On Sharding Permissioned Blockchains

Mohammad Javad Amiri  Divyakant Agrawal  Amr El Abbadi
Department of Computer Science, University of California Santa Barbara
Santa Barbara, California
{amiri, agrawal, amr}@cs.ucsb.edu

Abstract—Permissioned Blockchain systems rely mainly on Byzantine fault-tolerant protocols to establish consensus on the order of transactions. While Byzantine fault-tolerant protocols mostly guarantee consistency (safety) in an asynchronous network using $3f+1$ machines to overcome the simultaneous malicious failure of any $f$ nodes, in many systems, e.g., blockchain systems, the number of available nodes (resources) is much more than $3f+1$. To utilize such extra resources, in this paper we introduce a model that leverages transaction parallelism by partitioning the nodes into clusters (partitions) and processing independent transactions on different partitions simultaneously. The model also shards the blockchain ledger, assigns different shards of the blockchain ledger to different clusters, and includes both intra-shard and cross-shard transactions. Since more than one cluster is involved in each cross-shard transaction, the ledger is formed as a directed acyclic graph.

Index Terms—Permissioned Blockchain, Scalability, Data Sharding, Directed Acyclic Graph

I. INTRODUCTION

Blockchain, originally devised for the Bitcoin cryptocurrency [27], is a distributed data structure for recording transactions maintained by nodes without a central authority [9]. Nodes in a blockchain system agree on their shared states across a large network of untrusted participants. Blockchain has unique features such as transparency, provenance, fault tolerance, and authenticity that are used by many systems to deploy a wide range of distributed applications such as healthcare [5], IoT [18], and supply chain management [20] in permissioned settings. Unlike permissionless settings, e.g., Bitcoin [27], where the network is public, and anyone can participate without a specific identity, a permissioned blockchain consists of a set of known, identified nodes that still do not fully trust each other.

In a permissioned blockchain system, every node maintains a copy of the blockchain ledger and a consensus protocol is used to ensure that the nodes agree on a unique order in which entries are appended to the blockchain ledger. To establish consensus among the nodes, asynchronous fault-tolerant protocols have been used. Fault-tolerant protocols use the state machine replication algorithm [23] where nodes agree on an ordering of incoming requests. Since nodes in a blockchain do not trust each other and might behave maliciously, a Byzantine fault-tolerant protocol is needed. Byzantine fault-tolerant protocols, e.g., PBFT [10], mainly guarantee safety (consistency) in an asynchronous network using $3f+1$ nodes to overcome the simultaneous malicious failure of any $f$ nodes.

In many systems especially blockchains, the number of available nodes is much more than $3f+1$. In such systems, using all the nodes to establish consensus degrades performance since more messages are being exchanged without providing improved resiliency, e.g., in PBFT, the number of message exchanges is quadratic in terms of the number of nodes.

To tackle that issue, one solution is to use the active/passive replication technique [17] by relying on only $3f+1$ active replicas to establish consensus on the order of requests. When the requests are ordered and executed, the active replicas send the execution results to the passive replicas, so that their copies of the ledger become up to date. The active replicas might be either a fixed set or a rotating set where at some predefined times a different set of replicas become active. While this approach reduces the cost of establishing consensus among all nodes by relying on only the required number of nodes ($3f+1$), it does not utilize the extra replicas.

An alternative solution is to employ the extra replicas to enhance the performance of the protocol by reducing one phase of communication, e.g., Byzantine fault-tolerant protocol FaB [26] uses $5f+1$ replicas to establish consensus on the order of requests in two phases instead of three as in PBFT. This approach improves the performance of the system by using some of the extra nodes, e.g., $2f$ extra nodes in FaB, however, if the number of extra nodes is more than $2f$, they cannot be utilized and in the best case scenario the extra nodes become passive replicas.

Partitioning the data into multiple shards that are maintained by different subsets of nodes is a proven approach to enhance the scalability of databases [12]. In such an approach the performance of the database scales horizontally with the number of nodes. Databases are sharded such that the resulting shards are as independent as possible, i.e., each transaction accesses the records within only a single shard. An appropriate sharding usually needs to be workload-aware, i.e. has prior knowledge of the data and how it is accessed by different transactions. Data sharding strategies mainly try to improve the performance of systems in terms of throughput and latency by reducing the number of cross-shard transactions (transactions that access more than one shard).

In this paper, we present a model for permissioned blockchain systems which is designed specifically for networks with a large number of nodes ($\geq 3f+1$). The blockchain model utilizes the extra resources by clustering (partitioning) the nodes into clusters where each cluster includes $3f+1$ nodes. Furthermore, the data is sharded and data shards are assigned to the clusters. Each cluster then is responsible to process the transactions that access its correspond-
The application data is sharded over different clusters. We assume prior knowledge of the data and how it is accessed by different transactions. Hence, this knowledge is used in their execution techniques (e.g., speculative execution).

Data sharding techniques are commonly used in distributed databases in the presence of non-malicious failures [12] [15] [6]. Using data sharding techniques for permissioned blockchains is presented in Elastico [25] and Omniledger [19] where the mining network is uniformly partitioned into smaller committees and each committee processes a disjoint set of shards. While Elastico does not support the cross-shard transaction, Omniledger proposes an atomic protocol for cross-shard transactions using a locking-based method. In the permissioned settings, Fabric also addresses sharding by deploying different shards on different channels. In Fabric cross-shard transactions are handled using a trusted entity.

In this section, we introduce an infrastructure for blockchain systems where the nodes are partitioned into clusters and the application data including transactions and the ledger is sharded over clusters. The blockchain consists of a set of nodes in an asynchronous distributed system where nodes are connected by a network. We use a Byzantine failure model where faulty nodes may exhibit arbitrary, potentially malicious, behavior. We assume that a strong adversary can coordinate malicious nodes and delay communication to compromise the replicated service. However, the adversary cannot subvert standard cryptographic assumptions about collision-resistant hashes, encryption, and signatures, e.g., the adversary cannot produce a valid signature of a non-faulty node.

Nodes are connected by point-to-point bi-directional communication channels. Network channels are pairwise authenticated, which guarantees that a malicious node cannot forge a message from a correct node. i.e., if node $i$ receives a message $m$ in the incoming link from node $j$, then node $j$ must have already sent message $m$ to $i$.

Byzantine fault-tolerant protocols mainly guarantee consistency (safety) in an asynchronous network using $3f+1$ nodes [7] to overcome the simultaneous malicious failure of any $f$ nodes. As discussed earlier, we assume that the number of nodes, $N$, is much larger than $3f+1$. Therefore, to utilize the extra nodes we partition the nodes into clusters where each cluster includes $3f+1$ nodes (the last cluster might include more nodes). Nodes are assigned to the clusters either using their ids, e.g., $n_0, n_1, ..., n_{3f}$ are assigned to the first cluster, or based on their geographical distance. We denote the set of clusters by $P = \{p_1, p_2, \ldots\}$ where $|P| = \frac{N}{f+1}$.

The application data is sharded over different clusters. We assume prior knowledge of the data and how it is accessed by different transactions. Hence, this knowledge is used in data sharding to increase the probability of maintaining the
records which are accessed by a single transaction in the same shard [13]. Nevertheless, there might still be a portion of transactions that access records from different shards. As a result, the blockchain supports two types of transactions: intra-shard and cross-shard. An intra-shard transaction accesses the records within a single shard whereas a cross-shard transaction accesses records in at least two different shards.

Since there are $|P|$ clusters, the data is also sharded into $|P|$ shards, thus each cluster maintains a shard of the data. Within each cluster, the data is replicated over the nodes of that cluster. We denote shards by $d_1, \ldots, d_{|P|}$ where each shard $d_i$ is replicated over the nodes of cluster $p_i$.

Figure 1 presents the architecture of a blockchain system consisting of 12 nodes where $f = 1$. Thus, we have three clusters ($|P| = \frac{12}{3}$) where each shard is replicated on the 4 nodes of its cluster.

IV. BLOCKCHAIN LEDGER

The blockchain ledger is an append-only data structure recording transactions in the form of a hash chain where each block contains a batch of transactions. Batching transactions into blocks is a reason for the low performance of blockchains. Transactions were originally batched into blocks, first, to amortize the cost of cryptography, e.g., solving proof-of-work, and second, to make data transfers more efficient in a large geo-distributed setting [16]. However, in permissioned blockchains, since proof-of-work is not required and nodes are physically close to each other, batching transactions into blocks decreases performance. Thus, in our model, each block consists of a single transaction. To support both types of intra- and cross-shard transactions, we generalize the notion of a blockchain ledger from a linear chain to a directed acyclic graph (DAG) where the nodes of the graph are blocks and edges enforce the order of blocks.

Within each cluster, since transactions have access to the same data shard which is replicated over all nodes of the cluster, a total order between all the transactions that the cluster is involved in (both intra- and cross-shard) is enforced to ensure consistency. To capture the total order of transactions in the blockchain ledger, blocks are chained together, i.e., each block includes the cryptographic hash of the previous block. Since more than one cluster is involved in each cross-shard transaction, the ledger is formed as a directed acyclic graph.

In addition to intra- and cross-shard transactions, a unique initialization block, called genesis, is considered for the blockchain.

Fig. 2. (a): A blockchain ledger consisting of three shards, (b), (c), and (d): The views of the blockchain from different shards

Fig. 2(a) shows a blockchain ledger consisting of three clusters $p_1$, $p_2$, and $p_3$ (data shards $d_1$, $d_2$, and $d_3$) created in the model. In this figure, $\lambda$ is the genesis block of the blockchain. Intra- and cross-shard transactions are also specified. For example, $t_{10}$, $t_{11}$, $t_{13}$, $t_{14}$, and $t_{16}$ are the intra-shard transactions of cluster $p_1$. Note that each cross-shard transaction is labeled with $t_{01, \ldots, 0k}$ where $k$ is the number of involved clusters and $o_i$ indicates the order of the transaction among the transactions of the $i$th involved cluster. For example, $t_{12,22}$, $t_{24,33}$, and $t_{15,26,35}$, are cross-shard transactions where $t_{12,22}$ accesses data shards $d_1$ and $d_2$ (clusters $p_1$ and $p_2$), $t_{24,33}$ accesses data shards $d_2$ and $d_3$, and $t_{15,26,35}$ accesses all three $d_1$, $d_2$, and $d_3$. As can be seen, transactions that access a data shard are chained together, e.g., $t_{10}$, $t_{11}$, $t_{12,22}$, $t_{13}$, $t_{14}$, $t_{15,26,35}$, and $t_{16}$.

We denote the set of blocks (transactions) by $T$, the genesis block by $\lambda$, intra-shard transactions by $T_i$, and cross-shard transactions by $T_c$ where $T = \lambda \cup T_i \cup T_c$. We also define a function $\rho : T \rightarrow 2^P$ to specify the involved clusters (data shards) for each transaction where for an intra-shard transaction $t \in T_i$, $\rho(t)$ returns a single cluster (singleton set) and for a cross-shard transaction $t \in T_c$, $\rho(t)$ returns a set of (at least two) clusters.

**Definition:** A blockchain ledger is a directed acyclic graph $G = (\lambda, T, E)$ where

- $\lambda$ is the unique initialization block of the blockchain,
- $T$ is the set of transactions (blocks), and
- $E$ is the set of edges between blocks.

In addition to the data, the blockchain ledger is partitioned between different clusters. In fact, the entire blockchain ledger is not maintained by any cluster and each cluster only maintains its own view of the ledger including the transactions that access the data shard of the cluster. The blockchain ledger is indeed the union of all these physical views.
Definition: Given a blockchain ledger $G = (\lambda, T, E)$, let $p$ be a cluster in the blockchain. The view of $p$ is a linear graph $G_p = (\lambda, T_p, E_p)$ where
- $\lambda$ is the unique initialization block of the blockchain,
- $T_p = \lambda \cup \{ t \mid p \in \rho(t) \}$ is a set of transactions, and
- $E_p = \{ (t, t') \in E \mid t, t' \in T_p \}$ is a set of edges.

Fig. 2(b)-(d) show the views of the blockchain ledger for clusters $p_1$, $p_2$, and $p_3$ respectively. As can be seen, each cluster $p_i$ maintains only the part of the ledger consisting of the transactions that access data shard $d_i$. Those transactions (blocks) are chained together.

Nodes within a cluster follow the Byzantine failure model where faulty nodes may exhibit arbitrary, potentially malicious, behavior. Therefore, to achieve consensus on the order of the intra-shard transactions, a Byzantine fault-tolerant protocol, e.g., PBFT [10], is needed. However, achieving consensus on the order of the cross-shard transactions needs the participation of the nodes of all the involved clusters. Such a protocol might rely on a separate set of nodes, i.e., orderers, to establish consensus [3]. The protocol design is considered as a future work and a step towards developing a permissioned blockchain system.

V. CONCLUSION

In this paper, we proposed a model for a permissioned blockchain system which is designed specifically for networks with a large number of nodes ($\gg 3f + 1$). The model utilizes the extra resources by partitioning the nodes into clusters of size $3f + 1$ and processing the transactions on different clusters in parallel. Since the model supports both intra- and cross-shard transactions, the blockchain ledger is formed as a directed acyclic graph. Each cluster, however, maintains only a shard of the ledger that includes its intra-shard transactions and the cross-shard transactions that the cluster is involved in. As future work, we will develop a consensus protocol for this model to order intra- and cross-shard transactions.

ACKNOWLEDGEMENT

This work is funded by NSF grants CNS-1703560 and CNS-1815733.

REFERENCES