Blockchains and Distributed Databases: a Twin Study

Pingcheng Ruan[†], Gang Chen[◇], Tien Tuan Anh Dinh[◦],
Qian Lin[†], Dumitrel Loghin[†], Beng Chin Ooi[†], Meihui Zhang[‡]

[†]*National University of Singapore*
{ruanpc, linqian, dumitrel, ooibc}@comp.nus.edu.sg

[‡]*Beijing Institute of Technology*
meihui_zhang@bit.edu.cn

[◦]*Singapore University of Technology and Design*
dinhtta@sutd.edu.sg

[◇]*Zhejiang University*
cg@zju.edu.cn

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. A blockchain is also a distributed system, and as such it shares some similarities with distributed database systems. Existing works that compare blockchains and distributed database systems focus mainly 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 paper is to fill this important gap.

In this paper, we perform a twin study of blockchains and distributed database systems as two types of transactional systems. We propose a taxonomy that helps illustrate their similarities and differences. In particular, we compare the systems along four dimensions: replication, concurrency, storage, and sharding. We discuss how the design choices have been driven by the system's goals: blockchain's goal is security, whereas the distributed database's goal is performance. We then conduct an extensive and in-depth performance study on two blockchains, namely Quorum and Hyperledger Fabric, and three distributed databases, namely CockroachDB, TiDB and etcd. We demonstrate how the different design choices in the four dimensions lead to different performance. In addition, we show that for most workloads, blockchain's performance is still lagging far behind that of a distributed database. However, the gap is not as significant as previously reported, and under high contention or constrained workloads, blockchains and databases are even comparable. Our work provides a framework for exploring the design space of hybrid database-blockchain systems.

1. INTRODUCTION

The very first blockchain system, that is Bitcoin [47], 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 [42]. 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 [42] peer-to-peer (P2P) environment, without relying on a common trusted third party. But it can execute only simple transactions that move some states from one address (or user) to another. However, recent blockchains such as Ethereum [58] and Hyperledger Fabric [13] support general-purpose transactions. The key enabler is the *smart contract* which is a user-defined computation executed by all nodes in the blockchain. With smart contracts, blockchains can execute any transactional workload 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. Many companies and government agencies are exploring blockchains to replace, or to complement, their enterprise-grade databases [46, 45, 24].

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 [24, 59, 20, 61]. Their distinction is further compounded by the significant gap in performance [29], for instance Bitcoin processes around 10 transactions per second [43] while etcd — a state-of-the-art distributed NoSQL database — processes over $50,000$ operations per second [31].

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 [29] compares three permissioned blockchains, namely Hyperledger Fabric, Ethereum and Parity, with H-Store under two popular data processing workloads. It shows a large gap in performance, but provides no further analysis of the 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, or may even reverse.

We aim to provide a comprehensive comparison between blockchains and databases. Our approach is to position them within the same design space, that is, the design space of general distributed 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 performance and correctness tradeoffs 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.

Under our taxonomy, existing works such as [29] are incomplete: they only cover extreme points in the design space. In contrast, our work is comprehensive, as it presents and discusses many other points in the space. Our taxonomy helps illustrate the inherent similarities between blockchains and databases. Both systems address the same set of problems in distributed systems, including consistency, failure, and efficient data access. We observe that different design choices are made because of the systems' high-level goals: security for blockchains and performance for databases. Given the taxonomy, we conduct experiments to evaluate the effect of different design choices, thereby giving insights into the factors that contribute to the overall performance gap. Another benefit of our taxonomy is that it provides a framework for exploring new designs that merge blockchains with databases [48, 36, 11].

In summary, we make the following contributions:

- We compare blockchains and distributed databases as two different types of distributed, transactional systems. We propose a new taxonomy that characterizes the systems along four design dimensions: replication, concurrency, storage, and sharding.
- We conduct a comprehensive performance study of five systems under a variety of workloads. The five systems include two representative permissioned blockchains, namely Hyperledger Fabric and Quorum, and three popular database systems, namely CockroachDB, TiDB and etcd.
- We show that, although blockchains perform poorly compared to databases in most cases, the former can still outperform the latter under workloads with high contention and constraints.
- We show that for blockchains, beside the consensus protocols, the block validation and commit phase also have significant impact on the overall performance.

In the next section, we provide an overview of blockchains and distributed databases. Section 3 presents our qualitative comparison with the focus on the above four dimensions. Section 4 describes the experimental setup. Section 5 discusses the performance results. Section 6 reviews related works before Section 7 concludes.

2. BACKGROUND

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

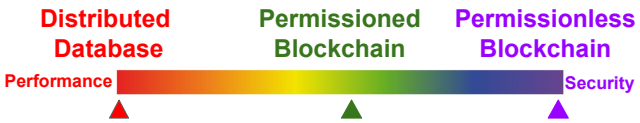


Figure 1: Blockchains versus distributed databases on the security-performance spectrum.

2.1 Blockchain

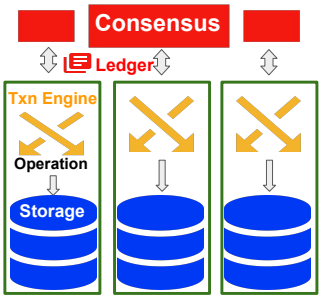
A blockchain is originally defined as a data structure consisting of a linked list of blocks, where the links are cryptographic hash pointers, and the blocks contain cryptocurrency transactions [47]. By this definition, the blockchain is a tamper-evident ledger for recording transactions. In this paper, we use a more recent and popular definition of blockchain, which is a distributed system consisting of multiple nodes some of which are Byzantine. The chain of blocks, or the ledger, is kept consistent at all nodes via a Byzantine fault-tolerant (BFT) consensus protocol.

In the earlier designs, a blockchain transaction is restricted to cryptocurrency and the states are modeled as Unspent Transaction Outputs (UTXO). For example, Bitcoin [47] and other similar altcoins use the UTXO model. Starting with Ethereum [4], 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 [13]. In these systems, a transaction on the ledger takes the form of a contract invocation, which modifies the ledger in a pre-agreed way 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, thus, cannot be compared to distributed databases.

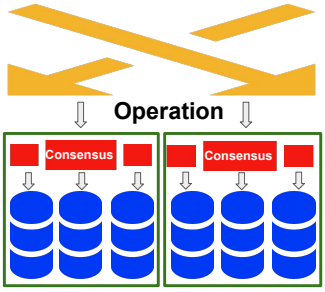
Permissionless vs Permissioned Blockchains can be broadly divided into two categories: permissionless (or public), and permissioned (or private). In the former, for example in Bitcoin and Ethereum, any node and user can join the system in a pseudonymous manner. In the latter, for example in Fabric and 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 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 [28, 29].

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,



(a) Blockchain architecture



(b) Distributed database architecture

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

remained mainstream throughput 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 [50]. Basically, there are two distinctive movements, namely NoSQL and NewSQL, under this new 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. They adopt the BASE principle for their transaction semantics. In the sense of CAP [33] theorem, these NoSQL systems compromise consistency for the sake of availability. A variety of their supported data models may include key-value store (e.g, Redis [18], etcd [3]), document store (e.g, CouchDB [12]), graph store (e.g, Neo4J [56]), column-oriented (e.g, Cassandra [39]) and so on. The most lenient consistency model is eventual consistency which makes no guarantees about the order of read and write operations. In the middle of the 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 stores offer them as the configurable options, where users can tradeoff 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*, emerge which aim to restore the relational model and ACID semantics without sacrificing much scalability. NewSQL has drawn attention since Google introduced Spanner [23], the first NewSQL system. It was followed by a few database vendors, such as CockroachDB [2] and TiDB [8]. In this paper, we consider both NoSQL and NewSQL systems.

3. TAXONOMY

In this section, we present a distributed system taxonomy that illustrates the important design choices made by blockchains and distributed databases. We highlight how the differences are driven by the fact that these systems aim to achieve different goals: security for blockchains, and performance for databases.

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, which involves running a consensus protocol among the replicas. Through this consensus protocol, the replicas reach agreement on the latest data. In this section, we characterize blockchains and distributed databases by the replication model, failure model, and consensus protocol.

3.1.1 Replication model

Blockchains replicate transactions, whereas databases replicate storage operations (reads and writes). As depicted in Figure 2, a blockchain replicates the ordered log of transactions by running a consensus protocol. Each node then executes the transactions against its local states (the ledger). On the other hand, a distributed database replicates 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. In other words, the transaction manager which coordinates the execution of a transaction must be trusted — a common assumption in databases. The blockchain, which assumes malicious nodes, does not have that trusted entity, therefore it must replicate the entire transaction.

3.1.2 Failure model

Fault tolerance is an important goal of any distributed system. Distributed databases assume crash failure, in which nodes only fail by crashing. In this model, referred to as crash fault tolerance (CFT), the system needs to tolerate hardware and software crash, as well as network partition. CFT is suitable for databases, because they are considered internal systems, protected by many layers of security. Blockchains, on the other hand, assume hostile environments, in which a node can behave arbitrarily. In this Byzantine fault tolerance (BFT) model, the system needs to tolerate any software and hardware failures, as well as any malicious user. This model is appropriate when the system

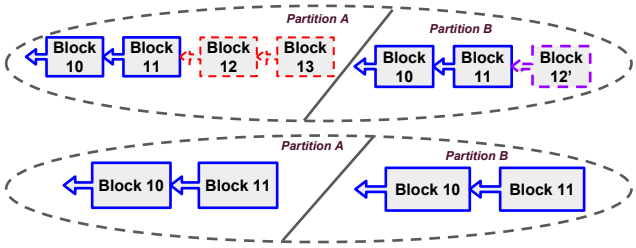


Figure 3: Permissonless vs. permissioned blockchains during network partition after Block 11. The former keeps appending blocks in each partition, while the later becomes unavailable but in consistency.

needs to operate correctly under security attacks. For example, the nodes may be compromised by an attacker and therefore deviate arbitrarily from the protocol.

CFT protocols have lower security guarantees than BFT, but they can achieve higher performance for a given number of failures. In particular, to tolerate f failures, CFT requires $2f+1$ replicas, whereas BFT requires $3f+1$. We observe that some permissioned blockchains support both failure models. For example, Quorum provides both Raft [49], a CFT protocol, and Istanbul consensus, a BFT protocol, implementation. These systems allow application developers to make different tradeoffs between security and performance.

3.1.3 Consensus protocol

Permissionless blockchains adopt proof-of-work (PoW) or other variants such as proof-of-stake (PoS) and proof-of-elapsed time (PoE). In contrast, both permissioned blockchains and distributed databases adopt classic CFT or BFT protocols, such as Paxos [41, 40], Raft [49], and PBFT [19]. Detailed comparisons between PoW and other consensus protocols can be found in [28, 17]. Here, we discuss the reason and implication of adopting PoW for permissionless blockchains. Our discussion also extends to other PoW variants.

In the permissionless setting, a node or user can have many identities. The lack of strong identity renders voting-based protocols — most classic CFT and BFT protocols in the literature are voting-based — infeasible. PoW overcomes the identity problem by relying on incentives. In PoW, a node's probability of solving a computational puzzle, thereby solving consensus and gaining rewards, is proportional to its physical resources which are difficult to forge. Thus, the node has no advantage in having multiple identities.

In an asynchronous network, FLP theorem [32] rules out any deterministic consensus protocol that can achieve both safety and liveness. Permissionless blockchains assume that nodes communicate over the Internet which is subject to unpredictable performance and frequent partitioning. They opt for liveness over safety, which means the system continues to work under network partitions, but these partitions may be in disagreement in the form of chain forks, as shown in Figure 3. For safety, permissionless blockchains rely on the network synchrony. In particular, when the partition time is longer than the block interval, nodes in each network partition independently append to the ledger. But when the network partition heals, transactions in shorter forks are discarded. This is the reason why permissionless blockchains require users to wait for transactions committed several blocks behind, before considering them in effect. We note that choosing liveness over safety is inevitable in the Internet environment because otherwise, the system will be unavailable for most of the time.

PoW protocols are less sensitive to network conditions than classic CFT and BFT protocols because the latter are communication bound. In particular, most CFT and BFT incur $O(N^2)$ message complexity, where N is the network size. The delay of a single message may trigger the expensive recovery mode, which may worsen the delay and lead to performance collapse [29]. PoW protocols, on the other hand, are computation bound, in which the difficulty of the computational puzzle is adjusted gradually to approximate a fixed block interval. By not relying heavily on network communication, PoW blockchains achieve higher availability than permissioned blockchains and distributed databases.

3.2 Concurrency

Most blockchains execute transactions sequentially, while distributed databases employ sophisticated concurrency control mechanisms to extract as much concurrency from the execution as possible. The reason for blockchains' choice of serial execution is two-fold. First, serial execution may not affect the overall performance because execution is often not the bottleneck. For example, in Bitcoin, the consensus protocol may take several minutes, while transaction execution takes only a few seconds. Second, enforcing 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, which makes the consistency model simple.

Unlike blockchains, concurrency remains a major research topic in databases. To exploit more concurrency inherent in the workloads, complex mechanisms are being proposed to ensure some forms of correctness. In particular, there exists a wide range of isolation levels [21, 15] which make different tradeoffs between correctness and performance. Most production-grade databases today offer more than one isolation level.

We note that some blockchains start to employ simple concurrency techniques used in databases. In Hyperledger Fabric, for example, transactions are simulated in parallel against the ledger states before sent for ordering. The system uses a simple optimistic concurrency control to achieve serializability which aborts transactions whose simulated states are stale. More established techniques to reduce abortion have also been proposed [54].

3.3 Storage

In this section, we describe how blockchains and distributed databases build different models and data structures on the underlying storage.

3.3.1 Storage model

The storage in blockchains exposes an append-only ledger abstraction. The ledger records historical transactions and the changes made to the global states. Some systems allow applications to access only the latest states, for example, Hyperledger Fabric v0.6. Novel storage systems have been proposed to enable access to any historical states during

smart contract execution [51]. The storage in distributed databases, on the other hand, exposes direct access to data 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.

3.3.2 Index

One of the most important properties of blockchains is data integrity, which means any tampering with the data on the ledger must be detected. As a result, blockchains use Merkle tree index structures to provide both efficient data access and integrity protection. For example, Ethereum uses a prefix trie, named Merkle Patricia Trie (MPT) [5]. In the MPT, the states are stored at the leaves, and the ones with a common key prefix are organized into the same branch. Each node is associated with the cryptographic hash of its content, 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.

Indexes play such an instrumental role in databases that any small optimization on the index can translate to 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 [37] and PSL [60] which are designed for better cache utilization and multi-core parallelism.

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. In this section, we discuss two key challenges in any sharded systems, 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 put enough resources into one shard to break the security assumption. State-of-the-art sharded blockchains have different approaches. For example, Elastico uses PoW for shard formation [44], Omniledger [38] employs a complex cryptographic protocol, while AHL [25] uses trusted hardware. The last two systems perform regular shard reconfiguration, by re-executing the shard formation protocol, in order to guard against adaptive adversaries.

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, random partition and range partition, so that users can select the most suitable for their workloads. Some systems, for instance, Cassandra[39], 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 can touch data at multiple shards. Atomicity requires the cross-shard transaction to either commit at all shards or not at all. In databases, this problem is addressed by the two-phase commit (2PC) protocol. This protocol requires a dedicated transaction coordinator. This coordinator is trusted, but it may fail and leave the transaction blocked forever. Three-phase commit protocol (3PC) eliminates the blocking problem, but it relies on strong assumptions of the network.

Sharded blockchains face additional challenges in ensuring atomicity because under the Byzantine failure model the coordinator cannot be trusted. To overcome this, both [25, 36] propose to implement a 2PC state machine in the 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 service high availability, therefore mitigating the blocking problem. But the Byzantine setup in blockchains imposes considerable overhead to the 2PC process.

3.5 Discussion

We have so far discussed the similarities and differences in the *design* of blockchains and distributed databases. Here, we describe an important, implementation-specific difference between the two.

Both systems require users to authenticate their requests. Databases support user sessions, meaning that the user only needs to authenticate once per session which usually consists of multiple requests. Blockchains, on the other hand, do not support user sessions, and the user has to sign every request. We show in Section 5.2.2 how this affects performance of read-only transactions. We note that adding support for user sessions in blockchains is possible, but would incur significant storage and computation overhead because session information must be maintained by the smart contracts.

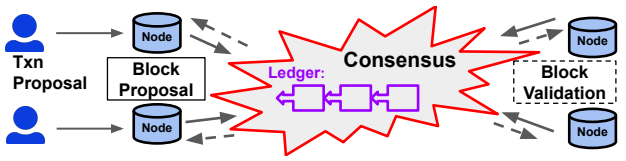
4. EXPERIMENTAL SETUP

The previous section presented a qualitative comparison of blockchains and distributed databases. We now demonstrate how the design choices affect system performance by conducting a comprehensive performance evaluation of five different systems. In this section, we describe the setup of our experiments.

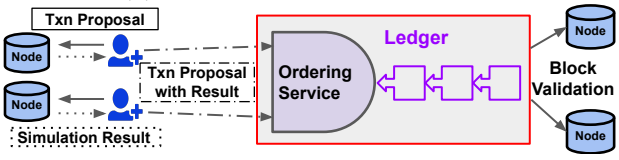
4.1 Systems

We select five representative systems: two permissioned blockchains, namely Quorum and Hyperledger Fabric, and three distributed databases, namely CockroachDB, TiDB and Etcd. Together, they cover a large area of the design space laid out in Section 3.

Quorum [7] (version 1.8.12) is a permissioned blockchain based on Ethereum. It targets the financial sector which



(a) Quorum transaction lifecycle



(b) Fabric transaction lifecycle

Figure 4: Transaction execution in Quorum versus Hyperledger Fabric. In Quorum, a node assembles pre-executed transactions into blocks and send them through 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.

requires greater efficiency and data privacy than what is provided by Ethereum. Quorum replaces Ethereum's Proof of Work (PoW) with a CFT protocol, namely Raft, and a BFT protocol called Istanbul BFT. However, its execution model is similar to Ethereum's. First, a leader executes transactions speculatively and assembles a new block. Next, the leader starts the consensus protocol so that the other nodes agree on the block. Finally, all the nodes execute the block.

Hyperledger Fabric [13] (version 1.3) is a popular permissioned blockchain started by IBM and the Linux Foundation. It has a modular design and features a novel execute-order-validate execution model. The system has two types of nodes: *peers* which execute smart contracts and validate blocks, and *orderers* which order transactions. A transaction is executed in three phases. In the Proposal phase, a client requests the peers to execute the transaction speculatively. The client collects the results and signatures from the peers and sends them to the orderers, triggering the next phase. In the Order phase, orderers order the transactions and batch them into blocks. In the Commit phase, each peer pulls blocks from the orderers and independently validates each block before persisting the results. The block validation process involves verifying signatures and checking for conflicts in the read/write sets. Read-only transactions use only the Proposal phase. Figure 4 compares Fabric's execution with Quorum's.

CockroachDB [2] (version 2.1.1) is a distributed NewSQL relational database that supports both ACID transactions and horizontal scalability. It relies on the Raft consensus protocol to synchronize replicas. As an open-source solution inspired by Spanner [23], CockroachDB utilizes a synchronized clock to coordinate two-phase commit (2PC) for transaction atomicity and serializability.

TiDB [8] (version 2.1.0) is another distributed NewSQL database system that supports both SQL semantics and horizontal scalability. While CockroachDB takes a monolithic approach, TiDB is featured for its modular design. TiDB consists of three independent components, namely Placement Driver for coordinating cluster management, TiKV servicing as the replicated key-value storage, and TiDB-server for parsing and scheduling SQL queries in a stateless manner. TiDB also employs optimistic concurrency control and 2PC for transaction management, as well as Raft for replica consistency. However, its data isolation level only support snapshot isolation, more lenient than serializable in CockroachDB.

Table 1: Experiment variables.

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

Etcd [3] (version 3.3.13) is a NoSQL database used in many large-scale systems [6]. Etcd provides a simpler key-value data model with relaxed transactional restriction 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, data are fully replicated on each node.

4.2 Setup

For a fair comparison, we run the five systems in full replication mode in which each node has a complete copy of the states. In particular, for Fabric we use the endorsement policy under which a transaction is executed and endorsed by all peers. For CockroachDB and TiDB, we set the replication factor to be the same as the number of nodes.

We use CFT protocols in all five systems. In particular, we configure Quorum to use Raft, and Fabric to use Kafka which is a CFT protocol. We also disable all optional security features, such as data encryption and Transport Layer Security (TLS) communication.

We set the number of transactions per block to 100 in Fabric and the block interval to the minimum in Quorum. We scale all TiDB's modules with the number of nodes so that each node runs all three modules in different processes. For Fabric, we run the orderers on two nodes and Kafka on three nodes.

Unless otherwise specified, we use the YCSB and Smallbank workloads in our experiments. The experiment parameters for YCSB are summarized in Table 1 with the default values underlined. 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 3.5GHz CPU, 32GB RAM and 2TB hard disk. All the experiments are repeated three times and we report the average results.

4.3 Benchmark Driver

We note that existing benchmark drivers for databases are synchronous (or closed-loop), meaning that a next request is sent only when the current request is completed. In contrast, blockchain benchmark drivers, such as Fabric's Caliper and BLOCKBENCH driver [29], are asynchronous (or open-loop) in which a new request is sent as soon as the current request is acknowledged by a node in the blockchain. Figure 5 illustrates the difference between the two types of drivers. In a synchronous driver, a separate status thread

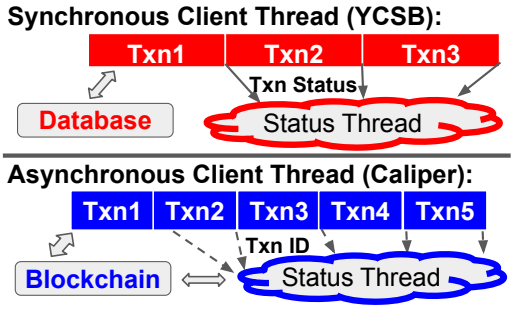


Figure 5: Differences between synchronous and asynchronous drivers. Asynchronous drivers can issue more transactions, as they do not have to wait for their completion, as opposed to the synchronous ones.

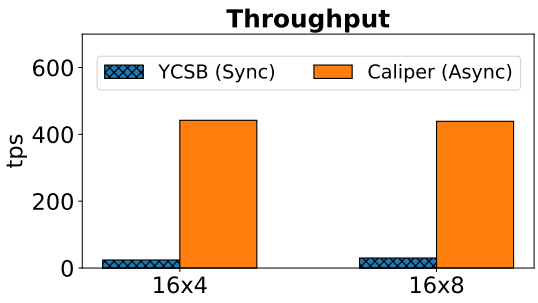


Figure 6: Fabric peak throughput measured from YCSB (synchronous) and Caliper (asynchronous) drivers. 16x4 (, 16x8) implies a total of 16 client nodes, each of which establishes 4 (, 8) connections.

is responsible for computing statistics. In an asynchronous driver, the status thread needs to keep track of outstanding requests and to periodically poll the blockchain for newly completed requests.

Asynchronous drivers are suitable for blockchains because of the long request latency which requires a large number of outstanding requests to saturate the system. In other words, benchmarking a blockchain with synchronous drivers would require many nodes to run the driver. To illustrate this, we implement a synchronous YCSB driver for Fabric using its Java SDK. Figure 6 compares the peak throughput using this driver with that obtained using Caliper. We observe that the synchronous driver severely underestimates Fabric's performance, even with a large number of connections.

For the database experiments we use the open-source driver for YCSB workload [9] and the OLTPBench [27] driver for Smallbank workload. Both Fabric and Quorum are benchmarked using Caliper [1]. We emphasize 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.

5. PERFORMANCE ANALYSIS

In this section, we present our comprehensive, quantitative comparison between blockchains and distributed databases. We start by comparing the systems' peak performance under the default configurations. We then analyze how the design choices discussed in Section 3 affect the overall performance.

5.1 Peak Performance

5.1.1 YCSB

We first analyze the peak performance of the five systems under the default configurations (Table 1). Specifically, we populated each system with 100K records, each of size 1 KB. We then measured the throughput and latency against three YCSB workloads: uniform update-only (100% writes), uniform query-only (100% reads), and uniform mixed (50% reads and 50% writes) workload. We also measure independently the performance of TiKV, the replicated storage of TiDB, and include it in this comparison.

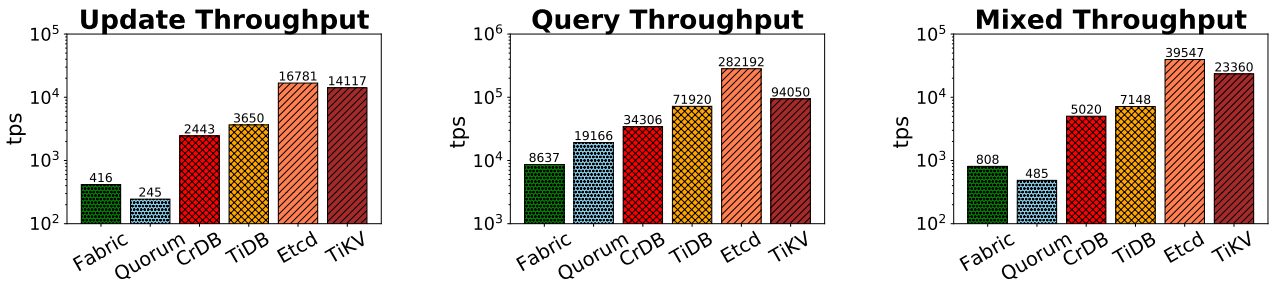
Figure 7 shows the peak throughput of the six systems against the three workloads. It can be seen that relational (NewSQL) databases outperform blockchains, and replicated storages (NoSQL) outperform relational databases. Specifically, the two blockchains achieve throughputs below 500 transactions per second (tps), whereas the two relational databases, namely CockroachDB and TiDB, achieve 2443 tps and 3650 tps, respectively. The two key-value storages achieve over 10,000 tps. We note that the blockchains have weaker guarantees for read-only transactions compared to those offered by the databases (linearizability). Despite this fact, they still have the worst performance. For example, Quorum's throughput is 10× lower than etcd's. Under the mixed workload, the peak throughputs of all systems increase, which indicates that the overhead of read requests is much lower than that of write requests. The two NoSQL systems outperform the two NewSQL systems because the former do not incur the overhead of supporting ACID transactions. The gap between TiDB and TiKV is also caused by the overhead of the TiDB-server that wraps around the key-value storage.

Our results further confirm the conclusion drawn in [29] that the performance of blockchains lags far behind state-of-the-art databases. However, we observe a smaller gap than reported in [29]. The key reason is that the previous work used H-Store, an in-memory, distributed database with no consensus-based replication. H-Store represents an extreme point in our design space that makes it rather dissimilar to blockchains. In contrast, all five systems considered in our work incur some overheads from the consensus protocols.

Figure 8a and 8b show the latency when the systems are unsaturated. Similar to throughput, we observe a clear gap between the blockchains and the databases. Responses to read requests in the former take longer (up to 6× in Fabric) than the linearizable reads in the latter. The write latency in Fabric and Quorum is 1000ms and 500ms respectively, while in databases it is below 100ms. One exception is TiDB which takes 189ms. This is due to the modular architecture of TiDB which introduces overheads in the form of communication among different modules.

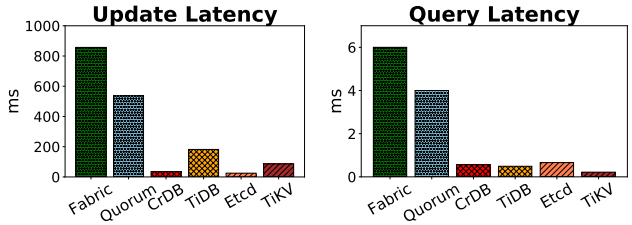
5.1.2 Smallbank

Figure 9 compares the systems' OLTP performance under the Smallbank workload. We do not include etcd because it does not support general transactional workloads. Compared to YCSB, a Smallbank transaction touches more records and imposes more constraints, for example, sufficient funds must be available for the payment transactions to succeed. However, the record size is smaller. Our experiments



(a) (b) (c)

Figure 7: Peak throughput comparison under YCSB. CrDB denotes CockroachDB.



(a) (b)

Figure 8: Latency comparison under YCSB

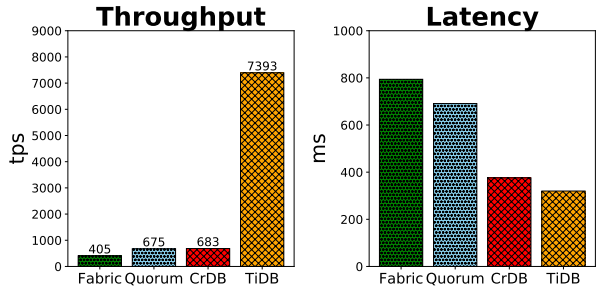


Figure 9: Performance under Smallbank.

show that TiDB has the highest performance, with a peak throughput of 7391 tps, because of the weak isolation level. Among the other systems, all of which provide serializability, we find that CockroachDB is comparable to the two blockchains. More specifically, CockroachDB achieves 683 tps, while the two blockchains achieve 405 and 675 tps, respectively. The results indicate that the transaction constraints in Smallbank limit the level of concurrency extractable under serializability guarantee.

Notably, while the performance of CockroachDB drops when switching from YCSB to Smallbank, we observe that the performance of Quorum improves. Its peak throughput reaches 675 tps with Smallbank, 2× greater compared to YCSB, as shown in Figure 7a. We attribute this improvement to Smallbank's smaller record size. As we shall see in Section 5.3.3, Quorum's performance is extremely vulnerable to larger transactions. Fabric delivers the lowest performance in terms of both throughput and latency.

5.2 Replication

5.2.1 *Effect of number of nodes*

Table 2: Throughput with varying number of nodes under full replication mode.

	3	7	11	15	19
Fabric	389	394	345	344	333
Quorum	237	236	229	217	219
CockroachDB	3060	2411	2482	2464	2282
TiDB	4150	6037	5793	5872	4535
Etcd	16492	16849	9123	7801	6076

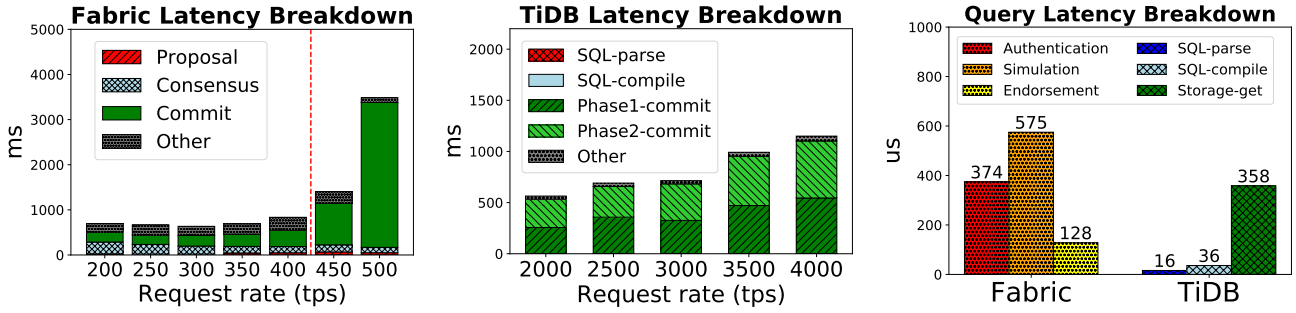
We analyze the impact of the number of nodes on the overall performance by running experiments with an increasing number of nodes. Recall that all five systems are in full replication mode, i.e., the number of replicas is the same as the number of nodes. Table 2 lists the peak throughputs under the uniform update workload.

Fabric's throughput drops slightly from 389 tps on 3 nodes to 333 tps on 19 nodes. By analyzing the timing logs, we find that there is a 15% increase in the block validation latency from 222ms on 3 nodes to 249ms on 19 nodes. This is expected because the endorsement policy requires a transaction to be endorsed by all peer nodes, hence more nodes lead to larger transactions and therefore longer validation. Since a block is validated sequentially, the increase in validation time translates to the decrease in throughput.

Quorum exhibits a relatively constant performance, which is unexpected because the Raft protocol does not scale well with the number of nodes. We find that the current implementation of Quorum underutilizes Raft. Specifically, the system does not pipeline consensus requests; instead, it only starts a new consensus round whenever the previous one finishes. Without pipelining, the throughput is only the inverse of the latency. In our experiments, the consensus latency in Quorum is relatively constant as the number of nodes increases, resulting in the overall throughput unchanged.

All databases show performance degradation when there are more nodes. In particular, CockroachDB peaks at 3 nodes, TiDB at 7 nodes and etcd at 7 nodes. This is as expected because the full replication mode incurs communication overheads that grow exponentially with the increment of nodes.

5.2.2 *Effect of replication model*



(a) (b) (c)

Figure 10: The breakdown of latency

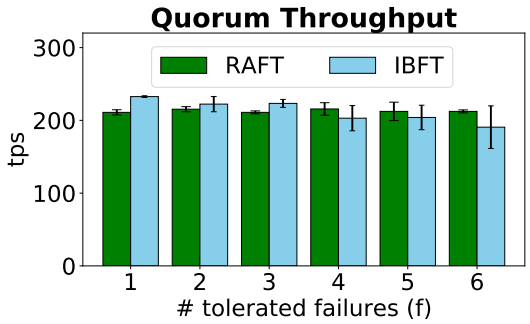


Figure 11: Quorum performance with CFT and BFT

To understand the impact of the replication model, we focus on Fabric and TiDB which support different transaction lifecycles. We instrumented the codebase to record detailed latency breakdown at every phase of the execution. In particular, for Fabric, we measured the latency of the proposal, consensus, and validation (or commit) phase. For TiDB, we recorded the latency of the SQL parsing, SQL compilation, and commit phase. Figure 10a and 10b show the detailed latency for the two systems under the uniform update workload.

Before Fabric is saturated (on the left side of the dashed line in Figure 10a), the consensus and validation phase take roughly the same time, while the proposal phase is almost negligible. Interestingly, the consensus latency decreases when the request rate increases. This is because the orderers wait for enough transactions (100 in our case) before creating a block. This wait time is lower when transactions are submitted more frequently. However, when the request rate exceeds the system capacity, the validation phase becomes the bottleneck, as shown by the significant increase in its latency. On the other hand, the latency of a TiDB transaction is dominated by the commit phase which runs 2PC over Raft consensus. Each of the two phases takes roughly the same time, as shown in Figure 10b. The latency also increases with the request rates.

Figure 10c shows the latency breakdown under the uniform read-only workload. The transaction lifecycle in TiDB is the same as in the update-only workload, except that the commit phase is replaced by a storage read operation. In Fabric, the transaction finishes at the proposal phase, without going through consensus and commit. We instrumented Fabric's codebase to split the proposal phase into three sub-phases: the authentication phase which authenticates the client, the simulation phase which executes the query against the local states, and the endorsement phase which signs the results.

We observe that Fabric spends half of the time (575us) in the simulation phase, another 374us and 128us for authentication and endorsement respectively. The last two phases are expensive because of the cryptographic computations. In contrast, TiDB incurs no cryptographic overheads, and most of its cost goes to the actual retrieval of data.

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. Figure 11 shows the peak throughput where f denotes the number of tolerated failures.

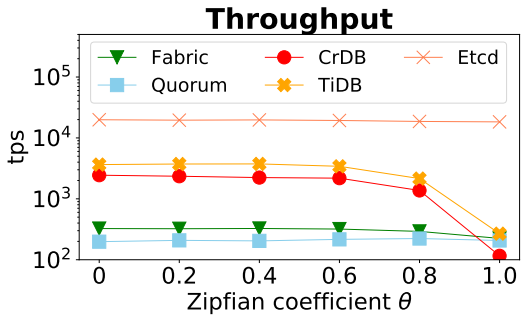
We observe that the throughputs remain relatively constant. This is because Quorum's implementation underutilizes Raft consensus, as explained in Section 5.1. IBFT's throughput does not degrade because it has not reached its capacity at $f = 6$. However, we observe that IBFT's throughput exhibits higher variance in larger networks, as evidenced by the greater error bar. This is due to the number of replicas needed in IBFT, which is $3f + 1$, compared to $2f + 1$ replicas needed in Raft for a given value of f. When f increases, the network becomes more unstable because of network delays, which leads to larger variances in performance.

5.3 Concurrency

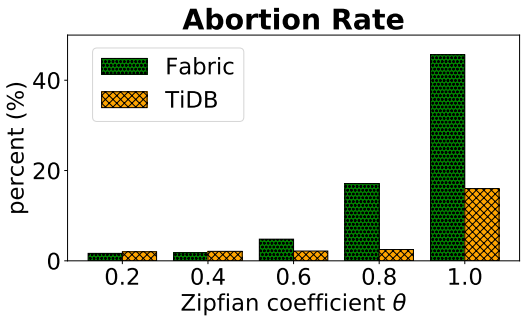
5.3.1 *Effect of skewness*

To demonstrate the effect of concurrency control mechanisms, we use skewed workloads in which each transaction modifies (first read, update and then write back) a single record. The record's key follows a Zipfian distribution with the skewness coefficient θ. Figure 12a and 12b show the throughputs and the corresponding abortion rates with increasing θ. The most important observation here is that blockchains and databases are comparable under a high contention workload.

Since there is only one write operation per transaction, more skewed workloads infer a larger amount of write-write conflicts. Both serializability and snapshot isolation perform poorly under write-write conflicts. This is demonstrated by

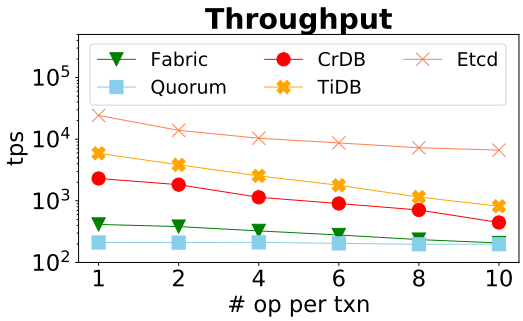


(a)

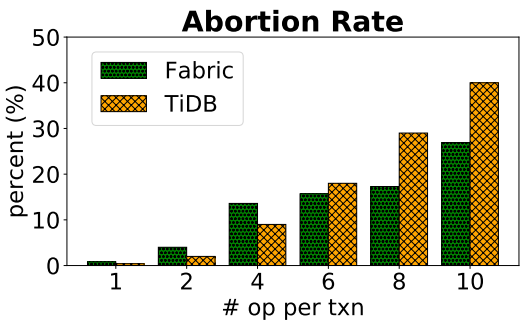


(b)

Figure 12: Throughput and abortion rate with skewed workloads. Each transaction modifies a single record



(a)



(b)

Figure 13: Throughput and abortion rate with uniformly modified records in a single transaction

the significant drop in throughput: CockroachDB from 1372 to 116 tps, and TiDB from 2151 to 286 when θ increases from 0.8 to 1.0. 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 uniform to a skewed workload with $\theta = 1$. It is due to Fabric's optimistic concurrency control, in which a transaction contains versions of the data accessed during the proposal phase, which are then checked at the validation phase. If the versions are not the latest, the transaction aborts. A skewed workload means many transactions are writing the same records, as a consequence increasing the probability of transaction abort. Indeed, Figure 12b shows that nearly 44% of the transactions 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 16% of TiDB's transactions fail but the throughput decreases by 80%. It is because, for each transaction, a coordinator must obtain a latch on the record. As a result, under highly skewed workload, the coordinator spends more time contending for the record than doing real work, therefore the throughput drops sharply.

5.3.2 *Effect of operation count*

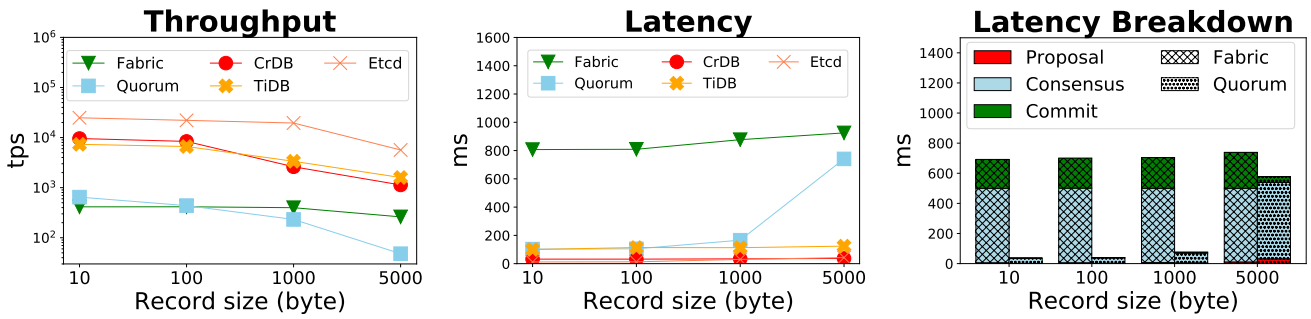
Another way to increase contention is to include more modification operations in each transaction. Figure 13a shows the impact of the number of operations per transaction on the overall performance. To remove any effect of transaction size, for a given number of operations we vary the record size such that the total transaction size is 1KB. For example, if a transaction writes to 10 records, then each record contains 100 bytes.

We observe a similar trend for TiDB and CockroachDB, as shown in Figure 12b. In particular, with 10 operations per transaction, these systems achieve only 14% to 20% of the throughput for single-operation transactions. Two sources of overheads contribute to this drop in performance. First, there are more conflicts when each transaction writes to more records, which leads to a higher abortion rate. Second, TiDB and CockroachDB use sharding, therefore a 10-operation transaction may span multiple shards. As there are more shards, the overhead of coordination (2PC) increases. Etcd and Quorum are unaffected because they do not have conflicts, and they do not support sharding.

Figure 13b shows the abortion rate of TiDB and Fabric as the number of operations per transaction increases. Both systems experience high abortion rates: 26.9% for TiDB and 40% for Fabric. Interestingly, most aborts in Fabric are caused by read operations, as opposed to by write operations in TiDB. Specifically, during the proposal phase in Fabric, a client must collect identical read results from the peers. It is because we mandate each transaction proposal must be simulated and endorsed by all peers. But different results may be returned, as the peers have the disjoint states, which is highly likely since they commit blocks at different rates. When a different result is received, the client immediately aborts. Featured for its optimistic concurrency control, TiDB, on the other hand, uses timestamps to check if the updated data is still at the latest.

5.3.3 *Effect of record size*

Figure 14a and 14b show the system performance with increasing record size under the uniform update workload.



(a) (b) (c)

Figure 14: Performance under uniform update workload with increasing record size.

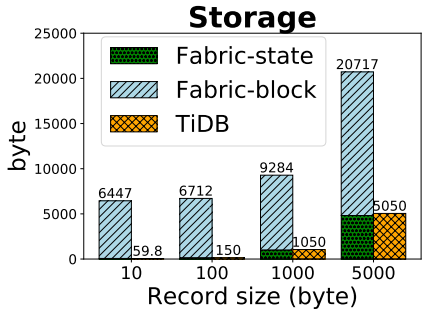


Figure 15: Storage breakdown in Fabric and TiDB

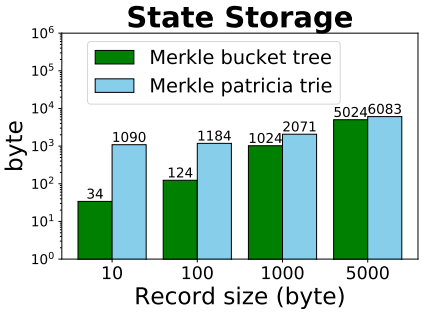


Figure 16: Storage overhead to achieve tamper evidence

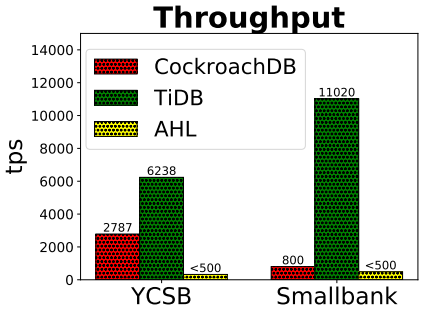


Figure 17: Throughput under identical sharding setup

All the databases exhibit a slight decrease in throughput and an increase in latency. However, the two blockchains behave differently. When the record grows from 10 to 5,000 bytes, Fabric's performance stays almost constant, but Quorum suffers a significant drop of throughput, from 647 tps to 48 tps, and a 6× increase in latency. To understand this, we analyze the transaction latency breakdown in Fabric and Quorum, the results of which are shown in Figure 14c. It can be seen that Fabric transactions have longer latency, and that in Quorum, the cost is dominated by the consensus phase. The consensus latency in Quorum grows with the record size because Raft protocol takes longer to broadcast larger blocks. In contrast, the Kafka ordering service in Fabric incurs less network communication than Raft, therefore it is less sensitive block size.

Figure 14c shows that the commit time in Fabric is largely constant, whereas in Quorum there is a significant jump from 2ms for 10-byte records to 40ms for 5000-byte records. By analyzing fine-grained timing logs, we find that in Fabric the commit time accounts for less than 30% of the overall latency, whereas in Quorum it accounts for 60% when the record size is 5000 bytes. For each commit, Quorum's virtual machine (EVM) needs to reconstruct a new MPT tree, which involves many expensive cryptographic hash computation. We observe that the cost of a hash function increases with the record size. In particular, we find that the cost of MPT computation increases from 56us to 2.5ms when the record size grows from 10 to 5000 bytes.

Another interesting observation from Figure 14c is that the delay of the proposal phase in Quorum grows at the same rate as 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, a transaction's lifecycle in Quorum suffers from the overhead of the sequential validation of in-block transactions **twice**. Such double batching delay could severely degrade the performance when the smart contracts are computational heavy (, which takes more execution time). In contrast, Fabric adopts a simulate-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 experiment.

5.4 Storage

5.4.1 *Effect of record size on storage*

Figure 15 shows the storage cost per record as we increase the record size. For this experiment, we populated the systems with 100k records. It can be seen that Fabric incurs much higher storage overhead than TiDB. For a 5000-byte record, the state storage consumes around 5000 bytes, while the block storage consumes 15,717 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. We inserted 10k records of different sizes and measured the state storage cost per record.

Figure 16 shows that MBT adds a constant of 24 bytes per record, while MPT adds over 1KB per record. Since both MBT and MPT store the 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 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 compared to MBT.

5.5 Sharding

To compare the impact of sharding on databases and blockchains, we disable full replication in CockroachDB and TiDB, and compare their performance with Attested Hyperledger (AHL) [25], a state-of-the-art sharded blockchain. We note that a comparison with Fabric or Quorum is not possible because these systems do not support sharding. AHL leverages trusted hardware to reduce shard size and to improve consensus throughput per shard. It supports cross-shard transactions by running a BFT shard that implements a 2PC state machine.

In our experiments, we set the number of replicas in the databases and the shard size in the blockchain to 3. Figure 17 shows the throughputs for 8 nodes. It can be seen that AHL's throughput is lower than CockroachDB and TiDB, for both YCSB and Smallbank workloads. The performance gap is due to the high cost of PBFT compared to Raft, and the cost of periodic shard reconfigurations.

6. RELATED WORK

Comparison. Existing works that compare blockchains and databases have highlighted their high-level differences. [28] demonstrates a significant gap in performance, while [24, 34, 59, 20, 61] 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 [59, 20, 61]. Our work presents a deeper and more comprehensive comparison, by looking at the fundamental designs of both systems. We consider them as different types of transactional distributed systems. Our taxonomy and quantitative, performance comparison illustrate both their similarities and their differences.

Performance benchmarking. There is a large number of works that conduct separate benchmarking of distributed databases [22, 14, 10] and blockchains [55, 16]. BLOCKBENCH [29] 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 [29], as we consider systems that are closely related to blockchains in their designs. We investigate the impact of more factors and interpret the results with more fine-grained measurements. Our results demonstrate a smaller performance gap between the two types of systems than previously reported. And such a gap could be smaller under specific workloads.

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 performance [26, 53] or to reason about smart contracts' behavior [52]. [57, 51] use database techniques to enhance the blockchain storage layer and expose richer information to smart contracts. [48, 11, 30] 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. For instance, we can design a hybrid system by starting with a database, then selecting a different choice in one of the four design dimensions discussed in Section 3.

There are recent proposals for solving the blockchain scalability issues by partitioning states into multiple chains. For examples, [35, 62, 36] extend database techniques for achieving atomicity to implement atomic transactions across the chains. However, most have not been fully implemented or tested.

7. CONCLUSIONS

In this paper, we presented a comprehensive comparison between blockchains and distributed databases. We viewed them as two different types of transactional distributed systems, and proposed a taxonomy consisting of four design dimensions: replication, concurrency, storage, and sharding. Using this taxonomy, we discussed how blockchains and distributed databases make different design choices that are driven by their high-level goals (security for blockchains, and performance for databases).

We then performed a quantitative, performance comparison using five different systems covering a large area of the design space. Our results illustrated the effects of different design choices on the overall performance. Our comparison confirmed a large gap in performance between the two classes of systems, but this gap is smaller than previously reported. There are corner cases where the former may outperform the latter, especially when the workload is skewed or the transaction entails more constraints. We also discovered that the sequential in-ledger block validation could limit the system throughput and negatively impact the latency, as in the respective cases of Fabric and Quorum. Finally, our work provides a framework for exploring database-blockchain hybrid designs. By providing deep insight into the behavior of blockchains and distributed databases, our paper opens new directions for future research.

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

8. REFERENCES

[1] Caliper. https://github.com/hyperledger/caliper.

[2] Cockroachdb. https://github.com/cockroachdb/cockroach.

[3] Etcd: Distributed reliable key-value store for the most critical data of a distributed system. https://github.com/etcd-io/etcd.

[4] Ethereum. https://github.com/ethereum/go-ethereum.

[5] Merkle patricia tree. https://github.com/ethereum/wiki/wiki/Patricia-Tree.

[6] The production users of etcd. https://github.com/etcd-io/etcd/blob/master/Documentation/production-users.md.

[7] Quorum. https://github.com/jpmorganchase/quorum.

[8] Tidb. https://github.com/pingcap/tidb.

[9] Ycsb. https://github.com/brianfrankcooper/YCSB.

[10] V. Abramova and J. Bernardino. Nosql databases: Mongodb vs cassandra. In *Proceedings of the international C* conference on computer science and software engineering*, pages 14–22. ACM, 2013.

[11] 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.

[12] J. C. Anderson, J. Lehnardt, and N. Slater. *CouchDB: the definitive guide: time to relax.* " O'Reilly Media, Inc.", 2010.

[13] 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 *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[14] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. Linkbench: a database benchmark based on the facebook social graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1185–1196. ACM, 2013.

[15] 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.

[16] A. Baliga, I. Subhod, P. Kamat, and S. Chatterjee. Performance evaluation of the quorum blockchain platform. *arXiv preprint arXiv:1809.03421*, 2018.

[17] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis. Sok: consensus in the age of blockchains. https://arxiv.org/pdf/1711.03936.pdf.

[18] J. L. Carlson. *Redis in action*. Manning Shelter Island, 2013.

[19] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[20] 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.

[21] 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.

[22] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[23] 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: Googles globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[24] M. Crosby, P. Pattanayak, S. Verma, V. Kalyanaraman, et al. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2(6-10):71, 2016.

[25] 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.

[26] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen. Adding concurrency to smart contracts. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 303–312. ACM, 2017.

[27] 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.

[28] 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.

[29] 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 *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1085–1100. ACM, 2017.

[30] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, and R. Ramamurthy. Blockchaindb: a shared database on blockchains. *PVLDB*, 12(11):1597–1609, 2019.

[31] etcd. Understanding Performance. https://bit.ly/2kzI8R2, 2019.

[32] 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.

[33] S. Gilbert and N. Lynch. Perspectives on the cap theorem. *Computer*, 45(2):30–36, 2012.

[34] F. Glaser. Pervasive decentralisation of digital infrastructures: a framework for blockchain enabled system and use case analysis. 2017.

[35] M. Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 245–254. ACM, 2018.

[36] M. Herlihy, B. Liskov, and L. Shrira. Cross-chain deals and adversarial commerce. *arXiv preprint*

arXiv:1905.09743, 2019.

[37] 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 *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 339–350. ACM, 2010.

[38] 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.

[39] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[40] L. Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

[41] L. Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[42] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[43] K. Li. The Blockchain Scalability Problem & the Race for Visa-Like Transaction Speed. http://archive.today/XnKJC, 2019.

[44] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[45] V. Morabito. Business innovation through blockchain. *Cham: Springer International Publishing*, 2017.

[46] W. Mougayar. *The business blockchain: promise, practice, and application of the next Internet technology*. John Wiley & Sons, 2016.

[47] S. Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[48] 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.

[49] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 305–320, Berkeley, CA, USA, 2014. USENIX Association.

[50] M. T. Özsu and P. Valduriez. *Principles of distributed database systems*. Springer Science & Business Media, 2011.

[51] 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.

[52] 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.

[53] A. Sharma, F. M. Schuhknecht, D. Agrawal, and J. Dittrich. How to databasify a blockchain: the case of hyperledger fabric. *arXiv preprint arXiv:1810.13177*, 2018.

[54] A. Sharma, F. M. Schuhknecht, D. Agrawal, and J. Dittrich. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *Proceedings of the 2019 International Conference on Management of Data*, pages 105–122. ACM, 2019.

[55] P. Thakkar, S. Nathan, and B. Viswanathan. 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)*, pages 264–276. IEEE, 2018.

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

[57] 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.

[58] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. ethereum project yellow paper, 2014.

[59] K. Wüst and A. Gervais. Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.

[60] 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.

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

[62] V. Zakhary, D. Agrawal, and A. E. Abbadi. Atomic commitment across blockchains. *arXiv preprint arXiv:1905.02847*, 2019.