. Beh-Raft-Chain [15] is a sharded blockchain which uses a multi-layered Raft consensus protocol [16] inside its shard instead of BFT. Although Raft has linear communication cost, it cannot be used in the presence of Byzantine nodes (such as in a blockchain-based 'real-world' environment), as Raft uses the strong assumption that shard leaders need to always be honest to maintain safety. This is not the case for BFT protocols as safety is maintained even if a leader is malicious, provided that the respective Byzantine fault threshold is not exceeded. Therefore, Beh-Raft-Chain devises a scoring metric and uses a sortition algorithm to rank the 'honesty' of nodes based on their behavior. Nodes with the highest 'honesty' scores are assumed to be non-faulty (i.e., honest) and are then chosen for leadership roles. A supervising committee exists in every shard, which monitors the shard leader (hence the multi-layer structure). Behavior is deterministically defined and monitors can report a leader if the latter lands in a malicious state.

Our work has two rounds of consensus in an epoch to validate and commit a block with finality by agreeing on a Merkle root [17] for that block. The current work uses single-layered classic BFT protocols (e.g., [5], [6]), both inside shards (intra-shard consensus in the first round) and among shards (inter-shard consensus in the second round), for ease of expression. However, other optimized and state-of-the art consensus protocols can be used in either round of block validation. Two different consensus protocols could also be used in the respective rounds of the same epoch.

C. *Faulty Block Detection in Sharded Systems during Transaction Processing*

Most contemporary systems have no mechanisms to detect faulty blocks (and consequently faulty shards) during transaction processing [4], [8], [10]. These systems instead assume that shards are 'perfect' and always output the correct result during an epoch. They use conservative fault parameters and shard sizes to minimize failure probability as discussed previously. OmniLedger [4] uses a partially synchronous intra-shard BFT algorithm and obtains shard sizes of 600-800 for a realistic shard failure probability (e.g., 1×10^{-10}). A shard size of 600-800 is considered to be very large and impacts intra-shard performance negatively as OmniLedger's consensus protocol has quadratic communication cost. RapidChain [8] attempts to improve upon OmniLedger by using a synchronous BFT protocol which is not responsive [18]. While RapidChain obtains smaller shard sizes of around 300-400 nodes, it trades off a key performance property of BFT algorithms called responsiveness for increased resiliency due to its use of synchrony [9], [18]. In the multi-round block validation approach presented in this paper, the aim is to obtain shard sizes that are much smaller than those of OmniLedger by splitting a contemporary shard into smaller fragments to gain in performance, but without compromising the collective fault tolerance of the entire system.

Chainspace [19] has auditing mechanisms in place to identify potentially faulty shards. Non-faulty nodes maintain enough information to be able to trace back inconsistencies to specific shards and therefore flag them. However, these faulty shards can only be detected after transaction processing, following which corrective measures can be undertaken. Furthermore, a full audit needs to be performed by an 'auditor' by potentially scanning the entire blockchain to detect such inconsistencies. This can obviously be very costly in terms of resources. In contrast, the multi-round block validation approach presented in this paper can detect potentially faulty shards with high probability during transaction processing, before an irreversible damage is done. Fault detection can also be relatively 'less costly' than Chainspace's mechanism, as faulty shards do not need to be specifically detected to prevent a faulty block from being appended to the blockchain. The amount of information and time required are significantly less, i.e., only final validation results from shards are required.

OptiShard [20] is a hierarchical and centralized blockchain system that uses the 'overlapping transactions' method to detect potentially faulty shards during transaction processing with some limitations, discussed in the following. A central authority chooses a set of transactions (overlapped set) and broadcasts it to all shards for validation. The overlapped set's results are then verified by the central authority. Shards that output incorrect results for the overlapped transaction are flagged as faulty. However, if a malicious shard manages to process the overlapped transaction correctly but corrupts other transactions in a block, the latter will go undetected and the faulty block will still be appended to the blockchain. The central authority can furthermore become a bottleneck. The multi-round consensus approach presented here mitigates such limitations and bottlenecks.

III. The Multi-Round Block Validation Approach

Problem Statement: All contemporary sharded systems known to us employ one-to-one mapping of transaction block to shard; hence they require a single round of consensus (intra-shard consensus) for block validation. In such systems, trade-offs must be made in terms of fault tolerance and performance, i.e., increasing shard size increases fault tolerance but decreases performance, and vice versa. We observe that by introducing a second round of consensus among the shards assigned to validate the same block, more flexibility can be achieved in adjusting fault tolerance. In this research, the second round of consensus is achieved in two broad ways: (1) assigning each transaction block to k contemporary shards and then have a second round of consensus among the k shards; or (2) split a contemporary shard into k smaller shards, assign each transaction block to k such smaller shards, and then have a second round of consensus among the k smaller shards. It can be shown that both the previous approaches can give better fault tolerance because the collective failure probability of the k

shards can be made smaller than the individual failure probability of a single contemporary shard, with suitable selection of parameters. Additionally, the second round of consensus gives fault-detection capability which enables detecting potentially faulty blocks with high probability during transaction processing. However, the first approach comes with the drawback of decreased throughput. On the other hand, the second approach does not affect throughput and can in fact can give better theoretical performance due to the use of smaller shards. In all the cases, fault-tolerance is increased, and faulty block detection during transaction processing is enabled. The previous discussion is elaborated in the following.

Consider a shard S of size M in a contemporary system. If S is faulty and includes some invalid transaction(s) (e.g., committing fraud or double-spending [21]) in a transaction block B, then B will be appended to the blockchain although it is an invalid block. This violates the 'safety' property [7] of the blockchain, and the entire system is considered to have failed, presumably with irreversible damage. There are no mechanisms to detect an invalid block during transaction processing (or during the epoch). Let $P_{ContemporaryShardFailure}$ be the probability that S is faulty and approves an invalid block. $P_{ContemporaryShardFailure}$ can be calculated using the cumulative Hypergeometric distribution (refer to (1) in Section II).

The following approaches can be used to prevent approval of an invalid block by a faulty shard.

A. *Approach 1: One-to-many Mapping of Transaction Block to Contemporary Shards followed by Second Consensus Round*

Consider an approach where each transaction block B is assigned to k contemporary shards for validation instead of a single shard (i.e., one-to-many mapping of transaction block to shard). Recall that in a contemporary system it is a one-to-one mapping of block to shard. Each shard individually validates the block (intra-shard consensus), followed by a second round of consensus among the k shards validating the same block (inter-shard consensus). If a faulty shard attempts to approve an invalid block, it could be flagged by the second round of consensus because its validation result will not match the validation results of the other non-faulty shards.

In the following discussion, the parameter k is called the *mapping factor* and the group of k shards validating the same block is called a *group*.

To simplify the discussion, let us start with 100% consensus requirement in the second consensus round. Subsection III C relaxes this consensus requirement. In such a scenario, a faulty transaction block will be validated and appended to the blockchain if and only if all the k shards in the group are faulty. Let the probability of all k shards being collectively faulty be represented by $P_{CollectiveFailure}$. Then,

$$P_{CollectiveFailure} = (P_{ContemporaryShardFailure})^k \quad (2)$$

Let P_{Valid} be the probability that a transaction block gets correctly validated, or a potentially invalid transaction block gets detected. It is given by the complement of (2):

$$P_{Valid} = 1 - P_{CollectiveFailure}$$
$$= 1 - (P_{ContemporaryShardFailure})^k \quad (3)$$

As a result, the following are deduced:

1) Capability of Faulty Block Detection and Increased Fault Tolerance

From (2) and (3), it can be inferred that as the mapping factor k increases, the likelihood of correctly validating that block increases as the probability that a group of k shards fail collectively decreases. This is also equivalent to saying that the likelihood of potentially faulty block detection increases. Note that $P_{CollectiveFailure} \ll P_{ContemporaryShardFailure}$.Therefore, the *1 to* k block to shard mapping together with the second consensus round among the k shards enables faulty block detection with high probability and increased fault tolerance by reducing the collective failure probability.

An obvious question arises: instead of the multi-round block validation approach, why not merge the k shards to create one bigger shard, use one-to-one mapping of blocks to bigger shards, and use a single round of consensus as in contemporary systems. It has two obvious drawbacks: firstly, a single round of consensus will disable faulty block detection. Secondly, larger shard size will degrade performance and consequently throughput. Merging is the opposite of splitting and the same discussion can be carried out in terms of splitting. Next subsection (subsection III B) describes a scheme based on splitting of contemporary shards to smaller shards, together with a detailed discussion on performance and fault tolerance.

2) Relaxation of the 'Perfect' Shard Assumption

In case of 100% consensus requirement in the second consensus round, all k shards need to fail collectively for the approval of a faulty block. The presence of even a single 'perfect' (non-faulty) shard giving the correct validation is sufficient to prevent faulty shards from gaining a 100% majority in a *group*. In other words, only one out of k shards per group needs to be 'perfect' to maintain safety. Hence there is a relaxation of the 'perfect' shard constraint in the entire system by a factor of k. Recall that all shards need to be 'perfect' in a contemporary system for maintaining safety and producing correct results. The 100 % consensus requirement in the second consensus round can be relaxed; however, it will result in decreased fault tolerance. This is discussed in detail in Subsection III C.

The obvious drawback of Approach 1 is the decrease in throughput T of a contemporary system by at least a factor of k, because the number of groups of k shards validating a distinct block per epoch becomes $\frac{T}{k}$. Fig. 1 illustrates a scenario with k = 2.

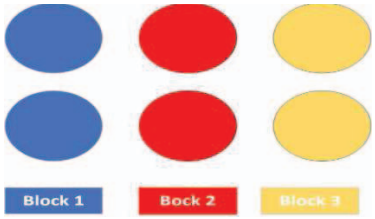

Fig 1. One-to-two mapping of a transaction block to shards.

The reduction in throughput is remedied by Approach 2 in the following.

B. *Approach 2: Splitting of Contemporary Shards to Maintain Throughput*

Consider a different approach where each contemporary shard S of size M is first split into k equal-sized smaller shards (called Transaction-processing shards or *Txp* shards) of size m_{Tx} each, i.e., $k \times m_{Tx} = M$. In a system comprising of T contemporary shards, number of *Txp* shards obtained after the split amounts to *(T x k)*, and there are T distinct groups of k *Txp* shards each. Approach *1* is now applied after the split, i.e., 1 to k mapping of a transaction block to k *Txp* shards in a group and second round of consensus between the k shards shards in each group. Each group of k *Txp* shards validates a different block every epoch. Therefore, the split enables to maintain throughput T as compared to the solution discussed in Approach *1*.

1) Increase in Fault Tolerance after Split as compared to Contemporary Systems

Intuition of split: in a contemporary system, if S is faulty then a potentially faulty block may be validated and subsequently be appended to the blockchain. On the contrary, if S is faulty but split into k smaller shards, it is not necessary that all the k smaller shards will be faulty, i.e., the likelihood (probability) of all k shards being faulty in the group formed by splitting S is lower than the likelihood that the original shard S is faulty. If the second consensus round requires 100% consensus among the k shards to validate a block (i.e., all k shards need to agree on the same Merkle root), the presence of a single non-faulty shard in the group is sufficient to detect a faulty block. Therefore, the likelihood of detecting a faulty block is higher after splitting S, as the split might result in at least one non-faulty shard with high probability (Fig. 2). Splitting S comes with the added benefit of maintaining throughput T.

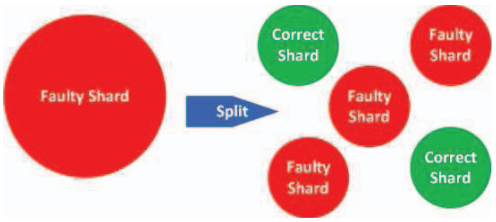

Fig 2. Formation of at least 1 non-faulty shard after splitting a faulty shard is sufficient to detect a potentially faulty block.

Let $P_{TxpFailure}$ be the probability that an individual *Txp* shard of size m_{Tx} is faulty. Let N be the total number of nodes in the system. Then, the probability $P_{CollectiveFailure}$ that a group of k *Txp* shards fails becomes:

$$P_{CollectiveFailure} = (P_{TxpFailure})^k \quad (4)$$

$$P_{Valid} = 1 - P_{CollectiveFailure} = 1 - (P_{TxpFailure})^k \quad (5)$$

Note that $m_{Tx} = \frac{N}{(T \times k)}$, and $P_{TxpFailure}$ can be calculated from (1). It can be seen that an increase in mapping factor k results in a decrease in individual *Txp* shard size m_{Tx}. A decrease in shard size results in an increase in the probability $P_{TxpFailure}$ that an individual shard is faulty. Hence increasing mapping factor k increases the likelihood that an individual *Txp* shard fails. However, due to the second consensus round validation step where, in case of 100% consensus, all k *Txp* shards in a group need to fail collectively for an invalid transaction block to be appended to the blockchain.

Failure probability $P_{TxpFailure}$ of an individual *Txp* shard after the split is much higher than the failure probability $P_{ContemporaryShardFailure}$ of an original contemporary shard S. However, when appropriate values of k are chosen, it can be observed that the collective failure probability of a group of k *Txp* shards can be lower than the failure probability of an individual contemporary shard S, i.e.,

$$P_{CollectiveFailure} \ll P_{ContemporaryShardFailure}$$

Use case analyses in Subsection III D and Section IV elaborate on this observation. The multi-round block validation approach with split can therefore not only be more resilient to faults as compared to contemporary systems but can also enable the detection of faulty transaction blocks during transaction-validation with high probability. It can furthermore relax the "perfect" shard assumption and tolerate faulty shards in the system up to a certain threshold without compromising safety (refer to Table 1).

2) Theoretical Improvement in Intra-shard Performance due to Split.

Classical BFT algorithms are quadratic in message complexity [5], [6]. For a contemporary shard S of size M, intra-shard message complexity is $O(M^2)$ per consensus. In the multi-round approach with split, intra-shard message complexity per consensus becomes $O(\frac{M^2}{k^2})$ for each of the *Txp* shards which work in parallel on the same transaction block. Hence, number of message exchanges decreases by a factor of k^2 inside *Txp* shards when one-to-k mapping is used with quadratic BFT algorithms. For large values of k, the reduction in number of message exchanges can considerably boost intra-shard performance. If a state-of-the-art linear BFT algorithm such as HotStuff [11] is used, message complexity per consensus of a *Txp* shard becomes $O(\frac{M}{k})$; this is a reduction by a factor of k which is still beneficial. However, it should be noted that the highest value of k does not necessarily yield the best system, because it may compromise on the liveness of the system and some other metrics discussed in the following. Proper selection of the value of k is elaborated in Section IV.

3) Bottleneck of 100% Consensus

Achieving 100 % consensus in a distributed system is challenging and usually requires a trusted central authority; however, it goes against the concept of blockchain. The central authority can be a bottleneck in terms of both fault tolerance and performance. Failure of the central authority means that the entire system has failed (i.e., single point of failure).

In a private or semi-private blockchain, the central authority can be chosen by the private entities or consortiums in charge of the blockchain and thus can be 'perfect'. However, in public blockchains where the primary adversary being dealt with tends to be Byzantine, sharded systems use variants of the Byzantine Fault Tolerance (BFT) protocol (e.g., [5], [6], [11]) to achieve consensus. When BFT consensus is used, the requirement of having 100% consensus among k shards in a group becomes infeasible because classical BFT algorithms are already optimal in terms of quorum sizes and the number of faults they can

withstand inside a shard. Increasing quorum size to k votes for 100% consensus would actually compromise on the safety of the system while decreasing quorum size could compromise on liveness. Other types of BFT consensus algorithms such as Flexible BFT can offer more flexibility [22], [23] in terms of quorum sizes, but they come with different fault model(s) and other considerations, and are beyond the scope of this paper.

In the following discussion, we relax the consensus requirement from 100% and use BFT protocol in the second round of distributed consensus to reach agreement among the k shards in a group.

C. Using Classic BFT in the Second Round of Consensus

In every epoch, an inter-shard BFT consensus is carried out among the k shard leaders of a group assigned to validate a block (Fig. 3). Multiple leader-selection algorithms already exist in the literature and are beyond the scope of this paper.

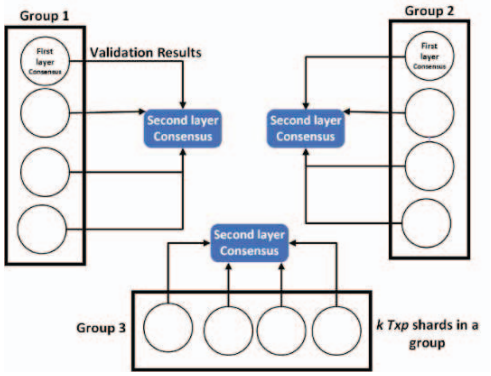

Fig. 3. Three shards S of size M are each split into $k = 4$ *Txp* shards of size m_{Tx} each.

The consensus requirement depends on the type of the BFT algorithm used. For instance, partially synchronous BFT can withstand less than 1/3 faulty nodes (Byzantine nodes) in a shard, while the fault threshold for synchronous BFT is slightly less than 1/2. In other words, in partial synchrony [9], [24], more than $\frac{2k}{3}$ shards in a group need to have the same validation result (i.e., Merkle root) for a transaction block to be approved (i.e., 67% consensus requirement). In synchronous BFT [9], 51% consensus majority is required and hence more than $\frac{k}{2}$ shards in a group need to have the same validation result for a transaction block approval.

1) Increase in number of 'Perfect' Shards required for Safety when Second-round Consensus Requirement is Relaxed

Based on the previous discussion, the number of 'perfect' *Txp* shards required to maintain safety in a group of k shards increases from one (in case of 100% consensus) to at least $\left\lceil\frac{k}{3}\right\rceil$ *or* $\left\lceil\frac{k}{2}\right\rceil$ depending on which type of BFT protocol is used. In general, the collective failure probability for a group of k shards can be modelled using the binomial distribution and is given by:

$$P_{CollectiveFailure} = \sum_{i}^{k} \begin{pmatrix} \binom{k}{i} \\ \times \ (P_{TxpFailure})^{i} \\ \times \ (1 - P_{TxpFailure})^{k-i} \end{pmatrix} \quad (6)$$

In (6), i can take the values of k (100% consensus using central authority), $\left\lceil\frac{2k+1}{3}\right\rceil$ (67% consensus), or $\left\lceil\frac{k+1}{2}\right\rceil$ (51% consensus). Note that decreasing (or relaxing) the consensus requirement of a group of k shards increases its collective failure probability, $P_{CollectiveFailure}$, and hence decreases its fault tolerance and the 'likelihood' of detecting faulty transaction blocks. Table 1 illustrates the various trade-offs.

Table 1. Trade-offs between different consensus requirements.

	Number of 'perfect' shards required	
Consensus Requirement	To Maintain Safety	To Make Progress
100 % (Central Authority only)	1 out of k	k out of k
67% (PartiallySynchronous BFT)	$\left\lceil\frac{k}{3}\right\rceil$	$\left\lceil\frac{2k+1}{3}\right\rceil$
51 % (Synchronous BFT)	$\left\lceil\frac{k}{2}\right\rceil$	$\left\lceil\frac{k+1}{2}\right\rceil$

2) Theoretical Performance in an Epoch

Although approach 2 processes the same number of transaction blocks T per epoch as contemporary sharded systems, there can be a potential decrease in number of message exchanges per epoch due to the following: intra-shard communication complexity decreases by a factor of k or k^2 depending on the consensus algorithm used. However, this decrease is offset by the distributed second consensus round which can have message complexity of $O(k)$ or $O(k^2)$ depending on the BFT protocol used; though k is rather small. Assuming perfect concurrency, the communication complexity per epoch of a group of k shards in approach 2 is $O\left(TB + \frac{M^2}{k^2} + k^2\right)$ for quadratic BFT, where TB is the average transaction block validation time.

D. Use Case Analysis

The following use case assumes the partially synchronous network model within shards. A faulty node is considered to be Byzantine only. Hence, each shard can tolerate less than 1/3 faulty nodes (i.e., $f < M/3$). The maximum number of Byzantine faults F that can be present in the system is set to $N/4$. This threshold is used to adhere to the parameters that other partially synchronous sharded blockchains use [4], [10]. These parameters and the fault model can be changed to obtain different levels of fault-tolerance and performance [22], [23].

Consider the use case with total nodes $N = 2000$ and maximum number of Byzantine faults present in the system, $F = N/4 = 500$. Approach 2 with split (Sections III B and C) is used in this scenario such that a second round of consensus among the k shards in a group is required for validation of a block with finality. The following discussion quantitatively demonstrates that the Approach 2 can be made more fault tolerant as compared to a contemporary sharded system by making proper choices of the parameters, e.g., m_{Tx}, T, and k. Moreover, fault can be detected with high probability during block validation in an

epoch, rather than after, so that an invalid transaction block is not appended to blockchain by a potentially faulty shard.

1) Impact of Mapping Factor k on Collective Fault Tolerance of a Group of k Txp Shards

Let the desired throughput T of the system be 4. In a contemporary sharding system, 4 big shards S, each of size M =500, would be formed. In contrary, in Approach 2, each shard S is split into k smaller *Txp* shards. Table 2 and Fig. 4 illustrate the results from these possible splits for k = 2, 4, 5, 10, and 20. Table 3 and Fig. 5 illustrate the results of possible splits when desired throughput T is 2, i.e., 2 big shards of size M=1000 are created in a contemporary system. It can be seen that as mapping factor k increases, individual *Txp* shard size gets smaller resulting in higher individual shard failure probability $P_{TxpFailure}$. However, the collective failure probability $P_{CollectiveFailure}$ of a group of k *Txp* shards decreases, therefore making the system more fault resilient. This increase in collective fault tolerance is the general trend for the 3 different consensus requirements discussed in this paper (i.e., 100%, 67%, and 51%) with increasing mapping factor k. Note that when k =1, Approach 2 becomes equivalent to a contemporary system.

Table 2. Failure probabilities with different mapping factors when M = 500.

Mapping factor, k	*Txp* shard size, m_{Tx}	Individual shard failure probability, $P_{TxpFailure}$	Collective group failure probability, $P_{CollectiveFailure}$		
			100% Consensus	67 % Consensus	51 % Consensus
1	500	6×10^{-7}	N/A	N/A	N/A
2	250	7×10^{-4}	5×10^{-7}	N/A	N/A
4	125	0.016	7×10^{-8}	2×10^{-5}	2×10^{-5}
5	100	0.025	9×10^{-9}	2×10^{-6}	1×10^{-4}
10	50	0.096	6×10^{-11}	7×10^{-6}	1×10^{-4}
20	25	0.15	3×10^{-17}	4×10^{-8}	3×10^{-5}
50	10	0.22	3×10^{-33}	8×10^{-12}	5×10^{-6}

Table 3. Failure probabilities with different mapping factors when M = 1000.

Mapping factor, k	*Txp* shard size, m_{Tx}	Individual shard failure probability, $P_{TxpFailure}$	Collective group failure probability, $P_{CollectiveFailure}$		
			100% Consensus	67 % Consensus	51 % Consensus
1	1000	2×10^{-18}	N/A	N/A	N/A
2	500	6×10^{-7}	4×10^{-13}	N/A	N/A
4	250	7×10^{-4}	2×10^{-13}	1×10^{-9}	3×10^{-6}
8	125	0.016	5×10^{-15}	5×10^{-10}	6×10^{-8}
10	100	0.025	8×10^{-17}	6×10^{-10}	4×10^{-8}
20	50	0.096	4×10^{-21}	1×10^{-10}	4×10^{-7}
40	25	0.15	6×10^{-34}	6×10^{-14}	3×10^{-8}
100	10	0.22	1×10^{-65}	2×10^{-21}	4×10^{-10}

2) Effect of Relaxing Second Consensus Requirement on Collective Fault Tolerance and on 'Perfect' Shard Constraint

It can be deduced from Tables 2 and 3, and Fig. 4 and 5 that, as the second consensus requirement is relaxed, collective fault tolerance of a group decreases (i.e., failure probability increases). For example, consider the mapping factor k = 10 in Fig. 4. For 100 % consensus requirement, $P_{CollectiveFailure}$ is 6×10^{-11}. When the requirement is relaxed to 67%, $P_{CollectiveFailure}$ becomes 7×10^{-6}, and further increases to 1×10^{-4} for 51% consensus.

Number of 'perfect' shards required to maintain safety per group therefore increases (Table 1 top to bottom) as consensus requirement is relaxed. In general, the stricter the consensus requirement is, the more the 'perfect' shard constraint can be relaxed to maintain safety. This is also equivalent to saying that the stricter the consensus requirement is, the greater number of faulty shards a system can withstand. Recall that in a contemporary system, the presence of even a single faulty shard means that the system has failed due to the strict 'perfect' shard assumption.

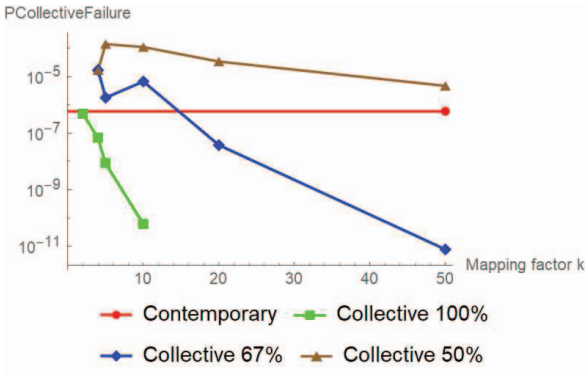

Fig 4. Effect on collective failure probabilities at different consensus requirements when S of size M = 500 is split into k smaller Txp shards.

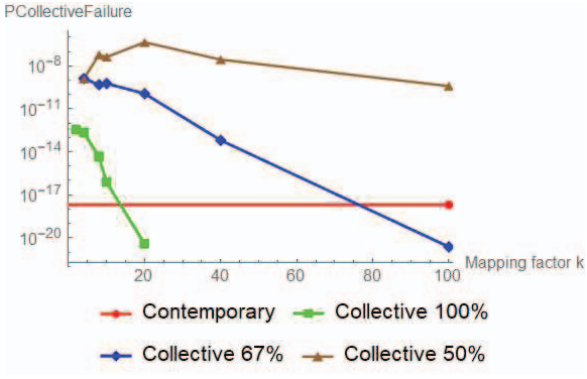

Fig 5. Effect on collective failure probabilities at different consensus requirements when S of size M = 1000 is split into k smaller Txp shards.

3) Impact of Throughput T on Collective Fault Tolerance

Recall that *Txp* shard size $m_{Tx} = \frac{N}{(T \times k)}$, where T is the desired throughput of the system. Rearranging the previous equation, we get $T = \frac{N}{(m_{Tx} \times k)}$. The previous equation shows that throughput T is inversely proportional to both mapping factor k and shard size m_{Tx}. With increasing T, the ranges of possible values for both k and m_{Tx} decrease. Based on the previous discussion, an increase in T results in an increase in the collective failure probability $P_{CollectiveFailure}$ of a group of k shards and therefore a decrease in fault tolerance of the system. Decreasing T has the opposite effect. Figures 6 and 7 illustrate the collective failure probabilities with different second-round consensus requirements (i.e., 100% and 67%) for the current use case, with some of the possible splits and desired throughput values (i.e., T = 2, 4, and 8).

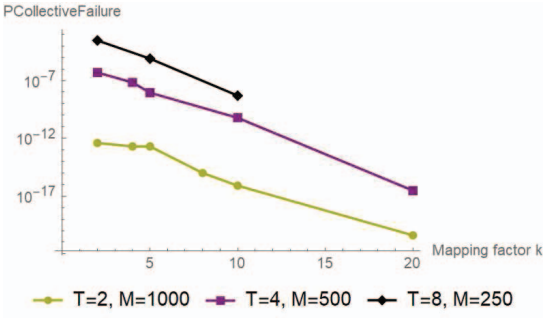

Fig. 6. Collective failure probabilities of a group of k shards at different throughputs and mapping factors (100% Consensus).

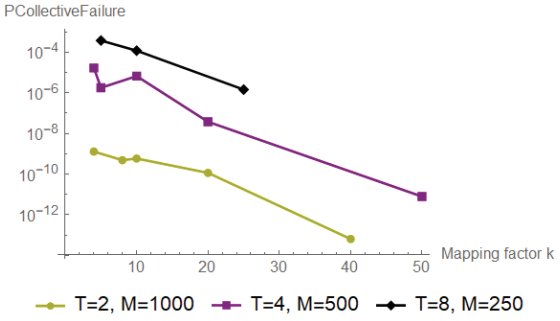

Fig. 7. Collective failure probabilities of a group of k shards at different throughputs and mapping factors (67% Consensus).

IV. Choosing a Good Mapping Factor *K*

A. Liveness Attack and Progress

Care should be taken in choosing a good mapping factor. Opting for the highest value of k does not necessarily lead to a better system in terms of both performance and fault tolerance.

Liveness Attack: In the case of 100% consensus requirement in the second round of consensus, among the k shards in a group to validate a transaction block, the presence of a single faulty shard is sufficient to halt the progress of that group. A single faulty shard can always send invalid results in every epoch such that the group it belongs to never reaches 100% consensus, i.e., that group never makes progress and never validates blocks although safety is maintained. We term this lack of progress as a *liveness attack*.

1) Impact of Mapping Factor k on Liveness

Recall from the previous discussion that when mapping factor k increases, the probability $P_{TxpFailure}$ that an individual *Txp* shard is faulty increases, however the collective failure probability $P_{CollectiveFailure}$ decreases. Although the collective fault tolerance of the system (safety property) increases with increasing value of k, the system becomes more prone to liveness attacks (no progress) as it becomes easier to corrupt a shard individually.

A group of k shards requiring 100% consensus can suffer from a liveness attack when at least 1 of its shards is faulty. Hence, the probability of liveness attack in that group is given by $P_{LivenessAttack}$ = *1 – P(all shards in the group are non-faulty)*. This can also be formulated as follows: $P_{LivenessAttack} = 1 - (1 - P_{TxpFailure})^k$ for 100% consensus. Figures. 8 and 9 support our analysis that a group is more prone to liveness attacks if it has more shards (i.e., if S is split into more fragments), because as the shards get smaller it becomes easier to corrupt them individually.

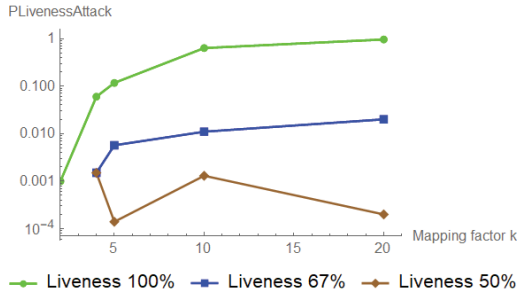

Fig. 8. Probabilities of liveness attack for different consensus requirements and k values when M = 500.

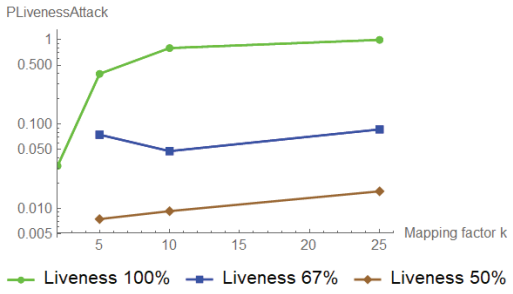

Fig. 9. Probabilities of liveness attack for different consensus requirements and k values when M = 250.

2) Impact of Relaxing Second-round Consensus Requirements on Liveness

In the case of 67% consensus, a group will not make progress if at least $\frac{k}{3}$ shards are faulty. In general, the probability of a liveness attack for any consensus requirement can be formulated as follows:

$$P_{LivenessAttack} = \sum_{j}^{k} \begin{pmatrix} \binom{k}{j} \\ \times (P_{TxpFailure})^{j} \\ \times (1 - P_{TxpFailure})^{k-j} \end{pmatrix} \quad (7)$$

In (7), j can take the values of 1 (100 % consensus), or $\frac{k}{3}$ (67% consensus), or $\frac{k}{2}$ (51% consensus). It can be inferred from Table 1 and (7) that as the consensus requirement is relaxed (as we go down Table 1), more shards need to be faulty and produce invalid results to prevent progress of a group. For example, only one shard needs to be faulty to prevent 100% consensus requirement while at least $\frac{k}{3}$ shards need to be faulty to prevent non-faulty shards from reaching 67% consensus. Therefore, when consensus requirement is relaxed, a group can withstand more faulty shards in terms of liveness, $P_{LivenessAttack}$ decreases, and it becomes easier to make progress (Fig. 8 and 9); however, it comes at the expenses of higher collective failure probability and reduced safety (Tables 1-3 and Fig. 4-5).

B. Expected Throughput E(T) per Epoch

Let $P_{Progress}$ be the probability that a group of k shards makes progress during an epoch, i.e., the group does not suffer from a liveness attack and therefore correctly validates a transaction block during that epoch. As a follow up of (7), it can be formulated as:

$$P_{Progress} = 1 - P_{LivenessAttack} \quad (8)$$

Let Y be the random variable denoting the number of groups of k shards out of all T groups in a system that make progress in that epoch. Since each group has only 2 possible outcomes (progress or no progress), and their outcomes are independent of other groups after being sharded, the binomial distribution can be used to calculate the probability that t groups out of T make progress in an epoch, where $0 \leq t \leq T$. If ($t > 0$), we call the system to be partially live. The probability that t out of T groups make progress in an epoch is given by:

$$P(Y = t) = \binom{T}{t} \times (P_{Progress})^{t} \times (P_{LivenessAttack})^{T-t} \quad (9)$$

Following from (9), the expected throughput $E(T)$ per epoch is given by:

$$E(T) = \sum_{t=0}^{T} t \times P(Y = t) \quad (10)$$

Equation (10) can be simplified to (11) below. Details of the calculation are omitted due to space constraints:

$$E(T) = T \times P_{Progress} \quad (11)$$

$E(T)$ represents the expected number of groups that will make progress during an epoch, which translates to the expected number of blocks that will be correctly validated in the system during that epoch. $E(T)$ is a useful metric that can be referred to when trying to find a good balance between fault tolerance and performance (throughput).

1) Impact of Increasing Mapping Factor k on Expected Throughput

Fig. 10 and 11 plot expected throughput values against mapping factor k for 3 different consensus requirements when desired throughput T is set to 4 (M=500) and 8 (M=250) respectively. We note that with increasing k, shards become more prone to individual failures. It becomes easier to corrupt at least 1 shard in a group requiring 100% consensus and hence more groups become vulnerable to liveness attacks. The same applies for the other consensus requirements. As a consequence, expected throughput of a system decreases with increasing values of k. Hence, a larger value of k does not necessarily yield a better system when performance also has to be considered.

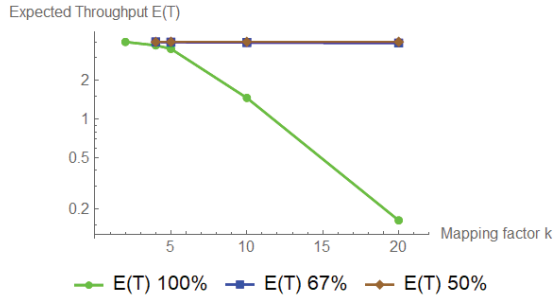

Fig. 10. Expected Throughput for different consensus requirements and mapping factors when T = 4 and M = 500.

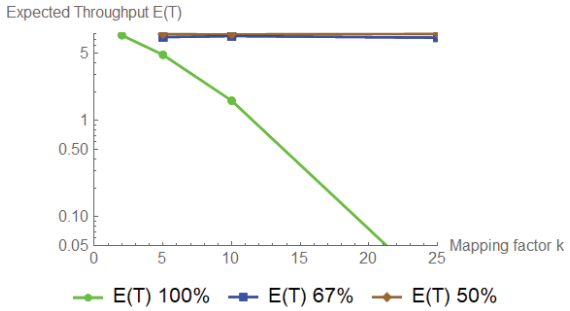

Fig. 11. Expected Throughput for different consensus requirements and mapping factors when T = 8 and M = 250.

2) Impact of Relaxing Second-round Consensus Requirement on Expected Throughput

As the consensus requirement is relaxed in the second round, it becomes more difficult to attack liveness and prevent progress as more shards need to be faulty in a group. For example, only one shard needs to be faulty to prevent progress when 100% consensus is used, while at least $k/3$ shards need to be faulty in the case of 67% consensus. Hence, relaxing consensus requirement makes it more difficult for liveness attacks and hence easier to make progress. As a result, Expected Throughput $E(T)$ of a system increases (Fig. 10 and 11).

C. Choosing a Good Mapping Factor k

We analyze Fig. 4 to 11 to come up with good mapping factor values k for the use case in Section 3 with N=2000, F=N/4=500, and partial synchrony (67% consensus) inside *Txp* shards. Note that the second consensus round can however be modeled as either synchronous (51% consensus) or partially synchronous (67% consensus) in this use case.

Consider the case where desired throughput T is 4. A contemporary system would form 4 shards S, each of size M = 500 which has failure probability $P_{ContemporaryShardFailure}$ = 6 x 10^{-7}. By splitting S into multiple smaller *Txp* shards, k values of 2

and beyond give better collective failure probabilities than unsplit S when 100% consensus is used in the second round (Fig. 4 and 6). As mapping factor k increases, liveness gets affected and hence Expected Throughput $E(T)$ of the system decreases. Possible k values with $E(T)$ close to T= 4 and good collective fault tolerance are k=2, 4, and 5 only (Fig. 10). Therefore, when 100% consensus is used, good fault tolerance can be obtained by splitting S of size M = 500 into 2, 4 or 5 fragments.

However, in the case of 67% consensus in the second consensus round, k values of 5, 10, 20 and 50 result in fault tolerance better than or comparable to $P_{ContemporaryShardFailure} = 6 \times 10^{-7}$. Liveness and consequently $E(T)$ is barely affected when the consensus requirement is relaxed from 100% to 67%, and hence $E(T)$ for all the 4 possible k values remain close to the desired throughput T = 4.

Consider another case where the desired throughput T is 8. A contemporary system would form 8 shards S, each of size M = 250 which has failure probability $P_{ContemporaryShardFailure} = 7 \times 10^{-4}$. By splitting S into multiple smaller *Txp* shards, k values of 5, 10, and 25 give better collective failure probabilities than unsplit S when 67% consensus is used in the second consensus round. Liveness is not affected significantly and the previous k values give $E(T)$ of 7.4, 7.6 and 7.3 respectively. Hence, the k values of 5, 10, and 25 are all good candidates to obtain good fault tolerance, including fault detection capability.

The same discussion and analysis can be carried out for other contemporary shard sizes and consensus requirements to choose optimal value(s) of k based on both fault tolerance and throughput.

V. Conclusion and Future Works

This paper presents a multi-round block validation approach in blockchain sharding that introduces a second round of consensus in an epoch, thus enabling faulty transaction block detection during transaction processing. The ability to detect potentially faulty transaction blocks in an epoch before they are appended to the blockchain can relax conservative assumptions such as 'perfect' shard requirement and large shard sizes. This can in turn allow the use of smaller shards which can theoretically lead to lower communication cost during intra-shard consensus. The paper introduces the novel concept of splitting a contemporary large shard into a group of k smaller shards (k is called the *mapping factor*), one-to-k mapping of a block to all k shards in a group, and finally a second round of consensus on the validation results of the group for the block to be appended to the blockchain. Three possible consensus requirements (100 %, 67%, and 51 %) and their trade-offs are discussed. Though there is an increase in failure probability of an individual shard due to decreased shard size after the split, the second round of consensus in a group can decrease the collective failure probability of the group, while still maintaining throughput. This is not the case for contemporary systems where gain in performance usually results in loss of fault tolerance, and vice-versa. Theoretical and use case analyses of the multi-round approach are presented which show that the introduction of the second round of consensus can make the system more fault resilient without compromising throughput. The work presented here assumes that each node has a full copy of the ledger (blockchain), i.e., the ledger is not distributed. Hence, each node/shard can independently validate a transaction block. Future research involves experimentally evaluating our work in a real blockchain environment, and splitting the ledger, which will need additional inter-shard communication and protocols for transaction block validation inside a group. These will be reported in our future works.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," May 2009.

[2] V. Buterin, *Why Sharding is great: Demystifying the technical properties.*

[3] T. Hanke, M. Movahedi and D. Williams, "DFINITY Technology Overview Series, Consensus System," *CoRR,* vol. abs/1805.04548, 2018.

[4] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta and B. Ford, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[5] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, USA, 1999.

[6] L. Ren, K. Nayak, I. Abraham and S. Devadas, "Practical Synchronous Byzantine Consensus," *CoRR,* vol. abs/1704.02397, 2017.

[7] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Transactions on Software Engineering,* Vols. SE-3, pp. 125-143, 1977.

[8] M. Zamani, M. Movahedi and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2018.

[9] I. Abraham, "Synchrony, Asynchrony and Partial synchrony," 1 June 2019. [Online]. Available: https://decentralizedthoughts.github.io/2019-06-01-2019-5-31-models/.

[10] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert and P. Saxena, "A Secure Sharding Protocol For Open Blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2016.

[11] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta and I. Abraham, "HotStuff: BFT Consensus with Linearity and Responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, New York, NY, USA, 2019.

[12] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malki, O. Naor, D. Perelman and A. Sonnino, "State Machine Replication in the Libra Blockchain," 2019.

[13] D. Boneh, B. Lynn and H. Shacham, "Short Signatures from the Weil Pairing," *J. Cryptol.,* vol. 17, p. 297–319, September 2004.

[14] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao and M. A. Imran, "A Scalable Multi-Layer PBFT Consensus for Blockchain," *IEEE Transactions on Parallel and Distributed Systems,* vol. 32, pp. 1146-1160, 2021.

[15] L.-e. Wang, Y. Bai, Q. Jiang, V. C. M. Leung, W. Cai and X. Li, "Beh-Raft-Chain: A Behavior-Based Fast Blockchain Protocol for Complex Networks," *IEEE Transactions on Network Science and Engineering,* vol. 8, pp. 1154-1166, 2021.

[16] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USA, 2014.

[17] H. Liu, X. Luo, H. Liu and X. Xia, "Merkle Tree: A Fundamental Component of Blockchains," in *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 2021.

[18] R. Pass and E. Shi, "Hybrid Consensus: Efficient Consensus in the Permissionless Model," in *International Symposium on Distributed Computing*, 2017.

[19] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn and G. Danezis, "Chainspace: A sharded smart contracts platform," *arXiv preprint arXiv:1708.03778,* 2017.

[20] S. Kantesariya and D. Goswami, "Determining Optimal Shard Size in a Hierarchical Blockchain Architecture," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020.

[21] G. Karame, E. Androulaki and S. Capkun, "Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin," *IACR Cryptol. ePrint Arch.,* p. 248, 2012.

[22] D. Malkhi, K. Nayak and L. Ren, "Flexible Byzantine Fault Tolerance," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2019.

[23] T. Ramburn and D. Goswami, "FlexiShard: a Flexible Sharding Scheme for Blockchain based on a Hybrid Fault Model," in *2022 21st International Symposium on Parallel and Distributed Computing (ISPDC)*, 2022.

[24] C. Dwork, N. Lynch and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *J. ACM,* vol. 35, p. 288–323, April 1988.