Bulletin of the Technical Committee on

Data Engineering

June 2022 Vol. 45 No. 2

IEEE Computer Society

Letters

Special Issue on Transparent Database Systems

News

Conference and Journal Notices

The TC on Data Engineering
Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin
The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

The June issue of the Data Engineering Bulletin features a collection of papers curated by Prof. Tien Tuan Anh Dinh on the topic of Transparent Database Systems. It is the first issue of the Bulletin dedicated to addressing the rising of blockchain technology and its implication for data management.

A blockchain is an immutable digital ledger of transaction records. It is decentralized, distributed, and encrypted, making it a potential solution for a variety of data-driven applications with varying transparency and privacy requirements.

Several papers in this issue concentrated on leveraging the benefits of blockchains for data management. For example, Korth discussed how to best balance transparency and privacy so that blockchains can provide users with the level of trust that centralized data management systems cannot. Peng et al., on the other hand, focus on the potential benefits for data providers and consumers should we succeed in developing a system that allows privacy-preserving verifiable data sharing using blockchain technology.

While blockchain databases have the potential to offer a variety of benefits such as transparency, privacy, ownership, and trust, currently, they lack meaningful query and analytics support, making it difficult for them to support many real-life applications. A digital ledger, to some extent, is a complicated "OLTP" system optimized for decentralized immutability. Applications that require "OLAP" support may be forced to rely on systems that re-centralize the distributed data, negating all the benefits that blockchains can provide. We are looking forward to discussing these challenges in future Bulletin issues.

Haixun Wang
Instacart

Letter from the Special Issue Editor

The success of blockchains prompts the database community to revisit the trade-offs between security and performance in data management systems. In fact, database researchers in the past few years have made significant contributions to the understanding and advancement of blockchains. This issue focuses on systems that recently emerged (or was resurrected) at the intersection of blockchains and databases. We call them *transparent databases*, which provide data security through transparency. In particular, these systems enable the users to securely verify that both the data and its history have not been tampered with. They achieve transparency by maintaining data in an append-only ledger (a core data structure in blockchains), and protecting the ledger with authenticated data structures such as Merkle trees (core data structures in both blockchains and secure outsourced databases). Although the security community have been deploying similar systems specifically for public key infrastructure, for example, key transparency and certificate transparency, our community's interest in transparent databases stems from the challenges in building general-purpose, high-performance systems that solve real-world data management problems. This issue contains perspectives from expert researchers working on this topic. They share their views on the state-of-the-art, the use cases, and the future research directions.

The issue opens with a contribution from Henry F. Korth, in which he reminds us of how transparency is often at odds with privacy, and more importantly, their trade-offs are made for us by a trusted party. He explains how blockchains, which removes the trust on opaque institutions, can revolutionize most data-driven applications. He highlights two building blocks that are vital to such revolutions: Merkle trees and zero knowledge proofs. The former allows for selective disclosure of information, and the latter for proving correct execution without revealing the data. When combined, they enable not only integrity protection of data and computation, but also of the data provenance. The second paper, by Zhe Peng, Jianliang Xu, Haibo Hu, and Lei Chen, demonstrates how these techniques can be used to give data owners control over their data. The authors present a timely example of COVID-19 data sharing, in which users want fine-grained control of what data to share and with whom. For this use case, a Merkle tree is built over the user data and its root is published on a blockchain. To selectively share some functions on some data, the user constructs a Merkle proof for the data, and a zero-knowledge proof showing that the output is computed correctly on the input whose Merkle root is on the blockchain.

Blockchains are an important component of transparent databases, because at very least they can serve as a public bulletin board where commitments are stored. The next two papers describe the latest techniques for improving performance and security of blockchains. Junchao Chen, Suyash Gupta, Sajjad Rahnama, and Mohammad Sadoghi, present interesting insights on the advantages and limitations of two types of consensus protocols. On the one hand, Byzantine Fault Tolerance (BFT) protocols have high performance, but require strong identities, and they can be broken when the attacker steals more than f private keys. On the other hand, Proof of Work (PoW) protocols are harder to break, but they are unsustainable. The authors then propose a new protocol, called Proof of Collaboration, that aims to have the best of both worlds. Deepal Tennakoon and Vincent Gramoli discuss the state-of-the-art on blockchain sharding — the popular database technique in which the data is partitioned into multiple shards. Sharding helps scale blockchain throughputs by distributing the works. However, the key challenge in blockchain sharding, which does not exist in traditional database settings, is the presence of malicious attackers that influence shard assignments in order to insert themselves to target shards. If successful, the attackers can break the fault tolerance threshold of the target shard and subsequently break the security of the blockchain. The authors explain how *probabilistic sharding* relies on trusted sources of randomness to avoid such attacks. They propose another layer of defense, which is to make sharding transparent such that users can verify how the shard is formed.

While transparent databases can be built directly on existing blockchains that are mainly designed for cryptocurrency or assets management applications, the next paper by Dumitrel Loghin describes another approach based on *blockchain databases*. Such systems integrate blockchain and database features, and are classified into permissioned blockchains, hybrid systems, and ledger databases. They share a similar architecture that consists of a ledger storage for data history, a database storage for the states, and a broadcasting service for coordination. Hybrid systems adopt either an out-of-blockchain database design, in which the system starts with a blockchain and builds database features on top of it, or an out-of-database blockchain design, in which the system starts with a database and builds blockchain features to it. The author compares the performance of different systems and shows that ledger databases achieve the highest throughputs. The last paper, by Meihui Zhang, Cong Yue, Changhao Zhu, and Ziyue Zhong, provides in-depth analysis of ledger databases. The authors identify a number of design choices that impact the overall security and performance. They then propose a benchmarking framework, named LedgerBench, for comparing different systems. The framework contains workloads that stress the unique features of ledger databases such as verification and auditing. It has flexible APIs such that new workloads and systems can be easily added. This paper also presents interesting experimental results on their implementations of three commercial systems: LedgerDB from Alibaba, QLDB from Amazon, and SQL Ledger from Microsoft. One of the main findings is that updating the ledger and verification constitute the main performance bottleneck.

It is a real pleasure working with the authors for this issue. Their insights are refreshing and I believe the readers will learn much from reading their contributions.

Tien Tuan Anh Dinh
Singapore University of Technology and Design

Employing Blockchain Properties for Transparent Databases

Henry F. Korth
Lehigh University
blockchain.cse.lehigh.edu

Abstract

Both transparency and privacy are important properties in most any information-management application, yet they are in many ways inherently opposed to each other. The tradeoff between these two goals is typically intermediated by a trusted system or organization. The resulting tradeoff is only as meaningful as the level of trust in the intermediary. Blockchains provide a decentralized single source of truth that allows for the reduction or elimination of the need for trust. This paper explores the options in trading off transparency and privacy in a blockchain database. It also considers the closely-related concepts of accountability and auditability and explores how cryptographic techniques allow the disclosures needed for regulatable information systems while preserving a high degree of privacy.

1 Transparency Versus Privacy and the Role of Trust

In a traditional enterprise information system, the database is the final source of truth. The data in the database are closely guarded by access controls and integrity checking. Privacy regulations often apply to the use of data (examples include the U.S. HIPPA and FERPA regulations for health and education records respectively). Transparency regulations are exemplified by requirements that enterprises provide financial statements at specific intervals with specific content and by requirements that enterprises disclose to each customer what data they hold on that customer. Despite these requirements, the entire process depends on a strong degree of trust in enterprises to provide consistent and truthful disclosures. Ensuring truthfulness requires an audit of enterprise data that, in turn, mandates further disclosures at least to the auditors. Increasing data disclosure can compromise privacy.

The blockchain properties of irrefutability, immutability, and anonymity [18] offer useful tools in managing the tradeoffs among privacy, transparency, and trust. However, much depends on how those tools are used. A simple move to a blockchain database presents a variety of challenges. A public blockchain is public to all. The only secret is the mapping between blockchain IDs and the identity of the corresponding real-world person or organization. That mapping can often be determined by correlation of on-chain and off-chain activity. The primary value of a blockchain in a transparent, privacy-preserving information system is its role as a trusted source of truth. The "truth" stored on that blockchain can take a variety of forms. From a database standpoint, the most important stored data are cryptographic commitments to protected data (using Merkle trees [16]) whose contents can be disclosed in part or in aggregate with a proof of the correctness of such disclosures. In such a

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

system, the blockchain holds mainly commitments. Data themselves are stored off-chain at much lower cost than on-chain storage.[1]

Disclosure of data is not the only trust consideration. When the disclosure is some function computed over the actual full dataset, one must trust the agent who did the computation. A flawed computation from correct data is of no more use than inaccurate data. Thus, beyond data, one must consider the provenance of data, that is, what data were used in the computation and what code was used to do the computation. Provenance in data has been studied at length for well over a decade [7]. Zero-knowledge proofs[2] (henceforth, ZK-proofs) are a cryptographic method for allowing a verifier to validate the correctness of an execution without having to know the actual data on which the execution operated. This makes it possible to prove that the disclosed result was produced by a specific program running on the same dataset to which a commitment was made. No information need be revealed about the data other than the result itself. The combination of provenance management and ZK-proofs of execution enable transparent trust not only in data but also in how those data were generated.

Our discussion of blockchain and database transparency is thus largely focused on trust. The ability of a blockchain to move the focus of trust from centralized authorities to a decentralized consensus creates trust in a process rather than in an opaque institution. This re-focusing of trust can revolutionize virtually all information-driven applications. A few examples follow:

- **Supply chain:** Businesses enter into supply-chain agreements via a legally binding contract. Traditional day-to-day tracking of those relationships either depends on a trusted member of the chain storing data or on a collection of separately managed repositories. A blockchain-based supply-chain information system replaces that with cryptographic signatures[8] for irrefutability, and verifiable, distributed updates for immutability. Trust in the final-product producer is limited only to that firm admitting the right set of users to the system. The blockchain is a single source of supply information that enables fast, effective product recalls with no "finger pointing" among suppliers trying to blame each other for supplying a defective component.

- **Real assets:** Governments maintain trusted records of asset ownership (real estate, vehicles, etc.). These systems require trust not only in government but also in the controls implemented by government on their databases. Each transaction becomes a complex process of validation of identity – of the parties involved and of the asset involved. In a blockchain-based system, the only function the government needs to provide is a single, permanent mapping between the real asset and a unique blockchain token (that is, a non-fungible token, or NFT[2]). Transactions regarding asset ownership then become simple blockchain transactions requiring no central intermediation by government. The amount of trust left to a central institution is limited to maintaining the physical-asset-to-NFT mapping.

- **Market making:** The creation of markets for trading stocks, exchanging currencies, etc. is a large, slow, high-overhead business. Most New York Stock Exchange trades take two business days to settle. International funds transfer via the SWIFT system may be anything but what that name suggests. The SWIFT system is a messaging system among correspondent banks, with funds transfers following the messaging. The process can still take days.

  In contrast, competing blockchain systems in this space take seconds (e.g., Stellar[13] and Ripple). Automated market-makers adjust prices with each trade. Automated arbitrageurs maintain price-equivalence across markets. No institution needs to be trusted; trust is in only publicly readable code.

[1]On-chain storage costs vary. On the Ethereum chain, based on gas price and ETH prices at the time this paper was written, storing 1MB of data on-chain would have cost about 30000 US dollars. Decentralized storage services like Filecoin employ cryptographically-secured off-chain storage.

[2]To be clear, our focus here is about assets like houses and cars, not cryptokitties, cyberpunks, bored apes, or something similar.

- **Lending:** Much like market-making, an automated system can manage real-time interest rates both for lenders and borrowers. Liquidators earn a fee for managing loans that become under-collateralized and doing so in an automated manner.

The list could go on. In each case, there is a gain in speed from taking humans out of the transaction path, and a gain in trustlessness by removing or reducing the need to trust institutions whose operation is opaque. Open-source security replaces institutional trust and "security by obscurity."

In what follows, we explore the implications of blockchain technology on traditional information-based applications from the standpoints of transparency, privacy, and trust.

2 Hiding Information in Public View

Implementing personal and business interactions require selective, limited disclosure of information. Disclosures may be limited in content and scope for a variety of sound reasons. For example, one does not publish one's salary, net worth, or medical history on one's web site, yet there are cases where certain disclosures are required. The technology and mathematics that has come together in blockchain systems enables separate, limited disclosures. Furthermore, these capabilities allow one to prove independent disclosures to be consistent with each other.

2.1 Securing Data Using a Merkle Tree

One key to information hiding is cryptographic hash functions, which have the property that although it is relatively easy to compute $H(x)$, given a value y, it is infeasible, given y, to find an x such that $H(x) = y$. Here, "infeasible" means that is strong evidence that there is no better algorithm to find x than guessing. Standard cryptographic hash functions have ranges on the order of 256-bit integers, meaning the likelihood of a successful guess is virtually zero. Blockchains use this to include in each block the hash of the prior block, making it infeasible to modify a block without modifying subsequent blocks. Merkle trees[16] create a tree structure of nodes that contain hashes of their children, down to leaves that hold actual data. The root of a Merkle tree (referred to as a "Merkle root") is thus a hash of the full dataset. The tree structure and the associated algorithms allow one to to prove the a single data item is, or is not, a member of the dataset, without disclosing anything more about the overall data in the dataset.

Placing a Merkle root in a transaction on a public blockchain creates a signed, public commitment by a user to a dataset without revealing anything about the dataset. Subsequently, that user can reveal specific data items (say to a tax authority) and show that those items are in the dataset to which a public commitment had been made. A future disclosure, if needed, can be proved similarly to be from that same base dataset. The result of this combination of Merkle trees and blockchain allows for a framework for selective information disclosure while ensuring privacy of associated data.

2.2 Proving Execution of Code Using Zero Knowledge

The selective-disclosure capability enabled by Merkle trees does not address transparency fully. Oftentimes, what is desired is aggregate reporting regarding a dataset rather than just disclosure of a few specific elements of that dataset. A Merkle tree alone cannot prove anything about an aggregation of data unless one is willing to disclose every data item that went into that aggregation. A better compromise between privacy and transparency would be achieved if one could show that an aggregate report was generated from a base dataset to which a public cryptographic commitment had been made, without actually disclosing any members of the underlying dataset. To state that precisely, let D be a dataset stored privately but with a Merkle root M published on a public blockchain. Let P be a program computing an aggregate report, and let $R = P(D)$ be the report generated when running P on dataset D. Using a ZK-proof, it is possible to prove that report R was generated, using program P, from a dataset for which M is the Merkle root. The result of this is that the user has disclosed only the actual report R, the report-generating software P, and the Merkle root M of the input. Neither the input data themselves (D) nor the details of this particular execution of P are disclosed, only the ZK-proof. It is relatively easy computationally to verify a ZK-proof, meaning that it is practical to verify aggregate reports based on a standard reporting methodology without the need to know anything about the underlying data. If there is a future need to audit that report, specific data from the underlying dataset can be revealed on demand without needed to reveal anything more than the minimum requested.

2.3 Transparent Provenance of Data

Generalizing from the above observations, one can consider a sequence of actions that lead to an action or to the generation of certain data items. Beyond just the certification of a single aggregate report, one can certify a sequence of events and prove in a publicly verifiable manner how the end result was obtained. The result is that transparency is not only about the current state of the database but also about the means through which the current state was generated. Such transparency can result in trustworthy information versus information based on trust in a centralized information provider. This high degree of transparency underlies the concept of *Web 3*, in which users control their own data and their exchange of data, based on an underlying blockchain infrastructure. The Web 3 vision stands in contract to most current web-based interactions (referred to as Web 2) in which centrally managed search engines, social networks, and information providers serve as intermediaries controlling user access to and interpretation of data.

This level of transparency allows for a public proof of how products are sourced. With a blockchain, rather than a corporation or government, serving as the central source of truth, one can provide a public proof that an end product being sold to a consumer arrived on the shelf via a validated supply chain in which the workers involved in producing the product were paid a fair wage. Starting from wages paid on a blockchain via cryptocurrency, through transfers in a supply chain documented in signed blockchain transactions, to the end consumer's purchase, there can be a public verification of the provenance of the product and the data related to it. There are many technical issues in getting to this point (including identity, which we discuss below in Section 4), and a variety of prototype projects underway, including one involving Lehigh Blockchain students, a local blockchain firm, and a Central American coffee producer with the goal of certifying fair-trade coffee on a blockchain.

A publicly verifiable proof of provenance is critical component of transparency. Given that transparency, the next problem is the evaluation of that provenance to determine whether it meets standards of correctness in the flow of data and work. This may seem not to be a major concern in the report-generation examples we have seen so far, but can be a larger problem in other cases, depending on the application.

2.4 Challenges

The above discussion of the possibilities of an apparently ideal mix of privacy and transparency does have its limitations. First, the viability of using a Merkle tree depends on agreed-upon and strict data formats. Changing even a bit in a data value changes its hash unpredictably. Adherence to such a specific, detailed degree of standardization may be challenging in practice. To test the practicality of the requisite standarization, we are currently prototyping a design of such a framework to automate major parts of accounting and audit.[11]

Another serious limitation is the difficulty in generating ZK-proofs efficiently. To generate a proof, one must compile the program into a low-level language, typically one consisting of simple arithmetic circuits (of the form A op B = result). An execution is represented as a polynomial over the program variables and temporary variables in the compiled code. Each statement in the execution defines a constraint on that polynomial. Finding these high-degree polynomials subject to the massive number of constraints is a huge computational problem. Production zero-knowledge systems have constraint sizes in the tens of millions. Billions (or more) are envisioned in future applications. Fortunately, the computations are parallelizable. In [24], a parallelized, ASIC-based approach is presented. As ZK-proofs become a widely used tool, not only for the transparency and privacy issues we discuss here, but also for blockchain performance acceleration, commodity parallel hardware such as GPUs are a promising tool for ZK-proof generation. ZK-proof computation moves from a numerical computation challenge to a database-style challenge when one considers the sheer volume of constraint systems. GPU parallelization alone is unable to overcome the performance impact of secondary-storage bottlenecks of large constraint systems. An alternative approach of nested parallelism using multiple GPUs and data sharing via RDMA is needed to exploit the parallelism of this database-scale problem[19].

3 Digital Currency

Although most modern financial transactions are processed digitally, physical cash and checks still are in wide use. Taking the last steps in a full transition to digital currency presents a variety of data management challenges and opportunities. One of the most notable opportunities is making bank-like service available to vast number of unbanked individuals.[3] We explore first data issues in a government-sponsored digital currency, referred to as a *central-bank digital currency*(CBDC). Next, we explore stablecoins, private currencies that aim to track the value of a government-backed fiat currency.

In both cases of digital currency, we face tradeoffs among transparency, privacy, regulatability, and performance.

3.1 CBDCs

The concept of a digital currency issued by a government central bank is gaining global attention[17]. The finance and policy details are beyond the scope of this paper, but the technical options available in CBDC applications serve to illustrate the tradeoffs among transparency, privacy, and regulatability.

At one end of the spectrum is the approach taken by the People's Republic of China[9], in which currency management is structured in a manner closer to a centrally administered database than to a decentralized blockchain. There, techniques associated with blockchains, such as digital signatures, are employed instead towards the goals of centralization. Other national central banks are studying digital currencies, many from a more decentralized standpoint. Of note is a recent statement by the U.S. central bank, the Federal Reserve[4], and the possibilities of achieving the benefits of some degree of decentralization similar to the current cash-based system[10]. Private currencies have launched as well, most notably Libra, which began with the backing of major financial (and other) firms, but quickly lost favor (despite its technical strengths – see, e.g.[45]) due to opposition from central banks, government leaders, and others. A key lesson from the Libra debacle is the need for digital currency solutions not only to be technically sound but also to provide policy makers with the politically desired degree of oversight and policy options.

Underlying all of these developments are database-centric issues of transparency and privacy. Regulation aimed at avoiding money laundering, "know-your-customer" rules aimed at avoiding funding illicit organizations, etc., require some level of disclosure of financial transactions. However, people generally seek to have a strong degree of privacy in their personal finances. The technical challenge here is to generate the requisite reporting and oversight while limiting not only the amount of personal data disclosed but also to whom those disclosures are made. Here, again, we see the concept of selective, limited disclosure that we discussed in Section 2.

Beyond the use of Merkle trees and ZK-proofs for reporting and disclosure, one must consider the ownership of the underlying data pertaining to digital-currency transactions. Centralized data ownership mandates a total trust in that central data owner. Most national financial systems have decentralized data ownership of financial data (e.g. credit-card companies, payment systems, and banks) with specific personal data available externally

[3]World Bank[20] data indicates that roughly 1.7 billion people globally are unbanked. While most are in the underdeveloped world, the problem is widespread. In the U.S., about 6% of the population is unbanked.

only in aggregate or via subpoena. A consequence of decentralized data ownership is decentralized control over transaction commit. That suggests use of a blockchain, since decentralized data ownership and decentralized control are foundations of blockchain systems. Translating those blockchain strengths to a database-scale framework like a digital-currency system is hard. Given the high transaction rates of a global-scale digital currency, consensus performance needs to be much greater than that of a typical blockchain. Parallelism and concurrency offer hopes for increased performance, but concurrency leads to further contention and performance impact. Decentralization of ownership and control was studied in the database research community in the 1980s and 1990s in the context of federated databases, or multidatabases[15, 23], that were assumed to be managed by independent semi-autonomous organizations. The results of that research present some important insights into the challenges of partially decentralized digital currencies. Tasks that are relatively simple in centrally administered distributed systems become hard. Global transaction commit via two-phase commit conflicts with autonomy, leading to relaxed notions of atomicity[12] including optimistic commit, and semantic correctness as an alternative to serializability. Approximate consistency, however, is not acceptable in a financial system beyond discrepancies that are deemed "non-material." Any non-material inconsistencies in a financial system must be safe from systemic exploitation that generates material inconsistencies.

Offline transactions are an important component of digital currency that presents a largely new problem to the database community. In the early days of mobile computing, there was discussion of a model of computing that included "frequent, foreseeable disconnection,"[1] but that concern quickly evaporated with the advent of virtually ubiquitous connectivity. In the world of personal financial transactions, the demand for offline transactions is likely to persist. Offline transactions could, admittedly, be for illicit purposes, but they also serve an important role for personal privacy and convenience. The argument that one should not fear any loss of privacy ("if you have nothing to hide...") would require trust in the government not to implement a surveillance state. For a currency with global aspirations, it is not reasonable to expect that level of trust to hold throughout the world. This observation is the key reason why the offline transaction problem is of great importance. The nation-state competition emerging to unseat the dollar as the global currency is likely to be influenced heavily the the ease of use of that currency in offline transactions around the world.

Offline transactions are a special case of the well-known issue of network partitioning. The CAP theorem[3] provides a strong formal statement of the issues in meeting the real-world requirement for offline transactions. The programmable-money concept[10] provides a model of storing information for subsequent reconciliation. The solution to this problem relies on a proper abstraction of the provenance of offline payments that allows for their eventual reconciliation with online data. Stated differently, we seek a form of eventual consistency with transaction properties that are more ACID-like and less BASE-like.[4]

While much has been written about the policy and geopolitical issues around digital currency, much remains to be done in turning the vision into reality. The database community can offer useful insight here not only in algorithms and systems, but also in providing a conceptual framework to allow policy-makers to make the levels of privacy and transparency, and concepts supporting data consistency explainable to the public.

3.2 Stablecoins

A stablecoin is a cryptocurrency not issued by a government, but one that comes with a promise that its value will track a government fiat currency (typically the U.S. Dollar). That "promise" may be based on transparency, trust, of some combination thereof. Stablecoin implementation can be split into two categories: reserve-backed and algorithmic. In a reserve-backed stablecoin, there is a promise that the stablecoin is backed by safe, secure assets in the underlying fiat currency (bank deposits, government-issued debt, and other assets generally accepted as low risk). Trust in a reserve-backed stablecoin rests on trust in the promised asset backing. An algorithmic stablecoin is backed not by fiat currency reserves, but rather an automated system that maintains the fiat-currency peg by

[4]The BASE properties are an analog to the well-known ACID properties of database transactions: Basically Available, Soft state, Eventually consistent.

automated trades and/or incenting certain actions by arbitrageurs. Trust in an algorithmic stablecoin rests on trust in its code and the mathematical model that the code implements.

Assessment of the backing of a reserve-backed stablecoin requires a high-degree of transparency regarding the holdings of the backer of the stablecoin. While such transparency is desirable, there is a substantial degree of opacity in the nature of the backing of certain present-day stablecoins. As stablecoins become more systemically important to the financial system, calls are emerging for bank-like regulation of stablecoins. An alternative model to traditional regulation is a highly-automated, globally verifiable audit system based on the concepts of Section 2. Such a model can provide complete transparency for the reserve, with frequent updates to the audit possible. As stablecoins come under closer public scrutiny, innovative approaches to transparent, publicly verifiable accounting and audit of stablecoin reserves are likely to become important research topics.

Algorithmic stablecoins do not have a reserve in the underlying fiat currency. Instead the stablecoin is paired with a second cryptocurrency that serves as a backing to the actual stablecoin. Algorithmically-enabled exchanges between these two cryptocurrencies aim to keep the stablecoin very close to its fiat peg. This was the structure of the recently failed TerraUSD UST stablecoin.

MakerDAO and the associated DAI stablecoin is backed by non-fiat overcollateralization and is run by an automated system implemented in a smart contract as a decentralized autonomous organization (DAO). The collateral provides a degree of security, but since that collateral is not in the associated fiat currency, there are risks that are algorithmically mitigated, e.g., by a liquidation mechanism. Such currencies' transparency is largely beyond the scope of databases, since the operation is algorithmically driven and thus backed by code rather than data.

3.3 Security of a High-Value Digital-Currency Database

Our discussion of digital currency has focused on routine transactions and the supporting database. Unlike enterprise databases (such as those of banks), digital currencies are directly open to the public with little or no intermediation. This creates a wide range of security vulnerabilities beyond those typically considered in a database setting. Whereas an enterprise database is private and thus secured by not only database authorization but also by the operating system's security and the enterprise network firewalls, a public blockchain is truly public. Anyone can join. Anyone can submit a transaction. This openness enables not only a high level of transparency but also a high-level of risk. In a controlled-access database environment, one need not worry about a denial-of-service attack because there are other mechanisms external to the database to control that. A digital-currency database is, by its nature, necessarily open and public, yet it presents a high-value target of attack, for example, by an enemy nation-state.

While it is beyond our scope here to go into the details of security, it is worthwhile to consider the tradeoff between transparency, privacy, and security for a digital currency. The virtues of a transparent, open digital currency including global use and banking for the unbanked, create potential openings for a malicious user or set of users to launch an attack. A permissioned blockchain, as is typical of an enterprise setting, can expel a malicious member because of the centralization of membership control. Applying that concept to an open digital currency results in some central authority granting (and, subsequently enforcing) the right of access. Designing such an authority in a way that makes it highly permissive yet secure requires careful real-time analysis of the behavior of anonymous users both individually and collectively. This is a hard data-analytics problem in its own right, but becomes a deep research challenge when coupled with the real-time performance constraints of a digital currency.

3.4 Quis custodiet ipsos custodes?

This famous quote ("who will guard the guards themselves?") from the Roman poet Juvenal is highly relevant in a large-scale financial system. A node that processes digital-currency transactions may have sufficient knowledge of the overall set of transactions to allow that node to reorder the sequence of events to another correct order more to that node's financial advantage. Thus, governance of the system's consensus mechanism is required to ensure that not only is consensus reached on a syntactically correct execution, but also that the agreed-upon execution satisfies broader constraints. Enforcement of global data consistency constraints of this sort was not a major focus in early blockchain design, and gaps in those constraints have been exploited[6]. Prevention, detection, and mitigation of such concerns is a further component of the design of consensus mechanisms in financial systems[5].

4 Identity Management

Identity is perhaps the single largest challenge in blockchain systems that interact with the physical world. Blockchain IDs identify the submitters of transactions and owners of crypto-assets, but ownership of physical-world assets must be enforced in the real-world. That requires some level of authority to map physical assets to blockchain assets (NFTs) and map blockchain identities to actual people or enterprises.

The permanence and universality of blockchain IDs creates an extreme transparency at the potential price of privacy. Once the mapping between a blockchain ID and an individual is established, that individual's entire transactional history becomes public. In the off-chain world, the ubiquity of a government-issued ID (e.g., U.S. social-security numbers), presents an even greater threat since disclosure of such an ID not only can violate privacy but also enable identity theft. Matters are less dire in the blockchain world since disclosure of who owns a blockchain ID does not enable computation of the corresponding private key. Thus the disclosure of the mapping between blockchain identity and real-world identity impact "only" privacy without enabling forging of transactions. Nevertheless, the potential of one's ID being disclosed is a substantial and unrepairable privacy threat.

The W3C verifiable credentials data model[21] provides a means of creating digital credentials that can be submitted as proof that the bearer has certain properties or qualifications (e.g., is over 21, is a licensed driver, holds a specific academic degree). A digital verifiable credential differs from traditional credentials in its use of cryptographic signatures and zero-knowledge. An issuer can provide a verifiable credential to an individual and sign it. The individual may then also sign it and present it for whatever purpose. The value of the verifiable credential rests in its ability to be verified digitally using the public key of each signer. Under such a scheme, a purchaser of alcoholic beverages could prove age without having to reveal other information (such as driver's license number). This verification scheme allows transparent proof that certain regulations, rules, etc. are being followed without disclosure of personal identifiers.

In the enterprise-blockchain environment, Hyperledger Aries (one of several projects under the Hyperledger umbrella – Hyperledger Fabric is the most widely known) is a toolkit for the use of verifiable credentials. It is gaining use in a variety of applications, especially supply chains in which a collection of firms cooperate with some degree of trust, but not absolute trust.

The ability to infer full provenance of data and the possibility of matching blockchain identity with real-world identity presents challenges beyond credentials. Consider an employee whose salary is paid in cryptocurrency. Each "paycheck" is public on that blockchain, creating the possibility that the employee's salary could become public. While salary information is revealed routinely to tax authorities, credit providers, etc., most individuals would not be comfortable publishing their salaries to the world. This legitimate desire of financial privacy runs counter to the transparency requirements of regulators seeking to prevent money laundering, funding of terrorism, etc.

In the traditional pre-blockchain world, identity management and protection is intermediated by a trusted provider. For the paycheck example, a bank or other financial institution accepts the payment and credits the funds to a database it owns and manages. The employees can spend income from that paycheck without there being any public connection between the paycheck and the spending pattern; only the trusted bank knows. Blockchains achieve a similar level of privacy by hiding the identity of a party to a transaction. Continuing the paycheck example, the employee would use one ID to receive pay, then have that ID send the funds through a transaction privacy tool to several other IDs owned by that same employee. Those latter IDs would be used for expenditures.

The above example may initially appear perfectly legitimate, but it could be used by a malicious actor for money laundering. If we assume that our example employee is not a money launderer, but rather just an individual who does not believe personal finances should be public, we then need a means of allowing that individual to provide proof of the propriety of transactions to authorities without needed to provide a full public disclosure. This is yet another example of the need for privacy with selective transparency. Using techniques similar to those we discussed in Section 2, Tornado Cash (tornado.cash) provides a tool for private transactions not visible to the public. Tornado Cash offers what it calls a "compliance tool" for a user to obtain a publicly verifiable proof of the source of specific funds. The user can then (presumably in a private, secure manner) submit that proof to authorities of the user's choice. Where this mechanism differs from the use of a traditional bank is that here, the intermediary (Tornado Cash) obtains no identity information from it users. All it certifies is a flow of funds among IDs. In our example, the user would have to prove ownership of the IDs to authorities, which can be done through the verifiable credentials approach discussed earlier.

5 Conclusion

Blockchain-driven information systems provide a foundation for a trust-free or trust-minimized environment for users to manage the inherent conflicts among transparency, privacy, and regulation/audit. The use of a blockchain is a necessary but not sufficient feature of such environments. A blockchain provides a crowd-certified source of truth (and thus trust), but the management of information itself requires further use of blockchain technologies from the realm of cryptography, including Merkle trees and zero-knowledge.

Initial blockchain applications have been able to succeed with modest transaction performance levels and modest data volumes. However, enterprise applications are bringing the scale of enterprise databases into the realm of blockchain-centric transparency/privacy tradeoffs. Similarly, the rise of digital currencies, especially CBDCs, are soon to generate transactions rates well beyond those of current database applications.

The technical challenges of scaling blockchains to database levels are made greater by the more open, thus attack-prone, setting of public blockchains. Finding the right tradeoff between the controls of a private, permissioned blockchain, and the social benefits of open access to a financial system is a challenge spanning computer science and public policy. Blockchain databases present a challenge to the database community to revisit traditional research issues in a different framework: different computational model, different data structures, and different target applications.

Acknowledgments

Lehigh Blockchain (blockchain.cse.lehigh.edu) is supported by gifts from Steel Perlot, the Stellar Foundation, the Oracle Corporation, and a Lehigh CORE grant.
We acknowledge our collaborators in Lehigh's Scalable Systems and Software Research Group (sss.cse.lehigh.edu), Lehigh's Institute for Data, Intelligent Systems, and Computation (https://idisc.lehigh.edu/), and Lehigh's Center for Financial Services (business.lehigh.edu/centers/center-financial-services).

References

[1] R. Alonso and H. F. Korth. Database System Issues in Nomadic Computing. Proc. ACM SIGMOD International Conference on Management of Data, 1993.

[2] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything Provable is Provable in Zero-Knowledge. *Advances in Cryptology — CRYPTO' 88.*, Springer, 1988.

[3] E. Brewer. CAP Twelve Years Later: How the "Rules" Have Changed. *Computer*, vol. 45, no. 2, pp. 23-29, Feb. 2012.

[4] Board of Governors of the Federal Reserve System. *Money and Payments: The U.S. Dollar in the Age of Digital Transformation*, Jan 2022.

[5] J. Byers. *Combating Front-running in the Blockchain Ecosystem.* Masters Thesis, Lehigh University, 2022.

[6] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. *2020 IEEE Symposium on Security and Privacy*

[7] J. Cheney and S. Chong, N. Foster, M. I. Seltzer, S. Vansummeren. Provenance: a future history. OOPSALA, 2009.

[8] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22:6, 1976.

[9] Y. J. Fanusie and E. Jin. China's digital currency. *Technical report, Center for a New American Security*, 2021.

[10] *Knox Networks Provides Feedback to the Federal Reserve on a Potential US Digital Dollar*, May 2022. https://medium.com/@knoxnetworks/knox-networks-provides-feedback-to-the-federal-reserve-on-a-potential-digital-dollar-fb7e77418fca

[11] H. F. Korth, N. M. Snow, and B. B. Fanuscu. Provably Correct Financial Disclosures. *In Preparation*. 2022.

[12] E. Levy, H. F. Korth, and A. Silberschatz. A Theory of Relaxed Atomicity. Proc. 1991 ACM Symposium on Principles of Distributed Computing (PODC).

[13] G. Losa, E. Gafni, and D. Mazières. Fast and secure global payments with Stellar. Proc. 27th ACM Symposium on Operating Systems Principles (SOSP), 2019.

[14] O. Malekan. *Re-Architecting Trust: The curse of history and the crypto cure for money, markets, and platforms*. Bookbaby, Pennsauken, NJ, 2022.

[15] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. The Concurrency Control Problem in Multidatabases: Characteristics and Solutions. ACM SIGMOD International Conference on Management of Data, 1992.

[16] R C. Merkle. A Digital Signature Based on a Conventional Encryption Function. Advances in Cryptology — CRYPTO '87, Springer, 1987.

[17] W. Peracchio. *Design Considerations for Central Bank Digital Currencies*. Masters Thesis, Lehigh University, 2021.

[18] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, 7th edition*, chapter 26: Blockchain Databases. McGraw Hill Education, New York, NY, 2020.

[19] M. Vezenov. *Accelerating zkSNARKs on Modern Architectures*. Masters Thesis, Lehigh University, 2022.

[20] World Bank. *Global Findex Database.* global.findex.worldbank.org

[21] World-Wide Web Consortium. *Verifiable Credentials Data Model v1.1*, March 2022 https://www.w3.org/TR/vc-data-model/

[22] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. HotStuff: BFT Consensus with Linearity and Responsiveness Proc. 2019 ACM Symposium on Principles of Distributed Computing (PODC).

[23] A. Zhang and A. K. Elmagarmid. A Theory of Global Concurrency Control in Multidatabase Systems. *VLDB Journal* vol. 2 no. 3, 1993.

[24] Y. Zhang, S. Wang, X. Zhang, J. Dong, X. Mao, F. Long, C. Wang, D. Zhou, M. Gao, and G. Sun. PipeZK: Accelerating Zero-Knowledge Proof with a Pipelined Architecture. 48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021.

BlockShare: A Blockchain Empowered System for Privacy-Preserving Verifiable Data Sharing

Zhe Peng[*], Jianliang Xu[*(✉)], Haibo Hu[†], and Lei Chen[‡]
[*]Hong Kong Baptist University
[*]{pengzhe, xujl}@comp.hkbu.edu.hk
[†]Hong Kong Polytechnic University
[†]haibo.hu@polyu.edu.hk
[‡]Hong Kong University of Science and Technology
[‡]leichen@cse.ust.hk

Abstract

Data has become the most valuable resource, as it is essential to analytics, decision making, and artificial intelligence. To unleash the value of data, data sharing has become a prerequisite to bringing tremendous benefits for both data providers and data consumers. However, existing solutions for data sharing are mostly server-centric, i.e., they rely on the control of a trusted server, which increases security and privacy concerns among data users. The design of a privacy-preserving verifiable data sharing framework to simultaneously secure user privacy and data integrity has not been sufficiently studied and remains a grand challenge. In this paper, we propose BlockShare, a privacy-preserving verifiable data sharing system based on blockchain. First, we design a novel blockchain-based architecture, together with a new authenticated data structure scheme to efficiently verify any portion of a shared data record in a decentralized fashion. Second, we develop a zero-knowledge verification scheme that enables a user to prove a dynamic condition without disclosing the specific data attribute, minimizing privacy loss. We implement BlockShare and conduct experiments to evaluate the system performance. Experimental results demonstrate the effectiveness of our proposed system.

1 Introduction

Data has been considered the “oil" of the digital world, as it is essential to analytics, decision making, and artificial intelligence. Recent years have witnessed increasing availability of personal data such as health, financial, and geo-social data. Sharing such data with relevant stakeholders is a prerequisite for unleashing its value. For example, sharing medical records with healthcare organizations can improve the quality of patient care, reduce insurance fraud, advance clinical research, and timely predict epidemic outbreaks like COVID-19. To achieve secure data sharing, most existing systems rely on the security of cloud service providers, who claim to protect client data for storage and sharing.

However, with the increasing value of data and growing cybersecurity threats, these cloud systems cannot fully address users’ security and privacy concerns. On the one hand, there have been many reported cloud security

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

breaches, such as Apple's iCloud data leak in 2014 and Instagram's exposure of 49 million user accounts in 2019. On the other hand, the Facebook-Cambridge Analytica data scandal in 2018 has exposed the issue of corporate dishonesty to grab users' data without consent for her benefit. Both causes have affected users' willingness to share personal data using cloud-based solutions. New data privacy legislation, such as the European Union General Data Protection Regulation (GDPR) [1] or the EFF's call for information fiduciary rules for businesses [2], is an important step towards addressing data abuses, complemented by modern technological solutions that put users back into control.

Emerging blockchain or distributed ledger technology (DLT) has been adopted as a trustworthy data storage solution in various fields [3, 4, 5]. It allows trustless parties to collectively maintain a single version of data without a central authority. By features of decentralization, immutability, and transparency, blockchain enables the decentralized and secure exchange of digital information and assets without a central trusted server. As such, this new paradigm has fundamentally changed the way of data sharing and give rise to data markets controlled by users, instead of surrendering data governance to a few tech giants [6]. It has also increased willingness to share data and break data silos to enforce cross-organization or even cross-border data cooperation.

From these observations, we identify the following key challenges that a secure blockchain empowered data sharing system should address. First, *data privacy* protection, especially during data storage and distribution, is known to have a critical impact on the uptake of such a system [7]. Therefore, concealing user privacy becomes the very first requirement in the full life cycle of data sharing. Most current decentralized data sharing applications [8, 9] utilize blockchain and smart contracts to accomplish the sharing of personal data. However, existing approaches either forfeit availability guarantees for private data [10] or fall back on semi-centralized solutions for key management [11, 12], thereby subduing data privacy to a single point of failure or compromise. These potential system security issues might lead to serious private user data leakage.

Second, *data integrity* verification is crucial to ensure the correctness of the data shared by blockchain. This challenge is two-fold: (i) how to securely and efficiently store data on the blockchain without incurring unaffordable storage and computational overheads; and (ii) how to prevent data from being tampered with when sharing data with high granularity. Prior systems [13, 14, 15] mainly adopt distributed ledger technology to store more general data and leverage smart contracts to control the data sharing process. Nevertheless, since such general data (such as text, documents, and images) is usually large, it is not scalable to store raw data directly on-chain. Even worse, most approaches are not capable of fine-grained data sharing to generate customized data records for various data consumers on-demand. The integrity of a fragmented data record cannot be verified, which leaves space for data fraud.

To tackle the issues mentioned above, we propose BlockShare, a blockchain empowered data sharing system, which simultaneously secures user privacy and data integrity. The system allows the data owner to dynamically generate data records for sharing with tailored privacy protection on an as-needed basis, where a trusted central server is unnecessary. Concretely, we make the following contributions in the paper.

- We introduce a novel BlockShare framework, together with a new *authenticated data structure* scheme that can efficiently verify any portion of a shared data record in a decentralized fashion.

- We develop a *zero-knowledge verification* scheme that enables a user to prove a dynamic condition without disclosing the specific data attribute, minimizing privacy loss.

- We implement BlockShare using readily-available infrastructural primitives. Experimental results show that our system achieves verifiable sharing of personal data in a privacy-preserving manner.

2 Related Work

In this section, we briefly review related studies and discuss relevant techniques.

Blockchain and Smart Contract. Blockchain has recently raised major attention in both industries and academia, owing to the boom of cryptocurrencies such as Bitcoin [16]. As a distributed and cryptographically hardened ledge, blockchain is recognized as a revolutionary technology for data-intensive applications. Many new blockchain techniques, such as Ethereum [17] and Hyperledger fabric [18], also introduce smart contracts to develop and execute customized programs on blockchain virtual machines.

In order to protect data privacy in blockchain systems, Kosba et al. [19] propose a framework for building privacy-preserving smart contracts. The Hawk compiler could automatically compile the smart contract to a cryptographic protocol between contractual parties and the blockchain to retain transactional privacy. Many researchers also devote their efforts to designing solutions to support various queries on the blockchain. These methods, including VQL [20] and vChain [21], can help to greatly improve the query efficiency with data integrity guarantee for blockchain systems. In addition, some approaches have been designed to efficiently store and manage blockchain data, such as SlimChain [22], BlockchainDB [23], FalconDB [24], and CALYPSO [25]. Due to the immutability and transparency features, we could build a secure and decentralized data sharing system based on blockchain.

Blockchain-based Data Sharing. Data sharing has received extensive research attention with the advent of the big data era. To keep data security and privacy in sharing, data is usually stored after encryption and further attached with certain access policies [26, 27]. In the traditional cloud setting, the centralized architecture is compromised with potential single-point failures or insider attacks [28]. Recently, many works have revealed some insights to show that the introduction of blockchain can significantly enhance system security in a decentralized way [29, 30]. Various applications have been developed for data sharing in healthcare, smart vehicles, IoT, and e-finance, using tokens as on-chain credentials for personal data [31, 32].

In addition to committing data records and recording sharing, blockchain can also provide functionalities such as access control, participant incentive, transaction auditing, and user identification for underlying data marketplaces, as well as providing services like dispute arbitration and data warranties for upper-layer data applications [33, 34]. Many systems have been developed in recent years by leveraging blockchain and smart contracts to enhance data interoperability and unlock the economic benefits of data assets [35].

Nevertheless, given the unique properties that data can be replicated and redistributed, existing studies fail to address data privacy issues. While many works adopt encryption of managed data records, a one-size-fits-all record often contains superfluous information, violating the “minimum necessary" principle and creating privacy risks.

3 System Overview

In this section, we present the overview of our proposed BlockShare system for privacy-preserving verifiable data sharing.

System Model. As illustrated in Figure 1, the system model includes four parties: (i) *data source*, (ii) *data owner*, (iii) *blockchain*, and (iv) *data consumer*. To protect data privacy and make the whole system scalable, a decentralized storage architecture is employed. Specifically, raw data from various sources (e.g., health records, financial transactions, and social contacts) are generated and stored off-chain by their data owners (i.e., users). Each data object is modeled as a tuple $o_i =< id, V_i >$, where id denotes the object's ID, and V_i defines a set of data attributes. Meanwhile, an *authenticated data structure* (ADS) will be constructed for each data record and stored on the blockchain. The on-chain ADSs are immutable, serving as notarizations of the raw data.

In order to unlock the value of data, the data owner can prepare tailored data records with different privacy protection levels and share them with relevant stakeholders. For example, a vaccine certificate (or “vaccine passport") with only the coarse-grained vaccination information is issued when entering restaurants, while a vaccine passport with detailed vaccination and personal information will be made for use at the customs. After receiving the shared data record, the data consumer (e.g., hospital, customs, or insurance company) verifies the

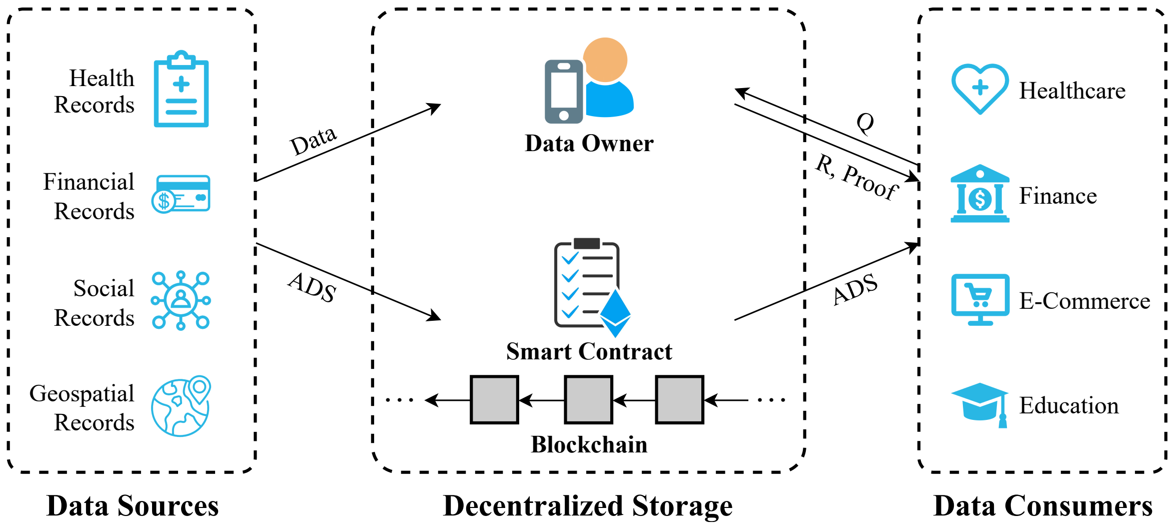


Figure 1: System Model of BlockShare

data integrity without violating the user's privacy. On the one hand, the integrity of the data record is checked with the corresponding ADS obtained from the blockchain such that no fraudulent data is transmitted. On the other hand, the data owner can generate a zero-knowledge proof for the data consumer such that personal information disclosure can be controlled with high granularity.

Threat Model. In our system, security threats come from two aspects: (i) the data owner is not fully trusted and might modify the requested data records intentionally or unintentionally; and (ii) the data consumer is curious and may attempt to infer some knowledge pertaining to its interests from the received data. To address these threats, we need to ensure data integrity without compromising users' privacy during data sharing. Concretely, we define three criteria of security constraints as follows:

- **Soundness.** All of the shared data records are not tampered with and are truly the results with respect to the data consumer's desires. This is a basic requirement for any data sharing service.
- **Completeness.** No valid data attributes are missing regarding the data sharing request. That is, the shared data record is complete during the process of fine-grained data sharing.
- **Zero-Knowledge Confidentiality.** Any information beyond the "minimum necessary" standard is protected. That is, the data owner can prove a data attribute satisfying a specific condition without disclosing the concrete attribute value or even its size.

In addition, we assume that there is no collusion between data sources, data owners, and data consumers. Regarding the blockchain, we also assume that the adversary cannot gain any advantage in attacking the consensus protocol and thus the execution integrity of the smart contract is guaranteed.

4 BlockShare Design

In this section, we present BlockShare, a privacy-preserving and verifiable data sharing initiative based on blockchain. We begin by introducing a dynamic data verification scheme to efficiently verify multiple versions of the shared data record with high granularity. Furthermore, we propose a zero-knowledge condition verification scheme to maximally protect the privacy of personal data under the umbrella of data integrity.

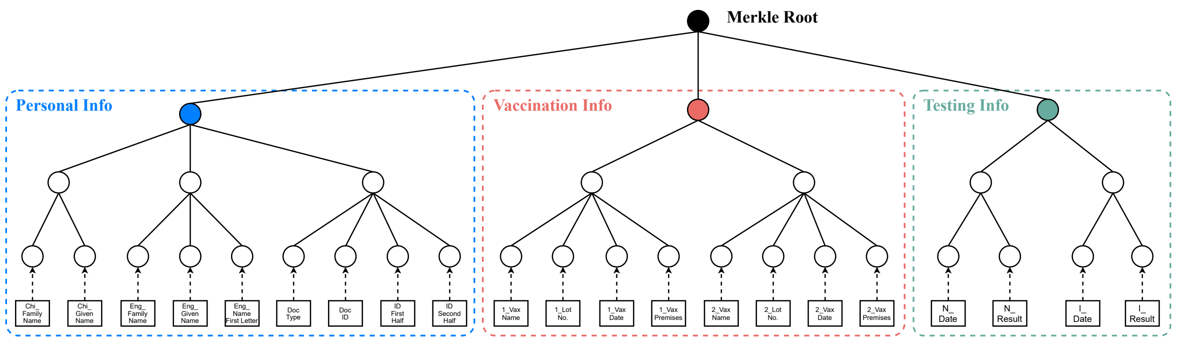


Figure 2: ADS Construction Example for the Vaccination Record

4.1 Dynamic Data Verification

ADS Construction and Storage. The verification for the shared data is built upon a blockchain-based decentralized storage model. Recall that in the proposed BlockShare architecture, original data records are generated from the trusted data sources and distributed to the relevant data owners for local storage. At the same time, an authenticated data structure (ADS) is constructed for each data record and kept on-chain as a proof of data integrity. Concretely, we utilize Merkle Hash Tree (MHT) [36] to derive a Merkle Root (serving as the ADS) to support data integrity verification.

We use Figure 2 as an example to show the ADS construction of the COVID-19 vaccination record. In view of the surge of COVID-19 cases in the community, many countries have developed an immunization information system to manage vaccination records for pandemic control. Generally, the vaccination record consists of three parts, including personal information, vaccination information, and testing information (e.g., nucleic acid test and serum IgG antibody test). In order to prove the virus immunity, citizens are required to show their vaccine passports when entering some premises. During this process, one challenge is that the user might forge or modify historical vaccination records to generate a fake vaccine passport for some purposes (e.g., avoiding quarantine, extending the expiration date of vaccination, or protecting privacy).

To remedy this problem, we build a semantic-oriented MHT to accurately model each part of the vaccination record. Specifically, to enhance data privacy and prevent rainbow attacks, a unique random number will be associated with each attribute in the data record to generate a "salted" record. As such, each leaf node contains a hash value h_{leaf} computed as $h_{leaf} = H(H(v)\|H(nonce))$, where $H(\cdot)$ is a cryptographic hash function, v is the value of attribute, $nonce$ is an attribute-specific random number, and "$\|$" denotes the concatenation operation. The hash value stored on each internal node is recursively calculated using its all child nodes. Finally, as illustrated in Figure 2, the Merkle Root is generated on the top of three independent sub-trees.

In addition, as the write operation on the blockchain is very extensive, the frequent on-chain storage of the whole MHT would incur massive gas consumption. At the same time, we observe that only the Merkle Root is needed from the blockchain during the data authentication. Therefore, an optimal method is to suppress all nodes in the MHT and only materialize the Merkle Root on the blockchain.

Information Masking and Verification. In practical data sharing, the diversity of privacy protection levels exists in many scenarios. The data owner might prefer to issue a tailored data record by sharing a portion of data attributes according to the dynamic requirements. For instance, the user can only indicate whether he/she has been vaccinated during a period without disclosing the entire ID number, vaccine name, vaccine lot number, and vaccination premises. It would be costly, if not impossible, to generate and store a specialized ADS for each version of the data record. As such, we propose a new "one-ADS-for-multiple-versions" paradigm in which the data source only needs to generate a single ADS for one data record and store the ADS on the blockchain.

More specifically, for a given attribute in a data record, many sub-attributes will be appended in the MHT as leaf nodes to support different privacy constraints. For example, as shown in Figure 2, two more sub-attributes (i.e., the first half of ID and the second half of ID) are derived from the ID number and added to the MHT. With this appended MHT, the data owner has multiple options to disclose the concrete content in the data record, while the Merkle Root keeps unique for verification. Furthermore, the data owner can mask unnecessary attributes into hash values to generate a specific version of the data record. For the shared attributes in the data record, the data owner generates a verification object (i.e., the Merkle Path) based on the unique ADS. Then, with the verification object and the on-chain ADS, the data consumer can verify the integrity of the received data record (e.g., the tailored vaccine passport).

4.2 Zero-Knowledge Privacy Protection

In the previous information masking scheme, although the unnecessary attributes can be masked to protect privacy, the shared data attributes might still contain more personal information than what is needed. For example, to show that the vaccination has been completed for a period (e.g., 14 days), the campus visitor has to disclose the concrete vaccination date. Thus, to further enhance privacy, we propose a general privacy protection scheme for the shared data attributes based on non-interactive zero-knowledge (NIZK) proof technology [37]. This scheme enables the data owner to prove a dynamic condition without revealing the specific value, limiting the disclosure of personal data to the minimum necessary while guaranteeing data integrity.

In this scheme, a constraint will be first defined for a specific attribute. Usually, this constraint represents the data information needed by the data consumer. Then, the data owner will generate a proof of satisfiability and pass it without the attribute value to the data consumer for verification. With this scheme, the data owner can indicate only a qualified period instead of a detailed date, an area instead of a concrete vaccination premise, or an age group instead of exact age. Given a constraint of an attribute, we present the formal definition of the zero-knowledge condition verification as follows:

Definition 1 (Zero-Knowledge Condition Verification): For the input attribute value v, a range R, a random number $nonce$ associated with the attribute, the hash value h of the attribute, ADS $mRoot$ of the data record o, and global parameters $G, H \in E(F_q)$, this scheme can prove to a verifier that the prover knows an assignment to v such that $hash(v, nonce) = h \ \wedge \ v \in R$, without revealing v. It consists of the following algorithms:

- $\{\mathbf{G} = (G_1, \cdots, G_n), \mathbf{H} = (H_1, \cdots, H_n), G, H\} \leftarrow \mathsf{Setup}(1^\lambda)$: Call the parameter generation algorithm Setup to generate public security parameters for zero-knowledge proof. On input a security parameter 1^λ, it outputs public parameters $\{\mathbf{G}, \mathbf{H}, G, H\}$ acting as implicit input for other functions.

- $\pi \leftarrow \mathsf{zkProofGen}(v, nonce, R, h)$: The prover calls the zero-knowledge proof generation algorithm $\mathsf{zkProofGen}$ to generate a proof for the shared attribute with zero knowledge. On input an attribute value v, the random number $nonce$ associated with the attribute, a range R for the attribute, and the hash value h of the corresponding leaf node, it outputs a zero-knowledge proof π.

- $mPath \leftarrow \mathsf{mkProofGen}(h, mTree)$: The prover calls the Merkle proof generation algorithm $\mathsf{mkProofGen}$ to generate a proof for the leaf node with the Merkle tree. On input the hash value h of the corresponding leaf node, and a Merkle tree $mTree$, it outputs a Merkle path $mPath$.

- $\{0, 1\} \leftarrow \mathsf{zkProofVer}(\pi, R, h)$: The verifier calls the zero-knowledge proof verification algorithm $\mathsf{zkProofVer}$ to verify the received zero-knowledge proof. On input a zero-knowledge proof π, a range R for the attribute, and the hash value h of the corresponding leaf node, it outputs 1 if the verification is valid.

- $\{0, 1\} \leftarrow \mathsf{mkProofVer}(h, mPath, mRoot)$: The verifier calls the Merkle tree authentication algorithm $\mathsf{mkProofVer}$ to verify the received Merkle proof. On input the hash value h of the corresponding leaf node, the received Merkle path $mPath$, and the public Merkle root $mRoot$, it outputs 1 if the verification is valid.

To summarize, owing to the dynamic data verification and zero-knowledge proof design, the data sharing process is verifiable while the data owner has full control over the information disclosure through the following features:

- **Multi-version support.** The data owner can dynamically generate multiple versions of a data record to meet different application needs.
- **Data integrity support.** The data owner can support integrity verification for multiple versions of a data record by using a single ADS.
- **Privacy protection support.** The data owner can quantify the information disclosure with different privacy protection levels using two predominant privacy metrics: (i) *Suppression*: the data owner can mask partial information of a data record with high granularity, while an adversary is unable to infer the preimage of the masked information. (ii) *Generalization*: a shared data attribute can be generalized to an arbitrary coarse granularity, for example, “more than 2 weeks" instead of a concrete date and “in the 20-30 age group" instead of exact age.

5 Implementations and Evaluation

In this section, we present the implementation of BlockShare and evaluate its performance in detail.

5.1 Experimental Implementation

We have designed and implemented a prototype of BlockShare, including the data source, data owner, and data consumer in JavaScript and Python, and the blockchain in Solidity. Arithmetic circuits for NIZK proofs, along with the core logic of circuit compilation, universal setup, proof generation and verification are implemented using Circom and Snarkjs. We perform the evaluation on a desktop computer with a 3.6 GHz Intel Xeon processor, 64 GB RAM, and 1 TB SSD. All experiments are conducted based on a synthetic dataset.

5.2 Performance Evaluation

1) ADS Construction: We first perform an evaluation on the time cost of ADS generation at three length settings of hash functions based on the data records of 1K, 2K, 4K, 8K, and 16K entries, respectively. As we can see from Figure 3, the time cost of ADS construction raises linearly in all cases as the amount increases. It only takes roughly 74ms to construct ADSs for 1K records and 0.8s for all of 16K records, when the hash value has 128 bits. In addition, although calculating a longer hash value usually needs more time, constructing ADSs with 512-bit length for all of 1K records can still be finished in 0.2s. When the amount of records increases to 16K, only 3s will be needed to finish the ADS construction process. These benefits come from the high efficiency of a hash function.

2) Proof Performance: We further evaluate the zero-knowledge verification scheme with three metrics: (i) NIZK proof size, (ii) proof generation time, and (iii) proof verification time. We use λ to indicate how many conditions need to be proved for corresponding attributes in a data record. For each metric, three comparison experiments are conducted, with different values of λ, including 1, 2, and 4, respectively.

Figure 4 shows the total size of NIZK proofs when the number of data records is varied from 100 to 1.6K. Our proposed scheme needs 80KB of storage space, if there are 100 data records and each record has one condition to be proved. Moreover, if we increase the dataset volume to 1.6K and each record has four conditions to be proved, the total storage cost for the proofs is limited to 2.7MB. It can be concluded that the proof size keeps succinct ($<$ 3 MB) in BlockShare.

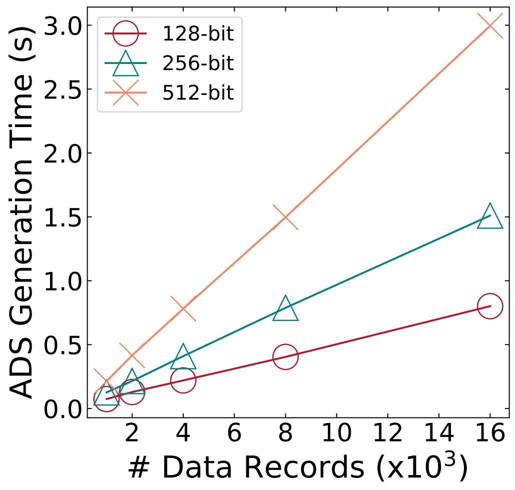


Figure 3: Time cost of ADS generation

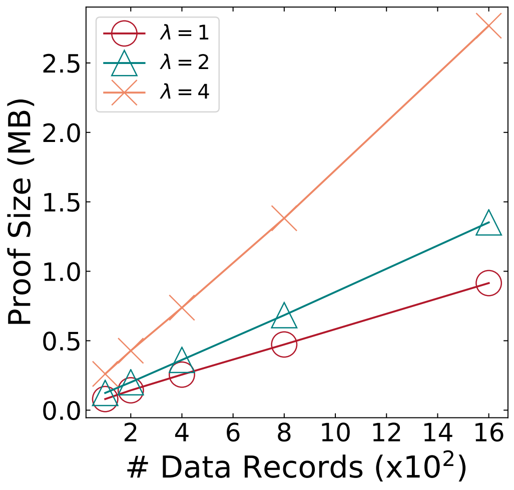


Figure 4: Storage cost of NIZK proof

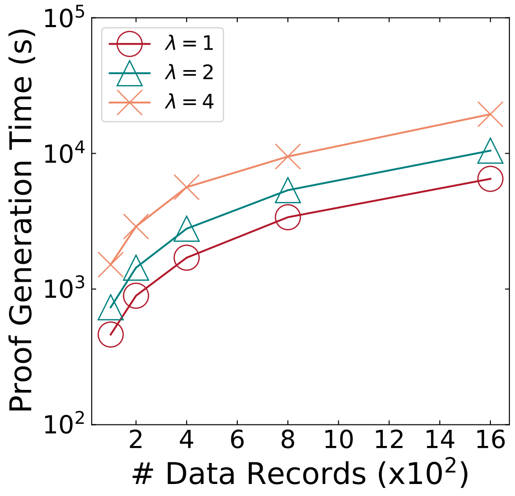


Figure 5: Time cost of proof generation

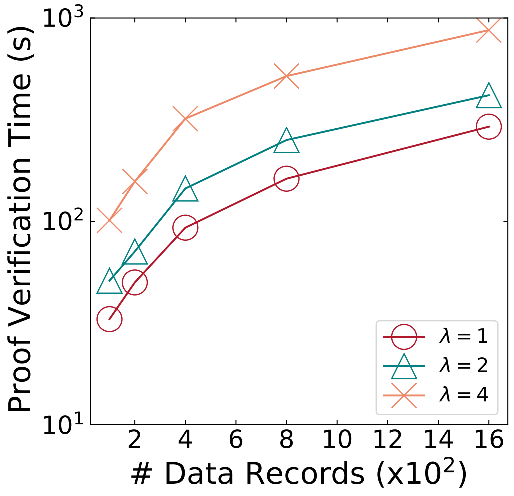


Figure 6: Time cost of proof verification

Figure 5 illustrates the total proof generation cost in terms of prover CPU time with a varied amount of data records. For a dataset consisting of 100 records, it takes about 7 minutes to complete the proof generation for all records when each record has an attribute to be protected with zero-knowledge privacy. We can find that the increase in proof generation time is noteworthy with the raising of dataset size and constraint volume. The reason is that our implementation of zero-knowledge condition verification based on arithmetic circuits trades the proof generation time for a succinct proof size to save communication bandwidth. To ease this issue, a potential method is to pre-generate and re-use the proof for a specific condition.

In Figure 6, we plot the total proof verification cost in terms of verifier CPU time with regard to the number of data records. The proof verification process could be completed in only 33s when there are 100 data records and one condition for each record. If we increase the number of records to 1.6K, 5 minutes will be spent to verify all of the data records. Compared with the proof generation process, we can see that our zero-knowledge proof verification is more efficient. High verification efficiency could be an advantage to encourage more data consumers to join the data sharing system.

3) Gas Consumption: The smart contract in BlockShare is deployed on the Goerli testnet of Ethereum, enforcing the storage and request of ADS on the blockchain. As shown in Table 1, the deployment is a one-time effort, costing approximately 530,548 gas. Gas spent for method invocation is also evaluated. Since ADSs only record and manage the metadata of data records, the gas spent for the ADS storage function is quite economical. It only costs 69,923 gas per time for the 256-bit ADS regardless of the size of a data record. Regarding the gas of the on-chain ADS request for data verification, it costs around 29,402 gas, i.e., approximately $0.06 when ETH is at the price of $2,000. The gas appears practically low because reading a value on the blockchain is very cheap in gas.

Table 1: Gas consumption of smart contract

Operation	Gas Consumed
Contract Deployment	530,548
ADS Storage	69,923
ADS Request	29,402

6 Conclusion

In this paper, we propose BlockShare, a privacy-preserving verifiable data sharing system based on blockchain. We first design a novel blockchain-based decentralized architecture, together with a new authenticated data structure scheme to efficiently verify any portion of a shared data record. Then, we develop a zero-knowledge verification scheme allowing a user to prove a dynamic condition without disclosing the specific data attribute, maximally protecting data privacy. We implement BlockShare and experimental results show that our system achieves verifiable sharing of personal data in a privacy-preserving manner.

Acknowledgement

This work was supported by Hong Kong RGC Grants C2004-21GF, 12202221, and 12201520.

References

[1] European Parliament and Council of the European Union, “General Data Protection Regulation (GDPR),” *Official Journal of the European Union (OJ) L119*, pp. 1–88, 2016.

[2] A. Schwartz and C. Cohn, ““Information Fiduciaries" Must Protect Your Data Privacy,” *Electronic Frontier Foundation*, 2018.

[3] Z. Peng, J. Xu, X. Chu, S. Gao, Y. Yao, R. Gu, and Y. Tang, “VFChain: enabling verifiable and auditable federated learning via blockchain systems,” *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 173–186, 2021.

[4] Z. Peng, C. Xu, H. Wang, J. Huang, J. Xu, and X. Chu, “P^2B-Trace: Privacy-preserving blockchain-based contact tracing to combat pandemics,” in *Proc. of ACM SIGMOD International Conference on Management of Data*, 2021.

[5] H. Wang, C. Xu, C. Zhang, J. Xu, Z. Peng, and J. Pei, “vChain+: Optimizing verifiable blockchain boolean range queries,” in *Proc. of IEEE International Conference on Data Engineering (ICDE)*, 2022.

[6] H. Subramanian, “Decentralized blockchain-based electronic marketplaces,” *Communications of the ACM*, vol. 61, no. 1, pp. 78–84, 2017.

[7] T. Salman, M. Zolanvari, A. Erbad, R. Jain, and M. Samaka, “Security services using blockchains: A state of the art survey,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 858–880, 2018.

[8] B. Shen, J. Guo, and Y. Yang, “MedChain: Efficient healthcare data sharing via blockchain,” *Applied Sciences*, vol. 9, no. 6, pp. 1–23, 2019.

[9] J. Kang, R. Yu, X. Huang, M. Wu, S. Maharjan, S. Xie, and Y. Zhang, “Blockchain for secure and efficient data sharing in vehicular edge computing and networks,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4660–4670, 2018.

[10] X. Liang, J. Zhao, S. Shetty, J. Liu, and D. Li, “Integrating blockchain for data sharing and collaboration in mobile healthcare applications,” in *Proc. of IEEE international symposium on personal, indoor, and mobile radio communications*, pp. 1–5, 2017.

[11] X. Cheng, F. Chen, D. Xie, H. Sun, and C. Huang, “Design of a secure medical data sharing scheme based on blockchain,” *Journal of medical systems*, vol. 44, no. 2, pp. 1–11, 2020.

[12] G. Zyskind, O. Nathan, *et al.*, “Decentralizing privacy: Using blockchain to protect personal data,” in *Proc. of IEEE Security and Privacy Workshops*, 2015.

[13] R. Matzutt, J. Hiller, M. Henze, J. H. Ziegeldorf, D. Müllmann, O. Hohlfeld, and K. Wehrle, “A quantitative analysis of the impact of arbitrary blockchain content on bitcoin,” in *Proc. of Financial Cryptography and Data Security (FC)*, 2018.

[14] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, “Medblock: Efficient and secure medical data sharing via blockchain,” *Journal of medical systems*, vol. 42, no. 8, pp. 1–11, 2018.

[15] Z. Su, Y. Wang, Q. Xu, and N. Zhang, “LVBS: Lightweight vehicular blockchain for secure data sharing in disaster rescue,” *IEEE Transactions on Dependable and Secure Computing*, 2020.

[16] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Decentralized Business Review*, p. 21260, 2008.

[17] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[18] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, *et al.*, “Hyperledger fabric: a distributed operating system for permissioned blockchains,” in *Proc. of ACM European Conference on Computer Systems (EuroSys)*, 2018.

[19] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,” in *Proc. of IEEE symposium on security and privacy (S&P)*, 2016.

[20] H. Wu, Z. Peng, S. Guo, Y. Yang, and B. Xiao, “VQL: Efficient and verifiable cloud query services for blockchain systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1393–1406, 2021.

[21] C. Xu, C. Zhang, and J. Xu, “vChain: Enabling verifiable boolean range queries over blockchain databases,” in *Proc. of ACM SIGMOD International Conference on Management of Data*, 2019.

[22] C. Xu, C. Zhang, J. Xu, and J. Pei, “SlimChain: scaling blockchain transactions through off-chain storage and parallel processing,” *Proc. of the VLDB Endowment*, vol. 14, no. 11, pp. 2314–2326, 2021.

[23] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, and R. Ramamurthy, “BlockchainDB: A shared database on blockchains,” *Proc. of the VLDB Endowment*, vol. 12, no. 11, pp. 1597–1609, 2019.

[24] 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*, 2020.

[25] E. Kokoris-Kogias, E. C. Alp, L. Gasser, P. Jovanovic, E. Syta, and B. Ford, “CALYPSO: Private data management for decentralized ledgers,” *Proc. of VLDB Endowment*, vol. 14, no. 4, pp. 586–599, 2020.

[26] X. Liu, Y. Zhang, B. Wang, and J. Yan, “Mona: Secure multi-owner data sharing for dynamic groups in the cloud,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1182–1191, 2012.

[27] T. F.-M. Pasquier, J. Singh, D. Eyers, and J. Bacon, “CamFlow: Managed data-sharing for cloud services,” *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 472–484, 2015.

[28] C. Wang, S. Wang, X. Cheng, Y. He, K. Xiao, and S. Fan, “A privacy and efficiency-oriented data sharing mechanism for iots,” *IEEE Transactions on Big Data*, 2022.

[29] B.-K. Zheng, L.-H. Zhu, M. Shen, F. Gao, C. Zhang, Y.-D. Li, and J. Yang, “Scalable and privacy-preserving data sharing based on blockchain,” *Journal of Computer Science and Technology*, vol. 33, no. 3, pp. 557–567, 2018.

[30] S. Qi, Y. Lu, Y. Zheng, Y. Li, and X. Chen, “CPDS: enabling compressed and private data sharing for industrial internet of things over blockchain,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2376–2387, 2020.

[31] K. Yu, L. Tan, M. Aloqaily, H. Yang, and Y. Jararweh, “Blockchain-enhanced data sharing with traceable and direct revocation in IIoT,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7669–7678, 2021.

[32] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, “MeDShare: Trust-less medical data sharing among cloud service providers via blockchain,” *IEEE Access*, vol. 5, pp. 14757–14767, 2017.

[33] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, “Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4298–4311, 2020.

[34] M. Shen, J. Duan, L. Zhu, J. Zhang, X. Du, and M. Guizani, “Blockchain-based incentives for secure and collaborative data sharing in multiple clouds,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1229–1241, 2020.

[35] J. Liu, X. Li, L. Ye, H. Zhang, X. Du, and M. Guizani, “BPDS: A blockchain based privacy-preserving data sharing for electronic medical records,” in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, 2018.

[36] R. C. Merkle, “A certified digital signature,” in *Proc. of Conference on the Theory and Application of Cryptology (CRYPTO)*, 1989.

[37] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, “Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting,” in *Proc. of International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2016.

Power-of-Collaboration: A Sustainable Resilient Ledger Built Democratically

Junchao Chen, Suyash Gupta[†], Sajjad Rahnama, Mohammad Sadoghi
Moka Blox LLC
Exploratory Systems Lab
University of California, Davis
[†]University of California, Berkeley

Abstract

*The recent surge in blockchain applications has accelerated the research in designing efficient decentralized currencies. Building a decentralized economy on the traditional byzantine fault-tolerant (*BFT*) protocols or the Proof-of-Work (*POW*) consensus protocol is inadequate as the immutability of the ledger created by the former is at the mercy of the long-term safe-keeping of private keys of participants, while the latter yields an extremely inefficient and environmentally unsustainable consensus. To ameliorate this situation, we envision the design of our* HYBRIDCHAIN *architecture, which offers the best of both worlds. Our* HYBRIDCHAIN *design runs a traditional* BFT *protocol to commit client transactions and employs our novel Power-of-Collaboration (*POC*) protocol to notarize the* BFT *chain. Unlike* POW*, our* POC *protocol advocates for participants to work together collaboratively instead of competing (often selfishly), which results in a safe, high-throughput, and resource-efficient consensus design.*

1 Introduction

The past decade has observed a surge in the design and deployment of decentralized systems. A key reason for this surge is the growing desire in the society to have self-governing democratic financial systems that are not under the control of a privileged set of entities. A central control often translates to a forced trust model with limited provision to support transparency and accountability. The adoption of Blockchain, for example, is a by-product of the ability to break away from the forced-central control in a trust-worthy fashion [8]. The emerging blockchain platforms facilitate a reliable execution of any digital contracts (i.e., transactions) in a decentralized manner despite the existence of malicious actors. At the core of any blockchain platform is a Byzantine fault-tolerant (BFT) consensus protocol and a tamper-proof replicated ledger [2, 8, 24]. The BFT protocol helps to achieve *consensus* on the order of incoming client requests among all the replicas, while the ledger logs this agreement.

Traditional BFT protocols expect a *permissioned* system where the identities of all the replicas (i.e., participants) are known prior to any consensus as they rely on having a verifiable voting right in a democratic setting. These protocols rely on a *communication-oriented* consensus model, where all the participants exchange endorsements across multiple rounds before they can reach a decision [1, 3, 4, 7, 9, 10, 11, 17, 19, 41, 45]. In

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

these protocols, a system of $\mathbf{n}$ replicas can reach a common decision if at most $\mathbf{f}$ of them are malicious, such that $\mathbf{n} \geq 3\mathbf{f} + 1$. The $\mathbf{n}$ parties are said to reach a decision when at least a majority of honest parties agrees to that decision. This decision is logged by requiring all the agreeing parties to *sign* the decision. Hence, the reached decision is considered *tamper-proof* because it has support of a majority of honest participants.

Despite being around for more than two decades, traditional BFT protocols did not see any major practical applications until the introduction of blockchain technology. We attribute *two* key factors for this lack of adoption. (i) To ensure that the malicious participants do not spawn multiple identities, these BFT protocols need an authority (i.e., *a forced trust gateway*) to verify and register every participant to verify every vote [6]; some participants may find this intrusive if they do not want to reveal their personal information. (ii) To overwrite the ledger, malicious participants just require access to the private keys of honest participants. In a sense, the proof of the validity of the ledger is not self-contained, and it operates on the assumption that the private-keys are kept safe externally indefinitely.

To resolve these challenges, initial blockchain platforms such as Bitcoin [16] and Ethereum [22] offer a *permissionless* model of consensus. These systems employ the *Proof-of-Work* (PoW) protocol [16, 22], which follows a *computation-oriented* consensus model and requires all the participants to compete with each other and try to solve a complex puzzle. Whichever participant solves the puzzle first gets to add a new entry (*block*) to the ledger. As a result, PoW protocol eliminates the three challenges seen by traditional BFT protocols. (i) Malicious participants can spawn multiple identities, but what actually matters is the available compute power. (ii) Each block includes the hash of the previous block; overwriting the ledger requires recomputing all the blocks making it computationally infeasible. (iii) Since reaching the consensus is based on presenting the proof of work that is embedded on the ledger (i.e., self-contained), there is no longer any need for external safe-keeping of private keys to sign endorsements.

These properties offered by PoW protocol help blockchain platforms to design a *decentralized economy*, where any person can participate in the consensus process, and the economy has a self-generating currency to monetize its participants. Monetizing the participants is necessary as the PoW protocol expects the participants to spend their resources to solve a complex puzzle. Clients of the Bitcoin platform, create transactions that exchange Bitcoins and send them to the participants (miners) in the PoW protocol. These miners check if the transaction is valid; the client has sufficient Bitcoins to transfer. If the transaction is valid, they run PoW protocol to include this transaction in the ledger. The winning miner of PoW gets a portion of the client's Bitcoin as *fees*, while the mining process (PoW) mints new tokens to fund the economy. This new token is transferred to the winning miner's account and is recorded as a transaction in the block.

The key challenge with platforms like Bitcoin is their *practicality*. These platforms have abysmally low throughput in the order of 10 transactions per second in part due to inadequate choice of small block sizes. Furthermore, as more miners join the network, the complexity of the puzzle has to be increased. For example, the complexity of the current Bitcoin puzzle is so high that the miners work in large groups to have any positive probability of creating the next winning block [8]. Moreover, as miners are competing with each other, it leads to massive wastage of computational resources (energy) as only the winning miner's efforts are recorded and rewarded. This results in an unsustainable ecosystem [5, 21].

We observe these challenges in the designs of existing BFT protocols and blockchain platforms and envision a HYBRIDCHAIN system that learns from these models and eliminates their key challenges. Essentially, we aim to establish a new research agenda; a new field of hybrid consensus protocols that depart from competitive consensus to a collaborative consensus that is both resilient and sustainable. Our HYBRIDCHAIN architecture takes a step in this direction by running two consensuses on each client transaction while ensuring there is no increase in the latency observed by the client. Each client request is first ordered through a state-of-the-art BFT consensus protocol (*commitment*), subsequently, this ordered request is engraved into the ledger through the PoW-style consensus (*settlement*). Specifically, HYBRIDCHAIN causes no increase in commitment latency while improving the settlement latency observed by existing protocols. Ordering the client transaction through a BFT consensus protocol first allows our HYBRIDCHAIN system to guarantee the following benefits: (i) clients receive

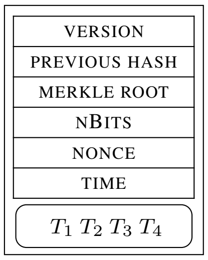


Figure 1: A Bitcoin-style block containing a header and a body with transactions (T_1, T_2, T_3 and T_4).

low-latency responses, and (ii) PoW participants no longer need to compete, resulting in a high-throughput sustainable chain. As a result, instead of employing the PoW for consensus, we design a novel protocol that allows miners to collaborate. We refer to this paradigm as *Power-of-Collaboration* (PoC).

Our PoC protocol splits the complex puzzle into disjoint slices and requires each miner to work on a distinct slice. This slice distribution significantly reduces the resource wastage and provides each honest miner with a reward for each new transaction added to the ledger. As each ledger entry is added collaboratively, any malicious entity that wishes to overwrite the ledger needs to match the computational power of all the existing miners making it practically impossible. These features of our HYBRIDCHAIN system make it lucrative; its design is the bedrock for a secure and efficient decentralized economy.

2 Preliminaries

We adopt the standard communication and failure model adopted by most BFT protocols [7, 3, 10]. We consider a service $\mathcal{S}$ of the form $\mathcal{S} = \{\mathcal{R}, \mathcal{M}, \mathcal{C}\}$. The set $\mathcal{R}$ consists of $\mathbf{n}_{\mathcal{R}}$ replicas of which at most $\mathbf{f}_{\mathcal{R}}$ can behave arbitrarily. The remaining $\mathbf{n}_{\mathcal{R}} - \mathbf{f}_{\mathcal{R}}$ are honest: they will follow the protocol and remain live. Similarly, the set $\mathcal{M}$ consists of $\mathbf{n}_{\mathcal{M}}$ miners of which at most $\mathbf{f}_{\mathcal{M}}$ can act maliciously. We assign each *anonymous miner* and replica a unique identifier, which can be obtained by a call to the function $\mathsf{id}()$. The range of these identifiers are $[0, |\mathcal{R}|]$ for replicas and $[0, |\mathcal{M}|]$ for miners. We further consider the existence of a finite set of clients $\mathcal{C}$ of which arbitrarily many can be malicious.

We assume *authenticated communication*: replicas employ standard cryptographic primitives such as MAC and digital signatures (DS) to sign messages. We denote a message m signed by a replica R using DS as $\langle m \rangle_{\mathbf{R}}$. We permit malicious replicas to impersonate each other but no entity can impersonate an honest replica. We assume that like Bitcoin miners are identifier through their public-keys (i.e., anonymous), while the identities of replicas is known before consensus (i.e., known verified identity). We employ a *collision-resistant* hash function $\mathtt{hash}(\cdot)$ to map an arbitrary value v to a constant-sized digest $\mathtt{hash}(v)$ [15]. Each replica only accepts a message if it is ***well-formed***.

We adopt the same partial synchrony model adopted in most consensus systems: safety is guaranteed in an asynchronous environment where messages can get lost, delayed, or duplicated. Liveness, however, is only guaranteed during the periods of synchrony [3, 7, 9, 10, 45].

Safety. If two honest replicas R1 and R2 order a transaction T at sequence numbers k and k', then $k = k'$.

Liveness. If a client sends a transaction T, then it will eventually receive a response for T.

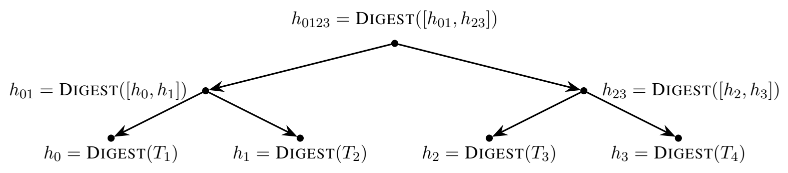

$h_{0123} = \text{DIGEST}([h_{01}, h_{23}])$
$h_{01} = \text{DIGEST}([h_0, h_1])$
$h_{23} = \text{DIGEST}([h_2, h_3])$
$h_0 = \text{DIGEST}(T_1)$
$h_1 = \text{DIGEST}(T_2)$
$h_2 = \text{DIGEST}(T_3)$
$h_3 = \text{DIGEST}(T_4)$

Figure 2: A Merkle tree over four transactions (T_1, T_2, T_3 and T_4) stored at the leaf nodes of the tree.

3 Background: Proof-of-Work Consensus

The *settlement* phase in our HYBRIDCHAIN architecture makes use of our novel POC protocol. The intuition behind POC's design stems from the POW protocol that helps create an immutable *chain of blocks*. The term immutable refers to the fact that each block appended to the chain requires miners to spend their resources. As a result, if an adversary attempts to over-write (or rollback) a part of the chain, it needs to create an alternate chain of all the desired blocks. However, to force all the honest miners to switch to this alternate chain, the adversary needs to compute this chain of blocks at a much faster rate than the original chain. This implies that the adversary needs to have much greater power than the honest miners. Prior works have illustrated that such an attack is hard to realize [12].

Prior to running the POW protocol, each miner $\text{M} \in \mathcal{M}$ needs to create a *block* of transactions. Although each miner M has access to the certificate $\mathfrak{C}$, it needs to arrange the contents of this certificate in the format of a block. To explain the format of a block, we follow the popular blockchain platform, Bitcoin, where each block includes a header and body (refer to Figure 1) [8]. The header includes: (i) *version* for this block, (ii) *hash* of the previous block, (iii) *merkle root* of all transactions, (iv) *nBits*, which determines the difficulty of the puzzle, (v) *nonce*, the solution for puzzle, and (vi) *time* at which block is created once the nonce is found.

Computing Merkle root of all the transactions is trivial (refer to Figure 2) and requires a miner M to compute a pairwise hash from the leaf to the root. This Merkle root helps to verify if a transaction was included to create the Merkle tree. However, the main challenge for a miner is to determine the nonce. In existing POW-based platforms, to solve the complex puzzle, each miner needs to calculate the *hash of the block* such that it contains a specific number of leading zero bits. For this purpose, the miners have to find a *nonce* value that yields the specific hash. This essentially makes the POW protocol like a *race* where all the miners are competing against each other to find the nonce. Whichever miner finds a valid nonce first, it gets the chance to propose the next block to be added to the chain. Hence, coming up with the correct number of leading zero bits in the hash is important as it sets the difficulty of the puzzle which simplicity controls the average time taken for each winning miner to propose a block. Once a miner finds the valid nonce, it can fill all the entries in the header, and it broadcasts the new block to all the other miners.

Notice that the ledger is essentially a chain of blocks (refer to Figure 3). In POW, it is possible that multiple miners propose the next block with valid nonces at approximately the same period of time. In such a case, the protocol states that each miner would only accept the first block it receives. This could lead to temporary branches or *forks*, all of which have the same previous hash. However, this condition resolves as time goes by because the protocol also expects the honest miners to stick to the *longest chain*—the one with the largest number of blocks. Eventually, all the shorter forks are discarded, and only the longest chain survives.

Incentives. Considering discarded forks and resources spent in the search for a valid nonce, what motivates a miner to participate in POW consensus? The answer is incentives. POW-based systems like Bitcoin reward the winning miner of the block present in the longest chain. These rewards help to offset the mining costs and maintain a sufficient number of honest miners. This requires a few more entries in the block header, which provide information such as the miner's account address and the reward amount.

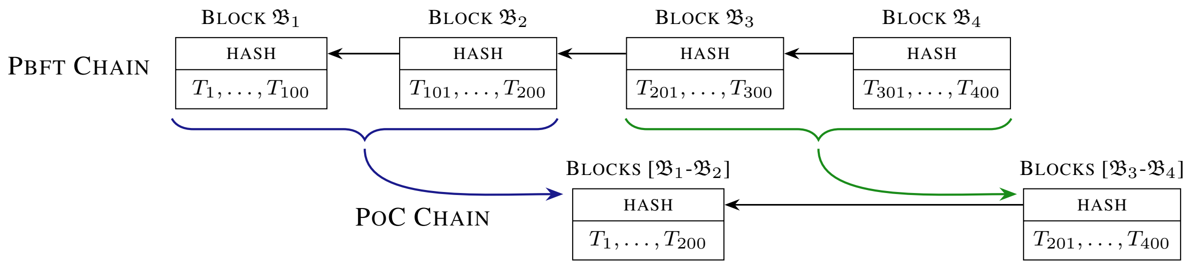

BLOCK $\mathfrak{B}_1$
BLOCK $\mathfrak{B}_2$
BLOCK $\mathfrak{B}_3$
BLOCK $\mathfrak{B}_4$
PBFT CHAIN
HASH
$T_1, \ldots, T_{100}$
$T_{101}, \ldots, T_{200}$
$T_{201}, \ldots, T_{300}$
$T_{301}, \ldots, T_{400}$
BLOCKS [$\mathfrak{B}_1$-$\mathfrak{B}_2$]
BLOCKS [$\mathfrak{B}_3$-$\mathfrak{B}_4$]
POC CHAIN
$T_1, \ldots, T_{200}$
$T_{201}, \ldots, T_{400}$

Figure 3: A schematic representation of a blockchain or ledger in the HYBRIDCHAIN architecture that consists of PBFT chain (i.e., *layer 1*) and POC chain (i.e., *layer 2*) that shows committed and settled transactions $T_1, \ldots, T_{400}$ on the PBFT and POC chains, respectively. The i^{th} block holds a *hash value* hash_{i-1} that identifies the preceding block and Block $\mathfrak{B}_1$ is the genesis block that does not have the preceding block.

3.1 POW Challenges

The key issue with running the POW consensus is that it leads to massive waste in energy and efforts: (1) Forks of the longest chain are subsequently discarded, which is a loss of resources for some miners who did find a valid nonce but did not receive any rewards. (2) Rational miners may acquire more resources to improve their chances of proposing the next block, but this leads to increasing the difficulty of hash computation to ensure fairness. (3) As the number of miners increases, the frequency of a single miner winning rewards decreases. (4) Several miners may work together in groups to find the valid nonce to increase their probability of winning rewards. This behavior significantly decreases the probability for a lone miner to propose the next block. (5) Malicious miners may attempt to perform attacks like selfish mining and double-spending, which can rollback client transactions and invalidate the rewards earned by honest miners [8].

These issues are so prevalent in blockchain systems like Bitcoin that, at present, almost every miner is trying to join some existing group. In these groups, miners *pool* their resources to find a valid nonce and propose the next block [18]. Every pool has its own participation rules and distributes rewards according to its policies. Despite this, the difficulty of hash computation is periodically increased or decreased in accordance with the average time to find a nonce by a pool of miners.

To address these challenges, in this paper, we aim to initiate a new avenue of research centered around hybrid consensus and collaborative mining. In particular, as a first step, we propose our novel *Power-of-Collaboration* (POC) protocol that re-imagines the POW consensus.

4 HYBRIDCHAIN Architecture

Our HYBRIDCHAIN runs two distinct consensus protocols in parallel. Specifically, it requires the replicas in set $\mathcal{R}$ to commit each client transaction through a BFT consensus protocol (commitment phase), following which the miners in set $\mathcal{M}$ run our POC consensus (settlement protocol). As all the BFT protocols follow the consensus dictated by PBFT [3], we use PBFT as the representative protocol for the ensuing discussions. *To summarize:* each client sends its request to the replicas running the PBFT protocol. Once these replicas commit this request, they forward it to the miners. These miners collaboratively run the POC consensus protocol, post which they add a block to the ledger. Next, we discuss each of these steps in detail.

4.1 Client Request and Transaction Ordering

PBFT follows the primary-backup model where one replica is designated as the *primary* while other replicas act as backups. Each consensus is led by the primary replica of the current *view*. In the case the primary is malicious, *view-change* takes place to replace the primary. We use Figure 4 to illustrate the three phases of PBFT.

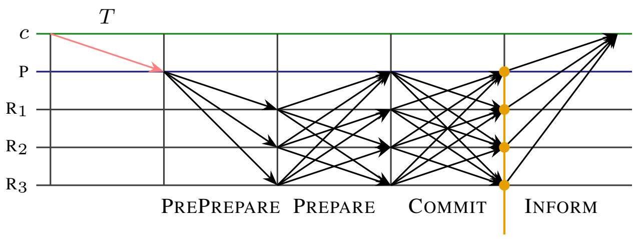


Figure 4: A schematic representation of the normal-case of the PBFT protocol with $\mathbf{n}_{\mathcal{R}} = 4$ and $\mathbf{f}_{\mathcal{R}} = 1$.

Client Request. A client c that wants to process a transaction T in our HYBRIDCHAIN architecture creates a request $\langle T \rangle_c$ and sends it to the replica designated as the primary of the view v. The client c uses DS to sign this message and adds a monotonically increasing timestamp to this message.

Pre-prepare. When the primary P replica receives a well-formed client request $m := \langle T \rangle_c$, it assigns m a sequence number k and creates and sends a PREPREPARE message to all the replicas. This PREPREPARE message also includes a digest $\texttt{hash}(m)$ of m, which is used in future communication to save space. During this phase, it is sufficient for the primary to sign the messages using MAC. When a replica R $\in \mathcal{R}$ receives a well-formed PREPREPARE message from the primary P of view v, it agrees to support the order k for m if it has not agreed to order another request at sequence number k. The replica R shows its support by broadcasting a PREPARE message.

Prepare. When a node R receives identical PREPARE messages from $2\mathbf{f}_{\mathcal{R}} + 1$ distinct replicas (can include its own message to reach the count), it marks the request m as *prepared* and broadcasts a COMMIT message. In HYBRIDCHAIN, we require each replica R to use DS to sign the COMMIT message.

Commit. When R receives identical COMMIT messages from $2\mathbf{f}_{\mathcal{R}} + 1$ replicas, it marks m as *committed*. If R has executed all requests with sequence number less than k, it executes m and sends a RESPONSE message to the client, which includes the result of execution r. The client c marks $\langle T \rangle_c$ as processed when it receives identical RESPONSE messages from at least $\mathbf{f}_{\mathcal{R}} + 1$ replicas.

Chain Communication. Post consensus on m, each replica R creates a certificate $\mathfrak{C}$, which includes: (i) the client request m, (ii) COMMIT messages for m from $2\mathbf{f}_{\mathcal{R}} + 1$ replicas, and (iii) the result r. Next, the replicas may follow the *cluster-sending* protocol [13] or delayed replication protocol [14] to communicate with the miners in set $\mathcal{M}$.[1] The cluster-sending protocol guarantees the delivery of at least one message between the two clusters, given that less than one-third of members of each cluster are malicious. To do so, each member from the sending cluster sends a message to a distinct member in the receiving member. In our case, we need the certificate $\mathfrak{C}$ to be sent to at least $2\mathbf{f}_{\mathcal{M}} + 1$ miners: replica R1 sends $\mathfrak{C}$ to miner M_1; replica R2 sends $\mathfrak{C}$ to miner M_2, and so on.

When a miner M $\in \mathcal{M}$ receives a certificate $\mathfrak{C}$ from a replica R $\in \mathcal{R}$, it broadcasts this certificate to all the other miners. As at least $2\mathbf{f}_{\mathcal{M}} + 1$ miners receive certificates, there is a guarantee that at least one honest miner will broadcast the certificate. As a result, each miner will have access to the certificate $\mathfrak{C}$, and it can proceed with the POC computation.

4.2 Collaborative Mining

Post PBFT consensus, our HYBRIDCHAIN system runs the POC protocol on the agreed transaction to securely bind it to the ledger. POC requires all the miners to *collaborate* and work together to compute the required hash. To do so, POC divides the POW hash computation into $\mathbf{n}_{\mathcal{M}}$ disjoint subproblems and requires each miner to work on a distinct predetermined subproblem.

[1]We can model chain communication as either push- or pull-based model using existing peer-to-peer communication primitives.

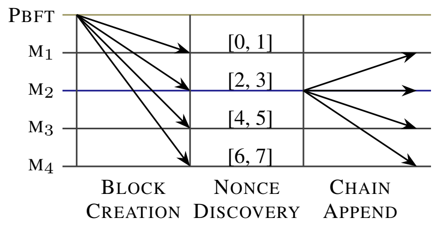


Figure 5: A schematic representation of the PoC protocol with $\mathcal{M} = \{\text{M}_1, \text{M}_2, \text{M}_3, \text{M}_4\}$. The total solution space $\mathcal{S}$ is $[0, 7]$ and is divided into four slices $([0, 1], [2, 3], [4, 5], [6, 7])$. Miners receive transactions from the PBFT replicas. Post creating a block, each Miner M_i, $i \in [1, 4]$ tries to discover the nonce in its slice $\mathcal{S}_i$. Assume the valid nonce is 2, then once M_2 discovers the nonce, it broadcasts the same to other miners.

Let, $\mathcal{S}$ represent the solution space; all the random numbers that a miner has to try to find a valid nonce. Without the loss of generality, let us divide $\mathcal{S}$ into $\mathbf{n}_{\mathcal{M}}$ equal slices, $\{\mathcal{S}_1, \mathcal{S}_2, \cdots \mathcal{S}_{\mathbf{n}_{\mathcal{M}}}\}$, such that

$$\mathcal{S}_1 \cap \mathcal{S}_2 \cap \cdots \cap \mathcal{S}_{\mathbf{n}_{\mathcal{M}}} = \varnothing \tag{1}$$

$$\mathcal{S}_1 \cup \mathcal{S}_2 \cup \cdots \cup \mathcal{S}_{\mathbf{n}_{\mathcal{M}}} = \mathcal{S} \tag{2}$$

Our PoC protocol assigns slice $\mathcal{S}_1$ to miner M_1, $\mathcal{S}_2$ to M_2, and $\mathcal{S}_i$ to M_i, $i \in [1, \mathbf{n}_{\mathcal{M}}]$. The key assumption is that if a miner takes time τ to find a valid nonce on the solution space $\mathcal{S}$, then if all the miners are honest and follow the PoC protocol, the time required to find the nonce should be of the order $\mathcal{O}(\frac{\tau}{\mathbf{n}_{\mathcal{M}}})$.

Further, PoC protocol ensures that each honest miner gets rewarded for their efforts; rewards are distributed among miners. If $\Diamond$ is the reward for a miner to find a valid nonce in PoW protocol, then in our PoC protocol, each i^{th} miner M_i receives a reward $\Diamond_i$ proportional to the size of its slice $\mathcal{S}_i$.

$$\Diamond_i = \frac{|\mathcal{S}_i|}{|\mathcal{S}|} * \Diamond \tag{3}$$

In the rest of this paper, for simplicity, we assume that all the slices have the same size.

4.2.1 PoC Protocol

Our PoC protocol works in rounds, and within each round each miner tries to find if a valid nonce exists in its slice of the block. In the rest of this section, we assume that the solution space $\mathcal{S}$ can be deterministically divided into $\mathbf{n}_{\mathcal{M}}$ disjoint equal slices by each miner. For example, in Figure 5, the solution space $\mathcal{S} = [0, 7]$ is divided into $\mathbf{n}_{\mathcal{M}} = 4$ slices; the slices are: $\mathcal{S}_1 = [0, 1]$, $\mathcal{S}_2 = [2, 3]$, $\mathcal{S}_3 = [4, 5]$, and $\mathcal{S}_4 = [6, 7]$. Designing optimal slice distribution schemes is an interesting research avenue, which we consider outside the scope of this work.

Certificate Dissemination. The PoC protocol starts when a miner M receives a certificate $\mathfrak{C}$ from a replica. The miner M checks if $\mathfrak{C}$ is well-formed and $\mathfrak{C}$ includes signatures from $2\mathbf{f}_{\mathcal{R}} + 1$ replicas; a proof that these replicas agreed to sequence this batch of transactions at a sequence number k. If this is the case, M broadcasts this certificate to other miners. Note: although while explaining PBFT we considered consensus on a single transaction, it can be trivially extended to a batch of transactions. This batching optimization is employed by all the existing BFT protocols to increase their throughputs [3, 7, 9].

Block Creation. When a miner M has the nonce for the block ordered at sequence number $k - 1$, it initiates the creation of a block at sequence k. It does so by generating a Merkle root of all the transactions in k^{th} batch and a new block header. As each miner M knows there are a total of $\mathbf{n}_{\mathcal{M}}$ miners, it creates $\mathbf{n}_{\mathcal{M}}$ slices and assigns itself the i^{th} slice $\mathcal{S}_i$ in round 0, where $i = \mathsf{id}(\text{M})$.

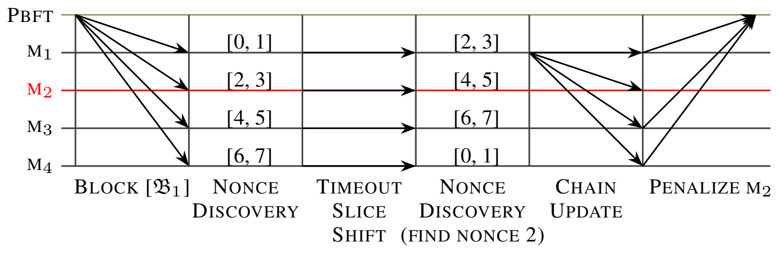


Figure 6: An illustration of the slice shifting procedure. Here, we assume 2 is the valid nonce of the block, and the miner M$_2$ is malicious. Hence, M$_2$ does not broadcast the block to other miners, which triggers slice shifting procedure. Post slice shifting, M$_1$ discovers the nonce and broadcasts to other miners.

Nonce Discovery. We assume that each miner M knows the characteristics of the expected hash (the number of leading zeroes). The miner uses this information to go over all the possible nonces in its slice range to find a valid nonce. Once a miner M computes the correct hash, it has access to a valid nonce. The miner M uses this information to complete the block header and forwards the block to all the miners.

Chain Append. When a miner receives a block from another miner, it first validates the nonce. If the nonce is valid, the miner appends this block to its local blockchain and assumes the PoC protocol for the corresponding batch as complete. Notice that if all the miners are well-behaving, then our PoC protocol requires only one round to find the valid nonce as the nonce is present in one of the slices. Post discovering the nonce, each miner starts working on the next block to be added to the chain.

4.2.2 Slice Shifting Protocol

In our HYBRIDCHAIN system, each miner receives certificates from the PBFT replicas. These certificates include client transactions that have been ordered by at least $2\mathbf{f}_{\mathcal{R}} + 1$ replicas. Our HYBRIDCHAIN architecture uses the PoC protocol to add these transactions to the ledger in the order defined by PBFT replicas. As a result, a malicious miner has limited attack opportunities; if a malicious miner finds a valid nonce in its slice, it can avoid forwarding this information to the honest miners. If such is the case, despite searching over its slice, each honest miner would not find any possible solution and would not be able to make progress.

To resolve this attack, our PoC protocol requires each miner to set a *timer* δ. Each miner M starts a timer δ when it receives a certificate from the PBFT replicas. M stops δ if it discovers the valid nonce or it receives a valid block from another miner. If M's timer δ expires, and it does not have access to the valid nonce, it initiates the *slice shifting* protocol. Once the slice shifting is endorsed by the majority of miners, then each miner searches for the nonce in the next slice. Specifically, if prior to slice shifting a miner M was working on the i^{th} slice $\mathcal{S}_i$, post slice shifting M will work on $((i+1) \mod \mathbf{n}_{\mathcal{M}})^{th}$ slice, $\mathcal{S}_{(i+1)}$. As each miner already has access to all the slices, this switch does not require any additional communication.

The key intuition behind the slice shifting procedure is that even if a malicious miner decides to hide the nonce, post switch, it will be discovered by another miner. However, it is possible that up to $\mathbf{f}_{\mathcal{M}}$ consecutive miners may be malicious. As a result, the honest miners will discover the valid nonce after $\mathbf{f}_{\mathcal{M}}$ shifts.[2]

4.2.3 Reward and Penalty Economy

Frequent slice shifting due to malicious miners will be detrimental to the performance of our PoC protocol; it forces honest miners to do more work and wastes their resources. Moreover, why would any rational miner want

[2]The search space can be salted deterministically upon shifting to expand the search space and guard against rare cases in which the original problem may have no solution irrespective of minors' behavior.

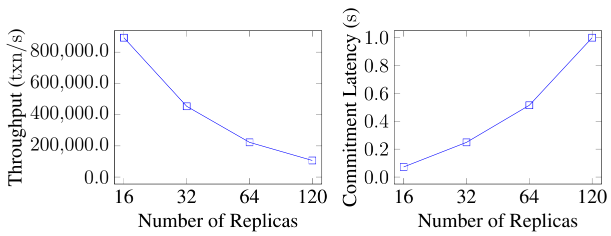


Figure 7: Evaluating peak throughput and commitment latency attained by PBFT consensus in HYBRIDCHAIN.

to join the PoC network and invest its computational resources? To make PoC protocol fruitful, we incentivize all the honest miners for their efforts; all miners are assumed honest until proven guilty.

First, like existing blockchain systems, such as Bitcoin and Ethereum, one of the aims of our HYBRIDCHAIN system is to establish a decentralized economy. To do so, like Bitcoin, in PoC, when a miner discovers a nonce and broadcasts the valid block to other miners, we assume the creation of a *new token*. For brevity, we skip diving into the crypto-economics of the token generation and disbursement, and refer to the existing literature on the same [16, 22]. However, the key goal is that this token is equally divided among the honest miners. Further, like Bitcoin and Ethereum, we expect each client to pay some fees for getting its transaction processed by our HYBRIDCHAIN system. This fees is also equally divided among all the miners working on the current block.

To disburse transaction fees and tokens among the miners, there are two possible approaches: (i) Like Bitcoin, each miner includes $\mathbf{n}_{\mathcal{M}}$ transactions that assigns an equal fraction of the reward to every other miner's public-key (account). These transactions can be deterministically created by each miner prior to mining and are included while creating the Merkle root. However, this will create unnecessary book-keeping, increasing the size of blocks. (ii) We assume that the genesis block of the ledger records the information about the founding miners and their respective initial slices. Further, miners can redistribute, resale, and divide their slices to other miners (similar to buying and selling of stocks), and any such transactions must be stored on the ledger before becoming effective. Assume that when a miner purchases a slice, sufficient tokens are reserved to enable penalization of misbehaving miners, which results in slashing their reserve funds similar to Proof-of-Stake designs [8]. Given all this information, when a block is formed by the PoC miners, we add the incentives to the accounts of the respective miners; miners can validate if they received incentives or not.

This rewarding process of PoC is similar to strategies adopted by *mining pools* in systems like Bitcoin. Most importantly, in PoC, the agreement on what to be included in the next block is strictly determined by PBFT chain, not miners. This substantially simplifies the design of PoC by making it deterministic, eliminating any lottery-based or leader-less consensus challenges that traditional PoW must cope with. Furthermore, we present the novel idea of slice shifting for the cases when no miner in a round broadcasts a valid nonce. As slice shifting requires each miner to work on the next slice to find the nonce, it is expensive. We mitigate the need for slice shifting by heavily *penalizing* malicious miners. Specifically, we require each miner to count the number of *shifts* it took to find a valid nonce and to identify the miner who failed to find the valid nonce. Further, as the order of all initial slice assignments is known to all the miners, so each miner can trivially determine which miner was responsible for previous shifts. Notice that any misbehaving miner will be discovered and penalized as it diminishes the returns for other honest miners.

5 Proof-of-Concept Evaluation

We now present an initial evaluation of our vision of HYBRIDCHAIN architecture, which we implement in the open-sourced RESILIENTDB fabric [7, 10, 9, 17].[3]

[3]For our proof-of-concept of HYBRIDCHAIN architecture, we employ an experimental version of RESILIENTDB with the codename of *NexRes* (Next Generation RESILIENTDB); this is an architectural rewrite of the RESILIENTDB 3.0 (the latest stable version).

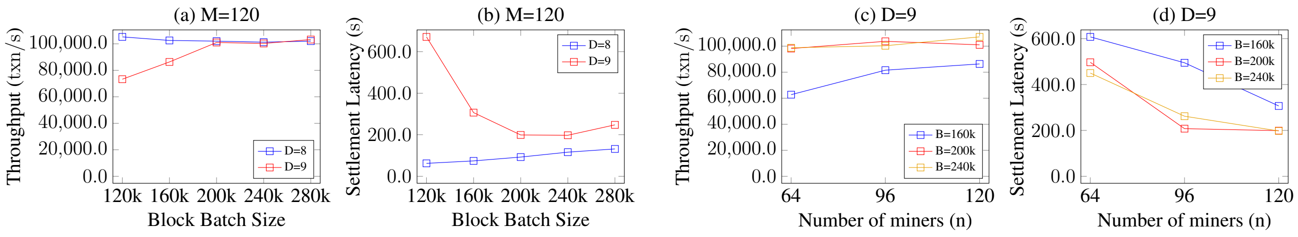


Figure 8: Evaluating PoC throughput and average settlement latency with different difficulty (D), miners (M) and block batch size (B).

Experimental Setup. We use Oracle Cloud Infrastructure's VM.Standard2.8 architecture to deploy miners and replicas (16 cores, 8.2 Gbps bandwidth, 120 GB Memory). In our experiments, we generate 400 million client requests of size 64 B each, while client response has size 17 B. We average results over three runs. We require clients to sign their messages using `ED25519` while replicas use `CMAC`.

Batching. We employ the standard practice of batching client transactions, which we refer to *transaction batching*, to optimize the PBFT consensus. Additionally, during the PoC consensus, each miner aggregates multiple batches from PBFT consensus, ' which we refer to *block batching*, prior to mining.

PBFT Scaling. In Figure 7, we gauge the peak throughput and *commitment latency* incurred by the PBFT consensus of our HYBRIDCHAIN architecture. This is an important metric as it informs the rate at which we can reply to the clients. For this experiment, we increase the number of replicas from 16 to 120 and require the primary to process a batch of 100 transactions per consensus; transaction batch size is set to 100. As expected, on increasing the number of replicas, there is a drop in peak throughput and an increase in incurred latency, which remains in the subsecond range. This phenomenon occurs because, at each setting, we are approximately hitting the network bandwidth; as the number of replicas increases, more messages are communicated per consensus. In summary, the peak throughput reaches well over 800 k transactions/second at 16 replicas while sustaining over 100 k transactions/second even when scaling to as many as 120 replicas.

PoC Scaling. Next, we study the impact of our PoC consensus protocol on the HYBRIDCHAIN architecture. We deploy 120 replicas for running PBFT consensus. For the PoC setting, we set the solution space parameter to 42 nonce bits and split it into equal disjoint slices based on the number of miners. Notice that the difficulty of each problem is the number of leading zeros in the hash. In Figure 8, we present our results; here D refers to difficulty (with the default of $D = 9$), M refers to the number of miners (with the default of $M = 120$), and B refers to the number of batches in a block. As stated earlier, for every B blocks produced by PBFT a single block is notarized and minted by PoC.

In Figures 8(a) and 8(b), we fix the number of miners to 120, which allows creating 120 equal slices, and increase the block size from 120 k to 280 k. We test at two difficulty levels: $D = 8$ and $D = 9$. Our results indicate that $D = 8$ is relatively easy, due to which each miner has to perform a smaller amount of work. As a result, any increase in batch size does not increase peak throughput. Hence, we test on $D = 9$ at which mining smaller batch size impacts the system throughput as miners have to participate in a larger number of consensus rounds. On further increasing the block size, we observe that the throughput hits the PBFT's peak as desired. Thus, notarizing the blocks by PoC no longer hinders the system throughput, it only prolongs the *settlement latency* as expected. When examining the end-to-end system throughput of HYBRIDCHAIN which includes both PBFT and PoC, we observe a sustained throughput of 105 k with a commitment latency of 1 second and the settlement latency of 198 seconds when the batch size is set to 200 k. *Note:* We could not test at difficulty beyond $D = 9$ as each nonce computation became prohibitively time- and resource-intensive given our available commodity hardware.

In Figures 8(c) and 8(d), we gauge the performance of PoC when there are 64 to 120 miners. For these experiments, we also test at three different block batch sizes. As expected, the degree of collaborative mining is directly proportional to the number of miners. As we increase the number of miners, there is an increase in peak throughput and a decrease in the settlement latency.

6 Conclusions

In this paper, we present the vision of our HYBRIDCHAIN system, which facilitates the creation of a safe and efficient decentralized economy. HYBRIDCHAIN achieves these guarantees by separating the life-cycle of a client transaction into two phases: commitment and settlement. In the commitment phase, HYBRIDCHAIN employs a traditional BFT protocol to order the client transactions. Post this, HYBRIDCHAIN requires a set of miners to run the settlement phase, where they notarize the ordered client transactions. Our HYBRIDCHAIN architecture does not impact the transaction latency observed by the client as each client receives the commitment response post BFT consensus. Further, to efficiently notarize transactions during the settlement phase, our HYBRIDCHAIN architecture introduces the notion of collaborative mining, where participants work together instead of competing with each other. These notarized transactions are written to a ledger and can be queried in the future.

7 Acknowledgments

This work was supported in part by (1) *Oracle Cloud Credits* and related resources provided by the Oracle for Research program and (2) the *NSF STTR* under *Award Number* 2112345 provided to *Moka Blox LLC*.

References

[1] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. *SharPer: Sharding Permissioned Blockchains Over Network Clusters*, page 76–88. Association for Computing Machinery, 2021.

[2] Mohammad Javad Amiri, Chenyuan Wu, Divyakant Agrawal, Amr El Abbadi, Boon Thau Loo, and Mohammad Sadoghi. The bedrock of BFT: A unified platform for BFT protocol design and implementation. *CoRR*, abs/2205.04534, 2022.

[3] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[4] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 International Conference on Management of Data*, pages 123–140. ACM, 2019.

[5] Alex de Vries. Bitcoin's growing energy problem. *Joule*, 2(5):801–805, 2018.

[6] John R. Douceur. The sybil attack. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, pages 251–260, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[7] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. Proof-of-Execution: Reaching consensus through fault-tolerant speculation. In *Proceedings of the 24th International Conference on Extending Database Technology*, 2021.

[8] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. *Fault-Tolerant Distributed Transactions on Blockchain.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2021.

[9] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. RCC: resilient concurrent consensus for high-throughput secure transaction processing. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 1392–1403. IEEE, 2021.

[10] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. ResilientDB: Global scale resilient blockchain fabric. *Proc. VLDB Endow.*, 13(6):868–883, 2020.

[11] Suyash Gupta, Sajjad Rahnama, Shubham Pandey, Natacha Crooks, and Mohammad Sadoghi. Dissecting BFT consensus: In trusted components we trust! *CoRR*, abs/2202.01354, 2022.

[12] Suyash Gupta and Mohammad Sadoghi. Blockchain transaction processing. In *Encyclopedia of Big Data Technologies*, pages 1–11. Springer, 2019.

[13] Jelle Hellings and Mohammad Sadoghi. Brief announcement: The fault-tolerant cluster-sending problem. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019*, volume 146 of *LIPIcs*, pages 45:1–45:3, 2019.

[14] Jelle Hellings and Mohammad Sadoghi. Coordination-free byzantine replication with minimal communication costs. In *23rd International Conference on Database Theory, ICDT*, pages 17:1–17:20, 2020.

[15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, 2nd edition, 2014.

[16] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[17] Sajjad Rahnama, Suyash Gupta, Rohan Sogani, Dhruv Krishnan, and Mohammad Sadoghi. RingBFT: Resilient Consensus over Sharded Ring Topology. In *Proceedings of the 25th International Conference on Extending Database Technology*, pages 2:298–2:311. OpenProceedings.org, 2022.

[18] Meni Rosenfeld. Analysis of Bitcoin pooled mining reward systems, 2011.

[19] Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolic. State machine replication scalability made simple. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems*, pages 17–33. ACM, 2022.

[20] Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, and Natacha Crooks. Basil: Breaking up bft with acid (transactions). In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 1–17. Association for Computing Machinery, 2021.

[21] Harald Vranken. Sustainability of bitcoin and blockchains. *Current Opinion in Environmental Sustainability*, 28:1–9, 2017.

[22] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. 2015.

[23] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019.

[24] Kaiwen Zhang and Hans-Arno Jacobsen. Towards dependable, scalable, and pervasive distributed ledgers with blockchains. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1337–1346, 2018.

Transparent Sharding

Deepal Tennakoon
University of Sydney
dten6395@uni.sydney.edu.au

Vincent Gramoli
University of Sydney
vincent.gramoli@sydney.edu.au

Abstract

Sharding is a well known technique to scale distributed systems horizontally. With the recent advent of blockchains, which typically run in open networks in the presence of malicious participants, new forms of sharding techniques have arisen. While the database sharding was typically seemless for the client of the system, blockchain sharding allows clients to select the location of their data, or the shardchain where their contract executes.

A critical requirement in this new adversarial environment is for clients to be able to consult the current sharding state without being fooled by malicious participants, a property we refer to as transparency.

In this chapter, we survey classic sharding techniques inherited from the database literature and more recent sharding techniques inherited from the blockchain literature. Finally, we focus on a recent technique that builds upon these techniques and exploits the smart contract logic to adjust sharding on demand and the transparency of the blockchain to let clients consult the current sharding state securely.

1 Introduction

Sharding is a term originally tossed in the context of massively multiplayer online games, in which parallel worlds use the same database source but evolve different database runtime dedicated to different players. One explanation for the term "shard" stems from the game Ultima Online whose fictional story mentioned shattering a crystal into shards, holding copies of a world continent, such that these copies evolve in parallel [15].

Due to the growing amount of transactions, sharding became popular to scale databases horizontally [6, 8]. The sharding technique consists of replicating a database structure across multiple machines while splitting its dataset into chunks, each maintained by a distinct set of machines, called a *shard*. In particular by assigning different machines to the maintenance of separate rows of a table, sharding lowers the size of the database index at each machine. This allows to speed up the information retrieval by searching into a smaller index. By adding resources, sharding helps increasing the performance of database services. In particular, sharding can be used to dynamically adjust the provisioning of resources based on the demand, a notion often referred to as *elasticity* [23, 22, 24, 2].

Due to the scalability limitations of classic blockchains [20, 29], sharding has naturally been applied to blockchains [19, 14, 32] in the hope of scaling blockchains horizontally. However, the context of blockchains differs significantly from the traditional context in which databases operate: instead of running in the closed networks of datacenters, blockchains typically run in open networks where users are incentivized to steal digital

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

assets. The arbitrary (byzantine) failure model that can mimic a coalition of malicious users is thus a central problem of blockchains and differs radically from the crash failure or isolated arbitrary failure models present in datacenters. As an example, Figure 1 illustrates two different contexts in which the techniques used for database sharding and blockchain sharding can be implemented.

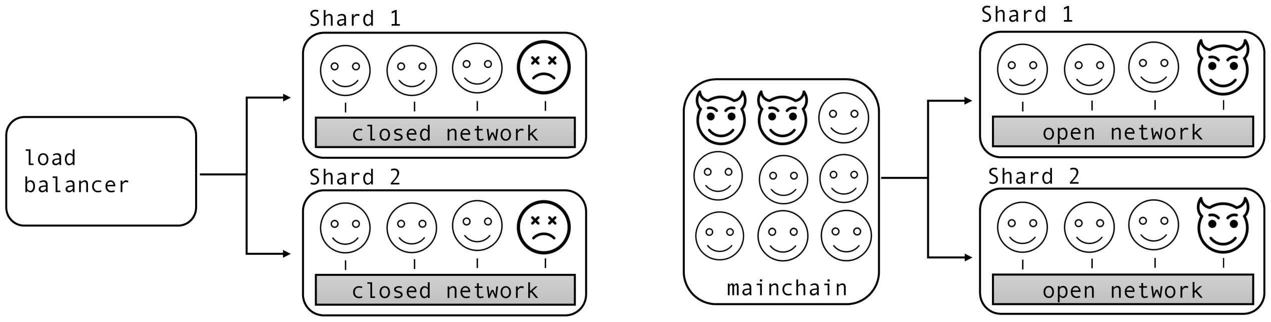


Figure 1: The database sharding (left) is typically using a load balancer to direct requests to the right shards running in closed networks whereas the blockchain sharding (right) typically uses different chains, like a mainchain and multiple shardchains, that run in an open networks where malicious participants can collude to attack the system.

Due to the growing demand for decentralized applications (DApps), the blockchain sharding techniques are evolving from static sharding techniques [25] to more dynamic sharding techniques [1]. These dynamic sharding techniques promise the elasticity of database sharding [23, 22, 24, 2]: they aim at adjusting the resource provisioning based on the demand. In some cases, the user could exploit this dynamism by deploying its popular DApp in a separate blockchain shard to avoid the congestion in other shards. In some other cases, one could migrate less demanded DApps in the same shard to decommission resources.

Unfortunately, most dynamic blockchain sharding solutions focus on creating or modifying shards [16], without considering how to access the existing sharding state. In other words the shard state is not *transparent*. If a user requests to adjust the amount of resources in a shard, then it typically ignores whether this adjustment was successful. The problem of offering transparency is not easy precisely because of the open networks in which the blockchains operate. A user could for example easily be fooled by a malicious participants that misinforms it of the success of its request, while the malicious participant never relayed it to the rest of the blockchain participants.

In this chapter, we survey existing sharding techniques and describe a new design that could leverage the inherent transparency of the blockchain to offer dynamic sharding that can be securely monitored, despite the presence of byzantine failures.

The rest of the chapter is organized as follows. In Section 2 we present the preliminaries. In Section 3, we present the database sharding techniques and in Section 4, we survey the blockchain sharding techniques. We present transparent sharding in Section 5. Finally, we present the related work in Section 6 and conclude in Section 6.

2 Preliminaries

We consider a distributed system of n *nodes* that exchange messages via a network. The failure model is different depending on the type of networks. The network can be *closed*, in which case nodes need to be authorized to join the network or *open*, in which case any nodes can join the network. Examples of closed and open networks are a local area network within a single datacenter and the Internet, respectively. It is typically more frequent to observe arbitrary failures in an open network than in a closed network, simply because one does not control the nodes joining the open network or their motivations.

In general, we consider that nodes communicate via point-to-point reliable channel. Hence if a node sends a message to another node and none of them fail, then the message is eventually received. Note that point-to-point channels can be implemented using secure channel. At times, we consider that every message takes less than some amount of time to be delivered, in which case we say that the network is *synchronous*. However, it is hard to predict the time a message will take in an open network due to various reasons (e.g., congestion, failures, disasters) that are out of the control of any administrator. Hence we often relax the synchrony assumption and assume that the network is *partially synchronous* [11] in that the bound exists but is not known by the protocol.

We assume the presence of up to f nodes that can fail among the n nodes of the group (we consider the group as either all nodes or just the nodes taking part in a shard). And we consider two types of failures: *crash* failures after which the faulty node stops acting and does no longer send messages and *byzantine* failures, when the faulty node behaves arbitrarily by not necessarily following the protocol. Interestingly, since byzantine nodes can store and send any arbitrary data to other nodes, it is important to provide a secure way for *correct* (i.e., non-faulty) nodes to retrieve the correct information. The property by which the system offers a node the possibility to retrieve correct information about the system is called *transparency*.

Blockchains are distributed systems [20]. Upon receiving client transactions, blockchain nodes first validate and broadcast them to the blockchain network. Then, all correct nodes in the blockchain agree upon the order to execute client transactions. This is known as reaching consensus on the order of transactions. Subsequently, all correct nodes execute transactions according to the order determined by consensus, reaching the same final state. After execution, the blockchain nodes store the transactions in blocks and each block points to the previous block forming a chain of blocks (i.e., an immutable ledger), hence the name “blockchain”. Finally, the client queries the blockchain to retrieve the result of the execution from the blockchain nodes. Just like databases, blockchains store the results of transaction executions, however, blockchains differ by the openness of the network in which they typically operate and the crytpography they require to avoid trusting a central entity.

3 Database Sharding

Sharding comes originally from the database literature where it proved instrumental for optimizing performance. It offers a natural way to adjust resource provisioning to serve the demand.

3.1 Storage Database Sharding

Google has been influential in the design of novel sharding techniques to scale its storage systems, in particular with BigTable [6] and Spanner [8].

BigTable [6] offers eventual consistency to NoSQL data by replicating the data on clusters of machines to offer horizontal scalability. It is optimized to offer low latency single-value reads and writes. The client requests a front-end server that redirects the request to a bigtable cluster, whose nodes, called *tablet servers* handle subsets of requests. One can improve the performance by adding tablet servers to the cluster. Bigtable stores data in tables, each being a key-value map, that is sharded into blocks of contiguous rows but tablet servers simply points to the data without hosting them, hence rebalancing tablets from one node to another is fast.

Spanner [8] is an SQL distributed database that scales horizontally. Rows are organized lexicographically by primary keys. Spanner replicates data across multiple zones. The data is sharded by ranges of rows across the multiple replicas of a zone. The replicas execute the Paxos consensus protocol to guarantee that consistent data are sufficiently replicated to tolerate potential crash failures before being committed. In case of failures, the data are migrated across machines to balance the load.

3.2 Dynamic Database Sharding

Dynamic sharding consists of modifying the sharding state at runtime. It is key to provide database elasticity.

Slicer [2] is a dynamic sharding solution for datacenters that proved effective to allocate resources to web-service frontends and increase the efficiency of web caches. Slicer derives a key from a request and exploits a centralized slicing service that monitors load and task availability to map key ranges or slices to tasks. Slicer combines with the Google Stubby's RPC system to balance tasks across datacenters. Slicer minimizes resource usage by 63% compared to a static sharding technique. Slicer uses a centralized algorithm rather than a client-based consistent hashing.

Accordion [22] offers the possibility to scale out and scale in a distributed database system by provisioning more or less servers as the load varies. Usually, databases are horizontally partitioned such that each partition is owned by exactly one server. The partitioning algorithm is key to the performance of the system but the placement of partitions is generally static. Accordion makes this mapping of partition to servers dynamic. E-Store [24] is an elastic technique for on-line transaction processing that exploits a two-tier horizontal partitioning technique that migrates data when load imbalance is detected to address the problem of the workloads being skewed towards the same resources.

Basil [23] presents a vertical database sharding approach for ACID transactions, maintaining a sharded key-value store in a byzantine fault-tolerant setting. This facilitates the execution of transactions concurrently in different shards. Basil requires the clients (i.e., the transaction senders) to decide to commit or abort the transactions based on the votes of replicas in a shard. Consequently, the client relays the outcome to the application. By ensuring byzantine isolation, whereby correct clients observe a state of the database produced by correct clients alone, and byzantine independence, whereby byzantine participants cannot collude to abort a client's transaction, Basil provides safety and liveness. However, the dynamism of Basil is limited as it does not explain how to change the number of shards at runtime. It also requires $5f + 1$ replicas per shard to ensure the aforementioned properties.

4 Blockchain Sharding

As blockchain typically operate in open networks, the blockchain sharding solutions must typically cope with byzantine failures. This is why multiple probabilistic techniques able to rotate the shard membership in an unpredictable way have become popular.

4.1 Deterministic sharding

Deterministic sharding [5, 7, 10] consists of assigning transactions to shards deterministically. The advantage of such an approach is that the current shard state is inherently transparent as anyone can infer the shard responsible for each transaction by simply computing a local deterministic function.

SharPer [5] creates shards deterministically based on geographical distribution. Nodes that are located close to each other are assigned to the same shard. Red Belly Blockchain [10] shards only the verification without sharding the consensus nodes. The motivation stems from the fact that the verification of cryptographic signatures is CPU intensive. Instead of having all nodes verifying all transactions, Red Belly Blockchain assigns deterministically each transaction, based on its hash, to two subsets of nodes: its $f + 1$ primary verifiers and its f secondary verifiers. The secondary verifiers wait for some time for the primary verifiers to verify the transaction. If the primary verifiers are too slow or unresponsive, then the secondary verifiers start verifying the transaction. A node detects whether a transaction is properly signed once it receives the same response from $f + 1$ distinct verifiers.

The drawback of deterministic sharding is that the outcome of the function is predictable, which makes the system vulnerable to attacks. In particular, an attacker can exploit this information in order to bribe the nodes that are responsible of a shard. If the attacker manages to bribe a sufficient portion of a shard then it can prevent the members of this shard from reaching consensus, potentially leading the system to an inconsistent state. Such inconsistent states are called "forks" and are exploited by various attacks [21, 12] to double spend. SSChain [7] allows nodes to freely join a shard deterministically. To avoid shard-takeovers SSChain follows a two-chained approach: a root chain that verifies the blocks coming from each shard before committing them, and a shardchain that agrees upon blocks to send to the root chain. The root chain is able to make an accurate verification of shard blocks by keeping the full state of the blockchain.

4.2 Probabilistic sharding

To cope with the predictability of deterministic sharding, probabilistic sharding protocols were proposed [14, 19, 32, 13]. Probabilistic sharding relies on a *mainchain* also known as beacon chain, final committee, or main committee, that performs administrative tasks like creating new shards and synchronizing the states of multiple shards. The mainchain creates each shard, also called *shardchain*, probabilistically which maintains separate state, transactions, blocks and a chain. Each shard verifies a unique subset of transactions and executes consensus separately on those transactions. Unlike deterministic sharding, the probabilistic creation of shards mitigate risks of shard-takeovers. For this purpose, probabilistic sharding mechanisms typically select a shard size and a number of shards to guarantee with high probability that the shard members can reach a consistent state through consensus. In particular, when the network is open, one cannot predict the time a message takes to be delivered, hence the sharding mechanisms must ensure that less than 1/3 of the shard members are byzantine nodes with high probability [11]. In probabilistic sharding to mitigate bribery from slowly-adaptive adversaries, shards are changed within a specific time period known as an epoch. This is to avoid a shard-takeover potentially causing double spending.

OmniLedger [14] is a permissionless sharded blockchain that creates shards probabilisitically based on the RANDHOUND protocol and a VRF. A shard remains active in a time period known as an epoch. OmniLedger assumes synchrony for shard creation and partial synchrony in a shard epoch. OmniLedger handles cross-shard transactions using an atomic commit-abort protocol run by clients sending transactions. However, clients can censor cross-shard transactions as they are tasked with creating cross-shard transactions. OmniLedger has performance enhancements such as concurrent processing of non-conflicting transactions in a shard as well as using state blocks as checkpoints to reduce the size of the downloaded blockchain when syncing. The fault tolerance of clients is not mentioned. Assuming synchrony for shard creation is not realistic for real-world cases.

RapidChain [32] assumes synchrony within an epoch but assumes partial synchrony in all other parts of the protocol. RapidChain's probabilistic shard creation involves a reference committee using proof-of-work (PoW) coupled with randomization to assign nodes to shards in a way that minimizes the probability of $f > n/3$ where f is the number of byzantine nodes and n is the number of shard nodes.

In contrast, Monoxide [27] assumes asynchrony and creates zones (i.e., shards) by assigning random identifiers to miners which assign those miners to zones. Each zone processes transactions, keeps state and executes consensus separately. Within a zone Monoxide uses PoW to agree on the order of transactions, hence making the consensus probabilistic. To mitigate adversaries centralizing their mining power to one zone to take over a zone, Monoxide [27] introduces a novel proof-of-work scheme known as Chu-ko-nu mining. This scheme allows a miner to create a block in any zone by solving a PoW puzzle, which evenly distributes the mining power across all zones, preventing it from being gathered to a single zone. Cross zone transactions are processed in an asynchronous and lock-free manner that allows the zones to concurrently process transactions. However, the number of zones in Monoxide is not adjustable at runtime to the best of our knowledge.

The next major release of Ethereum known as Ethereum 2.0 is said to contain a probabilistic sharding mechanism consisting of a fixed set of 64 shard chains and a single beacon chain [13]. Nodes require to escrow a deposit to Eth2.0 before assigning them to shardchains probabilistically using a random beacon. Eth2.0 requires a minimum of 111 nodes to be in a shard [28] to lower the probability of having $2/3$ adversarial nodes in a shard to 2^{-40}. Each shard runs a series of 64 Casper FFG consensus instances per epoch, after which a new block containing the shard states is appended to the beacon chain.

4.3 Probabilistic transaction sharding

Probabilistic transaction sharding creates shards probabilistically and inherits all characteristics of probabilistic sharding except it only shards transactions. In other words it assigns transactions to a subset of nodes (i.e., shards). The state, chain and blocks are not sharded.

Elastico [19] is a permissionless byzantine fault tolerant blockchain that partitions the network into shards that only process a subset of the entire set of transactions. Elastico assumes partial synchrony and achieves linear scalability. Since Elastico is permissionless, Sybil resistance is achieved by establishing identities using a PoW puzzle, public key and IP addresses. An adversary's capability to create multiple identities is limited since they require to solve a PoW puzzle. Elastico consists of a final committee and multiple committees each running its own BFT consensus. Based on the generated node identities and a random beacon generated by the final committee, nodes are assigned to shard committees per epoch. The final committee receives all agreed values from each committee at the end of an epoch and reaches consensus using a BFT consensus. Note that epochs in Elastico are based on an adjustable value N such that N is the number of blocks. To tolerate adaptive adversaries, Elastico rotates entire committees after an epoch, in contrast to the gradual or constant number of nodes rotated in other probabilistic approaches [13, 32]. As a result, Elastico nodes, despite being assigned to shards, require to store the entire state of all shards.

Zilliqa [25] exploits PoW and a random beacon to select nodes into a "DS committee", which is Zilliqa's mainchain. Every newly elected node in the DS committee churns out the oldest node making sure that at all times the most recently elected n nodes are in the DS committee. Nodes wanting to join shards use PoW and a random beacon generated by the DS committee to solve a puzzle and derive a nonce which is submitted to the DS committee. By reaching consensus on nonces, the DS committee assigns nodes to shards probabilistically where each shard processes a subset of transactions.

4.4 Probabilistic state sharding

Probabilistic state sharding creates shards probabilistically by only sharding states. It inherits all other characteristics of probabilistic sharding.

Al-Bassam et al. [3] presents ChainSpace that assigns smart contract objects to a set of nodes randomly based on $\Psi(o) = id(o) \mod K$ where K represents the constant number of shards and $id(o)$ is the $\mathrm{SHA256}$ hash of the object. Since smart contract objects are assigned to separate shards, each shard keeps a separate state corresponding to the smart contract objects. ChainSpace assumes asynchronous communications and uses BFT-Smart for consensus.

In the NEAR protocol[1], the set of nodes that have the highest stake in an epoch are randomly assigned to shards probabilistically. Each shard keeps a separate state. A node can be a member of one or many shards. When a node is a member of multiple shards it keeps the state of all those shards.

5 Transparent dynamic sharding

As recent sharding techniques offer blockchain users the ability to control the locations of their data or to select the shard in which they execute their contract, it has become crucial for sharding to be transparent.

5.1 Dynamic blockchain sharding

Dynamic blockchain sharding (DBS) [1] is a blockchain sharding protocol made transparent by exploiting the blockchain transparency itself. It shares commonalities with traditional sharding from the database literature [22, 24, 2, 23] that offer elasticity: new shards can be created and existing shards can be closed at runtime, hence

[1] `https://near.org/papers/economics-in-sharded-blockchain/`

adjusting potentially the provisioning of resources based on the demand. It differs from traditional sharding from the database literature by tolerating byzantine participants. It runs the Democratic Byzantine Fault Tolerant consensus protocol [9] so that any shard adjustment is decided by all correct nodes unanimously, despite partial synchrony and the presence of up to $f < n/4$ byzantine nodes, and it rotates shards probabilistically to cope with slowly-adaptive adversaries, similar to probabilistic sharding approaches [14, 19, 32, 13].

5.2 Transparency

To achieve transparency, DBS differs from other techniques by exploiting the smart contract logic to adjust the sharding state. Initially, when the blockchain starts, it is equiped with a built-in smart contract that exposes to client users the functions to create a new shard by spawning potentially more computational resources, to close an existing shard by decommissioning computational resources, and to adjust the size of the existing shards at runtime.

As each function invocation to adjust the shards consists of a blockchain transaction request that gets securely stored in the distributed ledger (like any other blockchain transactions), a node simply needs to consult the current state of the mainchain to derive the most current sharding state. This state indicates the amount of shards that exist, the number of nodes in each shard, the locations (e.g., static IP addresses, domain names) of these nodes. Note that if the client needs to retrieve the state of the shard (e.g., its DApps, past transactions), then the client would need to download the corresponding shardchain as well.

When a client wants to download the mainchain, it first contacts the nodes running the mainchain. Note that it is reasonable to assume that a client can retrieve the nodes of the mainchain, otherwise this client would not be able to use the service (classic blockchains like Bitcoin [20] and Ethereum [29] use hard-coded DNS seed for clients to retrieve blockchain nodes). The client then asks a copy of the blockchains to the mainchain nodes. Upon confirmation of the current state of the mainchain by $f + 1$ distinct mainchain nodes, then it knows that this mainchain state is correct. This is because f is the maximum number of byzantine nodes in the system by assumption, so there cannot be $f + 1$ malicious nodes responding to the client with a fake state. Note that we could hardcode directly the domain names of the nodes hosting the shards but every adjustment to the shard would require a lengthy DNS reconfiguration.

5.3 Mainchain and shardchains

DBS lets existing users of the initial blockchain, called the mainchain, become a user of a shard, called a shardchain, by depositing assets into the mainchain while invoking the shard creation function. These deposited assets remains frozen in the mainnet but can be used to transact in the shardchain for the lifetime of the shardchain. When the shardchain is closed, the balances are reconciled and the remaining deposits are returned to their users. DBS was shown instrumental to accelerate the performance of the blockchain almost linearly with the number of shards in good executions. In case of unexpected network delays, DBS may not succeed in adjusting the sharding during the first attempt, then it retries after allocating more time for the second attempt and so forth. As the network is partially synchronous, there is a point in the execution where DBS has allocated a sufficient amount of time for the sharding adjustment to succeed.

Once the client has successfully downloaded the mainchain, as explained in Section 5.2, then it is easy to reconstruct the current sharding state. The client can inspect the history of transactions stored in the mainchain and retrieves one by one the smart contract function invocations that created, closed and altered the shards. By replaying these transactions, the client can derive the current sharding state, by retrieving exactly the resources (computational nodes involved in running the shards) and the users (the users that deposited assets to access each shardchain).

6 Related work

There exist several surveys on blockchain sharding [26, 31, 30]. In this section, we discuss similarities and disparities of these surveys with our blockchain sharding survery.

In [26], the authors provide a systemic and comprehensive review of blockchain sharding methods. First, they introduce the basic concepts of blockchain sharding such as identity establishment, randomness generation for shard creation, intra-shard consensus, cross-shard transactions, epochs, and shard committee reconfiguration. Then they discuss the key characteristics of state-of-the art sharding methods and they summarized in a table based on shard creation method, network model, intra-shard consensus, inter-shard consensus, safety and performance. Our survey in contrast classifies database sharding and then blockchain sharding methods based on common characteristics.

Yu et al. [31] present a systemic survey of blockchain sharding techniques for permissionless blockchains. However they do not cover sharding blockchain works such as Red Belly Blockchain [10], NEAR [2] and Zilliqa [25]. This survey, similar to Sok [26] summarizes the surveyed blockchain sharding techniques according to their key characteristics but in contrast does not discuss database sharding approaches and its lead up to blockchain sharding.

Xi et al. [30] perform a comprehensive survey on blockchain sharding that includes Monoxide [27], Elastico [19], OmniLedger [14], RapidChain [32], ChainSpace [4], Ethereum2.0 [13] and TEEEChain [17]. They identify the following characteristics of each sharded blockchain: the network model (e.g. synchronous, partially-synchronous), node allocation method into shards (e.g. PoW, random beacon, deterministic, etc), transaction model (e.g. UTXO, account-based), intra-shard consensus algorithm, threat model, cross-shard transaction processing techniques and performance. Similar to our work, Xi et al. [30] classify blockchain sharding into three categories. Namely, network sharding, transaction sharding and state sharding. However, they do not explicitly assign sharded blockchains to these three categories.

In [18], the authors offer a systematic analysis of existing sharded blockchain systems. They decompose the blockchains that benefit from sharding into functional components, classify these systems, and analyze their components. They present a layered decomposition similar to the layers 0, 1, 2 of Xi et al. [30] but called them network, consensus and application layers. In their context, sharding consists of splitting the work so that each shard generates its own independent chains completely disconnected from other chains. As a result, they consider solutions that partially order transactions. This is a distinction with some of the sharding techniques we consider, like verification sharding, that totally order all transactions [10].

Interestingly, the large body of work on blockchain sharding explains how shards can be created or modified despite the presence of malicious participants, however, they do not explain how one can retrieve the current sharding state in a secure way. By contrast, we consider transparency as an important property to allow users to retrieve the current state of the blockchain sharding despite the presence of malicious participants.

7 Conclusion

In this chapter, we surveyed sharding techniques both in the database context and in the blockchain context. Although the blockchain sharding techniques are inspired by the database context, they raise an interesting challenge related to the openness of the network in which they execute. In this novel context, an interesting problem is the one of transparency where the users can retrieve the correct sharding state despite the presence of malicious coalitions. We presented a recent solution to this problem that lets users adjust the shards dynamically and consult the current sharding state securely through transparency.

[2]`https://near.org/papers/economics-in-sharded-blockchain/`

Acknowledgements

This research is supported under Australian Research Council Future Fellowship funding scheme (project number 180100496) entitled "The Red Belly Blockchain: A Scalable Blockchain for Internet of Things".

References

[1] Deepal tennakoon and vincent gramoli. In *Proceedings of the Fifth International Symposium on Foundations and Applications of Blockchain (FAB)*, 2022.

[2] Atul Adya, Daniel Myers, Jon Howell, Jeremy Elson, Colin Meek, Vishesh Khemani, Stefan Fulger, Pan Gu, Lakshminath Bhuvanagiri, Jason Hunter, Roberto Peon, Larry Kai, Alexander Shraer, Arif Merchant, and Kfir Lev-Ari. Slicer: Auto-sharding for datacenter applications. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, page 739–753, 2016.

[3] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778*, 2017.

[4] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. In *25th Annual Network and Distributed System Security Symposium NDSS*. The Internet Society, 2018.

[5] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. *SharPer: Sharding Permissioned Blockchains Over Network Clusters*, page 76–88. Association for Computing Machinery, New York, NY, USA, 2021.

[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2), 2008.

[7] Huan Chen and Yijie Wang. Sschain: A full sharding protocol for public blockchain without data migration overhead. *Pervasive and Mobile Computing*, 59:101055, 2019.

[8] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, 31(3), 2013.

[9] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: efficient leaderless Byzantine consensus and its application to blockchains. In *Proc. 17th IEEE Int. Symp. Netw. Comp. and Appl (NCA)*, pages 1–8, 2018.

[10] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Red Belly: a secure, fair and scalable open blockchain. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21)*, May 2021.

[11] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[12] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The Attack of the Clones against Proof-of-Authority. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'20)*, Feb 2020.

[13] The eth2 upgrades. Accessed: 2022-03-26.

[14] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. Cryptology ePrint Archive, Report 2017/406, 2017.

[15] Ralph Koster. Database "sharding" came from uo?, 2009. Accessed:2022-04-23 - `https://www.raphkoster.com/2009/01/08/database-sharding-came-from-uo/`.

[16] Jae Kwon and Ethan Buchman. Cosmos White Paper. Accessed:2022-05-30 - `https://v1.cosmos.network/resources/whitepaper`

[17] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter R. Pietzuch, and Emin Gün Sirer. Teechain: Scalable blockchain payments using trusted execution environments. Technical Report 1707.05454, arXiv, 2017.

[18] Yizhong Liua, Jianwei Liua, Marcos Antonio Vaz Sallesb, Zongyang Zhanga, Tong Lia, Bin Hua, and Rongxing Luc Fritz Hengleinb. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. Technical Report 2102.13364, arXiv, 2021.

[19] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 17–30, 2016.

[20] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008.

[21] C. Natoli and V. Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *47th IEEE/IFIP Int. Conf. Dependable Syst. and Netw. (DSN)*, Jun 2017.

[22] Marco Serafini, Essam Mansour, Ashraf Aboulnaga, Kenneth Salem, Taha Rafiq, and Umar Farooq Minhas. Accordion: Elastic scalability for database systems supporting distributed transactions. *Proc. VLDB Endow.*, 7(12), 2014.

[23] Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, and Natacha Crooks. Basil: Breaking up bft with acid (transactions). In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 1–17, New York, NY, USA, 2021. Association for Computing Machinery.

[24] Rebecca Taft, Essam Mansour, Marco Serafini, Jennie Duggan, Aaron J. Elmore, Ashraf Aboulnaga, Andrew Pavlo, and Michael Stonebraker. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proc. VLDB Endow.*, 8(3), 2014.

[25] The ZILLIQA Team. The zilliqa technical whitepaper. Technical report, Zilliqa, 2017. Accessed February 2022.

[26] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, page 41–61, 2019.

[27] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 95–112. USENIX Association, February 2019.

[28] SJ Wels. Guaranteed-tx: The exploration of a guaranteed cross-shard transaction execution protocol for ethereum 2.0. Master's thesis, University of Twente, 2019.

[29] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.

[30] Jinwen Xi, Shihong Zou, Guoai Xu, Yanhui Guo, Yueming Lu, Jiuyun Xu, Xuanwen Zhang, and Francesco Gringoli. A comprehensive survey on sharding in blockchains. *Mob. Inf. Syst.*, jan 2021.

[31] Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J. Andrew Zhang, and Ren Ping Liu. Survey: Sharding in blockchains. *IEEE Access*, 8:14155–14181, 2020.

[32] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. RapidChain: Scaling blockchain via full sharding. In *ACM CCS*, pages 931–948, 2018.

The Anatomy of Blockchain Database Systems

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

Abstract

Blockchains are around for more than ten years and currently, we are witnessing the adoption of blockchain techniques in databases, and vice-versa. For example, typical blockchain data structures, such as cryptographically-linked blocks and Merkle trees, have been integrated into verifiable databases. On the other hand, database techniques, such as sharding and concurrency control, have been integrated into blockchains. In this paper, we are looking at systems that combine both blockchain and database techniques. We classify these systems into (i) permissioned blockchains, (ii) hybrid blockchain database systems, and (iii) ledger databases. We present their anatomy, including features, techniques, and design choices, by analyzing a few representative systems. In the end, we highlight their limitations and discuss future research directions.

1 Introduction

In the last few years, the line between blockchain systems and distributed databases has been blurred to a certain degree [36, 40]. We have seen the adoption of blockchain techniques in databases. For example, typical blockchain data structures, such as cryptographically-linked blocks [4] and Merkle trees [29], have been integrated into verifiable ledger databases [9, 11, 10, 14] and hybrid blockchain database systems [24]. We have also seen database techniques used in blockchains. For example, sharding is used to scale blockchains [18], while optimistic concurrency control (OCC) is used to decrease the number of aborted transactions [40, 37].

By zooming into the design and implementation of systems that combine blockchain and database techniques, we propose classifying them into three categories. Going from systems that have stronger blockchain features to systems that are closer to databases, these three categories are (i) *permissioned blockchains*, (ii) *hybrid blockchain database systems*, and (iii) *ledger databases*. From a high-level view, all these systems consist of distributed server nodes that communicate via a broadcasting service based on some consensus protocol. Each server node has a ledger (blockchain data structure) and a local database. Both the server nodes and the users (or clients) that interact with these nodes need to be authenticated. The broadcasting service is implemented either with a Crash Fault Tolerant (CFT) consensus protocol, that is closer to distributed databases, or a Byzantine Fault Tolerant (BFT) consensus that resembles typical blockchains.

In this paper, we analyze a few representative systems and present their anatomy in terms of design, techniques, features, and limitations. We shall present more details on our classification in Section 2, and analyze a few systems in Section 3. We present challenges, limitations, and future research directions in Section 4, and conclude in Section 5.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Table 2: Categories, Features, and Examples.

	Permissioned Blockchains	**Hybrid Blockchain Database Systems**	**Ledger Databases**
Administration	Decentralized	Decentralized	Centralized
Broadcasting	CFT or BFT	Typically CFT	CFT
Local Database	Tightly-coupled	Loosely-coupled	Tightly-coupled
Ledger	Replicated	Replicated	Centralized
Examples	Fabric [4] Quorum [3] Corda [27] Diem [10]	Veritas [25] BigchainDB [2] BlockchainDB [19] Blockchain Relational Database [33] ChainifyDB [39] FalconDB [35]	Amazon QLDB [9] Alibaba LedgerDB [10] Microsoft SQL Ledger [11] Spitz [14] Immudb [5] IntegriDB [49]

2 Classification

When analyzing the systems that combine both database and blockchain techniques, we can distinguish three main categories. First, we have *permissioned blockchains* (also known as private, enterprise, or consortium) that have more blockchain features than databases. Second, we have *hybrid blockchain database systems* which can be further classified into out-of-blockchain databases and out-of-database blockchains [36]. Third, we have (centralized) *ledger databases*. Table 2 presents the features of such systems and a few examples of the state-of-the-art for each category.

Permissioned blockchains, as opposed to typical permissionless or private blockchains such as Bitcoin [32] and Ethereum [14], employ authentication for the parties using the blockchain (i.e., clients and peers). They are named permissioned or private blockchains because only authenticated parties can use them. These blockchains are typically used in enterprise setups and they are operated by a consortium of organizations, hence, they are called enterprise or consortium blockchains. In such setups, an organization hosts one or more blockchain peers (or nodes). Since more than one organization is in charge of administrating and operating the blockchain, a permissioned blockchain is a decentralized system where the ledger is replicated on all the nodes (or peers). Initially, some of these permissioned blockchains considered using Byzantine Fault Tolerant (BFT) consensus protocols to replicate the ledger. For example, Hyperledger Fabric v0.6 [19, 20] used PBFT [23] and Quorum provides support for IBFT [38]. However, these BFT protocols degrade the performance of a blockchain concerning throughput and latency [20, 30]. That is why most of the current permissioned blockchains use Crash Fault Tolerant (CFT) consensus mechanisms, such as Raft [34] and Apache Kafka [1].

Hybrid Blockchain Database Systems are very similar to permissioned blockchains but they have different motivations, use cases, and database integration. These systems are motivated by the need of organizations to share a database or parts of a database. In general, this database already exists and it is loosely-coupled to the hybrid blockchain database system. For example, in a supply chain scenario, there should be a shared database with shipping options and costs. Shipping companies update this database, while the other parties just read the data. In such a case, we need a ledger to keep track of the updates in a transparent and tamper-evident way. An authentication mechanism is needed to access the ledger and the broadcasting service. Given this, most of the proposed hybrid blockchain database systems consider only CFT broadcasting services. As expected, if a BFT consensus is used instead, the performance of the system significantly degrades [24].

Ledger Databases are at the other end of the centralized-decentralized administration spectrum since they are hosted and operated by a single organization. In such a centralized model, the users need to trust that organization. To increase the trust, ledger databases use tamper-evident data structures and publish the hashes of the append-only ledger or provide proofs for current states in the database. Such systems may be distributed to increase fault tolerance and improve performance. However, they are not distributed to increase the trust as is the

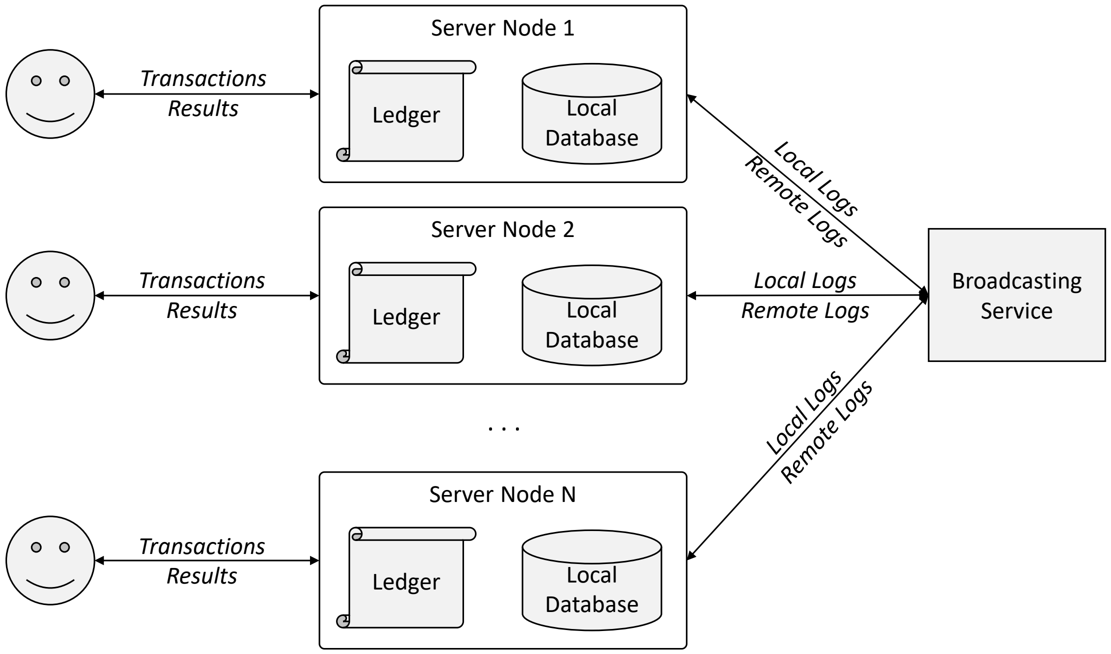


Figure 1: A Generic Hybrid Blockchain Database System

case for the other two categories. Moreover, the database and the ledger are tightly-coupled to the server nodes. While such systems require higher trust from the users, they provide higher performance and zero administration efforts compared to the other two categories.

3 Anatomy

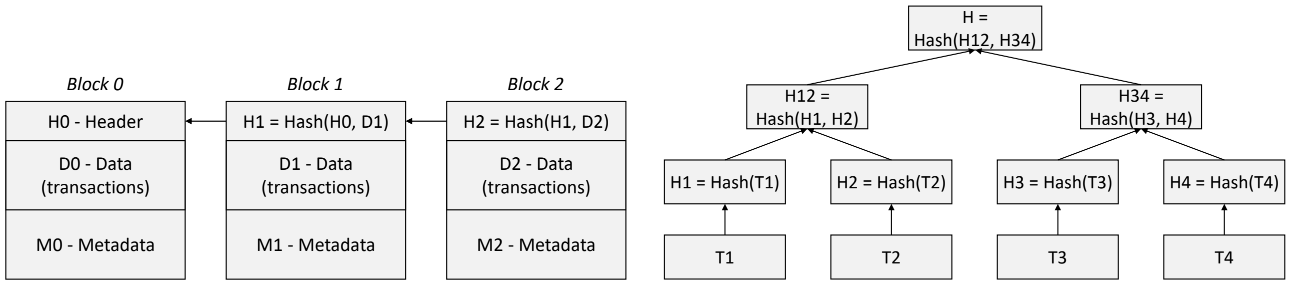
In this section, we start with the similarities among the three proposed categories, after which we present the particularities of each category together with the details of a few representative systems.

3.1 Overview

Typically, systems that combine blockchain and database features have an architecture similar to the one depicted in Figure 1. The system consists of some distributed server nodes (or peers), where each node handles user requests and coordinates with the other nodes via a broadcasting service. The users (or clients) need to be authenticated before sending requests to the nodes. A server node sends local updates (or logs) to and receives remote updates (or logs) from the broadcasting service. This broadcasting service can also be distributed across a few nodes, not necessarily the same as the server nodes. Moreover, the broadcasting service is implemented with a CFT or BFT consensus protocol. For example, the latest version of Fabric uses Raft [34], which is CFT, while Quorum supports, among others, IBFT [38].

Each server node connects to a local database and keeps a copy of the distributed ledger. Note that the local database and the ledger are different. The former keeps the latest version of the data (e.g., states, accounts, assets), while the latter keeps the entire update history using tamper-evident data structures. For example, Fabric uses LevelDB or CouchDB as its local database, which is also called World State. On the other hand, the ledger in Fabric is a linked list of blocks where the header of a block is linked to the header of the previous block using a cryptographic hash. Other systems use data structures based on Merkle trees [29] to represent the ledger.

We briefly compare these two ledger data structure, as illustrated in Figure 2. The hashed blocks data structure,



(a) Hashed Blocks (b) Merkle Tree

Figure 2: Ledger Data Structures

as shown in Figure 2a, is a linked list of blocks where a block points to its predecessor using a cryptographic pointer, except for the first block which is typically called the *genesis* block. Each block consists of data, metadata, and header sections. The data section contains all the transactions that are part of the block. The header is a digest of the block computed using a hashing function. Most of the blockchains use SHA3 or Keccak hashing algorithms. The header of all the blocks except the first one is computed as the hash of the concatenation between the hash of the previous block and the hash of the current block's data section.

A generic Merkle tree, as shown in Figure 2b, is a tree where the leaves are data representing transactions and the internal nodes are hashes. Each parent node contains the hash of the concatenation of all the hashes of its children. Hence, the root node contains the hash that represents all the underlying transactions. We note that Merkle trees can be combined with hashed blocks: the data section of a block can be organized as a Merkle tree. For example, Quorum uses this approach to store transactions in its ledger. In contrast, Fabric does not use a Merkle tree: it just hashes the transaction data as a chunk [9]. We direct the reader to [46] for an analysis of advanced Merkle tree data structures.

3.2 Permissioned Blockchains

Hyperledger **Fabric** [4] is a permissioned blockchain developed by Linux Foundation with a significant contribution from IBM. In Fabric, there are three types of nodes, namely, clients, peers, and orderers. A client sends a transaction request to a set of peers governed by an endorsement policy. For example, the *AND* policy includes all the peers in the network. That is, a client has to send the transaction to and receive endorsements from all the peers. A peer executes the transaction request in simulation mode and creates read and write sets to mark which world states are touched by the transaction. Since this is in simulation mode, the peer does not persist the changes to its local database. Then, the client sends the responses from the peers together with its transaction to the orderers. These orderers pack the transaction in a block and broadcasts the block to all the peers in the network. Currently, Fabric adopts the Raft CFT consensus among orderers. In the last phase, all the peers validate the block and persist the changes of valid transactions to the local database. Note that the peers do not need to re-execute the transaction: they only persist the write set. In the validation phase, the read set is also verified to check if any state has been modified since the transaction was simulated. In such a case, the transaction is aborted. In summary, Fabric implements an execute-order-validate (EOV or XOV) transaction lifecycle, as opposed to many other blockchains that adopt an order-execute (OX) lifecycle. Fabric supports both LevelDB (default) and CouchDB for the world states database. The ledger is stored on the filesystem as a linked list of blocks, where the block headers are linked together via hashing.

Fabric has been extensively analyzed and optimized by the database research community. Many works benchmark and analyze the performance bottlenecks of Fabric [17, 19, 20, 30, 31, 42]. In our recent work [31], we show that a Fabric network with up to 10 peers can reach around 1,000 transactions-per-second (TPS). Other works improve the rate of aborted transactions by relaxing the concurrency model (e.g., using optimistic concurrency control) and by re-ordering the transactions [37, 40].

Quorum [3] is a permissioned blockchain that draws its source code from Ethereum (implemented in the Go programming language). Naturally, Quorum supports Solidity smart contracts, but it replaces the energy-inefficient Proof-of-Work consensus with a few alternatives, out of which Raft is the default. Besides Raft, Quorum also supports IBFT (Istanbul BFT), QBFT (Quorum BFT), and Clique Proof-of-Authority (POA). IBFT is inspired by PBFT [23], while QBFT is an optimized version of IBFT which is also interoperable with Hyperledger Besu, an Ethereum client developed by the Hyperledger Foundation. As opposed to Fabric, Quorum has only peers and clients and adopts the traditional order-execute (OX) transaction lifecycle. That is, a transaction is first grouped into a block and then executed by each peer in the network. Similar to Fabric, Quorum uses LevelDB as its local database, but it adopts Merkle Patricia Trie for the ledger. In our recent work [31], we show that Quorum with Raft exhibits a throughput of 250 TPS, which is relatively low for a permissioned blockchain.

Corda [27] is advertised as a distributed ledger technology (DLT) for enterprises. For that reason, it is built on Java and Kotlin so it can be better integrated with existing Java enterprise systems. Besides nodes, a Corda network has notaries which are responsible for validating transactions in terms of uniqueness and validity. In essence, uniqueness prevents double-spending, while validity means that the transaction passes the input-output tests and it has all the required signatures. Notaries use a consensus protocol which is Raft-based in the default version of Corda. This default version uses H2, a relational database management system written in Java, for the local database. The ledger uses a custom version of Merkle trees to hide transaction details from the entities that are not involved in the transaction. A recent publication shows that the performance of Corda is very low, at 15 TPS [26]. Even when a single notary is used to minimize the impact of consensus, the performance is low due to a synchronous (blocking) transaction processing mechanism [26].

Diem [10] is a permissioned blockchain that was developed by a consortium of companies led by Facebook. It was previously known as the *Libra* blockchain [10]. The entire project has been discontinued in 2022. However, Diem implements some powerful features which are worth mentioning. For example, it uses LibraBFT [12], a BFT consensus based on Hotstuff [45] which further improves PBFT [23]. For the ledger, Diem uses Jellyfish Merkle tree [23] which is a sparse Merkle tree inspired by the Merkle Patricia Trie used in Ethereum. RocksDB [21], a fast key-value store derived from LevelDB and developed by Facebook, is used as the underlying database. A recent study shows that Diem achieves around 600 TPS on 4 nodes [47], which is a decent performance for a BFT-based blockchain.

3.3 Hybrid Blockchain Database Systems

Veritas [25] is an out-of-blockchain database that consists of a shared database (or table) and a blockchain ledger for keeping auditable and verifiable updates done on the shared database. Each node is operated by an organization. A node uploads its local update logs and downloads remote update logs to and from a broadcasting service. Veritas employs a concurrency control mechanism based on timestamps. The timestamp of a transaction represents the sequence number of that transaction in the log. A transaction is first verified locally by the node receiving it. If it passes the verification (e.g., multi-version concurrency control – MVCC), it is included in the logs and sent to the broadcasting service. Once the other nodes agree to the updates, they send acknowledgments, and once every node receives the acknowledgments, it persists the updates to the local database and appends them to the ledger. Note that this mechanism incurs $O(N^2)$ communication complexity [24].

The original design of Veritas uses Redis [15], an in-memory NoSQL database, and Apache Kafka [1], a CFT broadcasting service. The re-implementation of Veritas in [24] achieves around 30,000 TPS, making it the fastest system among all those analyzed in this paper.

BlockchainDB [19] is an out-of-blockchain database with prominent blockchain features: it is a shared database built over a blockchain. It is the only hybrid blockchain database that uses sharding to partition the shared database. Firstly, the blockchain represents the storage layer of a BlockchainDB node. By default, BlockchainDB uses Ethereum, but other blockchains can be used as well via a plugin interface. With Ethereum, the ledger structure is based on Merkle Patricia Trie. Secondly, a node has a database layer with a simple key-value interface. Thirdly, there is a shard manager that helps the database layer to identify the shard where a specific key is stored. Due to the use of such a slow blockchain, like Ethereum with Proof-of-Work (PoW) or Proof-of-Authority (PoA), BlockchainDB exhibits a throughput of around 50 TPS [24].

FalconDB [35] is another out-of-blockchain database that starts from a blockchain and provides a shared database to the clients. Different from other systems, FalconDB provides a relational database interface to the clients. In FalconDB, both the clients and the peers need to keep a digest of the data. The difference is that clients only keep the blockchain headers to save storage space. However, these headers are sufficient for checking the correctness of the data queried from the peers. FalconDB uses IntegriDB [49], a verifiable SQL database, to store the ledger, Tendermint for consensus, and MySQL as the local database. The throughput of the system with a YCSB write-heavy workload (50% reads and 50% writes) is around 3,000 TPS [35]. Note that a similar YCSB workload is used to evaluate Veritas, BigchainDB, and BlockchainDB [24].

Blockchain Relational Database (**BRD**) [33] has a similar design to Veritas, but it starts from a PostgreSQL [7] relational database. In this sense, BRD is an out-of-database blockchain. Also, different from Veritas, the broadcasting service orders blocks of transactions (updates) and does not serialize the transactions in a block. To speedup transaction execution, BRD implements concurrent execution with Serializable Snapshot Isolation (SSI). Note that BRD uses PostgreSQL [7] as its local database, which supports Serializable Snapshot Isolation. BRD also uses Apache Kafka as the broadcasting service. Different from Veritas, BRD keeps the ledger in the same relational database, namely PostgreSQL. According to the BRD paper [33], the system achieves a throughput of 2,500 TPS with a key-value workload.

BigchainDB [2] is another out-of-database blockchain. It starts from MongoDB [6], a NoSQL database, used as the local database. By using MongoDB, the main data abstraction in BigchainDB is an asset, represented in JSON format. Otherwise, the transaction lifecycle is similar to the one in Veritas. A transaction is verified locally by a node, then a request is sent to the broadcasting service. BigchainDB uses a BFT consensus middleware as the broadcasting service, namely Tendermint [13]. Once the majority of the nodes agree on the transaction, it is committed in the local database. BigchainDB relies on Tendermint to keep the ledger in the form of a Merkle tree. Our evaluation of the open-source BigchainDB code shows a maximum performance of around 200 TPS with the YCSB workloads.

ChainifyDB [39] is an out-of-database blockchain that starts from a relational database which can be either PostgreSQL or MySQL. Apache Kafka is used for broadcasting the transactions which are SQL statements. The ledger uses a custom representation based on LedgerBlocks. A LedgerBlock contains all the transactions that are part of a block, where a transaction is in its SQL form. Next, the LedgerBlock contains a list of bits representing the successful transactions, a SHA256 hash digest over the data changed by the transactions, and a hash value of the previous LedgerBlock that was added to the ledger. This representation is similar to the one used by Fabric. ChainifyDB achieves a throughput of around 1,000 TPS on three nodes using the SmallBank workload when all three nodes need to reach consensus. When only two out of three nodes need to reach consensus, the throughput increases to around 5,000 TPS [39].

3.4 Ledger Databases

Amazon Quantum Ledger Database **QLDB** [9] is a verifiable database developed by Amazon and provided as a cloud service. QLDB follows the structure depicted in Figure 1 by integrating a relational database and a ledger in its server node. The database keeps the current states and the history of those states, while the ledger is an append-only journal that keeps track of all the changes done to the database in an immutable way. While it is not clear what is the underlying database, the ledger in QLDB is implemented based on Merkle trees. Our preliminary evaluation of QLDB shows a throughput of 10,000 TPS, which is relatively low for a centralized system. However, we note that an update in QLDB changes both the database and the ledger, and these two changes are done sequentially.

Table 3: Summary of Systems, Features, and Performance

System	Broadcasting Service	Ledger Structure	Local Database	Throughput [TPS]
Fabric [4]	Raft (CFT)	Linked Blocks	LevelDB	1,000 [31]
Quorum [3]	Raft (CFT)	Merkle Patricia Trie	LevelDB	250 [31]
Corda [27]	Raft (CFT)	Merkle Tree	H2	*10* [26]
Diem [10]	LibraBFT (BFT)	Jellyfish Merkle Tree	RocksDB	*600* [47]
Veritas [25]	Kafka (CFT)	Sparse Merkle Tree	Redis	30,000 [24]
BlockchainDB [19]	PoW/PoA (BFT)	Merkle Patricia Trie	Ethereum(LevelDB)	50 [24]
FalconDB [35]	Tendermint (BFT)	Merkle Tree(IntegriDB)	MySQL	*3,000* [35]
BRD [33]	Kafka (CFT)	Relational	PostgreSQL	*2,500* [33]
BigchainDB [2]	Tendermint (BFT)	Merkle Tree(Tendermint)	MongoDB	200 [24]
ChainifyDB [39]	Kafka (CFT)	LedgerBlock	PostgreSQL/MySQL	*1,000* [39]
QLDB [9]	N/A	Merkle Tree	N/A	10,000
LedgerDB [10]	Master-Workers (CFT)	Merkle Tree	L-Stream	20,000
SQL Ledger [11]	N/A	Merkle Tree	SQL Server	*70,000* [11]
Spitz [14]	2PC + timestamp (CFT)	Merkle Tree	ForkBase	*70,000* [14]

LedgerDB [10] is a verifiable database developed by Alibaba and provided as a cloud service. LedgerDB updates the ledger, which is based on a Merkle tree, asynchronously. Specifically, the transactions are batched and the Merkle tree is updated with the batched transactions. Hence, this approach is called batch accumulated Merkle tree (bAMT). LedgerDB supports multiple underlying storage engines, but L-Stream, a custom storage developed by Alibaba, is the default one. L-Stream is an append-only filesystem created specifically for LedgerDB. In terms of distributed architecture, the server nodes in LedgerDB are coordinated by a master that ensures CFT and workload balancing. Our preliminary evaluation of LedgerDB shows a throughput of 20,000 TPS, two times higher compared to QLDB.

SQL Ledger [11] is a ledger database developed by Microsoft and offered as a service on its Azure cloud. It has a similar architecture to QLDB and LedgerDB, but it uses Microsoft's SQL Server as the underlying storage engine. SQL Ledger keeps a ledger data structure based on Merkle trees and two tables, namely, the Ledger Table and the History Table. The Ledger Table reflects the latest record for a given key, while the History Table records the previous version of that record. It is not clear what type of consensus is used to coordinate among multiple nodes in SQL Ledger. Moreover, the reported evaluation was done on a single server with 72 cores [11]. In this evaluation, SQL Ledger achieves a throughput of 70,000 TPS with TPC-C workloads. It is expected to see lower SQL Ledger performance in a distributed setting.

Spitz [14] is a verifiable database that uses ForkBase [43] at the storage level. The authors identify the source of low performance in the other systems as being the existence of separate sub-systems for the ledger and database. Hence, Spitz relies on ForkBase for both the ledger and database. Spitz consists of multiple transaction processing nodes and a common ForkBase backend. The processing nodes coordinate via a two-phase commit (2PC) protocol. A global timestamp service is used to ensure the order of the transactions. Hence, Spitz is a CFT system. The ledger is implemented in ForkBase with the help of a data structure inspired by Merkle trees. Spitz is evaluated on a key-value store application and it achieves up to 70,000 TPS on write operations with 10,000 records. The performance degrades to about 10,000 TPS with more than one million records [14].

3.5 Summary

We summarize the features and the performance of the systems analyzed in this paper in Table 3. In this table, the values in italic are taken from their respective papers, while the other values are based on our measurements [24, 31]. Note that for ledger databases, the *Broadcasting Service* feature is not accurate. However, we list the mechanisms used by the systems for coordination among distributed nodes under this feature. In the next section, we present the limitations of current systems and some challenges in designing and implementing systems that combine blockchain and database features.

4 Challenges and Limitations

When analyzing the systems that combine blockchain and database features, we observe the lack of open-source code for most of the hybrid blockchain databases and ledger databases. Hence, it is difficult to understand the exact implementation and to assess the performance of these systems. In our previous work [24], we re-implemented Veritas and BlockchainDB in a modular way that allows us to replace some of the components, such as the consensus mechanism and the local database. However, more needs to be done to achieve an open-source, flexible and modular hybrid blockchain database system where the consensus and the underlying database can be replaced in a plug and play manner.

At the same time, such systems should offer both key-value and relational interfaces to the users. We note that most of the existing systems offer simple key-value interfaces, with the exception of FalconDB, ChainifyDB, QLDB, and SQL Ledger. It remains to be analyzed what is the impact of having a flexible user interface on performance. For example, what is the impact of having a relational interface when the underlying database is NoSQL? For such designs, the server node needs to be flexible enough and yet exhibit good performance.

Regardless of a BFT or CFT consensus, sharding could be used to improve the scalability and performance of such hybrid systems. So far, only BlockchainDB considers sharding, but its use of blockchain as the underlying storage hinders its performance. In one of our previous works [18], we used sharding to scale Fabric v0.6 in a Byzantine environment. We have shown that Fabric can scale to around 1,000 nodes distributed world-wide, while achieving a performance of around 4,000 TPS. However, sharding comes with the downside of managing cross-shard transactions which have a negative impact on performance.

Last but not least, the effect of newer BFT consensus protocols or optimizations should be evaluated. The existing systems either use a version of PBFT [23] or Tendermint [13]. New consensus frameworks, such as HotStuff [45], Basil [41], and Leopard [28] among others, are claiming much higher throughput compared to PBFT. It remains to be analyzed if such systems can improve the performance of blockchains or hybrid systems. For example, Hotstuff claims more than 100,000 operations per second, while our evaluation of Veritas with Apache Kafka exhibits 30,000 TPS. If we replace Kafka with Hotstuff, can we achieve at least the same performance of 30,000 TPS?

5 Conclusions

In this paper, we analyzed systems that combine both blockchain and database techniques. We classify these systems into three categories, namely, (i) permissioned blockchains, (ii) hybrid blockchain database systems, and (iii) ledger databases. While sharing a similar architecture, each category and each system in a category has its own particularities. We then analyzed a few representative systems, such as Fabric [4], Quorum [3], Veritas [25], QLDB [9], and LedgerDB [10], among others. The exact performance of these systems is hard to evaluate due to the lack of open-source code. On the other hand, existing implementations are not flexible and modular enough. By designing and implementing a modular system where the user interface, consensus, and local storage are plug and play, we could answer more of the existing questions. For example, can we replace a CFT broadcasting framework with a newer BFT consensus framework while experiencing no performance loss? Such questions remain to be answered in the future.

Acknowledgements

This research/project is supported by the National Research Foundation, Singapore under its Emerging Areas Research Projects (EARP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. We would like to thank Prof. Beng Chin Ooi, Prof. Tien Tuan Anh Dinh, Dr. Pingcheng Ruan, Tianwen Wang, and Cong Yue for their help with this project.

References

[1] Apache Kafka, `https://kafka.apache.org/`, 2017.

[2] BigchainDB 2.0 The Blockchain Database, Technical report, 2018.

[3] GoQuorum, `https://github.com/ConsenSys/quorum`, 2021.

[4] Hyperledger Fabric, `https://www.hyperledger.org/use/fabric`, 2021.

[5] immudb, `https://codenotary.io/technologies/immudb/`, 2021.

[6] MongoDB, `https://www.mongodb.com/`, 2021.

[7] PostgreSQL, `https://www.postgresql.org/`, 2021.

[8] Amazon Quantum Ledger Database (QLDB), `https://aws.amazon.com/qldb/`, 2022.

[9] Hyperledger Fabric Ledger, `https://archive.ph/edzMi`, 2022.

[10] Z. Amsden, et al., The Diem Blockchain, `https://archive.ph/1xfcy`, 2021.

[11] P. Antonopoulos, R. Kaushik, H. Kodavalla, S. Rosales Aceves, R. Wong, J. Anderson, J. Szymaszek, *SQL Ledger: Cryptographically Verifiable Data in Azure SQL Database*, page 2437–2449, 2021.

[12] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, A. Sonnino, State Machine Replication in the Libra Blockchain, `https://archive.ph/Uxlb3`, 2019.

[13] E. Buchman, *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*, PhD thesis, The University of Guelph, 2016.

[14] V. Buterin, A Next-Generation Smart Contract and Decentralized Application Platform, `http://archive.fo/Sb4qa`, 2013.

[15] J. Carlson, *Redis in Action*, Manning Shelter Island, 2013.

[16] M. Castro, B. Liskov, Practical Byzantine Fault Tolerance and Proactive Recovery, *ACM Transactions on Computer Systems (TOCS)*, 2002.

[17] J. A. Chacko, R. Mayer, H.-A. Jacobsen, *Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric*, page 221–234, 2021.

[18] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, B. C. Ooi, Towards Scaling Blockchain Systems via Sharding, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 123–140, 2019.

[19] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, J. Wang, Untangling Blockchain: A Data Processing View of Blockchain Systems, *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1366–1385, 2018.

[20] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, K.-L. Tan, BLOCKBENCH: A Framework for Analyzing Private Blockchains, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 1085–1100, 2017.

[21] S. Dong, A. Kryczka, Y. Jin, M. Stumm, RocksDB: Evolution of Development Priorities in a Key-Value Store Serving Large-Scale Applications, *ACM Trans. Storage*, 17(4), oct 2021.

[22] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, R. Ramamurthy, Blockchaindb: A Shared Database On Blockchains, *Proc. VLDB Endow.*, 12(11):1597–1609, 2019.

[23] Z. Gao, Y. Hu, Q. Wu, Jellyfish Merkle Tree, `https://archive.ph/s7pPF`, 2019.

[24] Z. Ge, D. Loghin, B. C. Ooi, P. Ruan, T. Wang, Hybrid Blockchain Database Systems: Design and Performance, *Proc. VLDB Endow.*, 15(5):1092–1104, 2022.

[25] J. Gehrke, L. Allen, P. Antonopoulos, A. Arasu, J. Hammer, J. Hunter, R. Kaushik, D. Kossmann, R. Ramamurthy, S. T. V. Setty, J. Szymaszek, A. van Renen, J. Lee, R. Venkatesan, Veritas: Shared Verifiable Databases and Tables in the Cloud, *Proc. of 9th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2019.

[26] R. Han, G. Shapiro, V. Gramoli, X. Xu, On the Performance of Distributed Ledgers for Internet of Things, *Internet of Things*, 10:100087, 2020, Special Issue of the Elsevier IoT Journal on Blockchain Applications in IoT Environments.

[27] M. Hearn, R. G. Brown, Corda: A Distributed Ledger, `https://bit.ly/3iLajrI`, 2019.

[28] K. Hu, K. Guo, Q. Tang, Z. Zhang, H. Cheng, Z. Zhao, Leopard: Towards High Throughput-Preserving BFT for Large-scale Systems, 2021.

[29] L. Liu, M. T. Özsu, editors, *Merkle Trees*, pages 1714–1715, Springer US, 2009.

[30] D. Loghin, G. Chen, T. T. A. Dinh, B. C. Ooi, Y. M. Teo, Blockchain Goes Green? An Analysis of Blockchain on Low-Power Nodes, 2019.

[31] D. Loghin, T. T. A. Dinh, A. Maw, C. Gang, Y. M. Teo, B. C. Ooi, Blockchain Goes Green? Part II: Characterizing the Performance and Cost of Blockchains on the Cloud and at the Edge, 2022.

[32] S. Nakamoto, Bitcoin: A Peer-to-peer Electronic Cash System, http://archive.fo/CIl1Y, 2008.

[33] S. Nathan, C. Govindarajan, A. Saraf, M. Sethi, P. Jayachandran, Blockchain Meets Database: Design And Implementation Of A Blockchain Relational Database, *Proc. VLDB Endow.*, 12(11):1539–1552, 2019.

[34] D. Ongaro, J. Ousterhout, In Search of an Understandable Consensus Algorithm, *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, 2014. USENIX Association.

[35] Y. Peng, M. Du, F. Li, R. Cheng, D. Song, Falcondb: Blockchain-Based Collaborative Database, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 637–652, 2020.

[36] P. Ruan, T. T. A. Dinh, D. Loghin, M. Zhang, G. Chen, Q. Lin, B. C. Ooi, Blockchains vs. Distributed Databases: Dichotomy and Fusion, *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 1–14, 2021.

[37] P. Ruan, D. Loghin, Q.-T. Ta, M. Zhang, G. Chen, B. C. Ooi, A Transactional Perspective on Execute-Order-Validate Blockchains, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 543–557, 2020.

[38] R. Saltini, D. Hyland-Wood, Correctness Analysis of IBFT, 2019.

[39] F. M. Schuhknecht, A. Sharma, J. Dittrich, D. Agrawal, ChainifyDB: How to get rid of your Blockchain and use your DBMS instead, *Proc. of 11th Conference on Innovative Data Systems Research (CIDR)*, 2021.

[40] A. Sharma, F. M. Schuhknecht, D. Agrawal, J. Dittrich, Blurring the Lines between Blockchains and Database Systems: The Case of Hyperledger Fabric, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 105–122, 2019.

[41] F. Suri-Payer, M. Burke, Z. Wang, Y. Zhang, L. Alvisi, N. Crooks, Basil: Breaking up BFT with ACID (Transactions), *Proc. of ACM SIGOPS 28th Symposium on Operating Systems Principles*, page 1–17, 2021.

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

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

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

[45] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, I. Abraham, HotStuff: BFT Consensus with Linearity and Responsiveness, *Proc. of 2019 ACM Symposium on Principles of Distributed Computing*, page 347–356, 2019.

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

[47] J. Zhang, J. Gao, Z. Wu, W. Yan, Q. Wu, Q. Li, Z. Chen, Performance Analysis of the Libra Blockchain: An Experimental Study, 2019.

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

[49] Y. Zhang, J. Katz, C. Papamanthou, IntegriDB: Verifiable SQL for Outsourced Databases, *Proc. of 22nd ACM SIGSAC Conference on Computer and Communications Security*, page 1480–1491, 2015.

LEDGERBENCH: A Framework for Benchmarking Ledger Databases

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

Cong Yue
National University of Singapore
yuecong@comp.nus.edu.sg

Changhao Zhu
Beijing Institute of Technology
zhuchanghao@bit.edu.cn

Ziyue Zhong
Beijing Institute of Technology
ziyue_zhong@bit.edu.cn

Abstract

Ledger databases protect the integrity of data, history and query results. However, existing ledger databases adopt different design choices, and there is a lack of benchmarking tools to evaluate them comprehensively. Therefore, it is unclear how each design choice performs and it is difficult for users to choose a system that is appropriate for their use case in practice. In this paper, we first conduct a survey on the designs of existing systems. We categorize the design of ledger database into four components and discuss the design choices of each component. Based on the study, we then outline LEDGERBENCH*, a benchmarking framework for ledger databases, for both macro- and micro-benchmarking. To evaluate the system performance of ledger databases,* LEDGERBENCH *provides macro-benchmarks consisting of verification-aware workloads adapted from Smallbank. To evaluate the performance of each component of a system, it provides micro-benchmarks with respect to verification, verification delay, auditing and storage. Lastly, we conduct comprehensive experiments on existing ledger databases. From the results, we observe that updating the ledger structure and verification is the main bottleneck of ledger databases. The cost can however be significantly reduced by adopting deferred verification and carefully crafted asynchronous ledger update functions that enforce minimum locking.*

1 Introduction

With the extensive application of cloud computing, outsourced databases and collaborative applications involving multiple parties, users of applications are exposed to various threats caused by the malicious behaviours of outside adversaries and untrusted third-party service providers. There is an increasing demand to protect data security.

In recent years, the ledger database is gaining attention in protecting the integrity of data, history and query results. It maintains data or log in the form of append-only ledger, which can generate proofs for users to verify the integrity. Compared to conventional databases, a ledger database has several advantages. 1) It provides

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

efficient verification. The users only need to maintain a cryptographic hash, called digest of the ledger, and integrity check can be performed by comparing the digest against the reproduced hash computed from the proof. 2) It reduces the surface of misbehavior. Since the ledger is immutable, all historical data is protected by the digest. Adversaries cannot tamper with the history without changing the hashes. 3) It can be publicly verified. Everyone can verify the integrity of the data or the log with the ledger. Compared to blockchains, ledger databases offer high performance without the performance bottleneck at consensus layer. Ledger databases are therefore suitable for maintaining financial transactions, logistic orders, healthcare data, etc., where the integrity of data evolution history and proof of data lineage are important.

There are several types of systems that expose ledger APIs, i.e., the system maintains the data and full history in an append-only hash protected authenticated data structure, and can generate proof of each read, update, insert and delete operation to verify the integrity. Recently, a bunch of commercial ledger databases has been offered by major service providers, e.g., QLDB [9], LedgerDB [10], and SQL Ledger [11]. The systems build Merkle tree variances over the transaction logs, and data in the transaction logs are materialized to index structures with meta-information of the log entry. Hence, all queried data can be verified using the log. Another type of systems is certificate transparency [4, 6, 7, 8], where public keys and certificates are protected by Merkle tree variances. Consistency checks can be performed using Merkle proof. In addition, blockchains [1, 2, 3] also expose ledger API. The systems maintain a sequence of hash-chained blocks, and the integrity is guaranteed by replicating the blocks using a byzantine fault tolerant protocol [23]. However, they are not our focus due to having a different threat model and design space, and suffering from low performance.

Despite the development of ledger databases, there is a lack of benchmarking tools to systematically and fairly evaluate existing systems. Existing key-value and OLTP workloads are unaware of ledger-related functions such as verification and auditing, and therefore, cannot offer accurate analysis of ledger databases. In this paper, we outline LEDGERBENCH, a benchmarking framework for ledger databases. We first conduct a survey on the designs of existing systems. We categorize the design of ledger database into four components, namely ledger structure, query processing, verification, and auditing, and discuss the design choices of each component. We then design LEDGERBENCH to evaluate on all four components with micro-benchmarks and the system performance with macro-benchmarks consisting of verification-aware OLTP workloads adapted from Smallbank. Lastly, we re-construct existing ledger systems based on our understanding of the systems in a consistent manner. We subsequently conduct extensive experiments on existing ledger databases to illustrate the strengths and weaknesses of each design and system, which may be useful for the future development of ledger databases.

In summary, we make the following contributions.

- We outline LEDGERBENCH, the first benchmark for databases that expose ledger APIs. The benchmark is designed to evaluate the performance on all aspects of a ledger database.
- We implement an open source LEDGERBENCH, which can be used or further developed to evaluate other ledger databases.
- We conduct an extensive evaluation of re-constructed QLDB [9], LedgerDB [10], and SQL Ledger [11]. Our results pinpoint the performance bottleneck of existing ledger databases, and show the advantage of various design choices.

The remaining of the paper is organized as follows. Section 2 briefly surveys the existing ledger databases. Section 3 explains the design of LEDGERBENCH. Section 4 presents the performance evaluation of four representative systems using LEDGERBENCH. Section 5 discusses the related work before Section 6 concludes.

2 Ledger Databases

In this section, we will describe the details of ledger databases and the design choices of existing systems.

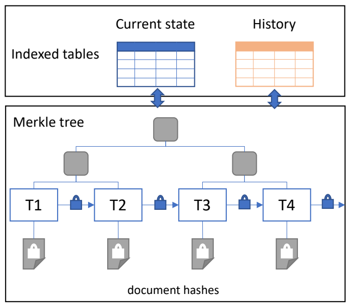


Figure 1: QLDB architecture

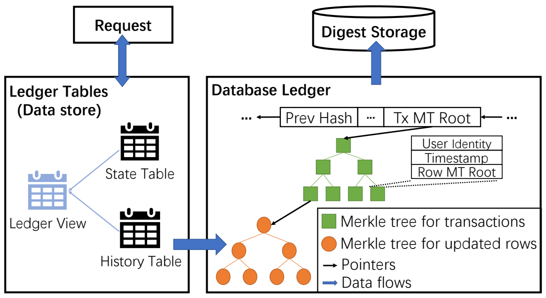


Figure 2: SQL Ledger architecture

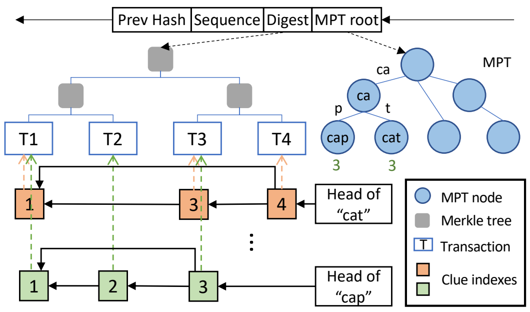


Figure 3: LedgerDB architecture

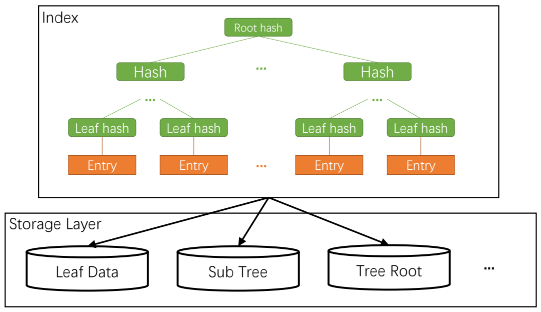


Figure 4: Trillian architecture

2.1 Threat Model

Ledger databases adopt a threat model assuming that a single-party service provider can be malicious or compromised. Users are only able to detect the malicious behaviors instead of preventing them. Many systems also rely on a group of trusted auditors to help verify the data integrity and notify the users of malicious behaviours, and therefore, relieve the burden on users.

2.2 Ledger Structure

Ledger is the key data structure of ledger databases. It maintains all current and historical data in an authenticated data structure (ADS), where proofs can be generated for verifying the data integrity. The ADS is usually a Merkle variance. For example, QLDB builds a Merkle tree over hashes of all data as shown in Figure 1. SQL Ledger constructs a Merkle tree over the modified data for each transaction, and another Merkle tree over the transaction entries batched in a block. The root hash of the latter Merkle tree is stored in the block entry with the previous block entry's hash to form a hashed chain as shown in Figure 2. LedgerDB adopts a batched accumulated Merkle tree, which adopts copy-on-write when new transactions are appended to reduce the contention as shown in Figure 3. Trillian shown in Figure 4 adopts a sparse Merkle tree to store the data, therefore, it does not allow different versions of the same key. CONIKS stores the data in Merkle prefix trees, the root hashes of which are linked in a linear hash chain as shown in Figure 5. To improve the efficiency of verification for the latest versions, Merkle^2 shown in Figure 6 constructs a forest of full Merkle trees over data in chronological order, and each internal node contains the root hash of a prefix tree built over data in lexical order.

2.2.1 Chronological order vs. lexical order

Systems such as QLDB, LedgerDB, and SQL Ledger construct ADS over data in chronological or transaction order. The ADS is used only for integrity proof, and separate index structures are required to query the data.

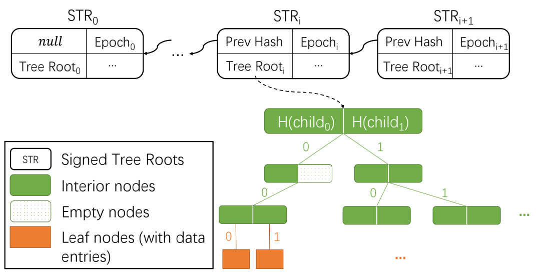


Figure 5: CONIKS architecture

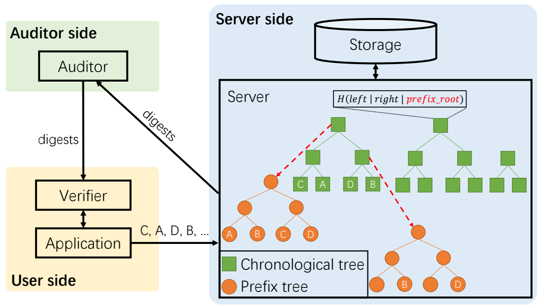


Figure 6: Merkle^2 architecture

This causes two problems: 1) updating and proof generation become slower when the ADS grows larger; 2) It requires additional protection of the indexes, e.g., LedgerDB uses a Merkle Patricia Trie(ccMPT) to protect its clue indexes, while the index tables in QLDB and SQL Ledger are not hash protected, rendering its inability to guarantee that the value is the latest. While systems like Trillian, CONIKS, and Merkle^2 embed additional Merkle variance in lexical order, which could serve as protected indexes of data.

2.3 Query Processing

2.3.1 Abstraction

Certificate transparency logs, for the purpose of storing keys and certificates, usually expose simple key-value store abstractions. While commercial ledger databases, which target on exchanging of assets or general business logic, support transaction with ACID properties. In particular, QLDB and LedgerDB directly build the ledger over the transaction logs. Alternatively, SQL Ledger commits the transaction to a separate transaction log for failure recovery, and creates the ledger at a later stage.

2.3.2 Batching

The update of the ledger is expensive since it involves calculating intensive cryptographic hashes and incurs high contention, especially for the root node, making it hard for parallelism. To reduce the hash calculation and mitigate the read/write contention against proof generation operations, LedgerDB updates its ADSs by batches of transactions and adopts copy-on-write when updating the ADSs. SQL Ledger constructs an individual Merkle tree for each transaction and block, making it contention-free when appending new blocks to the ledger. Similar to LedgerDB, SQL Ledger batches multiple transactions in a block to reduce the overhead of calculating block-level hashes.

2.4 Verification

To guarantee the integrity of data and query results, ledger databases provide proofs that can be publicly verified by the users or third-party verifiers for each request. The proofs of a ledger database typically consist of a digest which is a hash summary of the entire ledger structure, a proof containing the hashes of the Merkle tree nodes along the path from the root to the leaf node the data located. The client, upon receiving the proof, can reproduce the digest by calculating the hashes recursively. It then verifies the integrity of data by comparing the original and reproduced digest.

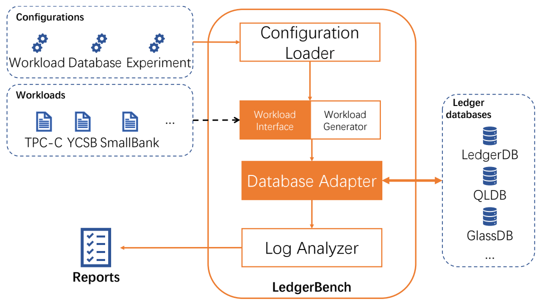


Figure 7: LEDGERBENCH architecture

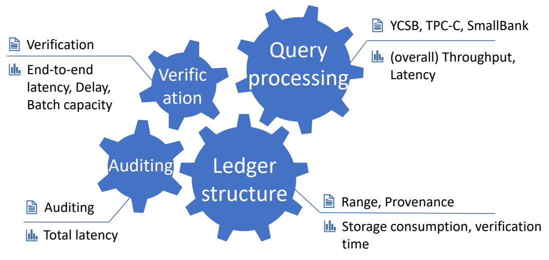


Figure 8: Abstraction of ledger databases and the corresponding workloads and metrics of each component.

2.4.1 Deferred verification

It is expected that providing proof for each operation is costly. LedgerDB [10] and SQL Ledger [11] adopt deferred verification. The system, on completion of the operation, will return a promise containing the data and block sequence the data resides to the users for future verification. The client can batch the proof generation request and verification process for higher performance. However, there is a trade-off between performance and security. There is a verification time window that the integrity of data could be temporarily violated.

2.5 Auditing

Systems such as certificate transparency, LedgerDB, and SQL Ledger rely on auditors to check the consistency of the ledger and detect malicious behaviors. The auditors will rebuild the ledger based on the logs and compare the digest of the rebuilt ledger and that requested from servers. It will notify the users of the misbehavior if any mismatching is found. The auditing process is typically expensive. Any third-party entities or powerful users can play the role of the auditor. The systems require at least one honest auditor so that any malicious behaviours can be detected and notifications can be sent to the users. QLDB does not rely on auditors to check the consistency of the ledger. Consequently, users have to check the consistency of the ledger if required.

3 Design

3.1 Architecture

Figure 7 shows the architecture of LEDGERBENCH. It contains a configuration loader, a workload generator, database adapters, and a log analyzer. The configuration loader loads the parameters of the benchmarks. The workload generator creates tasks by generating the operations and corresponding arguments based on the parameters. It sends the generated tasks to the database adapters through HTTP requests. The database adapters, on receiving the tasks, initialize database clients for execution. It logs the metrics collected, which are later processed by the log analyzer for final results.

Configuration loader. Configuration Loader is responsible for loading all experiment-related configurations at runtime. It reads the parameters from the command line and configuration files. The configurations include workload configurations, e.g., distribution of operations, database configurations, e.g., block time, and experiment configurations, e.g., number of testing threads, request rate, etc.

Workload generator. Workload generator issues the tasks to the database adapter via HTTP. According to the workload configuration, it creates the tasks consisting of the operation type and a list of parameters. Users can control the load by specifying in the experiment configurations the number of nodes and threads to run the workload generators and the request rate of each workload generator. The workload generator is implemented in a way that is easy to extend. It exposes a uniform workload interface, based on which a driver is implemented for each workload type. Users can add customized workloads by simply implementing the workload interface. Depending on the configuration, the workload generator initiates different drivers.

Database adapter. Database adapter maintains a pool of database client instances, which connect to the ledger database being evaluated. Upon receiving the requests from the task generators, database adapters store the tasks in a queue, from which the client instances fetch tasks and send them to the database for execution. Similar to workload generator, it is easy for users to extend new database systems to LEDGERBENCH as it exposes a general interface for each workload operation. Users only need to implement the function with the database clients to be tested. Database adapters can be implemented in any programming language since the tasks are sent via HTTP.

Log analyzer. Measurements are taken and logged before and after the execution of the workload tasks. The log analyzer processes the logs and calculates the required metrics.

3.2 API

In this section, we describe the APIs of the workload generator and database adapter. Users are able to include customized workloads and ledger databases by simply inheriting the interfaces. First, we describe the API of the workload generator as follows.

- `NextTask(conf)`. The interface takes the workload configuration as input and generates the next task including the operation type and a list of parameters. The configuration usually contains the ratio, distribution, and ranges of operations, keys and values. For example, a configuration of Smallbank workload includes $D(operation) = uniform, D(account) = uniform, R(account) = [0, 100000], R(ammount) = [0, 1000]$, where D is the distribution, and R is the range. The interface returns the generated tasks, e.g., $< SendPayment, 1, 2, 200 >$, which represents account 1 paying \$200 to account 2.

Next, we introduce the APIs of the database adapter. Depending on the database under evaluation, the database adapter may be implemented in different programming languages.

- `ExecuteTransaction(task, db)`. The interface takes the task generated by `NextTask` and the database adapter, `db`, as the input and returns the status of execution. The inherit function implemented by the user shall execute the transaction with a sequence of `Put`, `Get`, and `Verify` operations provided by the database adapter.

- `Put(keys, values)`. This interface defines the operation to update or insert a list of keys and values in the database. It will return the proof optionally for databases that do not support deferred verification.

- `Get(keys)`. This interface defines the operation to get the values of a list of keys from the database. It will return the proof optionally for databases that do not support deferred verification.

- `Verify(keys, block_seqs)`. This interface is for deferred verification, where a batch of keys could be verified together. The input of the interface is a list of keys and block sequences the keys are located. The inherit function shall get the proofs from the database, verify the proofs, and return the verification result.

- `Verify(proof)`. This interface is for immediate verification, where the input `proof` can be obtained from the `Get` and `Put` operations. The interface returns the verification result.

To extend LEDGERBENCH with a new workload, users need to implement the `NextTask` and `TransactionX`. To include a new database for evaluation, users need to implement the `Put`, `Get`, and `Verify`.

3.3 Metrics

LEDGERBENCH evaluates ledger databases on their four components described in Section 2. Figure 8 shows the metrics with respect to each component. The query processing component is responsible for query execution and data commitment. It is critical to know how fast the system can process data. Therefore, we measure the system-level throughput and latency of running key-value or OLTP workloads. While for ledger structures, the time- and space-efficiency of the access and verification methods are the key factors. Hence, the storage consumption and execution time of verification performed by the clients are taken as metrics. To evaluate the verification performance, we take the end-to-end latency of user verification requests, the number of keys each verification request will process for a batched verification, and vary the delay time for a deferred verification. Lastly, LEDGERBENCH evaluates the auditing process by taking the latency of auditing a batch of transactions.

3.4 Workloads

In this section, we describe the workloads used in LEDGERBENCH. We include a verification-aware workload adapted from Smallbank. We also implement range query workloads in addition to the point queries covered in Smallbank. Users can easily extend LEDGERBENCH with more workloads with the API described in Section 3.2.

SmallBank. There are two tables in our SmallBank workload, namely saving and checking, to simulate bank services such as querying balances, depositing and transferring money, and amalgamating assets among 100,000 bank accounts. Similarly, we implement all the six transactions in the context of verifiable databases, i.e., each transaction will return the corresponding proof to verify the integrity of this execution.

Range. To evaluate the effectiveness of indexing and block policies of a ledger database, we implement the range query workload. It queries a random range of keys. In our experiments, the queried keys follow a uniform distribution. Besides the corresponding values, we also request a promise, which is used later to validate the integrity of the ledger.

4 Evaluation

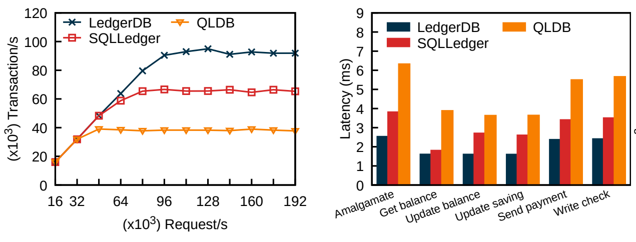
In this section, we benchmark the state-of-the-art ledger databases, namely, QLDB, LedgerDB, and SQL Ledger. QLDB is a commercial product offered by Amazon. LedgerDB is an industrial prototype implemented by Alibaba. SQL Ledger is a commercial product provided by Microsoft. We evaluate the systems with both macro-benchmarks and micro-benchmarks. Since there are no source codes for the systems, we re-implement the systems based on the online documentation and paper. We conduct the experiments on 24 machines equipped with 10×2 Intel Xeon W-1290P processors, 128GB RAM, and 10 Gbps Ethernet. We start 16 server nodes and 8 client nodes. Each client node will run 20 client processes.

4.1 Macro Benchmarks

This section evaluates the performance of an end-to-end query processing with Smallbank, and range workloads.

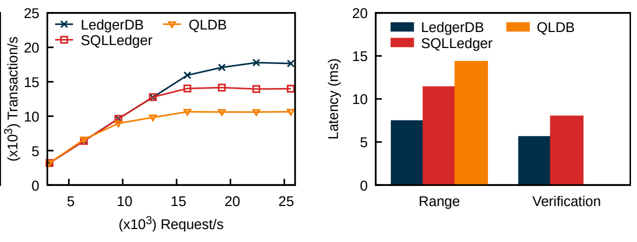
4.1.1 Smallbank

We initialize 100,000 users with 1000 dollars in their saving account and 50 dollars in their balance account. The experiment is conducted by feeding the system with transactions that consist of one of the six operations with equal probability. We first vary the total request rates from 16,000 to 192,000. Figure 9b shows the throughputs of the system with increasing client requests. LedgerDB outperforms SQL Ledger by up to $1.4\times$, and outperforms QLDB by $2.4\times$. LedgerDB and SQL Ledger outperform QLDB by adopting asynchronous ledger updates and deferred verification. In contrast, QLDB incurs significant overhead updating the ledger for each commit and



(a) Throughput (b) Latency

Figure 9: Performance for Smallbank workloads



(a) Throughput (b) Latency

Figure 10: Performance for range workload.

performing verification after each operation. LedgerDB performs better than SQL Ledger because the clue indexes used to access the data are built over versions, and therefore, much smaller compared to the ledger and history table indexes in SQL Ledger. The heads of the clue indexes are stored in memory for executing queries efficiently. Figure 9a shows the average latency of specific operations. The results show similar trends that LedgerDB has the lowest latency, while QLDB has the highest latency.

4.1.2 Range

We conduct the experiment with a dataset consisting of 100,000 keys. The ranges are selected with random sizes from 10 keys to 20 keys. The results are depicted in Figure 10. LedgerDB has the highest throughput. It builds a separate clue index for each key and requires scanning the head of all clue indexes when processing the range query. The overhead is mitigated since all clue index heads are stored in memory. SQL Ledger requires fetching additional metadata for future verification. The verification process is more costly due to the need of scanning the blocks. QLDB performs the worst as it verifies the data after each operation.

4.2 Micro Benchmarks

In this section, we evaluate each component of the systems with micro-benchmarks.

4.2.1 Verification

We evaluate the execution time and proof size for verifying one key from the client side, and show the results in Figure 11. LedgerDB has the highest proof size and verification time. This is because the verification of clue indexes is expensive. In particular, users need to verify the ccMPT to validate the number of entries stored in the clue index, and then verify each entry on the ledger. Though such tasks can be delegated to the auditors, users cannot guarantee the clue indexes are not tampered with after the auditing process. The proof sizes of QLDB and SQL Ledger are small, since they only contain a list of hashes on a path of the Merkle tree. However, QLDB and SQL Ledger fail to guarantee that the fetched data is the latest.

4.2.2 Delay

In this section, we evaluate how delay time affects the performance. We use a range of delay time from 10ms to 1280ms, and depict results in Figure 12. We can observe that the performance of all systems increases as the delay time increases in the beginning. This is because a higher delay time will result in a larger verification batch, which allows the servers to provide more efficient batched proof. As the delay time keeps increasing, the performance will drop because the overwhelming data cause significant overhead and high contention when generating the proofs.

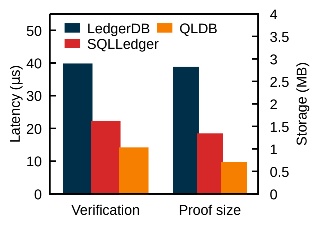


Figure 11: Verification latency and proof size.

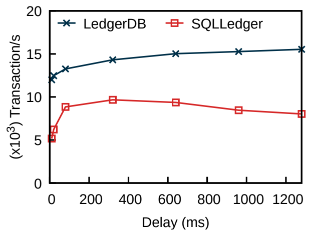


Figure 12: Performance with delay setting.

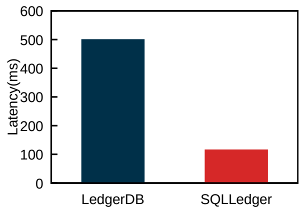


Figure 13: Latency for auditing one block.

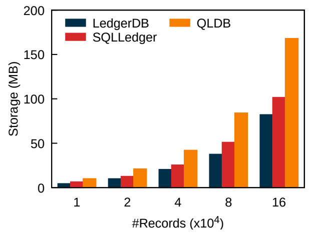


Figure 14: Storage consumption.

4.2.3 Audit

This section evaluates the cost of the auditing process for LedgerDB and SQL Ledger. We start with 16 server nodes and 8 client nodes with each client node running 20 client processes. We feed all clients with a read-write key-value workload. Then we start an audit client executing audit tasks every second. To fairly compare the two systems, we set the block time of both systems to be 10ms, i.e., all transactions within the period are collected to a block and appended to the ledger. The auditor will verify the block of transactions for each auditing request. Figure 13 shows the average latency for auditing one block. Both systems batch around 700 transactions in a block. The latency of LedgerDB is $4\times$ the latency of SQL Ledger. This is because verification for clue indexes is expensive in LedgerDB. Note that SQL Ledger does not verify its ledger table and history table, therefore, the integrity of data cannot be fully guaranteed. Though the queried data can be verified using the proof generated from the ledger, the system can always return stale data and still pass the verification.

4.2.4 Storage

We measure the storage consumption of the systems with respect to the total number of records stored in the systems, from 10,000 to 160,000. LedgerDB and SQL Ledger adopt transaction batching and update the ledger less frequently, and are therefore more space-efficient than QLDB. SQL Ledger consumes slightly more storage compared with LedgerDB because of the additional indexes.

5 Related Works

Existing works have been done to explore the opportunity of fusion designs of blockchain and database [15, 16, 17, 18, 19, 20, 21, 22]. However, the systems above try to enable database's query ability on top of blockchains, therefore suffering from the low performance of blockchains. While the ledger databases [9, 10, 11] use the ledger structures similar to the blockchain and build verifiable database with a central party, therefore, are more practical in processing OLTP workloads. Spitz [14] proposes to integrate a ledger in the HTAP system. This work focuses more on evaluating the ledger databases and their designs, though it can also be used to evaluate blockchains.

There are many existing benchmark frameworks. BlockBench [12] is the first framework to comprehensively evaluate the private blockchains from the application layer, execution engine, data model and consensus. Hyperledger Caliper [13] is another widely-used blockchain benchmark tool, supporting a range of systems such as Hyperledger Fabric [1], Ethereum [2], Fisco-Bcos [25], etc. OLTP-Bench [26] and YCSB [27] are benchmarking tools for relational database and key-value store, respectively. While LEDGERBENCH focuses on the ledger database specific designs to evaluate the systems in every aspect.

6 Conclusions

With the increasing digitization of businesses and cloud hosting, there have been increasing demands for transactions to be verifiable and auditable. Various commercial ledger databases have been designed to meet the demands by supporting the integrity of data, history and query results. We conduct a survey on the designs of some existing systems. We examine the design of these representative ledger databases, identify four major components and discuss the design choices of each component. We then outline LEDGERBENCH, a framework for benchmarking ledger databases, for both macro- and micro-benchmarking. We conduct extensive performance study and identify various bottlenecks and their possible causes. We hope the study and open source of the framework will facilitate further development in this important area.

References

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In EuroSys. 30.

[2] Gavin Wood et al . 2014. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151, 2014 (2014), 1–32.

[3] ConsenSys. 2020. ConsenSys/quorum: A permissioned implementation of Ethereum supporting data privacy. https://github.com/ConsenSys/quorum.

[4] Google. 2020. Certificate Transparency. https://www.certificate-transparency.org/.

[5] Google. 2020. Trillian: general transparency. https://github.com/google/trillian.

[6] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. 2015. CONIKS: Bringing Key Transparency to End Users. In Usenix Security.

[7] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung Jin Yang, Raluca Ada Popa. 2021. Merkle^2: A Low-Latency Transparency Log System. In IEEE Symposium on Security and Privacy.

[8] Mark D. Ryan. 2014. Enhanced Certificate Transparency and End-to-End Encrypted Mail. In NDSS.

[9] Amazon. 2019. Amazon Quantum Ledger Database. https://aws.amazon.com/qldb/.

[10] Xinying Yang, Yuan Zhang, Sheng Wang, Benquan Yu, Feifei Li, Yize Li, and Wenyuan Yan. 2020. LedgerDB: A Centralized Ledger Database for Universal Audit and Verification. PVLDB.

[11] Panagiotis Antonopoulos, Raghav Kaushik, Hanuma Kodavalla, Sergio Rosales Aceves, Reilly Wong, Jason Anderson, Jakub Szymaszek. 2021. SQL Ledger: Cryptographically Verifiable Data in Azure SQL Database. In SIGMOD.

[12] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In SIGMOD.

[13] Hyperledger foundation. 2018. https://www.hyperledger.org/use/caliper.

[14] Meihui Zhang, Zhongle Xie, Cong Yue, and Ziyue Zhong. 2020. Spitz: a verifiable database system. PVLDB.

[15] 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 SIGMOD.

[16] Pingcheng Ruan, Gang Chen, Tien Tuan Anh Dinh, Qian Lin, Beng Chin Ooi, and Meihui Zhang. 2019. Fine-grained, secure and efficient data provenance on blockchain systems. PVLDB.

[17] Johannes Gehrke, Lindsay Allen, Panagiotis Antonopoulos, Arvind Arasu, Joachim Hammer, Jim Hunter, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, Srinath T. V. Setty, Jakub Szymaszek, Alexander van Renen, Jonathan Lee, Ramarathnam Venkatesan.2019. Veritas: Shared Verifiable Databases and Tables in the Cloud. CIDR.

[18] Trent McConaghy, Rodolphe Marques, Andreas Müller, Dimitri De Jonghe, Troy McConaghy, Greg McMullen, Ryan Henderson, Sylvain Bellemare, Alberto Granzotto. 2016. Bigchaindb: a scalable blockchain database. white paper, BigChainDB.

[19] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. 2019. BlockchainDB: a shared database on blockchains. PVLDB.

[20] Zerui Ge, Dumitrel Loghin, Beng Chin Ooi, Pingcheng Ruan, and Tianwen Wang. 2022. Hybrid blockchain database systems: design and performance. PVLDB.

[21] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen. 2022. Blockchains: Decentralized and Verifiable Data Systems. Preprint.

[22] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi and Ji Wang. 2017. "Untangling Blockchain: A Data Processing View of Blockchain Systems," in TKDE.

[23] Miguel Castro, Barbara Liskov. 1999. Practical byzantine fault tolerance. In OsDI.

[24] Kai Mast, Lequn Chen, Emin Gün Sirer. 2018. Enabling strong database integrity using trusted execution environments. arXiv preprint arXiv:1801.01618.

[25] FISCO BCOS. The Building Block of Open Consortium Chain. https://www.fisco-bcos.org/

[26] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. OLTP-Bench: an extensible testbed for benchmarking relational databases. PVLDB.

[27] Brian Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In SoCC.

Open Calls for Nominations

The IEEE Technical Community on Data Engineering (TCDE) Nominations Committee requests nominations for qualified individuals to fill the TCDE Chair position for the 2023-2024 term.

Self-nominations are allowed. Nominations should be submitted to the Nominations Committee chair Erich Neuhold erich.neuhold@univie.ac.at with copies to Amr El Abbadi <amr@cs.ucsb.edu> and Farshad Firouzi <farshad.firouzi@duke.edu>.

The Nominations & Appointments Committee requests the following information regarding each candidate for consideration:

1. Candidate's name and contact information (as it should appear on all election materials), Business Title/Position, Business Name, Business Address, Preferred Phone, Email address, and a short Bio not exceeding 250 words.

2. A brief statement, not to exceed 250 words, on the candidate's prior achievements for the benefit of IEEE CS and IEEE TCDE.

3. A statement of candidacy (not to exceed 250 words) explaining the candidate's plans for using their office to strengthen IEEE TCDE.

Duties from the Technical and Confernce Activities Board Handbook:

The Chair is the leader for TC vitality, and is responsible for building a strong organization and officer structure in support of the TC's mission. The chair is responsible for making sure T&C Activities Board policy is carried out in relation to TC activities, and is the vital communication link between the TC and the T&C Activities Board. It is the responsibility of the Chair to see that the TC operates within T&C Activities Board guidelines, and that any deviations from T&C Activities Board policy are brought to the attention of the T&C Activities Board. Roles specific to the technical committee of interest should be discussed with the current or previous chair, as each TC is different. The Chair is accountable to both TC members and the T&C Activities Board.

The TC Chair has ultimate authority and responsibility for the conduct of a TC. While duties may be delegated to other parties, typically a Vice Chair or Executive Committee members, it is the Chair that bears responsibility for the successful operation of the TC. Only the Vice-President for Technical and Conference Activities or the CS President may overturn the decision of a Chair.

Erich Neuhold, Amr El Abbadi, Farshad Firouzi
TCDE Nominations Committee

Data Engineering

It’s FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name	
IEEE Member #	
Mailing Address	
Country	
Email	
Phone	

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

TCDE Chair

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au