Blockchains vs. Distributed Databases: Dichotomy and Fusion

Pingcheng Ruan
National University of Singapore
ruanpc@comp.nus.edu.sg

Tien Tuan Anh Dinh
Singapore University of Technology and Design
dinhtta@sutd.edu.sg

Dumitrel Loghin
National University of Singapore
dumitrel@comp.nus.edu.sg

Meihui Zhang
Beijing Institute of Technology
meihui_zhang@bit.edu.cn

Gang Chen
Zhejiang University
cg@zju.edu.cn

Qian Lin
National University of Singapore
linqian@comp.nus.edu.sg

Beng Chin Ooi
National University of Singapore
ooibc@comp.nus.edu.sg

ABSTRACT

Blockchain has come a long way — a system that was initially proposed specifically for cryptocurrencies is now being adapted and adopted as a general-purpose transactional system. As blockchain evolves into another data management system, the natural question is how it compares against distributed database systems. Existing works on this comparison focus on high-level properties, such as security and throughput. They stop short of showing how the underlying design choices contribute to the overall differences. Our work fills this important gap and provides a principled framework for analyzing the emerging trend of blockchain-database fusion.

We perform a twin study of blockchains and distributed database systems as two types of transactional systems. We propose a taxonomy that illustrates the dichotomy across four dimensions, namely replication, concurrency, storage, and sharding. Within each dimension, we discuss how the design choices are driven by two goals: security for blockchains, and performance for distributed databases. To expose the impact of different design choices on the overall performance, we conduct an in-depth performance analysis of two blockchains, namely Quorum and Hyperledger Fabric, and two distributed databases, namely TiDB, and etcd. Lastly, we propose a framework for back-of-the-envelope performance forecast of blockchain-database hybrids.

CCS CONCEPTS

• **Information systems** → **Distributed database transactions**; *Database performance evaluation*; • **General and reference** → Surveys and overviews.

KEYWORDS

Taxonomy; Benchmark; Blockchain; Transaction; Database

ACM Reference Format:
Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen, Qian Lin, and Beng Chin Ooi. 2021. Blockchains vs. Distributed Databases: Dichotomy and Fusion. 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, 14 pages. https://doi.org/10.1145/3448016.3452789

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452789

1 INTRODUCTION

The very first blockchain system, that is Bitcoin [64], is a decentralized ledger for recording cryptocurrency's transactions. The ledger consists of multiple blocks chained together with cryptographic hash pointers, each block containing multiple transactions. This chain of blocks is distributed across a network of nodes, some of which behave in a Byzantine (or malicious) manner [58]. The network runs a consensus protocol, namely *proof-of-work* (PoW), to keep the ledger consistent among the nodes.

Bitcoin is the first digital currency (or cryptocurrency) system that operates in a Byzantine [58] peer-to-peer (P2P) environment, without relying on a common trusted third party. But it can execute only simple transactions that move some coins from one address (or user) to another. Recent blockchains such as Ethereum [82] and Hyperledger Fabric [23] support general-purpose transactions. The key enabler is the *smart contract* — a user-defined computation executed by all nodes in the blockchain. With smart contracts, blockchains can execute transactional workloads which have so far been handled almost exclusively by databases. In other words, blockchains have evolved into transactional management systems, and therefore are comparable to distributed databases. Their advantages over the latter include data transparency and security against Byzantine failures. In fact, many companies and government agencies are exploring blockchains to replace, or to complement, their enterprise-grade databases [40, 62, 63].

The parallel between blockchains and distributed databases has not gone unnoticed. Existing works show that there are little similarities between the two. Blockchains are suitable when the applications are running in untrusted, hostile environments, whereas databases are suitable when performance is more important than security [36, 40, 83, 85]. Their distinction is further compounded by the significant gap in performance [45], for instance Bitcoin processes around 10 transactions per second [59] while etcd — a state-of-the-art distributed NoSQL database — processes over 50,000 operations per second [47].

On the other hand, we notice the trend of design fusion between databases and blockchains. Design principles and techniques that are traditionally used by databases are being adopted by blockchains. For example, concurrency control techniques attributed to databases are used to increase the performance of blockchains [42, 71, 74]. Moreover, sharding has been used to scale out permissioned blockchains [41]. At the same time, the security features of blockchains are used in hybrid blockchain-database systems to provide verifiable data [21, 46, 68].

One limitation of the existing works that compare blockchains and databases is that they only focus on application-level, observable and measurable properties, such as throughput and security. In particular, they show how the two types of systems differ without identifying the root cause. For example, BLOCKBENCH [45] compares three private blockchains, namely Hyperledger Fabric, Ethereum and Parity, with H-Store under two popular data processing workloads. The authors expose a large gap in performance, but provide no further analysis of that gap. As a consequence, the reported difference does not generalize to workloads other than the two used in the experiments. For instance, under high contention workloads, the performance difference may shrink drastically.

To overcome these limitations, we aim to provide a comprehensive dichotomy of blockchains and databases. Our approach is to position them within the same design space — that is, the design space of general transactional systems. We propose a taxonomy consisting of four design dimensions and discuss how the two types of systems make different design choices in each dimension. The first dimension is replication, which determines what data is replicated to what nodes, and the mechanism needed to keep the replicas consistent. The second is concurrency, which determines the tradeoffs between performance and correctness when executing concurrent transactions. The third is storage, which determines the data models and access methods. The final dimension is sharding, which determines how data is partitioned, and the mechanism for atomicity of cross-shard transactions.

The four dimensions in our taxonomy capture the fundamental similarities betwen blockchains and databases. In addition, their impact on the overall performance can be measured, therefore these dimensions form a framework for fine-grained, quantitiative comparison between these systems. We demonstrate how our taxonomy is useful in practice by applying it to compare the performance of recent hybrid database-blockchain systems [21, 46, 61, 65, 68, 72].

In summary, we make the following contributions in this paper:

- We compare blockchains and distributed databases as two different types of distributed, transactional systems. We propose a new taxonomy that characterizes both types of systems and their hybrids along four design dimensions: replication, concurrency, storage, and sharding.
- We conduct a comprehensive performance study of four popular systems, including two permissioned blockchains, namely Fabric [10] and Quorum [16], and two database systems, namely TiDB [18] and etcd [7]. The results demonstrate the impact of different design choices on performance.
- We use our taxonomy to analyze the security and performance of emerging hybrid blockchain-database systems. We propose a framework that explains their performance differences and estimates the performance of future hybrid systems.

Section 2 provides relevant background, followed by a qualitative comparisons on the above four dimensions in Section 3. Section 4 and Section 5 discuss the experimental setup and results, respectively. Section 6 reviews related works before Section 7 concludes.

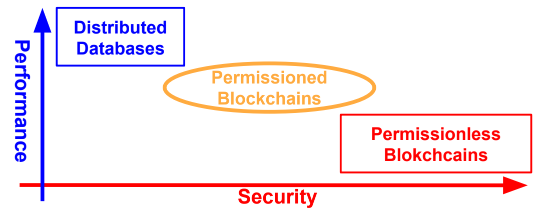


Figure 1: Blockchains vs. distributed databases in the security-performance coordinate.

2 BACKGROUND

In this section, we discuss relevant background on blockchains and distributed databases. Figure 1 shows a high-level comparison.

2.1 Blockchain

From a data structure perspective, a blockchain is a list of blocks linked by cryptographic hash pointers. These blocks contain cryptocurrency transactions [64]. By this definition, the blockchain is a tamper-evident ledger for recording transactions. With smart contracts, transactions are in the form of contract deployment and invocation. From a systems perspective, a blockchain is a distributed system consisting of multiple nodes, some of which are malicious. These nodes maintain a consistent ledger by using a Byzantine fault-tolerant (BFT) consensus protocol, such as PoW or PBFT [35].

In earlier designs, a blockchain transaction is restricted to cryptocurrency and the states are modeled as Unspent Transaction Outputs (UTXO). For example, Bitcoin [64] and other similar altcoins use the UTXO model. Starting with Ethereum [8], blockchains support *smart contracts* which allow users to encode and execute arbitrary Turing-complete computations on the ledger. The ledger states are modelled as accounts instead of UTXO. Other systems supporting smart contracts include Quorum, Parity and Hyperledger Fabric [23]. In these systems, a transaction on the ledger takes the form of a contract invocation. Transactions sequentially modify the system state based on their order in the ledger, determined by the consensus protocol. A read-only transaction can be carried out by any node, without undergoing the consensus and being included in the ledger. We only consider blockchains that support smart contracts in this paper, because earlier blockchains (without smart contracts) cannot support database transaction workloads and thus cannot be compared with distributed databases.

Permissionless vs. Permissioned. Blockchains can be broadly divided into two categories: permissionless (or public), and permissioned (or private). In the former, for example in Bitcoin and Ethereum, any node and user can join the system in a pseudonymous manner. In the latter, for example in Fabric and Quorum, the node and user must be authorized to join the system. With strong membership control and action regulation, permissioned blockchains are more suitable for enterprise applications and are particularly used in the financial sector. Figure 1 shows the security-performance tradeoffs in blockchains. It highlights how permissionless blockchains can achieve stronger security because they make

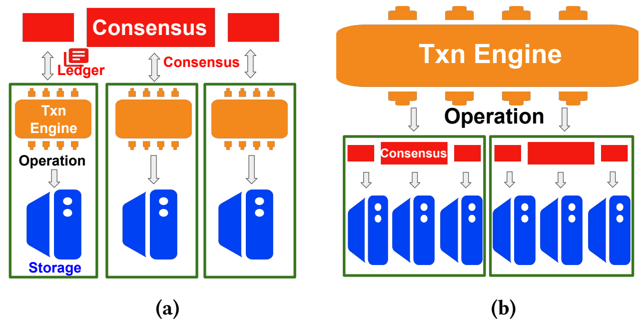


Figure 2: (a) Blockchains first reach consensus on the transaction history, then commit their effects into the storage. (b) Distributed databases replicate at the storage layer.

no identity assumption. In contrast, permissioned blockchains have weaker security because of the identity assumption, but can achieve higher performance because they can employ consensus protocols with higher efficiency. A more detailed discussion of permissionless versus permissioned blockchain designs can be found in [44, 45].

2.2 Distributed Databases

Unlike blockchains, database systems have been around for decades. Relational databases, which support easy-to-use SQL language and intuitive ACID transaction semantics, remained mainstream throughout the years. The recent demand of big data processing and the fact that Moore's law is reaching its limit are major factors behind the trend of scale-out database designs. Nowadays, both data and computation are distributed over multiple nodes in order to achieve high availability and scalability. Principles and techniques in designing and scaling distributed databases are described in detail in [67]. Basically, there are two distinctive movements, namely NoSQL and NewSQL, under this new distributed design direction.

NoSQL vs. NewSQL. For scalability, many distributed databases abandon the complex relational model and the strong ACID semantics. These systems are referred to as *NoSQL*. They support more flexible data models and weaker consistency. In the sense of the CAP theorem [50], these NoSQL systems compromise consistency for the sake of availability. A variety of their supported data models include key-value store (e.g, Redis [34], etcd [7]), document store (e.g, CouchDB [22]), graph store (e.g, Neo4J [80]), column-oriented (e.g, Cassandra [55]) and so on. The most lenient consistency model is eventual consistency which makes no guarantees about the order of read and write operations. Between eventual and strong consistency, researchers explore a variety of other abstractions, such as sequential, causal, and PRAM consistency. They standardize on the allowable operation behavior for the ease of reasoning. Most NoSQL databases offer configurable options, where users can trade off between performance and consistency.

The surge of NoSQL systems, however, does not obscure the cost in usability and the increase in application complexity. A new class of distributed database systems, called *NewSQL*, aim to restore the relational model and ACID semantics without sacrificing much scalability. NewSQL has drawn attention since Google introduced Spanner [39], the first NewSQL system. It was followed by a few database vendors, such as CockroachDB [5] and TiDB [18]. In this paper, we consider both NoSQL and NewSQL systems.

3 TAXONOMY

Table 1 compares the design choices of distributed databases and blockchains under each dimension in our taxonomy.

3.1 Replication

Replication is the technique of storing copies of the data on multiple nodes called replicas. The key challenge in such a system is to ensure consistency under failures. In this section, we characterize blockchains and distributed databases by what they replicate, how they keep the replicas consistent, and their failure models.

3.1.1 Replication model. The units of replication can be transactions or the read/write operations. Figure 2 shows that blockchains replicate an ordered log of transactions (or ledger). Distributed databases replicate the ordered log of read and write operations on top of the storage. The nodes in the database are oblivious to the transaction logic because they see only one operation at a time. One consequence of this model is that the transaction manager which coordinates the execution of a transaction must be trusted. In contrast, a blockchain does not have such trusted entity, therefore it replicates the entire transaction so that its execution can be replayed by each participant node.

By replicating transactions, the ledger contain application-level information, such as transaction context, client signature, execution timestamp, etc., making it easy to perform transaction verification. Due to this verifiability, blockchains are often used as a data and computing platform for mutually distrusting parties. On the other hand, replicating storage operations means there can be more concurrency, because operations can be replicated in different order but with the same effect on the storage.

3.1.2 Replication approach. There are two main approaches to maintain consistency among replicas. The first is primary-backup, which dedicates a replica as the primary which synchronizes its states with backup replicas. This is adopted by many databases. For example, Replex [77] that uses chain replication. Cassandra [55] uses the client as the primary to synchronize the replicas.

The second approach is state-machine replication, which essentially maintains an ordered log of operations/transactions on each replica. Each replica starts at the same initial state, then applies the operations/transactions in the log in the same order. Many systems use *consensus protocols*, such as Paxos [57], Raft [66], and PBFT [35], for the replicas to agree on the ordered log. Examples include Quorum [16], TiDB [18], Spanner [39]. Compared with the primary-backup, consensus protocols achieve the automatic primary failover, by introducing the view change. That is, when the progress halts, replicas may jointly agree to enter into a new epoch/view where a new primary is elected. Apart from the consensus, other systems rely on external services that provide a distributed *shared log* abstraction, such as Kafka [13] and Corfu [26]. Operations/transactions are appended to the log, and the replicas, as clients of the log, apply them independently. Examples of shared log systems include Fabric [10], Hyder [29], and Tango [27].

Primary-backup protocols are simpler, and can perform better than state-machine replication, when the states are small and there are no failures. For example, the chain replication protocol can spread the network cost more evenly among the replicas than a

Table 1: Design choices in blockchains and distributed databases

	Blockchains	**Distributed Databases**
Replication	Transaction replication Byzantine Fault Tolerant consensus: PBFT, PoW, etc.	State replication Crash Fault Tolerant consensus: Raft, Paxos, etc.
Concurrency	Serial execution	Concurrent execution
Storage	Append-only ledger abstraction Authenticated data structure: Merkle Tree, etc.	Direct access without historical query Hardware-conscious index: PSL, FAST, etc.
Sharding	Node-aware shard formation 2PC and BFT-based replication	Workload-aware shard formation 2PC with centralized coordinator

consensus protocol, and it achieves better read performance [77]. Systems based on shared log are expected to perform better than the ones based on consensus when there are no failures. This is because shared log decouples ordering from state replication, therefore it can be optimized to have high throughput. Furthermore, while the throughput of a consensus protocol decreases with more replicas, the throughput of a shared log system is expected to remain constant until the number of log consumers exceeds the capacity of the log producers [26].

3.1.3 Failure model. Replication protocols are complex because they need to maintain consistency under failure. Under the crash failure model, in which nodes only fail by crashing, the protocols need to tolerate hardware and software failures. Under the Byzantine failure model, in which nodes fail arbitrarily, the protocols need to tolerate any software and hardware failures, as well as any malicious behavior. This model is suitable for achieving security, since it considers attacks that fully compromise some nodes in the system.

An orthogonal dimension to the node failure model is the network assumption. The network is *synchronous* when the network delay is bounded and known. It is *asynchronous* when the network delay is unbounded. Protocols that tolerate crash failures, or CFT, require $f + 1$ replicas to tolerate f failures under the synchronous network model [32], and $2f + 1$ under the asynchronous network model [57, 66]. Protocols that tolerate Byzantine failure, or BFT, require $2f + 1$ and $3f + 1$ replicas to tolerate f failures under the synchronous and asynhronous network models, respectively [31, 35, 87].

Databases assume the crash failure model, since they are considered internal systems which are not subject to security attacks. For example, Spanner [17] uses Paxos [56], a CFT protocol. Permissioned blockchains support both failure models. For example, Quorum provides implementations for both Raft [66], a CFT protocol, and IBFT, a BFT protocol. These systems allow applications to make different tradeoffs between security and performance. Public blockchains, on the other hand, ubiquitously adopt BFT protocols because they admit any nodes to the system. In particular, PoW protocols are often used because they address one fundamental problem in the public settings: a node can have many identities. In PoW, a node's probability of solving a computational puzzle, thereby reaching consensus and gaining rewards, is proportional to its physical resources which are difficult to forge.

In an asynchronous network and under failures, the FLP theorem [48] rules out any deterministic consensus protocol that can achieve both safety and liveness. Public blockchains choose liveness over safety, meaning that the systems remain available under network partitions, but these partitions may be in disagreement which takes the form of forks. Here, availability refers to the system's behavior, that is, new blocks are appended to the ledger. Individual transactions may be censored and excluded from the blockchain.

PoW protocols have low throughputs mainly due to the resource requirements [45]. CFT protocols have better performance than BFT protocols, because the former incur $O(N)$ network cost, whereas the latter incur $O(N^2)$, where N is the number of nodes. As a result, BFT protocols do not scale to a large number of nodes, and their performance is more sensitive to network conditions at scale. Specifically, when N is large, BFT protocols are more likely to enter view change — an expensive phase of the protocol for replacing the current leader.

3.2 Concurrency

Concurrency refers to the extent to which transactions are executed at the same time. There are two choices: transactions are executed either serially (or sequentially), or concurrently. Most blockchains support only serial execution, while distributed databases employ sophisticated concurrency control mechanisms to extract as much concurrency as possible.

There are two reason behind blockchains' lack of support for concurrency. First, serial execution may not affect the overall performance because transaction execution is often not the bottleneck [45]. For example, in Bitcoin, the consensus protocol may take several minutes to complete, which is the block interval required by the protocol, whereas the transaction execution, which invokes the Bitcoin script to validate a cryptocurreny flow, can be done in milliseconds. Second, serial execution means the behavior of smart contracts is deterministic when the transaction execution is replicated over many nodes. The benefit of determinism is that it is easy to reason about the states of the ledger.

Unlike blockchains, concurrency remains a major research topic in databases, as it is the main source of performance improvement. The challenge in extracting concurrency is to ensure the correctness of the concurrent execution. In fact, there is a wide range of isolation levels [25, 37] which make different tradeoffs between correctness and performance. Most production-grade databases today offer more than one isolation level.

We observe that recent blockchains are adopting some simple concurrency techniques often found in databases. In Hyperledger Fabric, for example, transactions are simulated (executed) in parallel against the ledger states before being sent for ordering. During the later commit phase, the system uses a simple optimistic concurrency control to achieve serializability which aborts transactions whose simulated states are stale. More established techniques to reduce abort have also been proposed [71, 74].

3.3 Storage

3.3.1 Storage model. Storage can be built upon the latest states only, amenable for mutation, or upon all historical information, amenable for appending. The storage in distributed databases only exposes direct access to up-to-date records. In databases without explicit provenance support, historical data is maintained in limited forms, for example as write-ahead logs. We note that such logs are used primarily for failure recovery, and they are periodically pruned. Blockchains, besides the state storage, additionally expose an append-only ledger abstraction. The ledger, a chain of blocks, records historical transactions and the changes made to the global states. We note that such a ledger is hash protected to conserve historical integrity. Some blockchains allow applications to access only the latest states, for example, Hyperledger Fabric v0.6. Recently, novel blockchain-tailored storage systems have been proposed to enable access to any historical states during smart contract execution [69].

3.3.2 Index. Indexes play an instrumental role on the state storage to facilitate data access. Apart from the performance consideration, some security-oriented systems additionally rely on the index to compute a digest, which uniquely identifies the state contents. Distributed databases are more concerned by performance, i.e., any small optimization on the index can translate to a significant improvement in performance. Modern indexes are designed to be hardware-conscious in order to extract the most efficiency from the hardware. For example, in-memory databases abandon the disk-friendly B-tree structure for other structures such as FAST [53] and PSL [84] which are designed for better cache utilization and multi-core parallelism.

To compute the content-unique digest, blockchains employ an authenticated data structure, such as the Merkle tree index, to provide integrity protection on top of the state storage. For example, Ethereum uses a prefix trie, named Merkle Patricia Trie (MPT) [14]. In MPT, the states are stored in the leaves. The states with a common key prefix are organized under the same branch. Each node is associated with the cryptographic hash of its content in the storage engine, such that the root hash represents the complete global states. The access path serves as the integrity proof for the retrieved value. Older versions of Hyperledger Fabric use a Merkle Bucket Tree (MBT) in which the size of the tree is fixed. Unlike the ledger abstraction which is ubiquitous in blockchains, we note that not all blockchains adopt the authenticated data structure for the state organization. For example, Hyperledger Fabric abandons this design from version 1 onwards.

3.4 Sharding

Sharding is a common technique in distributed databases for achieving scalability, in which data is partitioned into multiple shards. Although it has been studied extensively in databases, sharding has only recently been introduced to blockchains to harness concurrency across shards. In this section, we discuss two key challenges in any sharded system, that are (i) how to form a shard, and (ii) how to ensure atomicity for cross-shard transactions.

3.4.1 Shard formation. A shard formation protocol determines which nodes and data go to which shard. The security of blockchains depends on the assumption that the number of failures is below a certain threshold. The shard formation protocol must, therefore, ensure that the assumption holds for every shard. In particular, the shard size must be large enough so that the fraction of Byzantine nodes is small. Furthermore, the attacker must not be able to influence the shard assignment, otherwise, it could reserve enough resources for one shard to break the security assumption. State-of-the-art sharded blockchains have different approaches. For example, Elastico uses PoW for shard formation [60], while the recent version of Ethereum uses Proof-of-Stake to select validators for each shard [9]. OmniLedger [54] employs a complex cryptographic protocol, while AHL [41] uses trusted hardware. These protocols are secured against Sybil attacks, and executed regularly, in the form of shard reconfiguration, to guard against adaptive attackers.

The goal of sharding in distributed databases is scalability. As such, the systems aim to assign data to shards in a way that optimizes the performance of certain workloads. In practice, they offer a variety of partitioning schemes, for example, hash partitioning and range partitioning, so that users can select the most suitable for their workloads. Some systems, for instance, Cassandra[55], even allow users to specify workload distributions so that data can be partitioned in a locality-aware manner. Unlike blockchains, shard reconfiguration is not necessary for databases, unless when there are significant changes in the workload distribution.

3.4.2 Atomicity. Sharding introduces the problem of transaction atomicity when a transaction touched data in multiple shards. Atomicity requires a cross-shard transaction to either commit or abort in all shards. In databases, this problem is addressed by the two-phase commit (2PC) protocol. This protocol requires a dedicated transaction coordinator that must be trusted, but may fail and leave the transaction blocked forever. A recent work proposed Parallel Commit to reduce the commit duration to a single round trip [76].

Sharded blockchains face additional challenges in ensuring atomicity because the coordinator cannot be trusted under the Byzantine failure model. To overcome this, Eth2 introduces a separate chain running Casper consensus [33], called Beacon Chain, that coordinates cross-shard transactions [9]. Similarly, [41, 51] propose to implement the 2PC coordinator as a state machine in a shard that runs a BFT protocol. The BFT protocol ensures that the shard is less vulnerable to attacks and does not become a point of failure. Any cross-shard transaction must involve this 2PC BFT replicated state machine to ensure atomicity. The consensus liveness guarantees the high availability of the coordinator, therefore mitigating the blocking problem. But the Byzantine setup in blockchains imposes considerable overhead to the 2PC process.

3.5 Fusion of Blockchains and Databases

The taxonomy above provides a comprehensive description of the design space of distributed transactional systems. This taxonomy

Table 2: System comparison based on our taxonomy with the benchmarked ones and their versions highlighted. For each hybrid system, we mark their security-oriented designs with red and performance-oriented designs with blue.

Category	System	Replication: Replication Model	Replication: Replication Approach	Replication: Failure Model (Consensus Protocol)	Concurrency	Storage: Ledger Abstraction	Storage: Index(Storage Engine)	Sharding Support (2PC)
Permissionless Blockchains	Ethereum [8]	Txn-based	Consensus	BFT(PoW)	Serial	✓	LSM Tree(LevelDB)+MPT	✗(✗)
	Eth2 [9]	Txn-based	Consensus	BFT(PoS + Casper)	Serial (in each shard)	✓	LSM Tree(LevelDB)+MPT	✓(✗)
Permissioned Blockchains	Quorum v2.2 [16]	Txn-based	Consensus	Raft(CFT)/IBFT(BFT)	Serial	✓	LSM Tree(LevelDB)+MPT	✗(✗)
	Fabric v2.2 [10]	Txn-based	Shared log	CFT(*Orderer* with Raft)	Concurrent Execution Serial Commit	✓	LSM Tree(LevelDB)	✗(✗)
	Fabric v0.6 [11]	Txn-based	Consensus	BFT(PBFT)	Serial	✓	LSM Tree(RocksDB) + MBT	✗(✗)
	EOS [6]	Txn-based	Consensus	BFT(DPoS)	Serial	✓	B-tree(MongoDB)	✗(✗)
	FISCO BCOS [12]	Txn-based	Consensus	CFT(Raft) BFT(PBFT)	Serial	✓	LSM Tree(LevelDB) +MPT	✗(✗)
NewSQL Databases	TiDB v4.0 [18]	Storage-based	Consensus	CFT(Raft)	Concurrent	✗	LSM Tree(TiKV)	✓(✓)
	CockroachDB [5]	Storage-based	Consensus	CFT(Raft)	Concurrent	✗	LSM Tree(RocksDB)	✓(✓)
	Spanner [17]	Storage-based	Consensus	CFT(Paxos)	Concurrent	✗	LSM Tree	✓(✓)
	H-store [52]	Storage-based	Primary-backup	CFT	Concurrent	✗	B Tree	✓(✓)
NoSQL Databases	Etcd v3.3 [7]	Storage-based	Consensus	CFT(Raft)	Serial	✗	B Tree(BoltDB)	✗(✗)
	Cassandra [4]	Storage-based	Primary-backup	CFT	Concurrent	✗	LSM Tree	✓(✗)
	DynamoDB [1]	Storage-based	Primary-backup	CFT	Concurrent	✗	B Tree	✓(✗)
Out-of-the Blockchain Databases	BlockchainDB [46]	Storage-based	Consensus	BFT(PoW)	Serial (in each shard)	✓	LSM Tree(LevelDB) +MPT	✓(✗)
	Veritas [21]	Storage-based	Shared log	CFT(Kafka)	Concurrent Execution Serial Commit	✓	Skip List(Redis)	✗(✗)
	FalconDB [68]	Storage-based	Consensus	BFT(Tendermint)	Concurrent Execution Serial Commit	✓	B Tree(MySQL) +Merkle Tree(IntegriDB)	✗(✗)
Out-of-the Database Blockchains	Blockchain Relational Database (BRD) [65]	Txn-based	Shared log	CFT(Kafka) BFT(BFT-SMaRt)	Concurrent	✓	B Tree(PostgreSQL)	✗(✗)
	ChainifyDB [72]	Txn-based	Shared log	CFT(Kafka)	Concurrent	✓	B Tree (MySQL/PostgreSQL)	✗(✗)
	BigchainDB [61]	Txn-based	Consensus	BFT(Tendermint)	Concurrent	✓	B Tree(MongoDB)	✗(✗)

helps in illustrating the similarities and differences between blockchains and distributed databases. It also serves as a principled framework for understanding the recently emerging hybrid blockchain-database systems. In this section, we discuss how these systems fit into the design space. We provide a deeper analysis of their performance in Section 5.6.

Out-of-the-blockchain Databases. One approach toward a hybrid design is to start with a blockchain (or a blockchain-like system) and build database features on top of it. Examples of this approach include BlockchainDB [46], Veritas [21], and FalconDB [68], which provide shared and verifiable databases for multiple distrusting parties. They use blockchains as an integrity-protected storage, and build other database components on top of it. In these systems, replication is transaction-oblivious, with duplicated states, logs, and meta-data. BlockchainDB replicates storage operations and uses PoW for consensus. It inherits the authenticated state organization from the underlying blockchain and employs multiple blockchains for storage. Therefore, it is amenable to sharding. However, transactions are executed sequentially within each shard. FalconDB and Vertias also adopt storage-based replication, but use Tendermint [31] for consensus and Kafka [13] as the shared log, respectively. They use a similar optimistic concurrency control mechanism as Fabric. FalconDB outsources the authentication task to IntegriDB [91], which enables a light-weight client to produce a proof without holding the entire ledger. Veritas relies on trusted verifiers for the state integrity.

Out-of-the-database Blockchains. Another hybrid design approach is to start with a database, then add blockchain features to it. Examples of this approach include BigchainDB [61], Blockchain Relational Database (BRD) [65], and ChainifyDB [72]. In these systems, each node has its own database and executes transactions on its database according to a global order achieved through consensus. These systems adopt the transaction-based replication model where the ledger serves as a secure shared log that stores transactions. The nodes execute the same sequence of transactions, but on different local databases. In particular, BRD uses PostgreSQL [75] and the transactions contain invocation contexts of stored procedures. BigchainDB uses MongoDB [15], thus its transactions are in JSON format. ChainifyDB allows heterogeneous relational databases, and transactions are in the form of standardized SQL statements. ChainifyDB uses a Kafka broker to share logs for efficiency. In contrast, BigchainDB uses the Tendermint consensus protocol which tolerates Byzantine failures at the expense of performance. BRD jointly uses Kafka [13] and BFT-SMaRt [30], an implementation of PBFT. These systems inherit the concurrency support of their underlying databases, with serializable constraints according to the ledger order. However, these systems do not protect the local states with Merkle trees and only rely on the integrity protection of the ledger. Finally, these systems do not support sharding.

In summary, out-of-the-database blockchains retain many design choices of distributed databases, as their main goal is performance. In contrast, out-of-the-blockchain databases inherit many blockchain features, as they are more security-driven. Some centralized and in-cloud databases, also learning from blockchains, rely on a hashed chain for verifiable transactions. Examples include Spitz [90], QLDB [2] and LedgerDB [86]. Some systems provide tailored optimizations on the ledger-like structure like LogBase [79].

3.6 Discussion

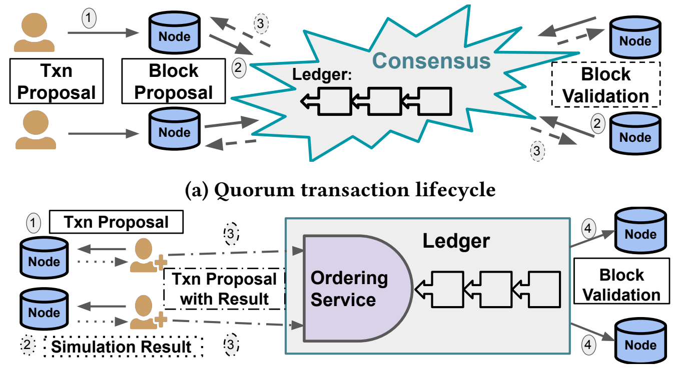
Table 2 summarizes some representative transactional systems and their design choices based on our proposed taxonomy. We only consider blockchains with generic smart contracts and NoSQL databases with key-value data model. We exclude permissionless blockchains from our quantitative analysis, as their security-performance tradeoffs have been extensively studied [49]. One can observe from Table 2 that the hybrid systems, just like permissioned blockchains, share some security-oriented design choices with blockchains and some performance-oriented design choices with databases.

4 EXPERIMENTAL SETUP

4.1 Systems

We select four representative systems: two permissioned blockchains, namely Quorum [16] and Hyperledger Fabric [10], and two distributed databases, namely TiDB [18] and etcd [7]. Quorum represents order-execute blockchains, while Fabric represents execute-order-validate blockchains. They also employ different replication approaches, as shown in Figure 3. Fabric employs an external ordering service while Quorum relies on Raft consensus. Quorum is a fork of *geth*, the Golang implementation of Ethereum. Quorum replaces the original Proof of Work (PoW) of Ethereum with a CFT protocol, namely Raft, and a BFT protocol called Istanbul BFT (IBFT). However, it inherits Ethereum Virtual Machine (EVM) to invoke smart contracts. Fabric is featured for its modularized design. In particular, a node role is separated into *orderer* and *peer*, as detailed in Figure 3b.

TiDB [18] and etcd [7] represent NewSQL and NoSQL distributed databases, respectively. TiDB consists of three independent modules, namely Placement Driver for coordinating cluster management, TiKV as the replicated key-value storage, and TiDB-server for parsing and scheduling SQL queries in a stateless manner. TiDB only supports snapshot isolation. Etcd provides a simple key-value data model with relaxed transactional restrictions but focuses on the tradeoff between availability and consistency. Similar to blockchains, etcd employs a single consensus instance to sequence all the requests. Without sharding, etcd fully replicates the data on each node. We also benchmarked CockroachDB [5], another NewSQL database. Since it exhibits similar performance trends as TiDB, we decide to omit it in this paper.



(a) Quorum transaction lifecycle

(b) Fabric transaction lifecycle

Figure 3: Transaction execution in Quorum vs. Fabric. In Quorum, a node assembles pre-executed transactions into blocks before running the consensus. In Fabric, a client collects simulation results and endorsements from peer nodes to form a transaction. Orderer nodes order the transactions and batch them into blocks, which are then pulled by the peer nodes for independent validation and commit.

4.2 Setup

For a fair comparison, we run all systems in full replication mode where each node has a complete copy of the states. In particular, for Fabric the endorsement policy is set such that a transaction is executed and endorsed by all peers. For TiDB, we set the replication factor to be the same as the number of nodes. In other words, even though TiDB partitions data to multiple shards and manages the shards separately, each node has a copy of the entire system state. We configure Quorum and Fabric to use Raft, a CFT consensus. For Fabric, we fix the number of orderers to three while scaling the peers. For TiDB, we scale all its modules with the number of nodes.

Unless otherwise specified, we use YCSB and Smallbank workloads in our experiments. The experiment parameters for YCSB are summarized in Table 3 with the default values underlined. For the database experiments we use the open-source driver for YCSB [19] and the OLTPBench [43] driver for Smallbank. Both Fabric and Quorum are benchmarked using Caliper [3]. We note that although there are differences in the types of drivers for benchmarking blockchains and databases, they alone do not account for the large performance gap reported in the following section.

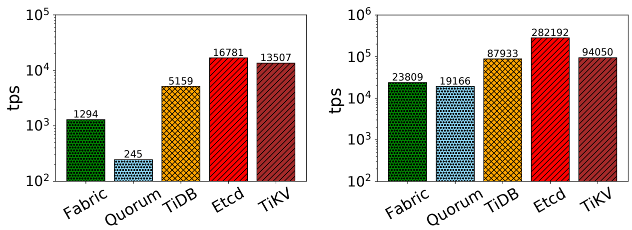
Our experiments are conducted on an in-house cluster consisting of 96 nodes connected via 1Gb Ethernet. Each node is equipped with Intel Xeon E5-1650 CPU, 32GB RAM, and 2TB hard disk. All the experiments are repeated three times and we report the average.

5 RESULT AND ANALYSIS

We first summarize the main findings, then provide the detailed experimental analysis. Based on these findings, we propose an empirical framework that compares the performance of recent hybrid blockchain-database systems. The framework not only explains the performance differences in existing systems, but it is also useful for understanding future hybrid systems.

Table 3: Experiments parameters

Variable	Values
Record size (Byte)	10, 100, 1000, 5000
Zipfian coefficient θ	0.0, 0.2, 0.4, 0.6, 0.8, 1.0
# of transaction operations	1, 2, 4, 6, 8, 10
# of nodes	3, 5, 7, 11, 15, 19



(a) Update **(b) Query**

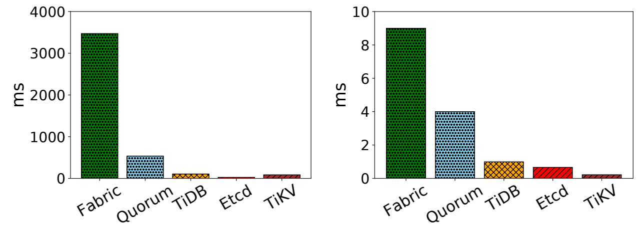
Figure 4: Throughput of YCSB workload (log scale).

- **Peak performance.** The performance gap between blockchains and distributed databases is large. However, the gap is not as significant as previously reported.
- **Replication.** The transaction-based replication model restricts concurrency, which limits the impact of different replication approaches and failure models on the system's peak performance.
- **Concurrency.** Execute-order-validate blockchains have low performance under workloads with high contention and constraints. The impact of workloads on the performance is prominent in NewSQL databases, where concurrency is on top of replication.
- **Storage.** The ledger abstraction in blockchains incurs significant storage overhead. On the other hand, the overhead needed to guarantee state tamper evidence is small.
- **Sharding.** The performance of sharded blockchains is far behind that of distributed databases, due to the security requirements on shard formation and periodic reconfiguration.

5.1 Peak Performance

5.1.1 YCSB. We first analyze the peak performance of the four systems under the default configurations shown in Table 3. Specifically, we populate each system with 100K records, each of size 1 KB. We then measure the throughput and latency against two YCSB workloads: uniform update-only (100% writes) and uniform query-only (100% reads). We also measure independently the performance of TiKV, the replicated storage of TiDB, and include it in this comparison.

Figure 4 shows the peak throughput of the five systems. The relational NewSQL database (TiDB) outperforms the blockchains, while the replicated storages (etcd and TiKV) outperform the relational database. Specifically, the two blockchains achieve update throughputs of below 1500 transactions per second (tps), whereas TiDB achieves 5159 tps. The two key-value storages, etcd and TiKV, achieve around 15, 000 tps. Both outperform the NewSQL database



(a) Update **(b) Query**

Figure 5: Latency of YCSB workload.

because they do not incur the overhead of supporting ACID transactions. This is evidenced by the gap between TiDB and TiKV, caused by the overhead of the TiDB-server that wraps around the key-value storage. But this gap is less evident under the query workload, as ACID semantics impose less constraints on read-only transactions.

Figure 5 shows the latency when the systems are unsaturated. Similar to throughput, we observe a clear separation between the blockchains and the databases. We note that the blockchains have weaker guarantees for read-only transactions compared to those offered by the databases (linearizability). Responses to read requests in the former still take longer (up to 6× in Fabric) than the linearizable reads in the latter. The update (query) latency in Fabric and Quorum is around 3500ms (9ms) and 500ms (4ms) respectively, while in databases it is below 100ms (1ms).

Our results confirm the conclusion drawn in [45] that the performance of blockchains lags far behind state-of-the-art databases. However, we observe a smaller gap than that reported in [45]. In particular, the relational database, TiDB, achieves 4× greater throughput than the fastest blockchain, Fabric, under the uniform update workload (5159 vs. 1294 tps). This is in contrast with [45], where H-Store exhibits more than 120× speedup over blockchains. The key reason is that H-Store is an in-memory, distributed database with primary-backup replication. H-Store represents an extreme point of the design space that makes it rather dissimilar to blockchains. In contrast, all systems considered in our work incur some overheads from the consensus protocols.

5.1.2 Smallbank. Figure 6 compares the OLTP performance under the Smallbank workload. The request key follows a Zipfian distribution with coefficient $\theta = 1$ on 1M records. We do not include etcd because it does not support general transactional workloads. Compared to YCSB, besides skewness, a Smallbank transaction imposes more constraints and may touch up to two records, but the record size is smaller. To our astonishment, the experiments show that the performance difference between blockchains and distributed databases is small. For example, Fabric and Quorum exhibit throughputs of 835 and 655 tps, respectively, while TiDB exhibits only 1031 tps. The performance of Fabric and TiDB drops when switching from YCSB to Smallbank, while the performance of Quorum improves with a peak throughput under Smallbank that is 2.5× greater compared to YCSB. We attribute this improvement to the smaller record size of Smallbank. As we shall see in Section 5.3.3, Quorum's performance is vulnerable to transactions that access large records. Likewise, the request skewness accounts for the throughput drop reported by Fabric and TiDB.

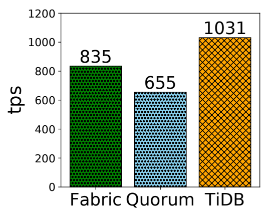
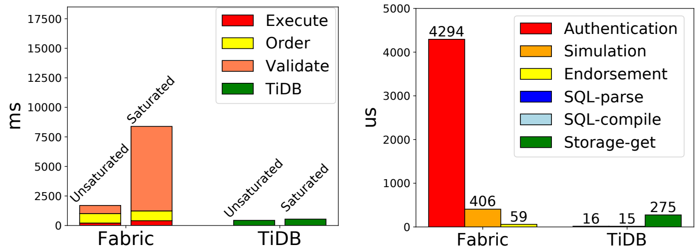


Figure 6: Throughput of the skewed Smallbank workload (1M records).



(a) Update **(b) Query**

Figure 7: Latency breakdown.

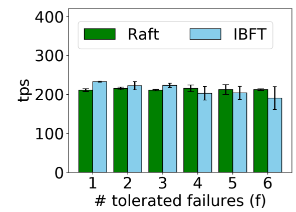


Figure 8: Quorum throughput with CFT(Raft) and BFT(IBFT).

Table 4: Throughput (in tps) with varying number of nodes under full replication mode.

	3	**7**	**11**	**15**	**19**
Fabric	1560	1288	1031	749	528
Quorum	237	236	229	217	219
TiDB	5697	7884	7544	6239	5526
Etcd	19282	16453	11243	7801	6076

5.2 Replication

5.2.1 Effect of replication model. To understand the impact of the replication model, we focus on Fabric and TiDB because they support different transaction lifecycles. Figure 7a compares the latency of a transaction when the systems are both unsaturated and saturated. Besides its higher latency compared to TiDB, Fabric exhibits a significant increase in latency when the system is saturated. To investigate this issue, we instrument Fabric codebase to record detailed latency breakdown at each phase of a transaction. In particular, we measure the latency of the execute, order, and validate phases. When Fabric is unsaturated, the order and validate phases take roughly 700ms each, while the execute phase takes below 500ms. But when the request rate exceeds the system capacity, validation phase becomes the bottleneck, as shown in Figure 7a.

We attribute this increase in latency to the serial validation of blocks in Fabric, where blocks pile up before committing their transactions. Even inside a block, transactions persist their effects sequentially based on their internal order. Worst still, substantial overhead in transaction processing is attributed to factors other than data processing. For example, we observe that Fabric, under the saturated scenario, spends 42% of the block validation time to verify the transaction signature. We note that serial validation is Fabric's implementation choice, i.e., it could commit transactions concurrently. However, most of the blockchains impose a strict transaction order to achieve deterministic execution for security reasons. In contrast, database transactions do not suffer from such strict sequentiality under their storage-based replication, nor do they incur security overhead.

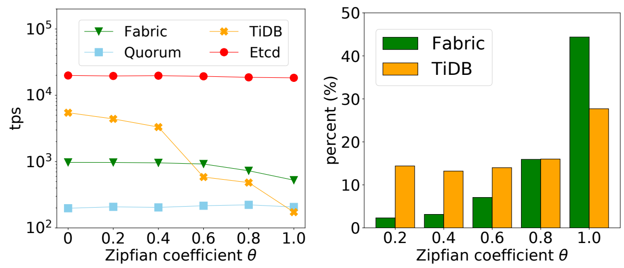
The security overhead is the most prominent in query transactions, which involve no consensus in both systems. We show in Figure 7b that Fabric spends most of the query time to authenticate the clients. In contrast, TiDB incurs no cryptographic overhead and most of its query time is spent on getting the data.

5.2.2 Effect of replication approach. We increase the number of nodes to compare the scalability of shared log and consensus-based systems under the full replication mode, and summarize the results in Table 4. Here, Fabric is the only shared log system. Even though Fabric employs the Raft consensus to obtain the transaction order, this is an external service with 3 fixed orderers. The increasing number of Fabric peers consume the same shared ordered log while for the other systems, all their nodes participate in the consensus.

Contrary to our expectations on the two blockchains, we observe neither a constant performance of the shared log system nor performance degradation in the consensus-based system. In particular, Fabric's throughput drops 3× from 3 to 19 nodes, while Quorum's throughput is roughly unchanged. In Fabric, we find a 38% increase in the block validation latency. This is because the endorsement policy requires a transaction to be endorsed by all the nodes. Hence, more nodes lead to transactions with more signatures and, therefore, longer validation. Due to the sequentiality in transaction-based replication, this increase in validation time translates to the decrease in throughput, as we explained in Section 5.2.1. On the other hand, Quorum underutilizes Raft, making its performance insensitive to the consensus group size. Specifically, Quorum first pre-executes transactions at the tip of the ledger, before batching these transactions into a block for the consensus. Thus, the block proposal rate is affected by the ledger's sequentiality.

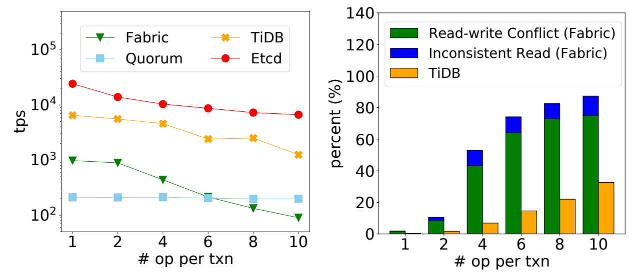
Under the same Raft protocol, the NoSQL database, etcd, achieves higher peak performance compared to the blockchains, but the performance degrades with the number of nodes. We attribute this to the consensus protocol. The NewSQL database does not exhibit either a constant or decreasing performance trend. Instead, TiDB reaches its peak performance on 7 nodes. This is because of the interplay between the transaction processing on TiDB servers and data storage on TiKV nodes. A detailed analysis can be found in our extended paper [70] . Finally, we conclude that the transaction-based replication model has an obvious impact on the performance of blockchains, while replication approaches have plain effects on the performance of distributed databases.

5.2.3 Effect of failure model. We compare the performance of Raft and Istanbul Byzantine Fault Tolerant (IBFT) consensus in Quorum to illustrate the impact of different failure models. Recall that Raft tolerates only crash failures, whereas IBFT can tolerate Byzantine failures. IBFT shares the crux of PBFT, which consists of a three-phase commit. But IBFT is heavily optimized for blockchains. For example, by embedding the consensus meta-data in the ledger, IBFT saves PBFT checkpointing efforts. IBFT additionally accommodates dynamic validators, while PBFT assumes fixed membership.



(a) Throughput (log scale) (b) Abort rate

Figure 9: Throughput and abort rate with skewed workloads. Each transaction modifies a single record.



(a) Throughput (log scale) (b) Abort rate

Figure 10: Throughput and abort rate with uniformly modified records in a single transaction.

Figure 8 shows similar peak throughputs that remain relatively constant when increasing the number of tolerated failures. However, we observe that IBFT's throughput exhibits higher variance in larger networks, as evidenced by the greater error bar. This is due to the larger quorums needed in IBFT, which are $2f + 1$ out of $3f + 1$, compared to $f + 1$ out of $2f + 1$ replicas needed in Raft. IBFT needs to contact more replicas in a time window compared to Raft to avoid the view change, during which the corresponding transaction processing is interrupted. When f increases, the probability of such interruption increases accordingly, hence, this leads to larger variances in performance.

5.3 Concurrency

5.3.1 Effect of skewness. To analyze the effect of concurrency control mechanisms, we use skewed workloads in which each transaction modifies (first read, then update and write back) a single record. The records' keys follow a Zipfian distribution that varies based on the skewness coefficient θ. Figure 9 shows the throughputs and the corresponding abort rates under different skewness. Our key observation here is that blockchains and databases are comparable under a high contention workload, given the fact that TiDB drastically drops from 5461 to 173 tps when θ increases from 0 to 1. Etcd and Quorum do not have concurrency control because they execute transactions serially. Thus, their performance is not affected by skewness.

Although Fabric commits transactions sequentially, we observe a 31% drop in throughput from a uniform to a skewed workload with $\theta = 1$. This is due to Fabric's optimistic concurrency control on read-write conflicts. That is, a transaction contains the versions of the records read during the proposal phase, which are then checked in the validation phase. If the versions are not the latest, the transaction aborts. A skewed workload means that many transactions are accessing the same records, leading to a higher probability of transaction abort. For example, Figure 9b shows that 44% of the transactions in Fabric abort when $\theta = 1$.

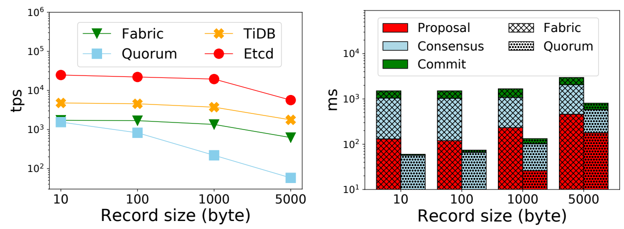
Another interesting observation is that TiDB's throughput drop is disproportional to its increase in abort rate. Specifically, when $\theta = 1$, only 30% of TiDB's transactions fail but the throughput decreases by 90%. This is because each transaction coordinator must obtain a latch on a primary record, whose write outcome determines the overall transaction status. But write must undergo the consensus for replication. Under a highly skewed workload, such a latching mechanism makes the transaction coordinator spend more time on contention resolution than the actual execution of the transaction payload, resulting in a remarkable decrement of the overall throughput. Hence we conclude that the workload skewness exerts a tremendous impact on storage-based replicated, concurrency-over-replication architectures.

5.3.2 Effect of operation count. We gradually include more update operations per transaction to analyze the impact of transaction atomicity on performance. To remove the effect of transaction size, for a given number of operations we vary the record size such that the total transaction size is 1000 bytes. For example, if a transaction writes 10 records, then each record contains 100 bytes.

As shown in Figure 10a, the performance of Fabric, TiDB, and etcd drops when the number of operations per transaction increases. In particular, with 10 operations per transaction, TiDB achieves only 32% of the throughput of single operation transactions. Two sources of overheads contribute to this drop in performance. First, there are more conflicts when a transaction writes to more records, which leads to a higher abort rate. Second, TiDB uses sharding, which means that a 10-operation transaction may span multiple shards. As there are more shards, the overhead of the 2PC coordination in TiDB increases. Etcd and Quorum are unaffected because they do not entail cross-shard transactions.

Figure 10b shows the abort rate of TiDB and Fabric as the number of operations per transaction increases. Both systems experience high abort rates: 26.9% for TiDB and 87% for Fabric. Interestingly, while TiDB aborts are mostly due to write-write conflicts, aborts in Fabric come from two sources: inconsistent reads and the read-write conflicts. On the one hand, during the proposal phase in Fabric, a client must collect identical read results from the peers. This is because we mandate that each transaction proposal must be simulated and endorsed by all peers. But different results may be returned, as the peers have disjoint states, which is highly likely since they commit blocks at different rates. In this case, the client immediately aborts the transaction. On the other hand, any of the modified records exhibiting a read-write conflict may render the transaction invalid. Under 10 operations per transaction, these two sources take up 14% and 86% of all the aborts, respectively.

5.3.3 Effect of record size. We enlarge the record in the uniform-update workload to increase the complexity per transaction without aggravating the inter-transaction conflicts. As shown in Figure 11a, all the databases exhibit moderate throughput decrease and latency increase. However, the two blockchains behave differently. When



(a) Throughput **(b) Latency breakdown**

Figure 11: Performance under uniform update workload with increasing record size. Both plots use log scale.

the record grows from 10 to 1,000 bytes, Fabric's performance remains roughly constant at 1400 tps and drops by half on 5,000 bytes. But Quorum suffers a significant drop in throughput, from 1547 tps on 10-byte to 58 tps on 5000-byte records. To understand this, we analyze the transaction latency breakdown in Fabric and Quorum, and present the results in Figure 11b. The block commit time in Fabric only doubles, whereas in Quorum there is a 70× increase from 3ms for 10-byte records to 221ms for 5000-byte records, reducing the proportion of the consensus from 88% to 50% in a transaction lifecycle. For each commit, Quorum's virtual machine needs to reconstruct an MPT data structure, which involves many expensive cryptographic hash computations. At the same time, the cost of a hash function increases with the record size. In particular, we find that the cost of MPT reconstruction increases from 56us to 2.5ms when the record size grows from 10 to 5000 bytes.

Another interesting observation from Figure 11b is that the delay of the proposal phase in Quorum grows at the same rate as the delay of the commit phase. This is due to Quorum's order-execute model, where transactions are firstly batched and serially executed during the proposal phase by the proposer. After consensus, the batched transactions are serially executed again by all the other nodes for validation and commit. Hence, Quorum's performance suffers from both double execution and the overhead of sequential validation of in-block transactions. In contrast, Fabric adopts an execute-order-commit model where transactions are executed concurrently during the proposal phase, before being ordered and batched in the consensus phase. The serial processing only occurs once during the commit phase. However, concurrency comes at the cost of potentially aborted in-block transactions that would break the serializability, as we saw in the previous section. Hence, when the transactions are computationally heavy, execute-order-commit blockchains outperform order-execute blockchains by introducing the sequentiality requirement later. But compared with blockchains, NewSQL databases with storage-based replication can harness more concurrency.

5.4 Storage

5.4.1 Effect of record size on storage. Figure 12 shows the storage cost per record as we increase the record size. Fabric incurs a much higher storage overhead than TiDB. For a 5000-byte record, the state storage consumes around 5000 bytes, while the block storage consumes 21,725 bytes. There is no additional storage used by TiDB because no historical information is maintained and the associated metadata is negligible. This result demonstrates that blockchains incur significantly higher storage costs than databases because of the underlying ledger abstraction.

5.4.2 Security overhead for tamper evidence. To quantify the overhead incurred by the integrity protection mechanism in blockchains, we compare the performance of Merkle Bucket Tree (MBT) from Hyperledger Fabric v0.6[1] and Merkle Patricia Trie (MPT) from Quorum. This comparison is done on the system behavior in its entirety. We refer readers to [88] for an in-depth analysis.

For this comparison, we insert 10K records of different sizes and measure the state storage cost per record. Figure 13 shows that MBT adds extra 24 bytes per record, while MPT adds over 1KB per record. Since both MBT and MPT store data records in the leaves, their differences come from the tree structures: the deeper the tree, the higher the storage overhead. The scale of MBT is fixed. Specifically, MBT first hashes all the records into 1,000 buckets, on top of which a Merkle tree with a given fan-out is built. Considering 1,000 buckets and a fan-out of 4 in our experiments, the depth of the tree is capped at 5 ($\lceil log_4 1000 \rceil$). As a prefix tree, the depth of MPT is affected by the key length, which is 16 bytes in our setting. Specifically, each internal MPT node holds 4 bits of the key, hence, the depth and fan-out can go up to 32 and 16, respectively. This explains why MPT needs more space.

5.5 Sharding

To compare the impact of sharding on databases and blockchains, we disable full replication in TiDB, and compare its performance with Spanner, a cloud-based NewSQL database, and Attested Hyperledger (AHL) [41], a state-of-the-art sharded blockchain based on Hyperledger Fabric v0.6. AHL leverages trusted hardware to reduce shard size and to improve throughput per shard. It supports cross-shard transactions by running a BFT shard that implements a 2PC state machine, and periodically reconfigures shards to mitigate adaptive adversaries. This experiment is run on Google Cloud Platform since Spanner is a cloud-only service. We set the number of nodes in a shard to 3 for all the systems, and we pre-populate the state with 1M 1KB-size records. We evaluate the systems with a skewed workload with a Zipfian coefficient of $\theta = 1$, in which each transaction modifies two records.

Figure 14 shows that TiDB achieves higher throughput compared to Spanner when increasing the number of nodes (and shards). This is because TiDB instantly aborts a transaction once detecting a conflict. In contrast, conflicting transactions in Spanner would contend for locks under the pessimistic concurrency control. To achieve stronger security, AHL with periodic shard re-configuration trades off 30% in performance compared to AHL with fixed shards. Nonetheless, the gap between AHL and both the databases is large, due to the high cost of PBFT and other security overheads.

5.6 Performance of Hybrid Systems

Based on our taxonomy and experimental results, we propose a framework for comparing the performance of existing hybrid systems. We emphasize that the framework only supports high-level, back-of-the-envelope comparison, and is not a replacement for detailed experimental analysis. It focuses on throughput as the key

[1] Fabric v1.0 and later relax the security model and no longer require tamper-evident indexes.

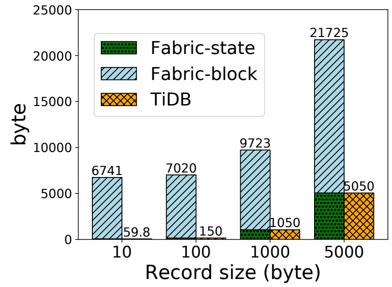


Figure 12: Storage breakdown in Fabric and TiDB.

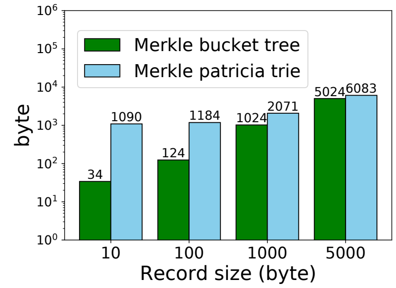


Figure 13: Storage overhead to achieve tamper evidence (log scale).

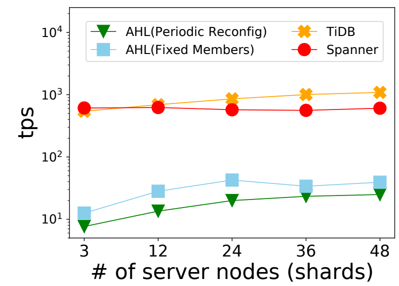


Figure 14: Throughput of the skewed workload (log scale).

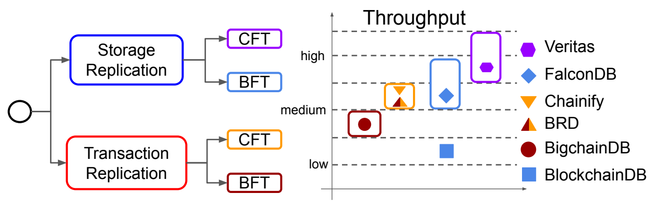


Figure 15: The framework for understanding the throughput of hybrid systems. The systems are color-coded based on the design choices.

performance metric and does not consider all the dimensions in our taxonomy. However, this framework explains the performance differences among systems according to their reported results. More importantly, it can guide the design of future hybrid systems.

Figure 15 presents our framework together with the reported performance of some hybrid systems. We note that the replication model is the deciding factor in determining the peak throughput. The results in Section 5.2.1 show that the replication model affects concurrency. In particular, transaction-based replication exposes lower concurrency than storage-based replication, which results in lower throughput. The next factor that affects throughput is the failure model. As explained in Section 3.1.3, CFT protocols are more efficient than BFT protocols due to their lower network overhead, therefore, systems using CFT are likely to have higher throughput. This is true especially when the CFT protocol is implemented as a shared log service. We note that even though our experiments do not show much difference between CFT and BFT in Quorum, it is because these protocols are not the bottleneck.

Figure 15 illustrates the reported performance of six hybrid systems within our framework. Using the two factors stated above, we can predict the throughput effectively. For instance, Vertias exhibits better throughput than Chainify (29*k* vs. 6.1*k*) because it uses storage-based replication and CFT protocols. But its performance has a high variance because, under high contention, the throughput can decrease significantly, as explained in Section 5.3.

6 RELATED WORK

Comparison. Existing works that compare blockchains and databases have highlighted their high-level differences. [44] demonstrates a significant gap in performance, while [36, 40, 83, 85] focus on the differences at the application layer. Some of these studies propose empirical flow charts to guide users in the quest of choosing solutions based on blockchains or databases [36, 83, 85]. In contrast, our work presents a deeper and more comprehensive comparison, by looking at the fundamental designs of both systems.

Surveys and benchmarking. There are some works that conduct separate surveys and benchmarking of distributed databases [20, 24, 38, 89] and blockchains [28, 78, 92]. BLOCKBENCH [45] is the first to compare them side-by-side and demonstrate that the performance of blockchain is still far behind that of distributed databases. Our work is more comprehensive than [45], as we consider systems that are related to blockchains in their designs. We conduct more fine-grained measurements and investigate a variety of factors.

Bridging blockchains and databases. There is a trend of integrating database designs into blockchains and vice versa. In particular, some works apply well-established concurrency control techniques to improve blockchain's performance [42, 74] or to reason about smart contracts' behavior [73]. [69, 81] use database techniques to enhance the blockchain storage layer and expose richer information to smart contracts. [21, 46, 65] propose hybrid designs that support the relational data model and strong security. Our work provides a novel framework for exploring the design space of hybrid, database-blockchain systems.

7 CONCLUSIONS

In this paper, we presented a comprehensive dichotomy between blockchains and distributed databases, viewing them as two different types of transactional distributed systems. We proposed a taxonomy consisting of four design dimensions: replication, concurrency, storage, and sharding. Using this taxonomy, we discussed how both system types make different design choices driven by their high-level goals, i.e., security for blockchains, and performance for databases. We then performed a quantitative performance comparison covering a large area of the design space. Our results illustrate the effects of different design choices to the overall performance. Finally, our work provides the first framework to explore future database-blockchain design fusions.

ACKNOWLEDGMENTS

The research is supported by the National Research Foundation, Singapore under its Emerging Areas Research Projects (EARP) Funding Initiative. Meihui Zhang's work is supported by National Natural Science Foundation of China (62072033) and CCF-AFSG Research Fund (RF20200015). Tien Tuan Anh Dinh is supported by Singapore University of Technology and Design's startup grant SRG-ISTD-2019-144.

REFERENCES

[1] Amazon dynamodb. https://aws.amazon.com/dynamodb.
[2] Amazon quantum ledger database (qldb). https://aws.amazon.com/qldb/.
[3] Caliper. https://github.com/hyperledger/caliper.
[4] Cassandra. https://cassandra.apache.org/.
[5] Cockroachdb. https://github.com/cockroachdb/cockroach.
[6] Eosio. https://eos.io/.
[7] Etcd: Distributed reliable key-value store for the most critical data of a distributed system. https://github.com/etcd-io/etcd.
[8] Ethereum. https://github.com/ethereum/go-ethereum.
[9] Ethereum 2.0 (eth2). https://ethereum.org/en/eth2/.
[10] Fabric. https://github.com/hyperledger/fabric.
[11] Fabric v0.6. https://hyperledger-fabric.readthedocs.io/en/v0.6/home.html.
[12] Fisco-bcos. https://http://fisco-bcos.org//.
[13] Kafka. https://kafka.apache.org/.
[14] Merkle patricia tree. https://github.com/ethereum/wiki/wiki/Patricia-Tree.
[15] Mongodb. https://www.mongodb.com.
[16] Quorum. https://github.com/jpmorganchase/quorum.
[17] Spanner. https://cloud.google.com/spanner.
[18] Tidb. https://github.com/pingcap/tidb.
[19] Ycsb. https://github.com/brianfrankcooper/YCSB.
[20] V. Abramova and J. Bernardino. Nosql Databases: Mongodb Vs Cassandra. In *Proc. of International C* Conference on Computer Science and Software Engineering*, pages 14–22. ACM, 2013.
[21] L. Allen, P. Antonopoulos, A. Arasu, J. Gehrke, J. Hammer, J. Hunter, R. Kaushik, D. Kossmann, J. Lee, R. Ramamurthy, S. Setty, J. Szymaszek, A. van Renen, and R. Venkatesan. Veritas: Shared Verifiable Databases And Tables In The Cloud. In *CIDR*, 2019.
[22] J. C. Anderson, J. Lehnardt, and N. Slater. *CouchDB: the Definitive Guide: Time to Relax*. "O'Reilly Media, Inc.", 2010.
[23] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al. Hyperledger Fabric: A Distributed Operating System For Permissioned Blockchains. In *Proc. of 13th EuroSys Conference*, pages 1–30. ACM, 2018.
[24] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. Linkbench: A Database Benchmark Based On The Facebook Social Graph. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 1185–1196. ACM, 2013.
[25] P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Highly Available Transactions: Virtues And Limitations. *PVLDB*, 7(3):181–192, 2013.
[26] M. Balakrishnan, D. Malkhi, V. Prabhakaran, T. Wobber, M. Wei, and J. D. Davis. Corfu: A Shared Log Design For Flash Clusters. In *Proc. of 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.
[27] M. Balakrishnan, D. Malkhi, T. Wobber, M. Wu, V. Prabhakaran, M. Wei, J. D. Davis, S. Rao, T. Zou, and A. Zuck. Tango: Distributed Data Structures Over A Shared Log. In *Proc. of 24th ACM Symposium on Operating Systems Principles*, pages 325–340, 2013.
[28] A. Baliga, I. Subhod, P. Kamat, and S. Chatterjee. Performance Evaluation Of The Quorum Blockchain Platform. *arXiv preprint arXiv:1809.03421*, 2018.
[29] P. A. Bernstein, C. W. Reid, and S. Das. Hyder – A Transactional Record Manager For Shared Flash. In *CIDR*, 2011.
[30] A. Bessani, J. Sousa, and E. E. P. Alchieri. State Machine Replication For The Masses With Bft-Smart. In *Proc. of 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362, 2014.
[31] E. Buchman. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. PhD thesis, The University of Guelph, 2016.
[32] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The Primary-Backup Approach. *Distributed systems*, 2:199–216, 1993.
[33] V. Buterin and V. Griffith. Casper The Friendly Finality Gadget. *arXiv preprint arXiv:1710.09437*, 2017.
[34] J. L. Carlson. *Redis in action*. Manning Shelter Island, 2013.
[35] M. Castro, B. Liskov, et al. Practical Byzantine Fault Tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
[36] M. J. M. Chowdhury, A. Colman, M. A. Kabir, J. Han, and P. Sarda. Blockchain Versus Database: A Critical Analysis. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1348–1353. IEEE, 2018.
[37] Computer, B. E. M. Association, et al. American National Standard For Information Systems-Database Language Sql. *NY, American National Standards Institute*, pages 27–28, 1986.
[38] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems With Ycsb. In *Proc. of 1st ACM Symposium on Cloud Computing*, pages 143–154. ACM, 2010.
[39] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. Spanner: Google'S Globally Distributed Database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):1–22, 2013.
[40] M. Crosby, P. Pattanayak, S. Verma, V. Kalyanaraman, et al. Blockchain Technology: Beyond Bitcoin. *Applied Innovation*, 2(6-10):71, 2016.
[41] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi. Towards Scaling Blockchain Systems Via Sharding. *arXiv preprint arXiv:1804.00399*, 2018.
[42] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen. Adding Concurrency To Smart Contracts. In *Proc. of ACM Symposium on Principles of Distributed Computing*, pages 303–312. ACM, 2017.
[43] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux. Oltp-Bench: An Extensible Testbed For Benchmarking Relational Databases. *PVLDB*, 7(4):277–288, 2013.
[44] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang. Untangling Blockchain: A Data Processing View Of Blockchain Systems. *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1366–1385, 2018.
[45] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan. Blockbench: A Framework For Analyzing Private Blockchains. In *Proc. of ACM International Conference on Management of Data*, pages 1085–1100. ACM, 2017.
[46] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, and R. Ramamurthy. Blockchaindb: A Shared Database On Blockchains. *PVLDB*, 12(11):1597–1609, 2019.
[47] etcd. Understanding Performance. https://bit.ly/2kzI8R2, 2019.
[48] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Technical report, Massachusetts Inst of Tech Cambridge lab for Computer Science, 1982.
[49] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On The Security And Performance Of Proof Of Work Blockchains. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.
[50] S. Gilbert and N. Lynch. Perspectives On The Cap Theorem. *Computer*, 45(2):30–36, 2012.
[51] M. Herlihy, B. Liskov, and L. Shrira. Cross-Chain Deals And Adversarial Commerce. *arXiv preprint arXiv:1905.09743*, 2019.
[52] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. Jones, S. Madden, M. Stonebraker, Y. Zhang, et al. H-Store: A High-Performance, Distributed Main Memory Transaction Processing System. *Proc. of VLDB Endowment*, 1(2):1496–1499, 2008.
[53] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, and P. Dubey. Fast: Fast Architecture Sensitive Tree Search On Modern Cpus And Gpus. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 339–350. ACM, 2010.
[54] 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)*, pages 583–598. IEEE, 2018.
[55] A. Lakshman and P. Malik. Cassandra: A Decentralized Structured Storage System. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.
[56] L. Lamport. Generalized Consensus And Paxos. *Technical Report MSR-TR-2005-33, Microsoft Research*, 2005.
[57] L. Lamport et al. Paxos Made Simple. *ACM Sigact News*, 32(4):18–25, 2001.
[58] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
[59] K. Li. The Blockchain Scalability Problem & the Race for Visa-Like Transaction Speed. http://archive.today/XnKJC, 2019.
[60] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A Secure Sharding Protocol For Open Blockchains. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.
[61] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto. Bigchaindb: A Scalable Blockchain Database. *white paper, BigChainDB*, 2016.
[62] V. Morabito. Business Innovation Through Blockchain. *Cham: Springer International Publishing*, 2017.
[63] W. Mougayar. *The business blockchain: promise, practice, and application of the next Internet technology*. John Wiley & Sons, 2016.
[64] S. Nakamoto et al. Bitcoin: A Peer-To-Peer Electronic Cash System. *Working Paper*, 2008.
[65] S. Nathan, C. Govindarajan, A. Saraf, M. Sethi, and P. Jayachandran. Blockchain Meets Database: Design And Implementation Of A Blockchain Relational Database. *PVLDB*, 12(11):1539–1552, 2019.
[66] D. Ongaro and J. Ousterhout. In Search Of An Understandable Consensus Algorithm. In *Proc. of USENIX Annual Technical Conference*, pages 305–320, 2014.
[67] M. T. Özsu and P. Valduriez. *Principles of distributed database systems*. Springer Science & Business Media, 2011.
[68] Y. Peng, M. Du, F. Li, R. Cheng, and D. Song. Falcondb: Blockchain-Based Collaborative Database. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 637–652, 2020.
[69] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang. Fine-Grained, Secure And Efficient Data Provenance On Blockchain Systems. In *VLDB*, 2019.
[70] P. Ruan, T. T. A. Dinh, D. Loghin, M. Zhang, G. Chen, Q. Lin, and B. C. Ooi. Blockchains vs. distributed databases: Dichotomy and fusion, 2021.
[71] P. Ruan, D. Loghin, Q.-T. Ta, M. Zhang, G. Chen, and B. C. Ooi. A Transactional Perspective On Execute-Order-Validate Blockchains. In *Proc. of ACM SIGMOD*

International Conference on Management of Data, pages 543–557, 2020.

[72] F. M. Schuhknecht, A. Sharma, J. Dittrich, and D. Agrawal. Chainifydb: How To Blockchainify Any Data Management System. *arXiv preprint arXiv:1912.04820*, 2019.

[73] I. Sergey and A. Hobor. A Concurrent Perspective On Smart Contracts. In *International Conference on Financial Cryptography and Data Security*, pages 478–493. Springer, 2017.

[74] A. Sharma, F. M. Schuhknecht, D. Agrawal, and J. Dittrich. Blurring The Lines Between Blockchains And Database Systems: The Case Of Hyperledger Fabric. In *Proc. of International Conference on Management of Data*, pages 105–122, 2019.

[75] M. Stonebraker and L. A. Rowe. The Design Of Postgres. *SIGMOD Rec.*, 15(2):340–355, 1986.

[76] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss, et al. Cockroachdb: The Resilient Geo-Distributed Sql Database. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 1493–1509, 2020.

[77] A. Tai, M. Wei, M. J. Freedman, I. Abraham, and D. Malkhi. Replex: A Scalable, Highly Available Multi-Index Data Store. In *Usenix ATC*, 2016.

[78] P. Thakkar, S. Nathan, and B. Viswanathan. Performance Benchmarking And Optimizing Hyperledger Fabric Blockchain Platform. In *Proc. of 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 264–276. IEEE, 2018.

[79] H. T. Vo, S. Wang, D. Agrawal, G. Chen, and B. C. Ooi. LogBase: A Scalable Log-Structured Database System in the Cloud. *Proc. of VLDB Endow.*, 5(10):1004–1015, June 2012.

[80] A. Vukotic, N. Watt, T. Abedrabbo, D. Fox, and J. Partner. *Neo4j in action.* Manning Publications Co., 2014.

[81] S. Wang, T. T. A. Dinh, Q. Lin, Z. Xie, M. Zhang, Q. Cai, G. Chen, B. C. Ooi, and P. Ruan. Forkbase: An Efficient Storage Engine For Blockchain And Forkable Applications. *PVLDB*, 11(10):1137–1150, 2018.

[82] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper , 2014.

[83] K. Wüst and A. Gervais. Do You Need A Blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.

[84] Z. Xie, Q. Cai, H. Jagadish, B. C. Ooi, and W. F. Wong. Parallelizing Skip Lists For In-Memory Multi-Core Database Systems. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 119–122. IEEE, 2017.

[85] D. Yaga, P. Mell, N. Roby, and K. Scarfone. Blockchain technology overview. Technical report, National Institute of Standards and Technology, 2018.

[86] X. Yang, Y. Zhang, S. Wang, B. Yu, F. Li, Y. Li, and W. Yan. LedgerDB: a Centralized Ledger Database for Universal Audit and Verification. *Proc. of VLDB Endow.*, 13(12):3138–3151, 2020.

[87] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: Bft Consensus With Linearity And Responsiveness. In *Proc. of ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[88] C. Yue, Z. Xie, M. Zhang, G. Chen, B. C. Ooi, S. Wang, and X. Xiao. Analysis Of Indexing Structures For Immutable Data. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 925–935, 2020.

[89] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang. In-memory Big Data Management and Processing: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1920–1948, 2015.

[90] M. Zhang, Z. Xie, C. Yue, and Z. Zhong. Spitz: A Verifiable Database System. *Proc. of VLDB Endow.*, 13(12):3449–3460, Aug. 2020.

[91] Y. Zhang, J. Katz, and C. Papamanthou. Integridb: Verifiable Sql For Outsourced Databases. In *Proc. of 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1480–1491, 2015.

[92] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.