Permissioned Blockchains: Properties, Techniques and Applications

Mohammad Javad Amiri
University of Pennsylvania
mjamiri@seas.upenn.edu

Divyakant Agrawal
University of California Santa Barbara
agrawal@cs.ucsb.edu

Amr El Abbadi
University of California Santa Barbara
amr@cs.ucsb.edu

ABSTRACT

The unique features of blockchains such as immutability, transparency, provenance, and authenticity have been used by many large-scale data management systems to deploy a wide range of distributed applications including supply chain management, healthcare, and crowdworking in permissioned settings. Unlike permissionless settings, e.g., Bitcoin, where the network is public, and anyone can participate without a specific identity, a *permissioned* blockchain system consists of a set of known, identified nodes that might *not* fully trust each other. While the characteristics of permissioned blockchains are appealing to a wide range of large-scale data management systems, these systems, have to satisfy four main requirements: *confidentiality*, *verifiability*, *performance*, and *scalability*. Various approaches have been developed in industry and academia to satisfy these requirements with varying assumptions and costs. The focus of this tutorial is on presenting many of these techniques while highlighting the trade-offs among them. We demonstrate the practicality of such techniques in real-life by presenting three different applications, i.e., supply chain management, large-scale databases, and multi-platform crowdworking environments, and show how those techniques can be utilized to meet the requirements of such applications.

ACM Reference Format:
Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. Permissioned Blockchains: Properties, Techniques and Applications. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3448016.3457539

1 INTRODUCTION

A blockchain is a distributed data structure for recording transactions maintained by several nodes without a central authority [18]. In a blockchain system, nodes agree on their shared states across a large network of *untrusted* participants. Blockchain was originally devised for Bitcoin cryptocurrency [47], however, recent systems focus on its unique features such as transparency, provenance, fault tolerance, and authenticity to support a wide range of distributed applications. Bitcoin and other cryptocurrencies are *permissionless* blockchain systems. In a permissionless blockchain, the network is public, and anyone can participate without a specific identity. Many other distributed applications, such as supply chain management and healthcare, are deployed on *permissioned* blockchain systems consisting of a set of known, identified nodes that still might not fully trust each other. The focus of this tutorial is on *permissioned* blockchain systems that support distributed applications across collaborative enterprises. These collaborative enterprises, however, do not necessarily trust each other. Hence, we address four different challenges regarding the *confidentiality*, *verifiability*, *performance*, and *scalability* requirements of distributed applications to make permissioned blockchain systems practical in real-life settings.

Confidentiality of data is required in many collaborative distributed applications, e.g., supply chain management, where multiple enterprises collaborate following Service Level Agreements (SLAs) to provide different services. To deploy distributed applications across different collaborative enterprises, a blockchain system needs to support the internal transactions of each enterprise as well as cross-enterprise transactions that represent the collaboration between enterprises. While the data accessed by cross-enterprise transactions should be *visible* to all enterprises, the internal data of each enterprise, which are accessed by internal transactions, might be *confidential*.

Besides confidentiality, in many cross-enterprise systems, e.g., crowdworking applications, participants need to verify transactions that are initiated by other enterprises to ensure the satisfaction of some predefined global constraints on the entire system. For example, the total work hours of a worker per week may not exceed 40 hours to follow *Fair Labor Standards Act*[1] (FLSA). Hence, if a worker works for multiple crowdworking platforms, e.g., a driver who works for both Uber and Lyft, verifying such global constraints requires access to data owned by other enterprises. Thus, the system needs to support *verifiability* while preserving the confidentiality of transactions.

In addition to confidentiality and verifiability, distributed applications, e.g., financial applications, require high performance in terms of throughput and latency, e.g., while the Visa payment service handles thousands of transactions per second, naive implementations of permissioned blockchains handle only hundreds of transactions per second. In general, "order" and "execution" are the two main phases of processing transactions in permissioned blockchains. Permissioned blockchains need to parallelize the processing of different transactions in the order or execution phase to improve the overall performance of the system.

Finally, scalability is one of the main obstacles to business adoption of blockchain systems. To support a distributed application, e.g., a large-scale database, a blockchain system should be able to scale efficiently by adding more nodes to the system. Partitioning the data into multiple shards that are maintained by different subsets of nodes is a proven approach to enhance the scalability of databases

SIGMOD '21, June 20–25, 2021, Virtual Event, China
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8343-1/21/06.
https://doi.org/10.1145/3448016.3457539

[1] https://www.dol.gov/agencies/whd/flsa

[23]. In such an approach, the performance of the database scales linearly with the number of nodes. While database systems use the sharding technique to improve the scalability of databases [23] in a network of crash-only nodes, the technique cannot easily be utilized by blockchain systems due to the possible untrusted nature of the nodes in the network.

In this tutorial, our goal is to present to the database community an in-depth understanding of state-of-the-art solutions for designing efficient permissioned blockchain systems. We start by discussing several large-scale data management applications that motivate the rising demand for permissioned blockchains as an infrastructure. We then provide a detailed description of the permissioned blockchain model and its basic cryptographic and distributed system components. Given the diverse needs of different applications, we describe in detail various techniques underlying the design of existing permissioned blockchain systems that address the fundamental challenges of confidentiality, verifiability, performance and scalability. Each section of the tutorial will end with a discussion of the trade-offs between the various techniques presented and the challenges practical systems face for supporting large-scale data.

2 TUTORIAL OUTLINE

Confidentiality, verifiability, performance, and scalability are the main requirements of large-scale data management applications that need to be achieved by permissioned blockchain systems. While confidentiality and verifiability are needed in cross-enterprise applications such as supply chain management and crowdworking (to preserve the confidentiality of enterprise data and to verify the transactions initiated by other enterprises while preserving the confidentiality of their data), performance and scalability are required in both single- and cross-enterprise applications.

In this section, we first present several practical large-scale data management applications to motivate permissioned blockchains. We next introduce permissioned blockchains and present different techniques proposed by permissioned blockchain systems to meet the requirements of large-scale data management applications.

2.1 Applications

We now briefly discuss several large-scale data management applications to motivate permissioned blockchains.

2.1.1 *Supply Chain Management*

Lack of trust between different parties is one of the most important problems in supply chain management. To tackle such an issue, a permissioned blockchain can be used to *monitor* the execution of the collaborative process and *check conformance* between the execution and SLAs. The utilized blockchain system needs to support both internal and cross-enterprise transactions where in contrast to the cross-enterprise transactions which are visible by all participants, the internal transactions of each enterprise are confidential, e.g., the internal transactions of the Manufacturer demonstrate its internal process for producing a product which the Manufacturer might intend to keep as a secret. Finally, the blockchain system has to address the performance aspect as well.

2.1.2 *Large-Scale Databases*

Sharding techniques are extensively used in distributed databases such as Google's Spanner [23] and Facebook's Tao [17] to address the scalability issue. Such systems mainly assume a crash failure model and rely on a trusted coordinator to process cross-shard transactions. In the presence of untrusted infrastructure, i.e., Byzantine nodes, a blockchain system can be used to achieve scalability while tolerating malicious failures. Scalability techniques can also be integrated with confidentiality techniques if the confidentiality of data shards is required.

2.1.3 *Multi-Platform Crowdworking*

Crowdworking empowers open collaboration over the Internet. A crowdworking system includes platforms, requesters, and workers where requesters submit their tasks and workers send their contributions for a particular task to the platforms. A crowdworking system might need to perform thousands of transactions per second, thus, has to be high performance. In addition, a multi-platform crowdworking system should scale appropriately with the increasing number of platforms, workers, and requesters. Moreover, the system has to provide verifiability of transactions against the predefined global constraints. For example, the total work hours of a worker, who might work for multiple crowdworking platforms, per week may not exceed 40 hours to follow *Fair Labor Standards Act*[2] (FLSA) or in California, *California Proposition 22*[3] imposes its own set of regulations on minimal hours worked for health benefits, e.g., if a driver works at least 25 hours per week, companies (i.e., platforms) are required to provide healthcare subsidies. In summary, a multi-platform crowdworking system requires all four confidentiality, verifiability, performance, and scalability properties.

2.2 Permissioned Blockchain

Blockchain systems are global scale peer-to-peer systems that integrate many techniques and protocols from cryptography, distributed systems, and databases. In a blockchain system, nodes agree on their shared states across a large network of possibly *untrusted* participants. Bitcoin [47] and other cryptocurrencies are permissionless blockchain systems. *Permissionless* blockchain systems are public where computing nodes without a priori known identities can join or leave the blockchain network at any time. On the other hand, a *permissioned* blockchain system uses a network of a priori known and identified nodes to manage the blockchain. The main underlying data structure in blockchain systems is the *blockchain ledger*, an append-only fully replicated structure that is shared among all participants and guarantees a consistent view of all user transactions by all participants in the system. Each block in the blockchain ledger includes a batch of transactions and the total order of transaction blocks in the blockchain ledger is captured by *chaining* blocks together, i.e., each block includes the cryptographic hash of the previous block. Figure 1 illustrates an example of a permissioned blockchain system consisting of five nodes where each node maintains a copy of the blockchain ledger.

[2]https://www.dol.gov/agencies/whd/flsa

[3]https://ballotpedia.org/California_Proposition_22,_App-Based_Drivers_as_Contractors_and_Labor_Policies_Initiative_(2020)

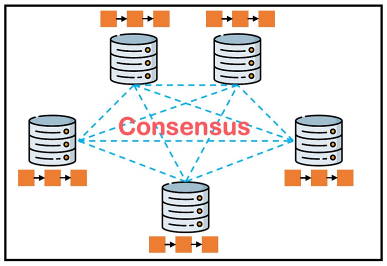

Figure 1: A Permissioned Blockchain System

The Blockchain Architecture consists of a set of nodes in an asynchronous large distributed system. Nodes in the system might crash, i.e., when a node fails it stops processing completely, or may behave maliciously, i.e., when a node fails it may act arbitrarily, often referred to as the Byzantine failure model. To ensure consistency among the data replicated on different nodes, blockchain systems use the State Machine Replication (SMR) algorithm [39], where nodes agree on an ordering of incoming transactions, to ensure the copies of the distributed ledger are identical. SMR regulates the deterministic execution of client transactions on different nodes such that every non-faulty node executes every transaction in the same order [53][39]. In a permissioned blockchain system, nodes establish consensus on this unique order in which transactions are appended to the blockchain ledger using asynchronous fault-tolerant protocols, e.g., Paxos [40] or PBFT [19].

2.3 Techniques

Over the past few years permissioned blockchain systems have proposed various techniques to address the four main requirements of large-scale data management systems. In this section, we discuss these techniques in detail and present the systems that utilize these techniques to address the four main requirements: confidentiality, verifiability, performance, and scalability.

2.3.1 *Confidentiality*

Confidentiality of data is required in many collaborative distributed applications, e.g., supply chain management, where multiple enterprises collaborate following Service Level Agreements (SLAs) to provide different services. To deploy distributed applications across different collaborative enterprises, a blockchain system needs to support the internal transactions of each enterprise as well as cross-enterprise transactions that represent the collaboration between enterprises. While the data accessed by cross-enterprise transactions should be *visible* to all enterprises, the internal data of each enterprise, which are accessed by internal transactions, might be *confidential*. In particular, in collaborative distributed applications, each enterprise can maintain its own independent disjoint blockchain and use techniques such as atomic cross-chain transactions [34][62] or Interledger protocol [58] to support cross-enterprise collaboration. Such techniques are often costly, complex, and mainly designed for permissionless blockchains. Techniques that support collaborative enterprises on a single blockchain, on the other hand, either do not support internal transactions of enterprises resulting in data integration issues or suffer from confidentiality issues since the entire ledger is visible to all enterprises, e.g., single-channel Fabric [15].

To achieve confidentiality, cryptographic and view-based (i.e., sharding-based) techniques have been proposed. In cryptographic techniques, the data is encrypted or hashed, hence, irrelevant parties cannot access the data. Alternatively, view-based techniques have been used to achieve confidentiality where each party (i.e., an enterprise or a group of enterprises) maintains only its own view of data (including records that are accessible to the party), hence, there is no need to use cryptographic techniques. We present Caper [8], multi-channel Hyperledger Fabric [16], and private data collections [6] (used within each channel of Hyperledger Fabric).

View-based approaches. We start by discussing *view-based approaches*. In Caper [8] each enterprise maintains two types of private and public data and the system supports both internal and cross-enterprise transactions where internal transactions are executed by a single enterprise, while cross-enterprise transactions are executed by all enterprises. In Caper, each enterprise orders and executes its internal transactions locally while cross-enterprise transactions are public and visible to every enterprise. In addition, the blockchain ledger of Caper is a directed acyclic graph that includes the internal transactions of every enterprise and all cross-enterprise transactions. Nonetheless, for the sake of confidentiality, the blockchain ledger is not maintained by any node. In fact, each enterprise maintains its own *local view* of the ledger including its internal and all cross-enterprise transactions. Since ordering cross-enterprise transactions requires global agreement among all enterprises, Caper introduces different consensus protocols to globally order cross-enterprise transactions.

Hyperledger Fabric [15][16] introduces channels to preserve confidentiality. A multi-channel Hyperledger Fabric consists of multiple channels where each channel has its own set of enterprises. Within a channel, each enterprise has its own set of executor (i.e., endorser) nodes where the transactions of the enterprise are executed by its endorser nodes. As a result, the enterprise logic which is implemented in its smart contracts is private from other enterprises. Enterprises within a channel, however, share the same blockchain ledger and blockchain state (i.e., datastore), hence, any transaction in a channel will be replicated on the ledger of all channel members (i.e., enterprises). Different channels, on the other hand, are completely separated and access neither the blockchain ledger nor the blockchain state of other channels. Different channels still might share the same set of orderer nodes. Orderers establish consensus on the order of transactions of a channel. Since orderers are able to access the transaction data, they should be trusted by all channel members. It should also be noted that an enterprise might be a member of different channels, e.g., a manufacturer that is involved in different supply chain management scenarios. Moreover, processing a (public) transaction among two channels requires a trusted channel among the participants or an atomic commit protocol.

Cryptographic techniques. Cryptographic techniques can be used when in a cross-enterprise application, a subset of enterprises wants to make confidential transactions and keep the transaction data private from other enterprises. In particular, in Hyperledger fabric, if a subset of enterprises on a channel needs to keep data confidential from other enterprises on the same channel, they have the option to create a new channel comprising just the enterprises who need access to the data. Creating separate channels, however, results in additional administrative overhead and data integrity (between public and private data) issues. Hyperledger Fabric [15] proposes *private data collections* [6] to manage confidential data that two or more enterprises on a single channel want to keep private from other enterprises on the same channel. Private data collections use hashing which is a cryptographic technique. By defining a private data collection, a subset of enterprises on a channel stores their confidential data in a private database replicated on each authorized peer. A *hash* of the private data, is still appended to the blockchain ledgers of every peer on the channel. The hash serves as evidence of the transaction and is used for state validation. Using the hash, other enterprises still able to check read-write conflicts during the validation phase. An enterprise might be involved in different private data collections where for each of them a private database is replicated on its peers.

Discussion. In summary, view-based techniques are costly in managing views, e.g., configuring channels in Hyperledger Fabric. Furthermore, processing public transactions requires establishing consensus among all involved views (e.g., enterprises, channels). Caper in comparison to Hyperledger Fabric preserves confidentiality (both logic and data) at the enterprise level. Cryptographic techniques, on the other hand, while they reduce the cost of managing views, result in the overhead of maintaining data in the blockchain ledger and blockchain state of irrelevant enterprises.

2.3.2 *Verifiability*

In many cross-enterprise systems, enterprises need to verify transactions that are initiated by other enterprises to ensure the satisfaction of global constraints in a privacy-preserving manner. This may arise in a crowdworking environment where multiple platforms that do not trust each other need to collectively enforce global regulations, e.g., a worker can work at most 40 hours per week. The blockchain system, therefore, needs to employ verifiability techniques while preserving the privacy of participants. Verifiability is also needed in cryptocurrencies with enhanced privacy, e.g., Zcash [7][35], where transaction data is confidential and nodes need to verify the transaction without knowing the sender, receiver or transaction amount.

To achieve verifiability cryptographic techniques (zero-knowledge proofs [26]) have been proposed. In cryptography, a zero-knowledge proof is a method by which one party (the prover) can prove to another party (the verifier) that they know a value x, without conveying any information apart from the fact that they know the value x. Verifiability can also be achieved using token-based techniques where a centralized entity generates verifiable tokens based on global constraints and distributes them to the corresponding participants. We present Quorum [20] and Separ [12] and discuss how verifiability has been addressed in these two systems.

Cryptographic techniques. Quorum [20] as an Ethereum-based [3] permissioned blockchain introduces two consensus protocols: a crash fault-tolerant protocol based on Raft [49] and a Byzantine fault-tolerant protocol called Istanbul BFT [5][50]. Quorum supports public and private transactions where both public and private transactions are ordered using the same consensus protocol. Quorum uses the zero-knowledge proof technique to ensure verifiability of private transactions. Zero-knowledge proofs enable the transfer of digital assets on a distributed ledger without revealing any information about the sender, recipient, or quantity of assets while ensuring that: sender is authorized to transfer ownership of the assets, assets have not been spent previously (double-spend), and transactions inputs equal to its outputs (mass conservation). Zero-knowledge proofs have also been used in crowdworking platforms, e.g., ZebraLancer [41], ZKCrowd [64], and Prio [24] to provide verifiability in a single-platform context.

Token-based techniques. Separ [12] is a multi-platform blockchain-based crowdworking system that uses a token-based technique to ensure verifiability. In Separ, a centralized trusted authority models global regulations using anonymous tokens and distributes them to participants. For example, if a global constraint declares that the total work hours of a worker per week must not exceed 40 hours to follow FLSA, the authority assigns 40 tokens to each worker where a worker can consume its tokens whenever the worker contributes to a task. Separ consisting of a privacy-preserving token-based system on top of a blockchain ledger shared across platforms where the global state of the system is managed among crowdworking platforms using distributed consensus protocols.

Discussion. In summary, cryptographic techniques are truly decentralized, hence, there is no need for a trusted entity. Zero-knowledge proofs, however, have considerable overhead [15]. Using such techniques especially in an environment where most transactions might be local, is not beneficial due to its overhead. Token-based techniques, on the other hand, require a centralized authority to generate tokens. The centralized authority must be trusted by all participants. There is, however, no need to replicate all transactions on every node resulting in improved performance.

2.3.3 *Performance*

Many large-scale data management applications require high performance in terms of throughput and latency, e.g., financial applications. Permissioned blockchain systems process transactions using either an optimistic or pessimistic approach. These approaches offer trade-offs in performance depending on the degree and frequency of contention and conflict among transactions. In the optimistic approach, nodes execute transactions without running a consensus protocol to definitively establish an ordering whereas in the pessimistic approach transactions are first ordered and then executed. From an architectural point of view, three main architectures have been proposed for permissioned blockchain systems. The order-execute (OX) and the order-parallel execute (OXII) architectures follow the pessimistic approach while the execute-order-validate (XOV) architecture follows the optimistic approach. We show how Tendermint [38], ParBlocckhain [10], Hyperledger Fabric [15], Fast Fabric [28], Fabric++ [54], FabricSharp [52], and XOX Fabric [27] address the performance challenge of permissioned blockchain systems.

Pessimistic approaches. In order-execute permissioned blockchains, a set of nodes (i.e. orderers) establishes agreement on a unique order of the incoming transactions using fault-tolerant protocols. Depending on the failure model of nodes, a Byzantine , e.g., PBFT [19], Hotstuff [61], a crash, e.g., Paxos [40], Raft [49], or even a hybrid, e.g., SeeMoRe [14], UpRight [22], fault-tolerant protocol can be used. Orderer nodes then generate and multicast blocks to other nodes (i.e., executors). Executor nodes execute the transactions of a block sequentially in the same order, append transactions to the blockchain ledger, and update the blockchain state (i.e., datastore). The order-execute architecture is widely used in different permissioned blockchain systems such as Tendermint [38], Quorum [20], Multichain [29], Chain Core [1], Hyperledger Iroha [4], and Corda [2]. In particular, Tendermint [38] uses a PBFT-based consensus protocol which differs from the original PBFT in several ways. First, only a subset of nodes, called validators, participates in the consensus protocol where to become a validator, nodes need to lock their coins. Second, Tendermint uses the leader rotation technique and switches the leader after each round (i.e., each attempt to construct a block) in a round-robin manner. Third, Tendermint implements a Proof-of-Stake consensus mechanism. In fact, in Tendermint, validators do not have the same "weight" in the consensus protocol, and the voting power of a validator corresponds to the number of its bounded coins. As a result, one-third or two-thirds of the validators are defined based on the proportions of the total voting power not the number of validators.

The order-parallel execute (OXII) architecture, similar to order-execute architecture, follows the pessimistic approach. In OXII architecture, a disjoint set of nodes (orderers) establishes agreement on the order of incoming transactions and constructs blocks. Once a block is constructed, orderer nodes generate a *dependency graph* for the transactions within a block. A dependency graph gives a partial order based on the conflicts between transactions and enables the parallel execution of *non-conflicting* transactions. The transactions are then executed by executor nodes following the generated dependency graph. ParBlockchain [10] follows the OXII architecture and is able to support multi-enterprise systems. In a multi-enterprise system, each enterprise has its own set of executor nodes where the transactions of each enterprise are executed by the corresponding executor nodes.

Optimistic approaches. Finally, Hyperledger Fabric [15] presents the optimistic XOV architecture (which was first introduced by Eve [36] in the context of Byzantine fault-tolerant SMR) by switching the order of the execution and order phases. In Fabric, transactions of different enterprises are first executed in parallel by executor nodes (i.e., endorsers) of each enterprise. Transactions are then ordered by a consensus protocol (currently a Raft-based [49] protocol) and multicast to all endorser nodes. Endorsers then validate the transactions and append them to the ledger.

While Fabric improves performance by executing transactions in parallel and supports non-deterministic execution of transactions, in the presence of any contention, i.e., conflicting transactions, in the workload (which is common in distributed applications), it has to disregard the effects of conflicting transactions which negatively impacts the performance of the blockchain. This happens because Fabric executes transactions in the first step and validates them in the last step, hence, if there is any read-write dependencies between transactions of the same block, it is not detected until the last step. Different techniques have been proposed to improve the performance of Fabric while still following its XOV architecture [27][28][51][52][54][55][56][57][60].

FastFabric [28] uses different data structures and caching techniques, and parallelizes the transaction validation pipeline to increase Fabric's throughput for conflict-free transaction workloads. Fabric++ [54] employs concurrency control techniques from databases to early abort transactions or reorder them after the order phase to reconcile the potential conflicts. FabricSharp [52] goes one step further and presents an algorithm to early filter out transactions that can never be reordered and also presents a reordering technique that eliminates unnecessary aborts (resulting from strong serializability guarantees of Fabric++ while Fabric requires serializability guarantees). Finally, XOX Fabric [27] model consists of a pre-order and a post-order execution step where the post-order execution is added after the validation step to re-execute transactions that are invalidated due to read-write conflicts.

Discussion. In summary, the OX architecture suffers from low performance due to the *sequential execution* of all transactions whereas, both OXII and XOV architectures are able to execute transactions in parallel. OXII also supports contentious workloads by detecting conflicting transactions during the order phase and generating dependency graphs while XOV validates read-write conflicts last resulting in poor performance. XOV, on the other hand, supports non-deterministic execution of transactions by executing transactions first and detecting any inconsistencies early on, while in OXII, transactions are executed in the last step, hence, if the results are inconsistent, aborting transactions would be costly.

2.3.4 Scalability

Scalability is one of the main obstacles to business adoption of blockchain systems, especially in financial and large-scale database systems. Permissioned blockchain systems mainly use *clustering* to improve scalability. In clustering approaches, e.g., Blockplane [48], nodes are partitioned into fault-tolerant clusters where each cluster processes (or at least orders) a disjoint set of transactions. Permissioned blockchain systems use *single-ledger* or *sharded-ledger* techniques to enhance scalability. In the single-ledger technique, the entire ledger is replicated on all clusters and all nodes execute every transaction. In the sharded-ledger technique, on the other hand, the ledger is partitioned into multiple shards that are maintained by different clusters. Sharded-ledger permissioned blockchain systems process two types of *intra-shard* and *cross-shard* transactions. Cross-shard transactions can be processed either in a centralized manner using the coordinator-based approach or in a decentralized manner using the flattened approach. We discuss ResilientDB [31][32][33], AHL [25], SharPer [11], Saguaro [13], and Multi-channel Fabric [16] in detail.

Single-ledger approaches. ResilientDB [32] uses a topological-aware clustering approach and partitions the network into local fault-tolerant clusters to minimize the cost of global communication. All clusters, however, replicate the *entire* ledger on every node and, at every round, each cluster locally establishes consensus on a single transaction and then multicasts the locally-replicated transaction to other clusters. All clusters then, execute all transactions of that round in a predetermined order. Since all transactions are executed by all clusters there is no concept of intra- and cross-shard transactions in ResilientDB.

Sharded-ledger Approaches. AHL [25] uses the sharding technique to enhance scalability. In AHL, similar to permissionless blockchains Elastico [42], OmniLedger [37], and Rapidchain [63], nodes are randomly assigned to clusters (called committees). To ensure safety with a high probability, each committee must include at least 80 nodes (instead of ~600 nodes in OmniLedger). To decrease the number of required nodes within each committee, AHL employs trusted hardware (the technique presented in [21][59]) that restricts the malicious behavior of a node. Using the trusted hardware, a malicious node cannot multicast inconsistent message, e.g., messages with inconsistent sequence numbers to different nodes, AHL processes cross-shard transactions in a centralized approach by relying on an extra set of nodes (called a *reference committee*) to play the coordinator role. The reference committee processes cross-shard transactions of the involved clusters using the classic two-phase commit (2PC) and two-phase locking (2PL) protocols.

SharPer [11][9] is another sharded permissioned blockchain system where the system consists of a set of fault-tolerant clusters each maintains a shard of the blockchain ledger. SharPer, in contrast to AHL, provides deterministic safety guarantees either by considering pre-determined fault-tolerant clusters or by assuming that the number of nodes is much larger than the number of failures. SharPer processes cross-shard transactions in a decentralized manner among the involved clusters (without requiring a reference committee) using decentralized flattened consensus protocols.

In Saguaro [13] nodes are organized in a hierarchical structure following the wide area network infrastructure from edge devices to edge, fog, and cloud servers where nodes at each level are further clustered into fault-tolerant clusters. At the lower level, Saguaro similar to SharPer, maintains a shard of the blockchain ledger on each cluster. Saguaro, however, benefits from the hierarchical structure of the network in the processing of cross-shard transactions. For each cross-shard transaction, the internal cluster with the minimum total distance from the involved clusters, i.e., the lowest common ancestor of all involved clusters, is chosen as the coordinator resulting in lower latency.

Finally, while in multi-channel Fabric [15][16], channels are mainly introduced to enhance confidentiality, they can be used to shard the system and data as well. A channel is in fact a shard of the full system that is autonomously managed by a (logically) separate set of nodes but is still aware of the bigger system it belongs to [16]. Using channels, Fabric processes intra-shard transactions efficiently using a fault-tolerant protocol. Cross-shard transactions in multi-channel Fabric are processed in a centralized manner and require either the existence of a trusted channel among the participants to play the coordinator role or an atomic commit protocol [16].

Discussion. Sharded-ledger approaches mainly differ in how they process cross-shard transactions. Centralized processing of cross-shard transactions is simpler and closer to the traditional two-phase commit, i.e., instead of a single coordinator node, a coordinator cluster is needed to tolerate Byzantine failure, however, a large number of intra- and cross-cluster communication phases is needed. On the other hand, the decentralized approach does not require an extra set of nodes, processes transactions in less number of phases among the involved clusters, and is able to process cross-shard transactions with non-overlapping clusters in parallel. However, if the involved clusters are distant, establishing cross-shard consensus among involved clusters that needs multiple rounds of message passing results in high latency. Single-ledger approaches, e.g., ResilientDB, on the other hand, do not suffer from the latency of processing cross-shard transactions by replicating the entire data on every cluster. However, exchanging messages between all clusters for every single transaction still results in high latency.

3 TUTORIAL INFORMATION

This is a **three hours** tutorial targeting researchers, designers, and practitioners interested in permissioned blockchains and their applications in large-scale data management systems. The target audience with basic background about blockchain and distributed systems should benefit the most from this tutorial. For the general audience and newcomers, the tutorial explains the design space of permissioned blockchains in large-scale data management systems.

This tutorial differs from previous tutorials on the same topic in database conferences. The tutorial presented by Maiyya et al. at VLDB 2018 [43] was mainly on permissionless blockchains, covered Bitcoin, the details of PoW consensus and several Bitcoin alternatives to improve the throughput of Bitcoin. The tutorial then presented solutions such as atomic swap and lightning networks to solve the challenges stemming from the rise of multiple blockchain systems. The next version of that tutorial, presented at SIGMOD 2019 [44], partially covered permissioned blockchains where several systems such as Hyperledger Fabric, ParBlockhain and Caper, were simply presented as examples of permissioned blockchains. In this tutorial, however, more than 20 permissioned blockchains and their underlying techniques to satisfy the requirement of large-scale data management systems is presented and different trade-offs between the various techniques are discussed. This tutorial also differs from C. Mohan's tutorial at VLDB 2017 [45] and ICDE 2018 [46] where he explicitly states that the scope of his tutorial "is general in nature without getting into the nitty gritty of, e.g., cryptographic algorithms or the distributed consensus protocols". Finally, this tutorial is different from the tutorial presented by Gupta et al. [30] at VLDB 2020 where the focus of that tutorial was on exploring high throughput consensus protocols for permissioned blockchains.

4 BIOGRAPHICAL SKETCHES

Mohammad Javad Amiri is a Postdoctoral Researcher in the Computer and Information Science department at the University of Pennsylvania. He received his Ph.D. from the University of California at Santa Barbara in 2020. His research mostly lies at the intersection of Data Management and Distributed Systems. The focus of his current research is on managing large-scale data in cloud infrastructures and blockchains.

Divyakant Agrawal is a Professor of Computer Science at the University of California at Santa Barbara. His current interests are in the area of scalable data management and data analysis in cloud computing environments, security and privacy of data in the cloud, scalable analytics over big data, and Blockchain. Prof. Agrawal is an ACM Distinguished Scientist (2010), an ACM Fellow (2012), an IEEE Fellow (2012), and an AAAS Fellow (2016).

Amr El Abbadi is a Professor of Computer Science at the University of California, Santa Barbara. Prof. El Abbadi is an ACM Fellow, AAAS Fellow, and IEEE Fellow. He was Chair of the Computer Science Department at UCSB from 2007 to 2011. He has served as a journal editor for several database journals and has been Program Chair for multiple database and distributed systems conferences. He has published over 400 articles in databases and distributed systems and has supervised over 40 Ph.D. students.

ACKNOWLEDGMENTS

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

REFERENCES

[1] [n.d.]. Chain. http://chain.com.
[2] [n.d.]. Corda. https://github.com/corda/corda.
[3] [n.d.]. Ethereum blockchain app platform. https://www.ethereum.org. 2017.
[4] [n.d.]. Hyperledger Iroha. https://github.com/hyperledger/iroha.
[5] [n.d.]. Istanbul byzantine fault tolerant consensus protocol. https://github.com/ethereum/EIPs/issues/650.
[6] [n.d.]. Private Data Collections: A High-Level Overview. https://www.hyperledger.org/blog/2018/10/23/private-data-collections-a-high-level-overview.
[7] [n.d.]. Zcash. https://z.cash/.
[8] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. CAPER: a cross-application permissioned blockchain. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1385–1398.
[9] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. On Sharding Permissioned Blockchains. In *Int. Conf. on Blockchain*. IEEE, 282–285.
[10] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. Par-Blockchain: Leveraging Transaction Parallelism in Permissioned Blockchain Systems. In *Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 1337–1347.
[11] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. SharPer: Sharding Permissioned Blockchains Over Network Clusters. In *SIGMOD Int. Conf. on Management of Data*. ACM.
[12] Mohammad Javad Amiri, Joris Duguépéroux, Tristan Allard, Divyakant Agrawal, and Amr El Abbadi. 2021. Separ: Towards Regulating Future of Work Multi-Platform Crowdworking Environments with Privacy Guarantees. In *Proceedings of The Web Conference*.
[13] Mohammad Javad Amiri, Ziliang Lai, Liana Patel, Boon Thau Loo, Eric Loo, and Wenchao Zhou. 2021. Saguaro: Efficient Processing of Transactions in Wide Area Networks using a Hierarchical Permissioned Blockchain. *arXiv preprint arXiv:2101.08819* (2021).
[14] Mohammad Javad Amiri, Sujaya Maiyya, Divyakant Agrawal, and Amr El Abbadi. 2020. SeeMoRe: A Fault-Tolerant Protocol for Hybrid Cloud Environments. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 1345–1356.
[15] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, et al. 2018. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *European Conf. on Computer Systems (EuroSys)*. ACM, 30.
[16] Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. 2018. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 111–131.
[17] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, et al. 2013. TAO: Facebook's Distributed Data Store for the Social Graph. In *Annual Technical Conf. (ATC)*. USENIX Association, 49–60.
[18] Christian Cachin and Marko Vukolić. 2017. Blockchain Consensus Protocols in the Wild. In *Int. Symposium on Distributed Computing (DISC)*. 1–16.
[19] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *Symposium on Operating systems design and implementation (OSDI)*, Vol. 99. USENIX Association, 173–186.
[20] JP Morgan Chase. 2016. Quorum white paper.
[21] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. 2007. Attested append-only memory: Making adversaries stick to their word. In *Operating Systems Review (OSR)*, Vol. 41-6. ACM SIGOPS, 189–204.
[22] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. 2009. Upright cluster services. In *Symposium on Operating systems principles (SOSP)*. ACM, 277–290.
[23] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, et al. 2013. Spanner: Google's globally distributed database. *Transactions on Computer Systems (TOCS)* 31, 3 (2013), 8.
[24] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 259–282.
[25] 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 *SIGMOD Int. Conf. on Management of Data*. ACM.
[26] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989), 186–208.
[27] Christian Gorenflo, Lukasz Golab, and Srinivasan Keshav. 2020. XOX Fabric: A hybrid approach to transaction execution. In *Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–9.
[28] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. 2019. Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second. In *Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 455–463.
[29] Gideon Greenspan. 2015. MultiChain private blockchain-White paper. *URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf* (2015).
[30] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2020. Building high throughput permissioned blockchain fabrics: challenges and opportunities. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3441–3444.
[31] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. 2021. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing. In *Int. Conf. on Data Engineering (ICDE)*. IEEE.
[32] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. ResilientDB: Global Scale Resilient Blockchain Fabric. *Proceedings of the VLDB Endowment* 13, 6 (2020), 868–883.
[33] Suyash Gupta, Sajjad Rahnama, and Mohammad Sadoghi. 2020. Permissioned Blockchain Through the Looking Glass: Architectural and Implementation Lessons Learned. In *Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE.
[34] Maurice Herlihy. 2018. Atomic cross-chain swaps. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 245–254.
[35] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. Zcash protocol specification. *GitHub: San Francisco, CA, USA* (2016).
[36] Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi, Mike Dahlin, et al. 2012. All about Eve: Execute-Verify Replication for Multi-Core Servers.. In *Symposium on Operating systems design and implementation (OSDI)*, Vol. 12. USENIX Association, 237–250.
[37] 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 *Symposium on Security and Privacy (SP)*. IEEE, 583–598.
[38] Jae Kwon. 2014. Tendermint: Consensus without mining. *Draft v. 0.6, fall* (2014).
[39] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 7 (1978), 558–565.
[40] Leslie Lamport et al. 2001. Paxos made simple. *ACM Sigact News* 32, 4 (2001), 18–25.
[41] Yuan Lu, Qiang Tang, and Guiling Wang. 2018. Zebralancer: Private and anonymous crowdsourcing system atop open blockchain. In *Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 853–865.
[42] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *SIGSAC Conf. on Computer and Communications Security (CCS)*. ACM, 17–30.
[43] Sujaya Maiyya, Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. 2018. Database and distributed computing fundamentals for scalable, fault-tolerant, and consistent maintenance of blockchains. *Proceedings of the VLDB Endowment* 11, 12 (2018), 2098–2101.
[44] Sujaya Maiyya, Victor Zakhary, Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. Database and distributed computing foundations of blockchains. In *SIGMOD Int. Conf. on Management of Data*. ACM, 2036–2041.
[45] C Mohan. 2017. Tutorial: blockchains and databases. *PVLDB* 10, 12 (2017), 2000–2001.
[46] C Mohan. 2018. Blockchains and databases: A new era in distributed computing. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1739–1740.
[47] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[48] Faisal Nawab and Mohammad Sadoghi. 2019. Blockplane: A global-scale byzantizing middleware. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 124–135.
[49] Diego Ongaro and John K Ousterhout. 2014. In search of an understandable consensus algorithm. In *Annual Technical Conf. (ATC)*. USENIX Association, 305–319.
[50] PegaSys. [n.d.]. Scaling Consensus for Enterprise: Explaining the IBFT Algorithm. https://media.consensys.net/scaling-consensus-for-enterprise-explaining-the-ibft-algorithm-ba86182ea668.
[51] Ravi Kiran Raman, Roman Vaculin, Michael Hind, Sekou L Remy, Eleftheria K Pissadaki, Nelson Kibichii Bore, Roozbeh Daneshvar, Biplav Srivastava, and Kush R Varshney. 2018. Trusted Multi-Party Computation and Verifiable Simulations: A Scalable Blockchain Approach. *arXiv preprint arXiv:1809.08438* (2018).
[52] 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 *SIGMOD Int. Conf. on Management of Data*. ACM, 543–557.
[53] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *Computing Surveys (CSUR)* 22, 4 (1990), 299–319.
[54] 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 *SIGMOD Int. Conf. on Management of Data*. ACM, 105–122.
[55] Joao Sousa, Alysson Bessani, and Marko Vukolic. 2018. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 51–58.
[56] Parth Thakkar and Senthil Nathan. 2020. Scaling hyperledger fabric using pipelined execution and sparse peers. *arXiv preprint arXiv:2003.05113* (2020).
[57] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. 2018. Performance benchmarking and optimizing hyperledger fabric blockchain platform. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 264–276.
[58] Stefan Thomas and Evan Schwartz. 2015. A protocol for interledger payments. *URL https://interledger.org/interledger.pdf* (2015).

[59] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. 2013. Efficient byzantine fault-tolerance. *Transactions on Computers* 62, 1 (2013), 16–30.
[60] Lu Xu, Wei Chen, Zhixu Li, Jiajie Xu, An Liu, and Lei Zhao. 2020. Locking Mechanism for Concurrency Conflicts on Hyperledger Fabric. In *International Conference on Web Information Systems Engineering*. Springer, 32–47.
[61] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 347–356.
[62] Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. 2020. Atomic commitment across blockchains. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1319–1331.
[63] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling blockchain via full sharding. In *SIGSAC Conf. on Computer and Communications Security*. ACM, 931–948.
[64] Saide Zhu, Zhipeng Cai, Huafu Hu, Yingshu Li, and Wei Li. 2019. zkCrowd: a hybrid blockchain-based crowdsourcing platform. *Transactions on Industrial Informatics* (2019).