Untangling Blockchain: A Data Processing View of Blockchain Systems

Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, *Member, IEEE*, Gang Chen, *Member, IEEE*, Beng Chin Ooi, *Fellow, IEEE*, and Ji Wang

Abstract—Blockchain technologies are gaining massive momentum in the last few years. Blockchains are distributed ledgers that enable parties who do not fully trust each other to maintain a set of global states. The parties agree on the existence, values, and histories of the states. As the technology landscape is expanding rapidly, it is both important and challenging to have a firm grasp of what the core technologies have to offer, especially with respect to their data processing capabilities. In this paper, we first survey the state of the art, focusing on private blockchains (in which parties are authenticated). We analyze both in-production and research systems in four dimensions: distributed ledger, cryptography, consensus protocol, and smart contract. We then present BLOCKBENCH, a benchmarking framework for understanding performance of private blockchains against data processing workloads. We conduct a comprehensive evaluation of three major blockchain systems based on BLOCKBENCH, namely Ethereum, Parity, and Hyperledger Fabric. The results demonstrate several trade-offs in the design space, as well as big performance gaps between blockchain and database systems. Drawing from design principles of database systems, we discuss several research directions for bringing blockchain performance closer to the realm of databases.

Index Terms—Blockchain, distributed databases, survey

1 Introduction

Blockchain technologies are taking the world by storm, largely due to the success of Bitcoin [1]. A blockchain, also called distributed ledger, is essentially an append-only data structure maintained by a set of nodes which do not fully trust each other. Nodes in the blockchain agree on an ordered set of blocks, each containing multiple transactions, thus the blockchain can be viewed as a log of ordered transactions. In the database context, blockchain can be viewed as a solution to distributed transaction management: nodes keep replicas of the data and agree on an execution order of transactions. However, traditional databases assume a trusted environment and employ well known concurrency control techniques [2], [3], [4] to order transactions. Blockchain's key property is that it assumes nodes behave in arbitrary (or Byzantine) manner. Being able to tolerate Byzantine failure by design, blockchain offers stronger security than incumbent database systems.

In the original design, Bitcoin's blockchain stores *coins* as the system states. For this application, Bitcoin nodes implement a simple replicated state machine model which moves coins from one address to another. Since then, blockchain has grown beyond crypto-currencies to support user-defined states and Turing complete state machine models. For example, Ethereum [5] enables any decentralized, replicated applications known as *smart contracts*. More importantly, interest from the industry has started to drive development of new blockchain platforms designed for private settings where participants are authenticated. Blockchain systems in such environments are called private (or *permissioned*), as opposed to the early systems operating in public environments (or *permissionless*) where anyone can join and leave. Applications such as security trading and settlement [6], asset and finance management [7], [8], banking and insurance [9] are being built and evaluated. These applications are currently supported by enterprise-grade database systems like Oracle and MySQL, but blockchain has the potential to disrupt this status quo because it incurs lower infrastructure and human costs [9]. In particular, blockchain's immutability and transparency help reduce human errors and the need for manual intervention due to conflicting data. Blockchain can help streamline business processes by removing duplicate efforts in data governance. Goldman Sachs estimated 6 billion USD saving in current capital market [9], and J.P. Morgan forecast that blockchains will start to replace currently redundant infrastructure by 2020 [8].

Amid the growing commercial and academic interest, a large number of blockchain systems have sprung up, each claiming some unique capabilities. Both private and public sector are clamoring to adopt blockchains, but they face overwhelming choices. While challenging, it is important to have a firm grasp on what the technology can and cannot

- *T. T. A. Dinh, R. Liu, B. C. Ooi, and J. Wang are with the Department of Computer Science, National University of Singapore, Singapore 119077, Singapore. E-mail: {dinhtta, liur, ooibc, wangji}@comp.nus.edu.sg.*
- *M. Zhang is with the Beijing Institute of Technoloy, Haidian Qu, Beijing 100081, China. E-mail: meihui_zhang@bit.edu.cn.*
- *G. Chen is with Zhejiang University, Hangzhou, Zhejiang 310000, China. E-mail: cg@zju.edu.cn.*

Manuscript received 16 Aug. 2017; revised 7 Nov. 2017; accepted 18 Nov. 2017. Date of publication 4 Jan. 2018; date of current version 1 June 2018.
(Corresponding author: Meihui Zhang.)
Recommended for acceptance by X. Lin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2781227

do. A quest for understanding blockchain must ultimately answer the following questions:

1) What is a blockchain? Specifically, what are its unique properties that benefit current and future applications?
2) How do current blockchains differ from each other, both qualitatively in the design and quantitatively in their performance?
3) What are the current challenges? And what do future blockchains look like?

To answer these questions, in this paper, we start by distinguishing two major classes of blockchain systems, namely public and private blockchains. We then explain four key technical concepts by which current systems can be categorized: distributed ledger, cryptography, consensus protocol and smart contract. Next, we describe BLOCKBENCH [10], our benchmarking framework for quantitatively evaluate and compare private blockchains. Using BLOCKBENCH, we conduct comprehensive evaluation of three major blockchains: Ethereum [5], Parity [11] and Hyperledger [12]. The results show that current blockchains' performance is limited, far below what a state-of-the-art database system can offer. Finally, we draw from our experience in building large-scale database systems several design principles that can improve future blockchains.

Other in-depth surveys of blockchain systems [13], [14] focus on crypto-currencies, which is an important blockchain application but it does not fully represent the potential of blockchain as a data processing platform. Our work presents a deeper, and broader view of the technology, with special emphasis on its performance. In summary, our contributions are:

1) We provide an in-depth survey of blockchain systems. We discuss state of the art, and categorize current systems along four dimensions: distributed ledger, cryptography, consensus protocol and smart contract.
2) We describe our benchmarking framework, BLOCKBENCH, that is designed for understanding performance of private blockchains against data processing workloads.
3) We present a comprehensive evaluation of Ethereum, Parity and Hyperledger. The results show the limitation of blockchains as data processing platforms. They identify several performance bottlenecks, and therefore can serve as a baseline for future blockchain research and development.

In the next section, we provide an overview of blockchain systems, separating them into public and private settings. Section 3 explains the four building blocks which are used in Section 4 to categorize existing blockchains. Section 5 describes BLOCKBENCH, followed by the evaluation of three blockchains in Section 6. Section 7 discusses a number of lessons learned from the performance study, and how to bring design principles from databases to improve blockchains. Section 8 concludes.

2 Blockchains: Private versus Public

A typical blockchain system consists of multiple nodes which do not fully trust each other. Some nodes exhibit Byzantine behavior, but the majority is honest. Together, the nodes maintain a set of shared, global states and perform transactions modifying the states. Blockchain is a special data structure which stores historical states and transactions. All nodes in the system agree on the transactions and their order. Fig. 1 shows the blockchain data structure, in which each block is linked to its predecessor via a cryptographic pointer, all the way back to the first (genesis) block. Because of this, blockchain is often referred to as a distributed ledger.

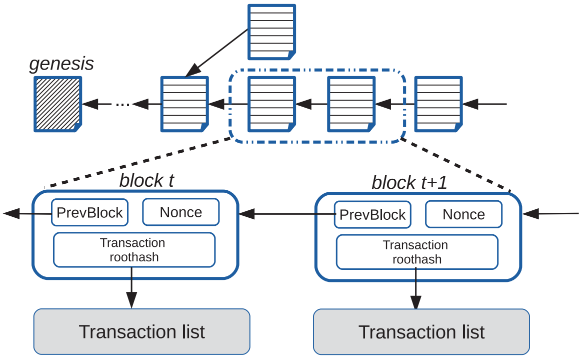


Fig. 1. Blockchain data structure. Transactions are packed into blocks which are linked to previous blocks.

A transaction in a blockchain is the same as in traditional databases: a sequence of operations applied on some states. As such, the blockchain transaction requires the same ACID semantics. The key difference is the failure model under consideration. Current transactional, distributed databases [15], [16] employ classic concurrency control techniques such as two-phase commit to ensure ACID. They can achieve high performance, because of the simple crash failure model. In contrast, the original blockchain design considers a much hostile environment in which nodes exhibit Byzantine behavior. Under this model the overhead of concurrency control is much higher [17].

At a high level, a blockchain system can be categorized as either public or private. In the former, any node can join and leave the system, thus the blockchain is fully decentralized, resembling a peer-to-peer system [18]. In the latter, the blockchain enforces strict membership. More specifically, there is an access control mechanism to determine who can join the system. As the result, every node is authenticated and its identity is known to the other nodes.

2.1 Public Blockchain

Bitcoin [1] is the most well known example of public blockchains. In Bitcoin the states are digital coins (crypto-currencies), and a transaction moves coins from one set of addresses to another. Each node broadcasts a set of transactions it wants to perform. Special nodes called *miners* collect transactions into blocks, check for their validity, and start a consensus protocol to append the blocks onto the blockchain. Bitcoin uses *proof-of-work* (PoW) for consensus: only a miner which has successfully solved a computationally hard puzzle (finding the right nonce for the block header) can append to the blockchain. PoW is tolerant of Byzantine failure, but it is probabilistic in nature: it is possible that two blocks are appended at the same time, creating a *fork* in the blockchain. Bitcoin resolves this by only considering a block

TABLE 1
Examples of Distributed Ledgers

Data Model	Number of ledgers	Owner	Example
Accounts	One	Administrator	Traditional ledgers used in financial institutions.
Assets	Many	Group of users	Private ledger used within a financial institution, or between small groups of financial organizations, e.g., global financial services.
Coins or accounts	One	Any user	Crypto-currencies like Bitcoin or Ethereum.

as confirmed after it is followed by a number of blocks (typically six blocks). This probabilistic guarantee leads to security and performance issues: attacks have been demonstrated by an adversary controlling only 25 percent of the nodes [19], and Bitcoin transaction throughput remains very low (7 transactions per second [20]).

Most public blockchain systems employ variants of PoW for consensus. PoW works well in the public settings because it guards against Sybil attacks [18]. However, being non-deterministic and computationally expensive, it is unsuitable for applications such as banking and finance which must handle large volumes of transactions in a deterministic manner.

2.2 Private Blockchain

Hyperledger [12] is among the most popular private blockchains. Since node identities are known in the private settings, most blockchains adopt one of the protocols from the vast literature on distributed consensus. Zab [21], Raft [22], Paxos [23], PBFT [17] are popular protocols that are in active use today. Hyperledger directly uses PBFT,[1] while others like Parity [11], Ripple [6] and ErisDB [24] develop their own variants. PBFT is a three-phase protocol. In the *pre-prepare* phase, a leader broadcast a value to be commit by other nodes. Next, in the *prepare* phase, the nodes broadcast the values they are about to commit. Finally, the *commit* phase confirms the committed value when more than two third of the nodes agree in the previous phase. PBFT is communication bound, but it achieves both safety and liveness in partially synchronous networks. Besides deterministic consensus, another key property of private blockchains is that they support smart contracts which can express highly complex transaction logics. These properties are particularly desirable in business and financial systems. Indeed, private blockchains evoke such interest from major banking and financial institutions that some even claim that they have the potentials to disrupt current practices in data management [8], [9].

3 Key Concepts

Categorizing blockchains as public or private is useful for identifying major characteristics of many blockchains. However, understanding their subtle differences warrants a finer taxonomy. This section introduces four underpinning concepts, based on which a more detailed classification of the systems can be obtained.

3.1 Distributed Ledger

A ledger is a data structure that consists of an ordered list of transactions. For example, a ledger may record monetary transactions between multiple banks, or goods exchanged among known parties. In blockchains, the ledger is replicated over all the nodes. Furthermore, transactions are grouped into blocks which are then chained together. Thus, the distributed ledger is essentially a replicated append-only data structure. A blockchain starts with some initial states, and the ledger records entire history of update operations made to the states.

A system supporting distributed ledgers can be characterized in three dimensions, as illustrated in Table 1. First, the application built on top of the ledger determines the data model of what being stored in the ledger. The data model captures key data abstractions, making it easy for the application to express its logic. For example, a crypto-currency application may adopt the user-account model resembling traditional banking systems. On the other hand, a general-purpose blockchain may use low-level model such as table or key-value. Second, the system may have one or more ledgers which may be connected to each other. A large enterprise, for example, may own multiple ledgers, one for each of its departments: engineering, customer care, supply chain, payroll, etc.. Third, ledger ownership may vary from completely open to public to strictly controlled by one party. Bitcoin, for example is completely open, and as a consequence requires expensive consensus protocol to identify who can update the ledger. Parity [11], on the other hand, pre-determines a set of owners who can write to the ledger simply by singing the blocks.

3.2 Consensus

The content of the ledger reflects historical and current states maintained by the blockchain. Being replicated, updates to the ledger must be agreed on by all parties. In other words, multiple parties must come to a consensus. Note that this is not the case in many real-world applications such as fiat currency, in which one entity (e.g., the bank or the government) decides the updates.

One key property of a blockchain system is that the nodes do not trust each other, meaning that some may behave in Byzantine manners. The consensus protocol must therefore tolerate Byzantine failures. The research literature on distributed consensus is vast, and there are many variants of previously proposed protocols being developed for blockchains [25]. They can be largely classified along a spectrum. One extreme consists of purely computation based protocols that use proof of computation to randomly select a node which single-handedly decides the next operation. Bitcoin's proof-of-work (PoW) is an example. The other extreme is purely communication based protocols in which

[1] Hyperledger has two main releases: v0.6.0 and v1.0.0-rc1. The former supports PBFT, but the latter adopts a no-Byzantine consensus protocol based on Kafka.

nodes have equal votes and go through multiple rounds of communication to reach consensus. These protocols, PBFT [17] being the prime example, are used in private settings because they assume authenticated nodes.

In between the two extremes are hybrid protocols which aim to improve performance of PoW and PBFT. Some are used in public blockchains to address the inefficiency of PoW. One example is Proof-of-Elapsed-Time (PoET) which replaces PoW with another protocol based on trusted hardware such as Intel SGX. Other examples are Elastico [26] and Algorand [27] which improve PoW by randomly sampling a small set of nodes at each round. Other hybrid protocols, for example Proof-of-Authority (PoA) [28], Stellar [29] and Ripple [6], are used in private blockchains where they improve PBFT by executing consensus in smaller networks called federates.

3.3 Cryptography

Blockchain systems make heavy use of cryptographic techniques to ensure integrity of the ledgers. Integrity here refers to the ability to detect tampering of the blockchain data. This property is vital in public settings where there is no pre-established trust. For example, public confidence in crypto-currencies like Bitcoin, which determines values of the currencies, is predicated upon the integrity of the ledger; that is the ledger must be able to detect double spending. Even in private blockchains, integrity is equally essential because the authenticated nodes can still act maliciously.

There are at least two levels of integrity protection. First, the global states are protected by a hash (Merkle) tree whose root hash is stored in a block. Any state change results in a new root hash. The tree's leaves contain the states, the internal nodes contain the hashes of their children. For instance, Hyperledger v0.6 uses a bucket hash tree, in which states are grouped (by hashing) into a pre-defined number of buckets. Ethereum, on the other hand, employs a Patricia-Merkle tree which resembles a trie and whose leaves are key-value states. Second, the block history is protected, that is the blocks are immutable once they are appended to the blockchain. The key technique is to link the blocks through a chain of cryptographic hash pointers: the content of block number $n+1$ contains the hash of block number n. This way, any modification in block n immediately invalidates all the subsequent blocks. By combining Merkle tree and hash pointers, blockchain offers a secure and efficient data model that tracks all historical changes made to the global states.

Blockchain's security model assumes the availability of public key cryptography. Identities, including user and transaction identities, are derived from public key certificates. Secure key management, therefore, is essential to any blockchains. As in other security systems, losing private keys means losing access. But in blockchain applications such as crypto-currencies, losing the keys has direct and irrevocable financial impact. We discuss in Section 4.2 different schemes for key and identity management.

There exist many research systems that extend the original blockchain design with novel and complex cryptographic protocols. They aim to improve security and performance with esoteric techniques such as zero-knowledge proofs, group signatures and trusted hardware. We discuss them in greater detail in Section 4.2.

```
contract Doubler {
  struct Partitipant {
    address etherAddress;
    uint amount;
  }
  Partitipant[] public participants;
  uint public balance = 0;
  ...
  function enter() {
    ...
    balance+= msg.value;
    ...
    if (balance >
              2*participants[payoutIdx].amount) {
      transactionAmount = ...
      participants[payoutIdx].
              etherAddress.send(transactionAmount);
      ...
    }
  }
  ...
}
```

Fig. 2. An example of Ethereum smart contract, written in Solidity, which implements a pyramid scheme.

3.4 Smart Contracts

A smart contract refers to the computation executed when a transaction is performed. It can be regarded as a stored procedure invoked upon a transaction. The inputs, outputs and states affected by the smart contract execution are agreed on by every node.

All blockchains have built-in smart contracts that implement their transaction logics. In crypto-currencies, for example, the built-in smart contract first verifies transaction inputs by checking their signatures. Next, it verifies that the balance of the output addresses matches that of the inputs. Finally, it applies changes to the states. In the rest of the paper we do not refer to such built-in logics as smart contracts. Instead, we only consider smart contracts that can be defined by users.

One way to characterize a smart contract system is by its language. At one extreme, Bitcoin provides fewer than 200 *opcodes* from which users can write stack-based scripts. For example, the following script verifies if 2 out of 3 valid signatures are available.

```
OP_2 <Pub1> <Pub2> <Pub3> OP_3 OP_CHECKMULTSIG
```

At the other extreme, Ethereum smart contracts can specify arbitrary computations, i.e., they are Turing complete code. Fig. 2 shows a snippet of a real smart contract running on Ethereum. It implements a pyramid scheme: users send money to this contract which then pays interests to early participants. The contract has its own states, namely the list of participants, and exports a function called `enter`. A user invokes the contract by sending his money through a transaction. When executed, the contract can access the input address (user account) via `msg.sender` and the transaction value via `msg.amount`. It updates the accumulated balance, computes the interest for each participants. Finally, payment is made by invoking `etherAddress.send`.

In between the two extremes are smart contract systems that offer more expressiveness than Bitcoin's opcodes, but they reject Turing-completeness. Kadena [30] and BigchainDB [31] support contracts with complex, but constrained semantics so that they can be formally checked for safety.

TABLE 2
Comparison of Blockchain Systems

	Application	Smart contract execution	Smart contract language	Data model	Consensus
Hyperledger v0.6.0 [32]	General applications	Dockers	Golang, Java	Key-value	PBFT
Hyperledger v1.0.0 [33]	General applications	Dockers	Golang, Java	Key-value	Ordering service (Kafka)
Bitcoin	Crypto-currency	Native	Golang, C++	Transaction-based	PoW
Litecoin [34]	Crypto-currency	Native	Golang, C++	Transaction-based	PoW (memory)
ZCash [35]	Crypto-currency	Native	C++	Transaction-based	PoW (memory)
Ethereum [5]	General applications	EVM	Solidity, Serpent, LLL	Account-based	PoW
Multichain [36]	Digital assets	Native	C++	Transaction-based	Trusted validators (round robin)
Quorum [37]	General applications	EVM	Golang	Account-based	Raft
HydraChain [38]	*General applications*	*Python, EVM*	*Solidity, Serpent, LLL*	*Account-based*	*Trusted validators (majority)*
OpenChain [39]	*Digital assets*	-	-	*Transaction-based*	*Single validator*
IOTA [40]	*Digital assets*	-	-	*Account-based*	*IOTA's Tangle Consensus*
BigchainDB [31]	Digital assets	Native	Python, crypto-conditions	Transaction based	Trusted validators (majority)
Monax [24]	General applications	EVM	Solidity	Account-based	Tendermint [41]
Ripple [6]	Digital assets	-	-	Account-based	Ripple consensus
Kadena [30]	*Pact applications*	*Native*	*Pact*	*Table*	*ScalableBFT [42]*
Stellar [29]	Digital assets	-	-	Account-based	Stellar consensus
Dfinity [43]	*General applications*	*EVM*	*Solidity, Serpent, LLL*	*Account-based*	*Threshold relay*
Parity [11]	General applications	EVM	Solidity, Serpent, LLL	Account-based	Trusted validators (round robin)
Tezos [44]	Michaleson applications	Native	Michaleson	Account-based	Proof of Stake
Corda [45]	Digital assets	JVM	Kotlin, Java	Transaction-based	Raft
Sawtooth Lake [46]	General applications	Native	Python	Key-value	Proof of elapsed time

Ones in italics are deemed inactive or at early phases of development.

Another way to categorize smart contract systems is by their runtime environments. Most systems execute smart contracts in the same runtime as the rest of the blockchain stack. We refer to them as employing native runtimes. For example, Kadena parses contracts written in its Haskell-like language and executes them directly as Haskell programs. Ethereum, on the other hand, comes with its own virtual machine for executing Ethereum bytecodes. Hyperledger, opting for portability, employs Docker containers to execute its contracts.

4 State of the Art

In this section we compare current blockchains using the four concepts discussed in Section 3. We explain their design in more detail and highlight their subtle differences. We also discuss research problems that are being tackled.

A list of blockchains and their properties are shown in Table 2. Major systems are included, but we stress that the list is not exhaustive, especially given the growing commercial and academic interest in blockchains. Systems shown in italics are either no longer in active development, or are still in initial phases of development. For examples, Hydrachain [38] codebase was last updated about 8 months ago at the time of writing, and IOTA's current codebase consists of only a reference implementation.[2] The table has no column for cryptography, since all systems (except for ZCash) employ standard techniques described in Section 3. Novel cryptographic protocols that are not yet integrated are discussed in Section 4.2.

4.1 Distributed Ledger

Recall that a system supporting distributed ledgers is characterized by its target applications, by the number of ledgers, and by the ledger ownership. In the following, we group the systems listed in Table 2 by their target applications.[3]

4.1.1 Crypto-Currency

The most successful adoption of blockchain technology is crypto-currency. In the wake of Bitcoin's success, multiple competing currencies appear. Most of these alternative currencies (or alt-coins) such as Litecoin or Dodgecoin, adopt similar data models to Bitcoin's. Ethereum, departs from Bitcoin's transaction-based model and instead implements an account-based model. The nature of currency applications requires that the ledger must be open and the system maintains only one ledger.

4.1.2 Digital Assets

Crypto-currency is one instance of digital assets—pieces of data with attached real-world values. Unlike crypto-currencies which are created on and derive their values directly from the blockchains, digital assets are often issued

[2] As many other blockchain projects do, IOTA is raising fund for development via token sale.

[3] We note that each system can also support other applications, limited only by the power of the supported smart contracts.

by real world entities and blockchains are merely a medium to record their existence and exchanges. Multichain [36], BigchainDB and Corda offer ledgers for storing and tracking asset history. Like Bitcoin, their data models are transaction-based which are centered around assets. These systems target private settings, in which multiple organizations can spin up a network to trade assets among each other. The organizations are the ledger owners, and it is common to have more than one ledger among them. Stellar, Ripple and IOTA issue their own assets (tokens) and offer their ledgers as a medium of exchange or a platform for micro-payment. IOTA, in particular, allows for zero-fee micropayment via its tokens, which makes the ledger useful for exchanges among IoT devices. The ledgers in these systems adopt account-based data models. One ledger exists per system and it is open; that is anyone can buy tokens and take part in the exchanges.

4.1.3 General Applications

Going beyond crypto-currency and asset management, some ledgers support running general, user-defined computations (or smart contracts). Ethereum and its derivatives, namely Hydrachain, Quorum, Monax, Parity and Dfinity let users write arbitrary business logics executed on top of the ledger. For example, Ethereum contracts range from simple crowdfunding campaigns to complex investment funds like the DAO [47]. Dfinity has a special type of contract—the governance contract—that enforces real-world regulations on Ethereum-like blockchains. Hyperledger and its close cousin Sawtooth Lake likewise support running Turing-complete code. They offer key-value data model, with which the applications can create and update key-value tuples on the blockchain. Kadena and Tezos restrict how powerful the applications can be by devising their own languages which are not Turing complete but can be formally verified. Tezos data model is account-based, whereas Kadena's is based on table. In particular, Kadena applications operate on *key-row* structures with schemas, versions and column history.

4.2 Cryptography

4.2.1 Identity Management

A user in a blockchain is uniquely identified by her public key certificate. In public settings, the user first generates a key pair (the default option being ECDSA based on the Secp256k1 elliptic curve), then derives the identity as the hash of the public key. This hash serves as a transaction address or an account number in crypto-currencies systems. To claim ownership of the transaction output or of the account, the user signs transactions with the corresponding private key. In private settings, there is an additional access control layer. Hyperledger separates this layer from the blockchain, in the form of a membership provider service and a certificate authority service. The administrator can implement arbitrary policies with these services to control who gets access to the blockchain. Signed requests sent to Hyperledger are first checked against these services before processed by the next (consensus) component. Multichain offers a simpler model with a fixed number of global permissions, while the remaining systems provide little detail of their protocols.

The problem of managing user keys is the same in private blockchains as in typical enterprise systems, thus existing solutions can be readily integrated. In public blockchains, however, the sheer scale and monetary impacts of losing private keys calls for more secure and more usable protocols. Bitcoin, in particular, embodies the challenges, as Bitcoin users themselves are tasked with managing large numbers of keys which are refreshed on new transactions. Eskandari et al. [48] evaluated six approaches for Bitcoin key management (or wallet): local storage, password protected storage, offline storage, air-gapped storage, password-derived keys, and hosted storage. The authors found that none of these approaches is satisfactorily usable, due to misuse of metaphors to traditional currencies, and also due to confusing abstractions.

4.2.2 Trusted Hardware

Recent distributed systems are leveraging trusted hardware such as Intel SGX and ARM TrustZone to improve performance at a slight cost of security [49], [50]. Sawtooth Lake proposes proof of elapsed time (PoET) as an efficient replacement for proof-of-work. TownCrier [51] employs SGX to implement a trusted party for vetting external contents and importing them to the blockchain. These systems are based on a trust model that is weaker than that of a purely cryptographic system. In particular, their security is dependent upon a trusted computing base (TCB) that is running inside the trusted hardware. Smaller TCBs mean better security.

All systems based on trusted hardware rely on *remote attestation* protocols. A key pair, called Endorsement Key (EK), is burnt into each device during manufacturing. Such a key pair serves as the root of trust, from which other short-term keys are derived. Before a piece of code is loaded, the hardware *measures* it by hashing the code content and signing it with one of the keys. The signed measurement together with the key certificate attest to a remote party what is being run in the local device. This protocol requires a certificate authority that maintains and endorses a list of known certificates and a list of revoked certificates. Highly complex attestation schemes, for example direct anonymous attestation [52], offer hardware anonymity without a certificate authority.

4.2.3 Transaction Privacy

Most blockchains are designed to protect transaction integrity, but they do not consider transaction privacy. A blockchain is said to have transaction privacy when (1) transactions cannot be linked from one to another, and (2) the transaction content is known only to its participants. In private settings, complete transparency of transaction history may not be a problem. Either transparency is desirable for the applications, such as financial auditing, or it is straightforward to add an access control layer to protect the blockchain data. In public settings, on the other hand, the need for transaction privacy is driven by two factors. First, deanonymization attacks have successfully recovered the underlying structure of the Bitcoin network [53], and even linked Bitcoin addresses to real-world identities [54]. Second, transaction linkability can undermine the currency's fungibility, rendering some coins more valuable than others due to their histories.

TABLE 3
Comparison of Consensus Protocols

Consensus Protocol	Network Settings	Description
PBFT-based	Private	Hyperledger uses the original PBFT [17]. Tendermint [41] enhances it by assigning unequal weights to votes. Other variants include Scalable BFT [63], Parallel BFT [64], Optimistic BFT [65], etc.
Stellar	Federated	Stellar network [66] proposes its own consensus protocol where the nodes form intersecting groups (federates). Consensus is agreed in each group, then propagated to the rest of the network.
Ripple	Federated	Ripple payment system [6] proposes a variant of PBFT where the nodes belong to intersecting groups, and in each group there is a large majority of non-Byzantine nodes.
Proof-of-Work (PoW)	Public	Bitcoin uses pure proof-of-work, which leads to scalability issues. Bitcoin-NG [67], Byzcoin [61] separate leader election from transaction validation in PoW, thus increase the overall performance.
Proof-of-Stake (PoS)	Public	Tendermint [41] uses PoS, in which a node's ability to create new block is determined by its stake in the blockchain, e.g., the amount of currencies it owns [68]. A set of high-stake owners uses another consensus mechanism, which is usually faster than PoW, to reach agreement on a new block.
Threshold Relay	Public	Dfinity [43] proposes *threshold relay* in which nodes form random group based on a public verifiable random function (Byzcoin [61] and Elastico [26] adopt similar approaches). The nodes in the group create a new block by signing it using threshold signature.
Proof-of-Authority (PoA)	Private	Parity [11] uses PoA, in which some pre-defined nodes are considered trusted authorities and they can propose the next blocks. It then uses round-robin scheduling to assign every authority node a time window during which it can propose blocks.
Proof-of-Burn (PoB)	Public	Slimcoin [69] uses PoB, in which a node destroys some base currencies it owns in another blockchain in order to get a chance of proposing a new block. Slimcoin supports PoB based on Peercoin [70].
Proof-of-Elapsed Time (PoET)	Private	Sawtooth [46] uses PoET, in which each node runs a trusted hardware, for example Intel SGX [71], that generates random timers. The first node whose timer has expired can propose the next block.
Others	Public	Other protocols based on PoW are of the form *proof-of-X*, for examples: Proof-of-Activity [72], Proof-of-Space [73], Proof-of-Luck [74], etc.

Zerocoin [55] is the first blockchain providing transaction unlinkability. It extends Bitcoin to allow for trading between bitcoins and special coins called *zerocoins*. Zerocoin essentially implements a cryptographic mixer that hides linkability between zerocoins and the corresponding bitcoins. Each zerocoin is a cryptographic commitment to two random values (s, r). When redeeming a zerocoin, the owner reveals s as the proof that the coin has not been spent, and a zero-knowledge proof of r. Transaction unlinkability is derived from the fact that the coin being redeemed can be any of the many unspent zercoins.

Zerocash [35] extends Zerocoin by improving efficiency of the latter's cryptographic operations. It functions as a stand-alone blockchain, as opposed to Zercoin being an extension of Bitcoin. Transactions in Zerocash, including split and merge transactions, are fully private. They are based on complex zero-knowledge proofs which only reveal the fact that there exists unredeemed coins whose sum is a specific value. Both Zerocoin and Zerocash are implemented using zkSNARK [56] and carry large overheads due to the underlying zero-knowledge protocols. Zerocash, for instance, requires a trusted party to securely create and distribute some public parameters whose sizes are in hundreds of megabytes.

4.2.4 Advanced Signatures

Bitcoin supports multi-signatures, in which a transaction can be redeemed when at least t out of n valid signatures are available. Multi-signatures are resilient against corrupt individuals by virtue of spreading the decryption or singing capabilities to a group of users. Lightning Network [57], an extension of Bitcoin with near-instant payment confirmation, relies on multi-signatures to first deposit some mutual funds on the blockchain. Once confirmed, payments from the funds can happen outside of the blockchain with immediate confirmation. Finally, the funds can be closed with corresponding transactions signed with all the required signatures. Extensions of Bitcoin multi-signature scheme can be built directly on top of ECDSA [58]. More advanced schemes, e.g., [59], [60], can be employed (albeit not without major changes in the current design).

Byzcoin [61] uses a group signature scheme called Cosi [62] to reduce communication overhead in PBFT. Cosi involves four rounds of communication at the end of which a collective signature is generated and verified by all members of the group. The signature is structured as a tree of Schnorr signatures. It significantly reduces the size of messages broadcast in the network during the prepare and commit phase of PBFT, because each node no longer needs individual signatures from all the other for verification.

4.3 Consensus

Recall that there is a spectrum of consensus protocols behind blockchain systems, starting from purely computation bound like PoW to purely communication bound like PBFT. Table 3 summarizes key properties of the major protocols which we now explore in detail.

4.3.1 Proof of Work Variants

All PoW protocols require miners to find solutions to cryptographic puzzles based on cryptographic hashes. Specifically, the solution is a random nonce n such that

$$H(n \| H(b)) \; < \; t,$$

for a cryptographic hash function H, a threshold t and the current block content b. The original protocol implemented in Bitcoin uses SHA-256 as the hash function. The availability of custom hardware (ASIC) that speeds up hash computation prompts other crypto-currencies to adopt memory-hard hash functions. Ethereum uses Dagger-Hashimoto function, Litecoin and Dodgecoin use scrypt, and ZCash uses Equihash function. These functions are resistant to ASIC, as they demand large investment in memory, but are easy to verify.

How fast a block is created is dependent on how hard the puzzle is. Bitcoin sets t to a value equivalent of 10 minutes per block. Litecoin, Dodgecoin and ZCash decrease t to achieve lower average block time to several minutes. t cannot be arbitrary small because it leads to unnecessary forks in the blockchain. Forks not only lead to wastage of resources but have security implication since they make it possible to double spend. Ethereum adopts GHOST [75] protocol which helps bring down block generation time to tens of seconds without compromising much security. In GHOST, the blockchain is allowed to have branches as long as the branches do not contain conflicting transactions.

4.3.2 Proof of Stake

PoW mining is hugely expensive. The process is particularly energy intensive, and has been estimated to consume enough electricity to power a small country like Denmark [76]. PoS is proposed to substantially reduce the cost of mining. Unlike Ethereum's GHOST, PoS maintains a single branch but changes the puzzle's difficulty to be inversely proportional to the miner's stake in the network. A stake is essentially a locked account with a certain balance representing the miner's commitment to keep the network healthy. Let s be the function that returns the stake, then a miner M can generate a new block by solving the puzzle of the following form

$$H(n \| H(b)) \; < \; s(M).t.$$

It can be seen that the greater the stake $s(M)$, the easier it is to find n.

Peercoin, forked from Bitcoin, is among the first systems with PoS. It bootstraps by running PoW to generate coins. The function $s(.)$ in Peercoin takes a coin C as input and returns $C.age(C)$ where $age(C)$ is the coin's age. Nxt [77], another PoS system, bootstraps by selling its tokens. The function $s(.)$ in Nxt considers both the miner's balance and the elapsed time from the last block. The longer it is since the last block, the easy it is to solve the puzzle. In particular

$$s(M, b_h) = bal(M).age(b_{h-1}),$$

where b_h is the current block at height h, $bal(M)$ returns how many coins in M's account, and age returns how much time has passed since the creation of a block at a certain height.

Ethereum's upcoming PoS protocol is implemented as a smart contract. Referred to as Casper, it allows miners to become *validators* by depositing Ethers to the Casper account. The contract then picks a validator to propose the next block according to the deposit amount. Its unique feature, however, is to force validators to behave correctly or else risk losing the entire deposit. In particular, each validator places a bet on whether a certain block will be confirmed in the future. If the block is confirmed, the validator gets a small reward. But if it is not, the validator loses its deposit. This mechanism avoids the *nothing-at-stake* problem in which validators can propose blocks in different branches. Tezos implements a simplified version of Casper in which the nodes *buy in* to become authorities which can then approve changes to the underlying blockchain. Tezos aims to provide an amendable blockchain in which soft forks and hard forks are inherent features of the blockchain.

4.3.3 PBFT Variants

PoW suffers from non-finality, that is a block appended to a blockchain is not confirmed until it is extended by many other blocks. Even then, its existence in the blockchain is only probabilistic. For example, eclipse attacks on Bitcoin [78] exploit this probabilistic guarantee to allow double spending. In contrast, the original PBFT protocol [17] is deterministic. Implemented in the earlier version of Hyperledger (v0.6), the protocol ensures that once a block is appended, it is final and cannot be replaced or modified. It incurs $O(N^2)$ network messages for each round of agreement where N is the number of nodes in the network. In practice, however, the original protocol scales poorly and collapses even before reaching the network limit [79]. We observe the same scalability issues in our evaluation of Hyperledger with BLOCKBENCH.

Tendermint proposes a small modification on top of PBFT. Instead of each node having an equal vote, in Tendermint each node may have different voting power, proportional to their stake in the network. To reach agreement in Tendermint it is necessary to only gather over $2/3$ of the total voting power. This may be cheaper than waiting for $2/3$ of the network to response when there is a small number of nodes with high stakes.

Recent works on improving PBFT have mainly focused on its performance. Zyzzyva [80] optimizes for normal cases (when there are no failures) via speculative execution. XFT [81], assumes a network less hostile than purely Byzantine, and demonstrates better performance by reducing the number of network messages. HoneyBadger [82], on the other hand, focuses on improving security under asynchronous networks. It employs a *randomized agreement* protocol which achieves safety with overwhelming probability even under network asynchrony. By optimizing the network layer, it is shown to outperform PBFT even when the network is synchronous. Both Zyzzyva, XFT and HoneyBadger hold great promise, but they have not been integrated into any blockchains.

4.3.4 Trusted Hardware

Most overheads of PoW and PBFT can be attributed to the assumption that nodes behave in Byzantine manners. The availability of Intel SGX [83] or ARM TrustZone [84], however, makes it possible to relax the trust model in the Byzantine settings. In particular, a node equipped with trusted hardware can be reliably checked for certain properties, for example, that it is running a specific software.

Sawtooth Lake leverages SGX to replace PoW with a more efficient protocol called PoET. Specifically, PoET runs inside an enclave protected by SGX. It starts by taking a block number as input and generating a timer of a random duration t. Afterward, it can produce certificates indicating how much time has passed since the timer starts. A node whose PoET generates the smallest t can append the block when the timer expires. In particular, the node attaches its PoET certificate to the block, and as long as t is smaller than what generated by any other node the block is accepted.

A2M [85] and Hybster [86] both exploit trusted hardware to reduce the number of replicas needed to tolerate f failures from $3f+1$ to $2f+1$. This means an N-node network can now tolerate up to $N/2$ adversarial nodes, as opposed to $N/3$ adversarial nodes in the original PBFT. A2M's and Hybster's safety are dependent on the trusted code bases (TCBs) that implement simple functions: a log data structure in the former and a monotonic counter in the latter.

4.3.5 Federated

Despite numerous improvements to the original protocol, PBFT-based consensus remains communication bound, thus it ultimately fails to scale beyond a certain number of nodes. To overcome this hard limit without scarifying safety, Stellar and Ripple adopt an approach that partitions the network into smaller groups called federates. Each federate runs a local consensus protocol among its members, which does not run into scalability problems because of the small network size. Local agreements are then propagated to the entire network via nodes lying in the intersections of the federates. Global consensus can be achieved under certain conditions. For Stellar, the condition is that every two federates intersect at non-Byzantine nodes. Ripple's safety conditions are that there is a large majority of honest nodes in every federate, and that the intersection of any two federates contain at least one honest node.

Both Stellar and Ripple assume federates are pre-defined and their safety conditions can be enforced by a network administrator. In a decentralized environment where node identities are unknown, such assumptions do not hold. Byzcoin [61] and Elastico [26] propose novel, two-phase protocols that combine PoW and PBFT. In the first phase, PoW is used to form a consensus group. Byzcoin implements this by having a sliding window over the blockchain and selecting the miners of the blocks within the window. Elastico [26] groups nodes by their identities that change every epoch. In particular, a node identity is its solution to a cryptographic puzzle. In the second phase, the selected nodes perform PBFT to determine the next block. The end result is faster block confirmation time at a scale much greater than traditional PBFT (over 1,000 nodes).

Similar to Byzcoin and Elastico, Dfinity [43] and Algorand [27] select at each round a random set of nodes that can propose blocks. Unlike the former, they dispense with PoW and instead use verifiable random functions (VRFs) to select the consensus group. In Dfinity, the VRF is based on the threshold signature of the previous block. In Algorand, it is based on a random seed published in the previous block and the node's secret key.

4.3.6 Non-Byzantine

The systems described so far in this section tolerate Byzantine failures, rendering them attractive for public settings and for private settings where the cost of engaging trusted parties (for example, for escrowing assets) is high. Some blockchains, however, assume trusted parties in order to simplify their designs. These blockchains have no safety guarantees when any of such parties behaves maliciously.

Openchain [39] relies on a single trusted party (called validator) that determines the next block. Consequently, it is most vulnerable to attacks as the validator is the single point of failure. Multichain and Parity have more than one trusted party which is referred to as authority in their systems. Each authority is given a time slice, via round-robin scheduling, during which it can append new blocks to the chain. This simple proof-of-authority (PoA) protocol avoids single point of failure while ensuring balanced workloads among the authorities. HydraChain and BigChainDb also have multiple authorities, but one authority cannot unilaterally decide the next blocks. Instead, the block is decided via majority voting. Quorum [37] employs Raft [22] as the consensus protocol among its authorities. Raft implements crash tolerant state machine replication, which is an important building block of modern distributed database systems. Using Raft, Quorum is able to make safe progress even when some authority nodes crash.

Corda's consensus protocol is executed by a set of trusted parties called *notaries* which check if a given transaction has been executed before. By delegating this check to an entity outside of the blockchain, Corda can justify using Raft for consensus. Transactions in Corda are sent to the notaries before being confirmed in the blockchain. The notaries then use Raft to ensure that the transactions are replicated among themselves and remain highly available despite crashes.

The latest release of Hyperledger (v1.0) outsources the consensus component to Kafka—another building block often found in distributed database systems. More specifically, transactions are sent to a centralized Kafka service which orders them into a stream of events. Every node subscribes to the same Kafka stream and therefore is notified of new transactions in the same order as they are published. Since there is only one Kafka service, the observed transaction sequence is the same at every node.

4.3.7 Others

IOTA [40] uses its own consensus protocol called Tangle in which the blocks form a direct acyclic graph (DAG) as opposed to a chain. In addition, a block in Tangle consists of only one transaction. When appended, the block must *approve* two other blocks creating links to them in the DAG. The block is confirmed when it is approved by many other blocks. Targeting IoT environments, Tangle's main goal is efficiency and low-cost payment. Although its security has not been rigorously analyzed, the low values of transactions (micropayments) in Tangle could in practice discourage Byzantine behavior.

Kadena [30] proposes an extension to Raft that handles Byzantine failures. It introduces various techniques on top of Raft, such as message signatures, client verification and incremental hashing. However, like Tangle, it is unclear whether the protocol guarantees safety and liveness.

4.4 Smart Contracts

Recall that a smart contract system can be characterized by its language expressiveness or by its execution environment. Except for Openchain, IOTA, Ripple and Stellar, all systems listed in Table 2 let users customize transaction logics to suit their applications. In the following, we group them by the contract language expressiveness.

4.4.1 Scripts

Bitcoin provides approximately 200 opcodes, but many of them are disabled in the latest implementation. Users can write stack-based programs with the opcodes. The most popular contracts in Bitcoin are related to multi-signatures. One example is the escrow contract that requires 2 out of 3 signatures before a coin can be released. The language can also implement bounty-hunting style contracts, for example, one that releases the reward coins when the pre-image of a hash value is found.

BigchainDB [31] adopts a more expressive language called *crypto-condition*. Developed as part of the Interledger Protocol project [87], crypto-condition allows specifying complex boolean expressions over many types of signatures. A crypto-condition script contains *conditions* and *fulfillments* which are treated as inputs and output of the script. The available conditions include *timeout* which enables time-release contracts. Crypto-condition's encoding is higher level than Bitcoin opcodes, making it easy to express complex logics.

4.4.2 Turing Complete

Ethereum is among the first blockchains offering Turing-complete smart contracts. Users write their contracts in either Solidity, Serpent or LLC language, which then get compiled to EVM bytecodes. EVM executes normal cryptocurrency transactions, and it treats smart contract bytecodes as a special transaction. Specifically, each smart contract is given its own memory to store local states. The memory is exposed as a key-value storage, though Solidity provides high-level data types such as map, array and composite structures. Resources consumed during execution of the contract, both in terms of CPU and memory, are tracked by EVM and charged to the transaction sender's account. EVM also keeps track of intermediate state changes and reverse them if there are insufficient funds to pay for the execution.

Hyperledger does not have its own bytecotes. Instead, it runs its language-agnostic smart contracts inside Docker containers. Specifically, a contract can be written in any language, which is then compiled into native code and packed into a Docker image. When the contract is uploaded, each node starts a new container with that image. Invoking the contract is done via Docker APIs. The contract can access the blockchain states via two methods *getState* and *putState* exposed by a shim layer. One benefit of Hyperledger is that it supports multiple high-level programming languages like Go and Java. However, its key-value interfaces with the blockchain necessitates extra application logics for mapping high-level data structures into key-value tuples.

Sawtooth Lake supports smart contracts in the form of transaction families. Each family is a user-defined Python class loaded into the ledger during start up. The contract is executed in the native runtime environment as a normal Python program.

One consequence of supporting Turing complete contracts is that software bugs are all but inevitable. While empowering, the Ethereum smart contract model receives strong criticism because it directly exposes Ethers against programming bugs. The security concerns indeed materialized in the DAO attack [47] in which the attacker stole $\$50M$ worth of asset. The attack exploits a concurrency bug in the DAO smart contract which allows one to repeatedly draw more money than what is specified in the transaction. Such bugs are inherent in a language like EVM which has weak or no formal specifications of its semantics. OYENTE [88] presents three major causes of security bugs: transaction order dependencies, timestamp dependencies and mishandled exceptions. It formalizes Ethereum semantics and proposes a tool for checking bugs directly on EVM bytecodes. The tool discovered over 8000 Ethereum contracts (worth over $\$60M$) with potential security bugs.

Like any other transactions on the blockchain, smart contract executions are transparent. It means the inputs, outputs and the states of the contract are visible to the network. Hawk [89] extends Zerocash to provide transaction privacy for smart contracts. The main challenge compared to Zerocash lies in the arbitrary transaction logics, whereas in Zerocash the logics are constrained by a small set of operations. Another challenge is to protect local states, which is not applicable in Zerocash. Given a contract, Hawk compiles it with zkSNARK to make it privacy preserving. Transaction inputs and outputs are pre- and post-processed via Hawk to hide the complex cryptographic details. Although the protocols incur large overhead both in time and space, Hawk represents a practical cryptographic system that achieves both transaction privacy and fairness.

4.4.3 Verifiable

Even before the DAO attack, some blockchains have rejected the models that allow for unconstrained computations. The languages of Kadena, Tezos and Corda are more powerful than Bitcoin scripts, but they trade Turing completeness for safety. Kadena's language is a Lisp-like functional language called Pact [30]. A Pact contract is stored in the ledger in human readable form, which is then parsed and executed in Ocaml. It is strongly typed and can be formally verified. Similarly, Tezos's stack-based language called Michelson comes with a strong type system and fully specified semantics. As a result, Tezos contracts can be statically checked for safety. In Corda, a contract is a sequence of *pure functions* that do not modify the states. Because the functions are merely constraints, the contract's safety can be formalized and verified.

5 BLOCKBENCH

The previous section has presented a thorough qualitative analysis of existing blockchains. In this section, we describe our benchmarking framework called BLOCKBENCH [10].

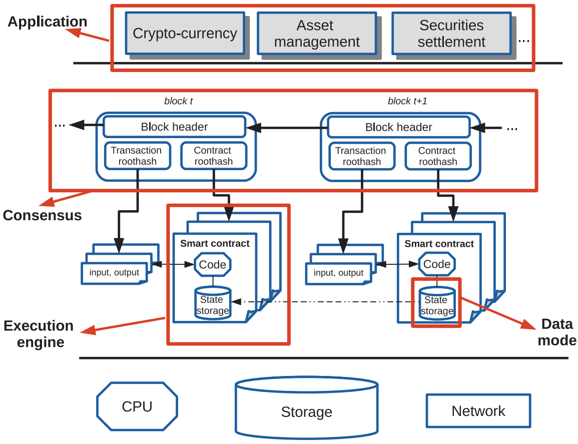


Fig. 3. Blockchain software stack on a fully validating node.

Designed for quantitative analysis of blockchains as data processing platforms, the framework targets private blockchains with Turing-complete smart contracts. BLOCKBENCH is open source [90] and contains data processing workloads commonly found in database benchmarks.

5.1 Layers

BLOCKBENCH targets blockchains that function as data processing platforms. Such a blockchain must have no restrictions on the application logics, thus it must support Turing complete smart contracts. Fig. 3 shows the logical components of the blockchain software stack, from which we refine the taxonomy described in Section 4 into four concrete layers shown in Fig. 4. For each layer there are multiple BLOCKBENCH workloads for evaluating it individually.

The consensus layer implements the consensus protocol. The data model layer contains the structure, content and operations on the blockchain data. The execution layer includes details of the runtime environment for executing smart contracts. Finally, the application layer includes classes of blockchain applications. Croman et. al. [20] proposed to divide blockchain into several planes: network, consensus, storage, view and side plane. While similar to BLOCKBENCH's four layers, the plane abstraction was geared towards crypto-currency applications and did not take into account the execution of smart contracts.

5.2 Implementation

BLOCKBENCH stack consists of a frontend interface for integrating new benchmark workloads, a backend interface for integrating new blockchains, and a driver for driving the workloads. A new blockchain can be integrated into the framework's backend by implementing the `IBlockchainConnector` interface. The interface contains operations for deploying the smart contract application, invoking it by sending a transaction, and for querying the blockchain states. Ethereum, Parity and Hyperledger are the current backends, while ErisDB (or Monax), Quorum and Sawtooth Lake integration are under development. A new benchmarking workload can be added by implementing the `IWorkloadConnector` interface.[4] The `Driver` takes as input a workload and sends transactions to the blockchain according to user-defined configurations (number of operations, number of clients, threads, etc.). It collects runtime statistics which are used to compute five important metrics.

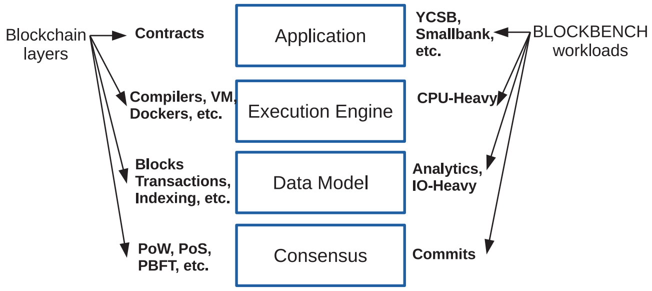


Fig. 4. Blockchain software layers and corresponding benchmark workloads.

- Throughput: the number of successful transactions per second. A workload can be configured with multiple clients and threads per clients to saturate the blockchain throughput.
- Latency: the response time per transaction. `Driver` implements blocking transactions, i.e., it waits for one transaction to finish before starting another.
- Scalability: the changes in throughput and latency when increasing the number of nodes and number of concurrent workloads.
- Fault tolerance: the changes in throughput and latency during node failure. We simulate crashes, network delays and random message corruptions.
- Security metrics: the ratio between the total number of blocks included in the main branch and the total number of confirmed blocks. The lower the ratio, the less vulnerable the system is from double spending or selfish mining.

5.3 Workloads

BLOCKBENCH comes with macro benchmark workloads for evaluating the application layer, and micro benchmark workloads for analyzing the lower layers. Smart contract implementations of the workloads shown in Fig. 4 are available and can be readily deployed on Ethereum, Parity and Hyperledger.

5.3.1 *Macro Benchmark Workloads*

We port two popular database benchmark workloads into BLOCKBENCH, namely YCSB and Smallbank. YCSB is widely used for evaluating NoSQL databases, for which we implement a simple smart contract which functions as a key-value storage. The `WorkloadClient` is based on the YCSB driver [91] which preloads each storage with a number of records, and supports requests with different ratios of read and write operations. For Smallbank [92], a popular benchmark for OLTP workload, we implement a smart contract that transfers money from one account to another.

Besides database workloads, BLOCKBENCH also provides three other workloads based on real Ethereum contracts. The first is *EtherId*, a popular contract implementing a domain name registrar. The second is *Doubler*, the pyramid scheme contract shown earlier in Fig. 2. The third is *WavesPresale* that implements a crowdfunding campaign via digital token sales.

[4] We assume that the smart contract implementing the workload's logic is already implemented and deployed on the blockchain.

```
type account_t struct {
    Balance     int
    CommitBlock int
}
type transaction_t {
    From string
    To   string
    Val  int
}
func Invoke_SendValue(from_account string,
        to_account string, value int) {
    var pending_list []transaction_t
    pending_list = decode(GetState("pending_list"))
    var new_txn transaction_t
    new_txn = transaction_t {
        from_account, to_account, value
    }
    pending_list = append(pending_list, new_txn)
    PutState(’pending_list’, encode(pending_list))
}
func Query_BlockTransactionList(block_number int)
     []transaction_t {
    return decode(GetState("block:"+block_number))
}
func Query_AccountBlockRange(account string,
        start_block int, end_block int)
     []account_t {
    version := decode(GetState(account+":latest"))
    var ret []account_t
    while true {
        var acc account_t
        acc = decode(GetState(account+":"+version))
        if acc.CommitBlock >= start_block &&
           acc.CommitBlock < end_block {
            ret = append(ret, acc)
        } else if acc.CommitBlock < start_block {
            break;
        }
        version -= 1
    }
    return ret
}
```

Fig. 5. Code snippet from the VersionKVStore smart contract for Analytics workload (Q1 and Q2).

5.3.2 Micro Benchmark Workloads

For the consensus layer, BLOCKBENCH provides *DoNothing* workload in which the smart contract accepts a transaction as input and simply returns. Since the contract execution involves minimal number of operations at the execution and data model layer, the overall performance will be determined by the consensus layer.

For the data model layer, BLOCKBENCH provides *Analytics* workload that is similar to an OLAP workload. In particular, it performs scan-like and aggregate queries whose performance is determined by the system's data model. Specifically, there are two queries:

Q1: *Compute the total transaction values committed between block i and block j.*

Q2: *Compute the largest transaction value involving a given state (account) between block i and block j.*

For Ethereum and Parity, both queries can be implemented via JSON-RPC APIs that return transaction details and account balances at a specific block. For Hyperledger, however, the second query must be implemented via a smart contract (VersionKVStore), because Hyperledger has no APIs for querying historical states. Fig. 5 shows the contract implementation in Hyperledger. To support historical data lookup, the contract appends a counter to the key of each account. To fetch a specific version of an account, the key `account:version` is used. The latest version is stored at the key `account:latest`. The contract also keeps keep a `CommitBlock` value in the data field for every version to point to the block number in which the current version is committed. To fetch the balances of a given account in a given block range, the contract scans all versions of this account and returns the corresponding balance when the version's CommitBlock value is in the specified range.

Another workload for the data model layer stresses the persistent storage. In particular, the *IOHeavy* workload evaluates the blockchain's IO performance by invoking a contract that performs a large number of random writes and random reads to the local states.

Finally, for the execution layer BLOCKBENCHprovides the *CPUHeavy* workload. It measures the efficiency of the execution layer for computationally heavy tasks by invoking a contract that executes quick sort algorithm over a large array.

6 EVALUATION

We selected Ethereum, Parity and Hyperledger for a comparative study using BLOCKBENCH. They occupy different positions in the design space, and are considered the most mature in terms of the codebase and user base. We used the popular Go implementation of Ethereum, *geth v1.4.18*, the Parity release *v1.6.0*. Unless otherwise specified, the Hyperledger version is *v0.6.0-preview*. We set up a private testnet for Ethereum and Parity by defining a genesis block and directly adding peers to the miner network. For Ethereum, we manually tuned the `difficulty` variable in the genesis block to ensure that miners do not diverge in large networks. For Parity, we set the `stepDuration` variable to 1. In both Ethereum and Parity, `confirmation-Length` is set to 5 seconds. The default batch size in Hyperledger is 500.

The experiments were run on a 48-node commodity cluster. Each node has an E5-1650 3.5 GHz CPU, 32 GB RAM, 2 TB hard drive, running Ubuntu 14.04 Trusty, and connected to the other nodes via 1 GB switch. For Ethereum, we reserved 8 cores out of the available 12 cores per machine, so that the periodic polls from the client's driver process do not interfere with the mining process. Our main findings are as follows:

- Hyperledger performs consistently better than Ethereum and Parity across the benchmarks. But it fails to scale up to more than 16 nodes.
- Ethereum and Parity are more resilient to node failures, but they are vulnerable to security attacks that forks the blockchain.
- The main bottlenecks in Hyperledger and Ethereum are the consensus protocols, but for Parity the bottleneck is caused by transaction signing.
- Ethereum and Parity incur large overheads in terms of memory and disk usage. Their execution engine is also less efficient than that of Hyperledger.
- Hyperledger's data model is low level, but its flexibility enables customized optimization for analytical queries.

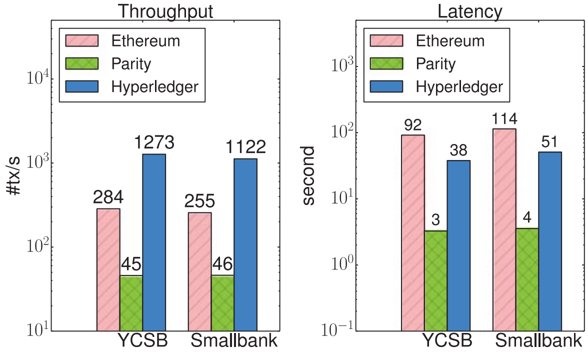


Fig. 6. Blockchain peak performance with eight clients and eight servers. The benchmark workload is YCSB. The performance against Smallbank workload is similar.

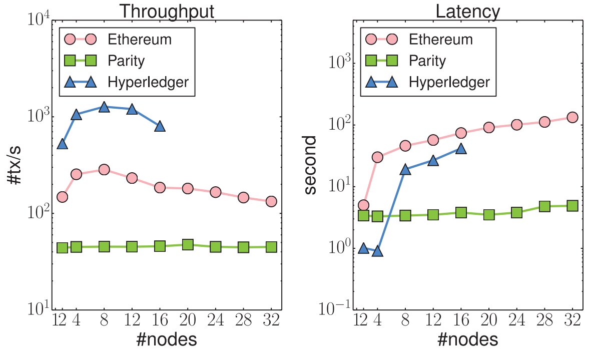


Fig. 8. Performance scalability (with the same number of clients and servers). The benchmark workload is YCSB. The scalability against Smallbank is similar, except that Hyperledger fails beyond eight nodes instead of 16.

6.1 Macro Benchmarks

This section discusses the performance of the blockchains at the application layer, using YCSB and Smallbank benchmarks.

6.1.1 *Throughput and Latency*

Fig. 6 shows the peak performance with 8 servers and 8 concurrent clients over the period of 5 minutes. We observe that in terms of throughput, Hyperledger outperforms the other two in both benchmarks. The gap between Hyperledger and Ethereum is due to the difference in the consensus protocols: one is based on PBFT while the other is based on PoW. With 8 servers, the communication cost from broadcasting messages is cheaper than block mining whose difficulty is set at roughly $2.5s$ per block. The gap between Parity and Hyperledger is not due to consensus protocols, as Parity's PoA protocol is expected to be simpler and more efficient than both PoW and PBFT. Instead, we observe that Parity processes transactions at a constant rate, and that it enforces a maximum client request rate at around 80 tx/s.

To put their performance in context, we compare the three blockchains against a popular in-memory database system, namely H-Store, using the YCSB and Smallbank workload. Blockchains and databases do not necessarily share the same design goal: the former are not designed for general data processing, nor do the latter protect data integrity against Byzantine failures. Nonetheless, we argue that the comparison offers useful insights into the design trade-offs and relative performance of the two systems. We ran H-Store's own benchmark driver and set the transaction rate at 100,000 tx/s. Fig. 7 shows at least an order of magnitude gap in throughput and two order of magnitude in latency. Specifically, H-Store achieves over 140K tx/s throughput while maintaining sub-millisecond latency. The gap in performance is due to the cost of consensus protocols. For YCSB, for example, H-Store requires almost no coordination among peers, whereas Ethereum and Hyperledger suffer the overhead of PoW and PBFT. An interesting observation is the overhead of Smallbank. Recall that compared to YCSB, Smallbank consists of more complex transactions in which multiple keys are updated in a single transaction. Smallbank is simple but is representative of the large class of transactional workloads such as TPC-C. We observe that in H-Store, Smallbank achieves 6.6x lower throughput and 4x higher latency than YCSB, which reflects the cost of distributed transaction management. In contrast, the blockchains suffer modest degradation in performance: 10 percent in throughput and 20 percent in latency. This is because each node in the blockchains maintains the complete states, therefore it pays no overhead in coordinating distributed transactions since the states are not partitioned.

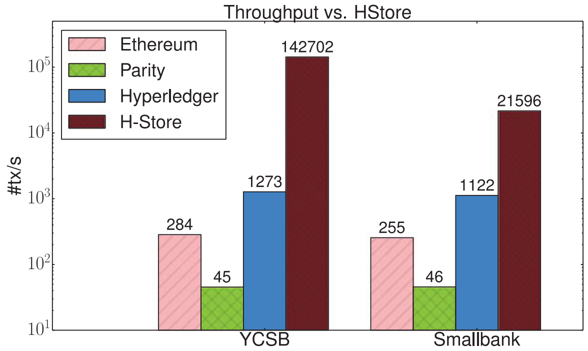


Fig. 7. Performance of the three blockchain systems versus H-Store.

6.1.2 *Scalability*

We fixed the client request rate (320 requests per second for Hyperledger, 160 requests per second for Ethereum and Parity) and increased both the number of clients and the number of servers. Fig. 8 illustrates how well the three systems scale to handle larger YCSB workloads. Parity's performance remains constant as the network size and offered load increase, due to the constant transaction processing rate at the servers. Interestingly, while Ethereum's throughput and latency degrade almost linearly beyond 8 servers, Hyperledger stops working beyond 16 servers.

To understand why Hyperledger failed to scale beyond 16 servers and 16 clients, we examined the system logs and find that the nodes never reached consensus on any batch of transactions. We observe a large number of messages being dropped even when there are fewer than 16 servers and clients. Furthermore, the servers repeatedly triggered view changes but never succeeded. At the client side, requests took longer to return as time passed, suggesting that the server or the network were saturated. Since the original PBFT protocol guarantees both liveness and safety, we can attribute this failure to scale to Hyperledger's implementation. Further investigation reveals that it is indeed the case.

Hyperledger uses gRPC for communication between servers. Each server keeps a separate message queue for every other servers in the network. The queue size is defined during initial setup (the default size being 1000

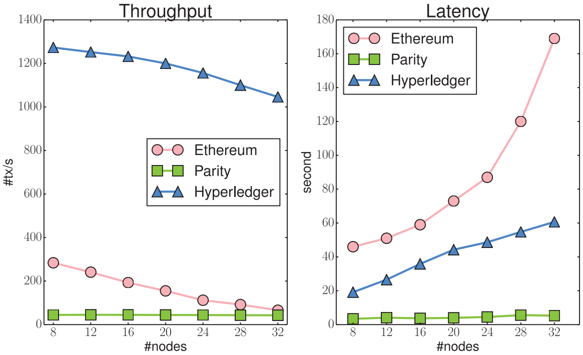


Fig. 9. Performance scalability (with eight clients).

messages) and the default behavior is to drop messages when the queue is full. In the current design, both client requests (transactions) and consensus messages (pre-prepare, prepare, commit, view changes) are sent on the same channel, that is they share the same queue. For large numbers of concurrent clients and servers, the channels are dominated by client requests, increasing the probability of consensus messages being dropped. Without a sufficient number of consensus messages, either a batch timer or a view-change timer will expire. In the first case, the PBFT leader resends messages of the current consensus round. In the second, the servers start the view change phase which broadcasts multiple rounds of consensus messages. As client requests are still occupying the network channels, both consensus or view change messages are dropped with high probability. Consequently, the network gets stuck in perpetual attempts to establish a stable view. The fact that PBFT is sensitive to network conditions has been observed in the past [79].

We note that in its latest release ($v1.0$) Hyperledger has replaced PBFT with a global ordering service. Implemented using Kafka, this new consensus engine may offer higher throughput than PBFT, but it offers no protection against Byzantine failures.

The results so far indicate that scaling both the number of clients and number of servers degrades the performance and even causes Hyperledger to fail. We next examined the costs of increasing the number of servers alone while fixing the number of clients to 8. Fig. 9 shows that the performance becomes worse as there are more servers, meaning that the systems incur some network overheads. For Hyperledger, having more servers means more messages being exchanged and higher overheads. In particular, to a confirm a batch of transaction in a larger network, the leader in Hyperledger needs to wait for larger sets of messages, therefore lowering overall throughputs. We note that with a fixed number of clients Hyperledger can scale up to 32 nodes, as oppose to failing after 16 nodes as in Fig. 8. This is because with fewer clients, the message queues at each node are not saturated with client requests and therefore consensus messages are less likely to get dropped.

For Ethereum, even though it is computation bound, it still consumes a modest amount of network resources for propagating transactions and blocks to other nodes. Furthermore, with larger network, the difficulty is increased to account for the longer propagation delays. We observe that to prevent the network from diverging, the difficulty level increases at a higher rate than the number of nodes. Thus, one reason for Ethereum's throughput degradation is due to network sizes. Another reason is that in our settings, 8 clients send requests to only 8 servers, but these servers do not always broadcast transactions to each other (they keep mining on their own transaction pool). As a result, the network mining capability is not fully utilized.

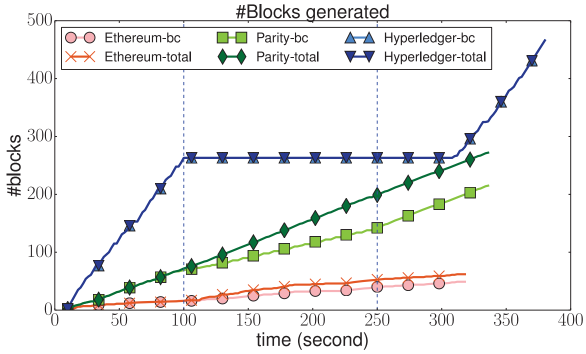


Fig. 10. Blockchain forks caused by attacks that partitions the network in half at 100^{th} second and lasts for 150 seconds. *X-total* means the total number of blocks generated in blockchain *X*, *X-bc* means the total number of blocks that reach consensus in blockchain *X*.

6.1.3 Fault Tolerance and Security

To evaluate how resilient the systems are to crash failures, we ran the systems with 12 and 16 servers, with 8 clients for over 5 minutes, during which we killed off 4 servers at 250th second. Due to space constraints, we only highlight key findings here and refer readers to [10] for more details. First, Ethereum is unaffected by the change, suggesting that the failed servers do not contribute significantly to the mining process. Second, Parity's throughput is also unaffected. It is because each node is given equal time slice during which it can generate block, thus failing 4 nodes in Parity means that the remaining 8 nodes are given bigger time slices. Third, Hyperledger stops generating blocks after the failure in the 12-server network, which is as expected because the PBFT can only tolerate fewer than 4 failures in a 12-server network. In the 16-server network, Hyperledger still generates blocks but at a lower rate, which were caused by the remaining servers having to stabilize the network after the failures by synchronizing their views.

We next simulated the attack that could make the blockchains vulnerable to double spending. The attack essentially creates network partition at 100th second that lasts for 150 seconds. We set the partition size to be half of the original.[5] Fig. 10 compares the vulnerability of the three blockchains with 8 clients and 8 servers. Recall that vulnerability is measured as the differences in the number of total blocks and the number of blocks on the main branch. We refer to this as Δ. Both Ethereum and Parity fork at 100th seconds, and Δ increases as time passes. For the attack duration, upto 30 percent of the blocks are generated in the forked branch, meaning that the systems are highly exposed to double spending or selfish mining attacks. When the partition heals, the nodes come to consensus on the main branch and discard the forked blocks. As a consequence, Δ stops

[5] We note that partitioning a N-node network in half does not mean there are $N/2$ Byzantine nodes. In fact, Byzantine tolerance protocols do not count network adversary as Byzantine failure.

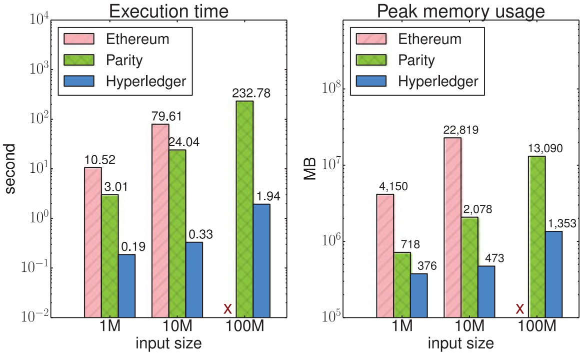


Fig. 11. CPUHeavy workload, '**X**' indicates Out-of-Memory error.

increasing shortly after 250th second. Hyperledger, in stark contrast, has no fork which is as expected because its consensus protocol is proven to guaranteed safety. We note, however, that Hyperledger takes longer than the other two systems to recover from the attacks (about 50 seconds more). This is because of the synchronization protocol performed after the partitioned nodes reconnect. In particular, when the nodes reconnect they enter the view change phase and exchange checkpointed states with each other in order to establish a new, stable view.

6.2 Micro Benchmarks

This section discusses the performance of the blockchains at the execution, data model and consensus layer. For the first two layers, the workloads were run with one client and one server. For the consensus layer, 8 clients and 8 servers were used.

6.2.1 *Execution Layer*

We deployed the CPUHeavy smart contract that is initialized with an integer array of a given size. The array is initialized in descending order. We invoked the contract to sort the array using quicksort algorithm, and measured the execution time and server's peak memory usage. The results for varying input sizes are shown in Fig. 11. Although Ethereum and Parity use the same execution engine, i.e., EVM, Parity's implementation is more optimized, therefore it is more computation and memory efficient. An interesting finding is that Ethereum incurs large memory overhead. In sorting 10 M elements, it uses 22 GB of memory, as compared to 473 MB used by Hyperledger. Ethereum runs out of memory when sorting more than 10 M elements. In Hyperledger, the smart contract is compiled and runs directly on the native machine within Docker environment, thus it does not have the overheads associated with executing high-level EVM byte code. As the result, Hyperledger is much more efficient in term of speed and memory usage. Finally, we note that all three systems fail to make use of the multi-core architecture, i.e., they execute the contracts using only one core.

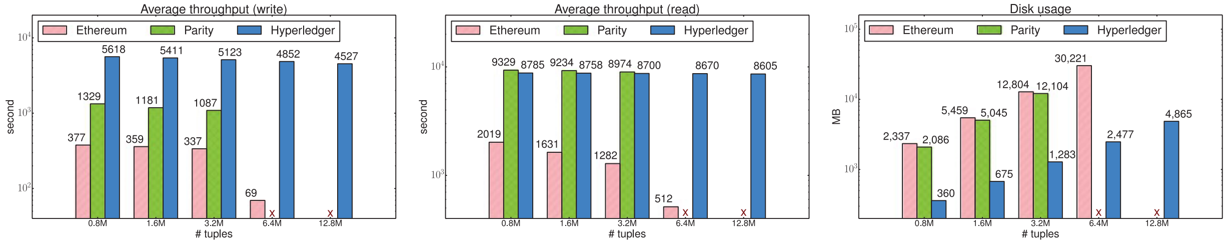
6.2.2 *Data Model - IOHeavy*

We deployed the IOHeavy smart contract that performs a number of read and write operations of key-value tuples. We used 20-byte keys and 100-byte values. Fig. 12 reports the throughput and disk usage for these operations. Ethereum and Parity use the same data model and internal index structure, therefore they incur similar space overheads. Both use an order of magnitude more storage space than Hyperledger which employs a simple key-value data model. Parity holds all the state information in memory, so it has better I/O performance but fails to handle large data (capped by over 3M states under our hardware settings). On the contrary, Ethereum only caches only parts of the state in memory (using LRU for eviction policy), therefore it can handle more data than Parity at the cost of throughput. Hyperledger leverages RocksDB to manage its states, which makes it more efficient at scale.

Hyperledger v1.0. We used the same IOHeavy smart contract to compare I/O performance of Hyperledger version $v1.0$ with the older version $v0.6$. As Fig. 14 illustrates, throughputs of $v1.0$ is an order of magnitude worse than that of $v0.6$. Furthermore, $v1.0$ crashes with more than 0.8 M operations, reporting exceptions about message oversizes. The significant gap can be attributed to the changes in the system architecture from $v0.6$ to $v1.0$. In the former, the nodes take part in PBFT to confirm a block. In this case, transactions in the IOHeavy workload incur no consensus overhead because there is only one node. In the latter, a new service, *the orderer*, is introduced into the network to order transactions and provide the consensus. With this new service, transactions in the IOHeavy workload now need to communicate with the orderer for them to be confirmed. More specifically, the nodes in $v1.0$ perform three more steps to finish a transaction compared to $v0.6$. As communication overhead increases, the throughputs decrease. This result suggests that replacing PBFT with a centralized service not only fails to protect the blockchain against Byzantine failures, but it may also impair the overall performance.

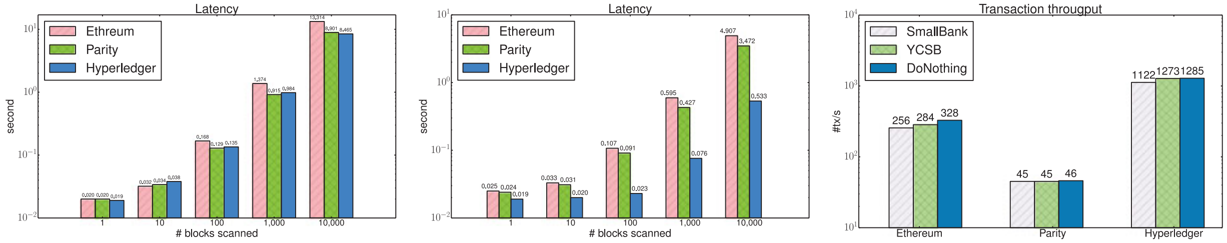
6.2.3 *Data Model - Analytics*

We implemented the analytics workload by initializing the three systems with over 120,000 accounts with a fixed balance. We then pre-loaded them with 100,000 blocks, each



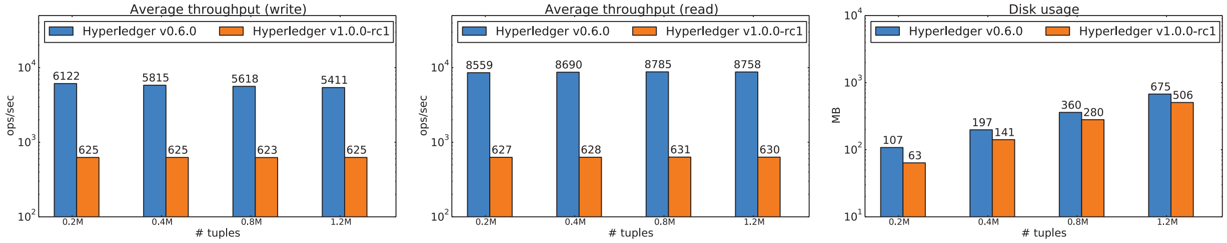
(a) Write (b) Read (c) Disk usage

Fig. 12. IOHeavy workload, '**X**' indicates Out-of-Memory error.



(a) Analytics workload (Q1) (b) Analytics workload (Q2) (c) DoNothing workload

Fig. 13. Analytics and DoNothing workloads.



(a) Write (b) Read (c) Read

Fig. 14. Hyperledger v0.6.0 versus v1.0.0 against IOHeavy workload.

contains 3 transactions on average. The transaction transfers a value from one random account to another random account. Due to Parity's overheads in signing transactions when there are many accounts, we considered transactions using only 1024 accounts. We then executed the two queries described in Section 5.3 and measured their latencies. Fig. 13 shows that the performance for Q1 is similar, whereas Q2 sees a significant gap between Hyperledger and the rest. We note that the main bottleneck for both Q1 and Q2 is the number of network (RPC) requests sent by the client. For Q1, the client sends the same number of requests to all systems, therefore their performance are similar. On the other hand, for Q2 the client sends one RPC per block to Ethereum and Parity, but only one RPC to Hyperledger because of our customized smart contract implementation. This saving in network roundtrip time translates to over 10x improvement in Q2 latency.

6.2.4 Consensus

We deployed the DoNothing smart contract that accepts a transaction and returns immediately. We measured the throughput of this workload and compare against that of YCSB and Smallbank. The differences compared to other workloads, shown in Fig. 13c is indicative of the cost of consensus protocol versus the rest of the software stack. In particular, for Ethereum we observe 10 percent increases in throughput as compared to YCSB, which means that execution of the YCSB transaction accounts for the 10 percent overhead. We observe no differences among these workloads in Parity, because the bottleneck in Parity is due to transaction signing (even empty transactions still need to be signed), not due to consensus or transaction execution.

7 Discussion

In this section, we first distill the lessons learned during the comparative studies of Ethereum, Parity and Hyperledger. We then discuss how design principles from database systems could help improve blockchain performance.

7.1 Lessons Learned from the Performance Study

Understanding blockchain systems. BLOCKBENCH aims to facilitate better understanding of the design and performance of different private blockchains. As more and more blockchain systems are being proposed, each offering different sets of feature, BLOCKBENCH's main value is that it narrows down the design space into four distinct abstraction layers. The layers are derived from our taxonomy presented in Section 4 which sufficiently captures the key and subtle characteristics of blockchain systems. By benchmarking these layers, one can gain insights into the design trade-offs and performance bottlenecks. For example, using the IOHeavy workload we identify that Parity trades performance for scalability by keeping states in memory. In addition, the workload reveals potential performance issues with the latest version of Hyperledger. Another example is the Analytics workload that demonstrates trade-offs in the data models. In particular, Hyperledger's simple key-value model means some analytical queries cannot be directly supported. However, it enables optimization that helps answering analytical queries more efficiently. Finally, we identify that the bottleneck in Parity is not due to the consensus protocol, but due to the server's transaction signing. We argue it is not easy to arrive at such insights without a systematic analysis framework.

Usability of blockchain. Our experience in working with the three blockchain systems confirms that in their current states, the blockchains are not yet ready for mass usage. Their designs and codebases are still being refined constantly, and there are no other established applications beside crypto-currency. Of the three systems, Ethereum is more mature both in terms of its codebase, user base and developer community. Another usability issue we encountered is in porting smart contracts from one system to another, due of their different programming models. This is likely to be exacerbated as more blockchain platforms are being proposed [6], [30], [44], [93].

Comparison to database systems. The comparison against H-Store presented in the previous section demonstrates that

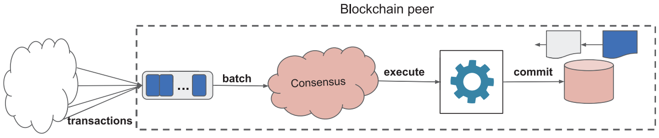


Fig. 15. The life of a blockchain transaction.

blockchains perform poorly at data processing tasks currently being handled by database systems. Although databases are designed without security and tolerance to Byzantine failures, we remark that the gap remains too big for blockchains to be disruptive to incumbent database systems. There are much for blockchains to learn from databases in terms of high-performance data processing, which we discuss next. Nevertheless, there are useful lessons that databases can take from the popularity and success of blockchains. Perhaps the most interesting lesson is that there is a clear need for Byzantine tolerant data processing systems which can accommodate a large number of users. Distributed databases have diverged from P2P system designs by assuming non-Byzantine failures [18], but with the increasing availability of faster and more trustworthy hardware, this may be the right time for the community to revise interest in high-performance, decentralized, P2P database systems.

7.2 Bringing Database Designs into Blockchains

The challenges in scaling blockchain by optimizing the consensus protocols are being addressed in many recent works [26], [61]. Nevertheless, we demonstrated in our comparative study that there are other performance bottlenecks beside consensus. Fig. 15 illustrates different stages that a transaction goes through before it is considered committed to the blockchain. Each stage could be a potential bottleneck and be subject to future optimizations. First, transactions are batch into a block. Next, the block becomes input to the consensus protocol, and if selected by the protocol it is sent to the execution engine. Finally, the engine executes the entire batch of transaction, creates new states and appends the block to the chain.[6] We note the striking resemblance with the flow of a transaction in a distributed database. In fact, the only difference being the consensus protocol: databases use two-phase commit or Paxos, whereas blockchains use Byzantine tolerant protocols or variants thereof. Given the similarity, we propose four approaches inspired by design principles in databases to improve blockchain performance.

Decoupling the layers and optimize them individually. One possible direction is to decouple storage, execution engine and consensus layer from each other, then optimize and scale them independently. Tezos and Corda, for examples, have decoupled the consensus layer by outsourcing it to separate parties. The data model layer could also be decoupled. For instance, current systems employ generic key-value storage, which may not be best suited to the unique data structure and operations in blockchain. UStore [94] demonstrates that a storage designed around the blockchain data structure is able to achieve better performance than existing implementations. Most importantly, we observe that current data models in Ethereum and Hyperledger are not ideal for answering analytics queries. In particular, both do not support fine-grained versioning at the transaction level: it is not possible to immediately query previous versions of a state except for querying at block level and processing the state of each block. Implementing a new data model is less complex when the storage layer is decoupled from the rest of the blockchain stack.

Embracing new hardware primitives. Many data processing systems are taking advantage of new hardware to boost their performance [95], [96], [97]. For blockchain, using trusted hardware, the underlying Byzantine fault tolerance protocols can be modified to incur fewer network messages [85]. With trusted hardware, the blockchain can tolerate more failures, and because of fewer network messages it can scale better. Systems like Parity and Ethereum can take advantage of multi-core CPUs and large memory to improve contract execution and I/O performance.

Sharding. Blockchain is essentially a replicated state machine system, in which each node maintains the same data. As such, blockchains are fundamentally different to database systems such as H-Store in which the data is partitioned (or sharded) across the nodes. Sharding helps reduce the computation cost and can make transaction processing faster. The main challenge with sharding is to ensure consistency among multiple shards. However, existing consistency protocols used in database systems do not work under Byzantine failure. Nevertheless, their designs can offer insights into realizing a more scalable sharding protocol for blockchain. Corda partitions the ledger into sub-ledgers belonging to smaller group, thus avoiding the need to broadcast transactions to the entire network. However, it still depends on an external, centralized component for consensus between sub-ledgers. Recent work [26] has demonstrated the feasibility of sharding the consensus protocol, making important steps towards partitioning the entire blockchain.

Supporting declarative language. Having a set of high-level operations that can be composed in a declarative manner makes it easy to define complex smart contracts. It also opens up opportunities for low-level optimizations that speed up contract execution. Hawk [89] is a recent example that applies compiler techniques to hide complexity of implementing privacy preserving smart contracts. However, it aims to strengthen security of the smart contracts rather than improve their performance.

8 Conclusion

In this paper, we have conducted a comprehensive survey on blockchain technologies. We laid out four underpinning concepts behind blockchains and analyzed the state of the art using these concepts. We presented our benchmarking framework, BLOCKBENCH which is designed to evaluate performance of blockchains as data processing platforms. Finally, we discussed four potential research directions, inspired by database design principles, for improving blockhchain performance. We hope that the survey and benchmarking framework would serve to guide the design and implementation of future blockchain systems that are not only secure, but scalable and usable in the real world.

[6] Note that the execution phase can be considered the last part of the consensus phase, because during execution a node may detect conflicting transactions and abort the current consensus round.

ACKNOWLEDGMENTS

This work is funded by the National Research Foundation, Prime Ministers Office, Singapore, under its Competitive Research Programme (CRP Award No. NRF-CRP8-2011-08). The authors would like to thank colleagues and the anonymous reviewers who have provided valuable feedback to help improve the paper.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008, http://bitcoin.org/bitcoin.pdf, Last accessed: 2017.

[2] Q. Lin, P. Chang, G. Chen, B. C. Ooi, K. Tan, and Z. Wang, “Towards a non-2PC transaction management in distributed database systems,” in *Proc. ACM Int. Conf. Manag. Data*, 2016, pp. 1659–1674.

[3] A. Thomson, T. Diamond, S. Weng, K. Ren, P. Shao, and D. J. Abadi, “Calvin: Fast distributed transactions for partitioned database systems,” in *Proc. ACM Int. Conf. Manag. Data*, 2012, pp. 1–12.

[4] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica, “Coordination avoidance in database systems,” *Proc. VLDB Endowment*, vol. 8, no. 3, pp. 185–196, 2014.

[5] Ethereum blockchain app platform. (2017). [Online]. Available: https://www.ethereum.org/

[6] Ripple, “Ripple.” (2017). [Online]. Available: https://ripple.com

[7] Melonport, “Blockchain software for asset management.” (2017). [Online]. Available: http://melonport.com

[8] J. P. Morgan and O. Wyman, “Unlocking economic advantage with blockchain. a guide for asset managers.” 2016, http://www.oliverwyman.com/our-expertise/insights/2016/jul/unlocking-economic-advantage-with-blockchain.html, Last accessed: 2017.

[9] G. S. Group, “Blockchain: Putting theory into practice,” 2016, http://www.goldmansachs.com/our-thinking/pages/blockchain/, Last accessed: 2017.

[10] T. T. A. Dinh, J. Wang, G. Chen, L. Rui, K.-L. Tan, and B. C. Ooi, “BLOCKBENCH: A benchmarking framework for analyzing private blockchains,” in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2017, pp. 1085–1100.

[11] Ethcore, “Parity: Next generation ethereum browser.” (2017). [Online]. Available: https://ethcore.io/parity.html

[12] Hyperledger, “Blockchain technologies for business.” (2017). [Online]. Available: https://www.hyperledger.org

[13] F. Tschorsch and B. Scheuermann, “Bitcoin and beyond: A technical survey on decentralized digital currencies,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 3, pp. 2084–2123, Jul.–Sep. 2016.

[14] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, “SoK: Research perspectives and challenges for bitcoin and crypto-currencies,” in *Proc. IEEE Symp. Security Privacy*, 2015, pp. 104–121.

[15] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, “The end of an architectural era (it’s time for a complete rewrite),” in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 1150–1160.

[16] J. C. Corbett, et al., “Spanner: Google’s globally-distributed database,” in *Proc. 10th USENIX Symp. Operating Syst. Des. Implementation*, 2012, pp. 261–264.

[17] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *Proc. 3rd USENIX Symp. Operating Syst. Des. Implementation*, 1999, pp. 173–186.

[18] Q. H. Vu, M. Lupu, and B. C. Ooi, *Peer-to-Peer Computing Principles and Applications*. Berlin, Germany: Springer-Verlag, 2009.

[19] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” in *Proc. 18th Int. Conf. Financial Cryptography Data Security*, 2014, pp. 436–454.

[20] K. Croman, et al., “On scaling decentralized blockchains,” in *Proc. 3rd Workshop Bitcoin Blockchain Res.*, 2016, pp. 106–125.

[21] F. P. Junqueira, B. C. Reed, and M. Serafini, “Zab: High-performance broadcast for primary-backup systems,” in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2011, pp. 245–256.

[22] D. Ongaro and J. K. Ousterhout, “In search of an understandable consensus algorithm,” in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 305–319.

[23] L. Lamport, “Paxos made simple, fast, and byzantine,” in *Proc. 6th Int. Conf. Principles Distrib. Syst.*, 2002, pp. 7–9.

[24] Monax: The ecosystem application platform. (2017). [Online]. Available: https://monax.io

[25] M. Vukolic, “The quest for scalable blockchain fabric: Proof-of-work versus BFT replication,” in *Proc. Int. Workshop Open Problems Netw. Security - iNetSec*, 2015, pp. 112–125.

[26] L. Luu, V. Narayanan, C. Zhang, K. Baweija, S. Gilbert, and P. Saxena, “A secure sharding protocol for open blockchains,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 17–30.

[27] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling byzantine agreements for cryptocurrencies,” in *Proc. 26th ACM Symp. Operating Syst. Principles*, 2017, pp. 51–68.

[28] Proof of authority chains. (2017). [Online]. Available: https://github.com/paritytech/parity/wiki/Proof-of-Authority-Chains

[29] Stellar. (2017). [Online]. Available: https://www.stellar.org/

[30] Kadena. (2017). [Online]. Available: http://kadena.io/

[31] Bigchaindb: A scalable blockchain database. (2017). [Online]. Available: https://github.com/bigchaindb/bigchaindb

[32] Hyperledger fabric v0.6.0. (2017). [Online]. Available: https://github.com/hyperledger/fabric/releases/tag/v0.6.0-preview

[33] Hyperledger fabric v1.0.0-rc1. (2017). [Online]. Available: https://github.com/hyperledger/fabric/releases/tag/v1.0.0-rc1

[34] Global decentralized currency. (2017). [Online]. Available: https://litecoin.org/

[35] E. B. Sasson, et al., “Zerocash: Decentralized anonymous payments from bitcoin,” in *Proc. IEEE Symp. Security Privacy*, 2014, pp. 459–474.

[36] Multichain: Open platform for blockchain applications. (2017). [Online]. Available: https://www.multichain.com/

[37] JPMorgan, “Enterprise-ready distributed ledger and smart contract platforms.” (2017). [Online]. Available: https://github.com/jpmorganchase/quorum

[38] Hydrachain: Permissioned distributed ledger based on ethereum. (2017). [Online]. Available: https://github.com/HydraChain/hydrachain

[39] Openchain: Enterprise-ready blockchain. (2017). [Online]. Available: https://github.com/openchain

[40] Openchain: Next generation blockchain. (2017). [Online]. Available: https://iota.org/

[41] Tendermint: Blockchain app development simplified. (2017). [Online]. Available: http://tendermint.com/

[42] C. Copeland and H. Zhong, “Tangaroa: A byzantine fault tolerant raft.” (2017). [Online]. Available: https://github.com/chrisnc/tangaroa

[43] Dfinity. (2017). [Online]. Available: https://dfinity.network/

[44] L. M. Goodman, “Tezos: A self-amending crypto-ledger position paper,” 2014, https://www.tezos.com/static/papers/position_paper.pdf, Last accessed: 2017.

[45] R. G. Brown, “Introducing R3 corda(TM): A distributed ledger designed for financial services,” *R3CEV Blog*, 2016, http://www.r3cev.com/blog/2016/4/4/introducing-r3-corda-a-distributed-ledger-designed-for-financial-services, Last accessed: 2017.

[46] Hyperledger, “Sawtooth distributed ledger,” 2016, https://github.com/hyperledger/sawtooth-core, Last accessed: 2017.

[47] Decentralized autonomous organization. (2017). [Online]. Available: https://www.ethereum.org/dao

[48] S. Eskandari, D. Barrera, E. Stobert, and J. Clark, “A first look at the usability of bitcoin key management,” in *Proc. Workshop Usable Security*, 2015.

[49] T. T. A. Dinh, P. Saxena, E. C. Chang, B. C. Ooi, and C. Zhang, “M2R: Enabling stronger privacy in MapReduce computation,” in *Proc. 24th USENIX Conf. Security, Symp.*, 2015, pp. 447–462.

[50] W. Zheng, A. Dave, J. Beekman, R. A. Popa, J. Gonzalez, and I. Stoica, “Oqaque: An oblivious and encrypted distributed analytics platform,” in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 283–298.

[51] F. Zhang, E. Cecchetti, K. Croman, and E. Shi, “Town crier: An authenticated data feed for smart contracts,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 270–282.

[52] B. Smyth, M. Ryan, and L. Chen, “Direct anonymous attestation: Ensuring privacy with corrupt administrator,” in *Proc. 4th Eur. Workshop Security Privacy Ad Hoc Sensor Netw.*, 2007, pp. 218–231.

[53] S. Meiklejohn, et al., “A fistful of bitcoins: Characterizing payments among men with no names,” in *Proc. 9th ACM SIGCOMM Conf. Internet Meas.*, 2013, pp. 127–140.

[54] Chainalysis - blockchain analysis. (2017). [Online]. Available: https://www.chainalysis.com

[55] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoins: Anonymous distributed e-cash from bitcoin," in *Proc. IEEE Symp. Security Privacy*, 2013, pp. 397–411.

[56] SCIPR Lab, "libsnark: A C++ library for zkSNARK proofs." (2017). [Online]. Available: https://github.com/scipr-lab/libsnark

[57] Lightning network: Scalable, instance bitcoin/blockchain transactions. (2017). [Online]. Available: https://lightning.network/

[58] S. Goldfeder, J. Bonneau, E. W. Felten, J. A. Kroll, and A. Narayanan, "Securing bitcoin wallets via threshold signatures," 2016, http://www.cs.princeton.edu/~stevenag/bitcoin_threshold_signatures.pdf, Last accessed: 2017.

[59] V. Shoup, "Practical threshold signatures," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2000, pp. 207–220

[60] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *J. Cryptology*, vol. 17, pp. 297–319, 2004.

[61] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *Proc. 25th USENIX Security Symp.*, 2016, pp. 279–296.

[62] E. Syta, et al., "Keeping authorities honest or bust with decentralized witess cosigning," in *Proc. IEEE Symp. Security Privacy*, 2016, pp. 526–545.

[63] J. Behl, T. Distler, and R. Kapitza, "Scalable BFT for multi-cores: Actor-based decomposition and consensus-oriented parallelization," in *Proc. 10th Workshop Hot Topics Syst. Dependability*, 2014, pp. 9–9.

[64] M. Zbierski, "Parallel byzantine fault tolerance," in *Soft Computing in Computer and Information Science*. Berlin, Germany: Springer, 2015, pp. 321–333.

[65] W. Zhao, "Optimistic byzantine fault tolerance," *Int. J. Parallel Emergent Distrib. Syst.*, vol. 31, no. 3, pp. 254–267, May 2016.

[66] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," 2016, https://www.stellar.org/papers/stellar-consensus-protocol.pdf, Last accessed: 2017.

[67] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse, "Bitcoin-NG: A scalable blockchain protocol," in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2016, pp. 45–59.

[68] Proof of stake. (2017). [Online]. Available: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ

[69] Slimcoin a peer-to-peer crypto-currency with proof-of-burn. (2017). [Online]. Available: www.slimcoin.club/whitepaper.pdf

[70] Peercoin - secure & sustainable cryptocoin. (2017). [Online]. Available: https://peercoin.net

[71] Proof of elapsed time (poet). (2017). [Online]. Available: https://intelledger.github.io/introduction.html#proof-of-elapsed-time-p%oet

[72] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake," *Int. Assoc. Cryptologic Res. Cryptology ePrint Archive*, vol. 2014, 2014, Art. no. 452.

[73] S. Park, K. Pietrzak, A. Kwon, J. Alwen, G. Fuchsbauer, and P. Gazi, "Spacemint: A cryptocurrency based on proofs of space," *Int. Assoc. Cryptologic Res. Cryptology ePrint Archive*, vol. 2015, 2015, pp. 528.

[74] M. Milutinovic, W. He, H. Wu, and M. Kanwal, "Proof of luck: An efficient blockchain consensus protocol," *Int. Assoc. Cryptologic Res. Cryptology ePrint Archive*, vol. 2017, 2017, Art. no. 249.

[75] Y. Sompolinsky and A. Zohar, "Accelerating bitcoin's transaction processing: Fast money grows on trees, not chains," Cryptology ePrint Archive, Report 2013/881, 2013. [Online]. Available: https://eprint.iacr.org/2013/881.pdf

[76] S. Deetman, "Bitcoin could consume as much electricity as denmark by 2020," 2016. [Online]. Available: https://tinyurl.com/yd8hq6n2

[77] NXT: Own your data, control your vote. (2017). [Online]. Available: https://nxt.org/

[78] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *Proc. 24th USENIX Security Symp.*, 2015, pp. 129–144.

[79] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making byzantine fault tolerant systems tolerate byzantine failure," in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2009, pp. 153–168.

[80] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," in *Proc. 23rd ACM Symp. Operating Syst. Principles*, 2007, pp. 45–58.

[81] S. Liu, P. Viotti, C. Cachin, V. Quema, and M. Vukolic, "XFT: Practical fault tolerance beyond crashes," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 485–500.

[82] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honeybadger of BFT protocols," in *Proc. 23rd ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 31–42.

[83] Intel, "Intel software guard extensions." (2017). [Online]. Available: https://software.intel.com/en-us/sgx

[84] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security, enabling trusted computing in embedded systems," ARM white paper, vol. 3, no. 4, 2004.

[85] B. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested append-only memory: Making adversaries stick to their word," in *Proc. 21st ACM Symp. Operating Syst. Principles*, 2007, pp. 189–204.

[86] J. Behl, T. Distler, and R. Kapitza, "Hybrids on steroids: SGX-based high performance BFT," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2017, pp. 222–237.

[87] Interledger: The protocol for connecting blockchains. (2017). [Online]. Available: https://interledger.org/

[88] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. 23rd ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 254–269.

[89] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy preserving smart contracts," in *Proc. 23rd ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 839–858.

[90] BlockBench: private blockchains benchmarking. (2017). [Online]. Available: https://github.com/ooibc88/blockbench

[91] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 143–154.

[92] M. J. Cahill, U. Röhm, and A. D. Fekete, "Serializable isolation for snapshot databases," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2008, pp. 729–738.

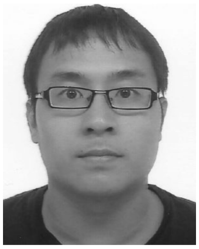
[93] Crypti, "A decentralized application platform." [Online]. Available: https://crypti.me

[94] A. Dinh, et al., "UStore: a distributed storage with rich semantics," (2017). [Online]. Available: https://arxiv.org/pdf/1702.02799.pdf

[95] K. Tan, Q. Cai, B. C. Ooi, W. Wong, C. Yao, and H. Zhang, "In-memory databases: Challenges and opportunities from software and hardware perspectives," *SIGMOD Rec.*, vol. 44, no. 2, pp. 35–40, 2015.

[96] H. Zhang, G. Chen, B. C. Ooi, K. Tan, and M. Zhang, "In-memory big data management and processing: A survey," *IEEE Trans. Knowledge and Data Eng.*, vol. 27, no. 7, pp. 1920–1948, Jul. 2015.

[97] A. Dragojevic, et al., "No compromises: Distributed transactions with consistency, availability, and performance," in *Proc. 25th Symp. Operating Syst. Principles*, 2015, pp. 54–70.



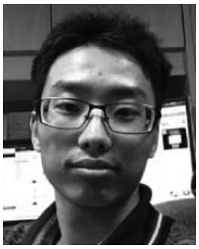
Tien Tuan Anh Dinh received the BEng and PhD degrees from the University of Birmingham, United Kingdom, in 2006 and 2010, respectively. He is currently a senior research fellow in the Department of Computer Science, National University of Singapore. His research interests include applied cryptography, trusted computing, distributed systems, big data analytics, databases, and machine learning.

Rui Liu received the BEng degree from the Northeastern University of China, and the MPhil degree from Hong Kong Polytechnic University, in 2012 and 2015, respectively. He was a research assistant in the Department of Computer Science, National University of Singapore. His research interests include distributed systems, data management, mobile computing, and ubiquitous computing.

Meihui Zhang received the BEng degree in computer science from the Harbin Institute of Technology, China, and the PhD degree in computer science from the National University of Singapore, in 2008 and 2013, respectively. She is currently a professor in the Beijing Institute of Technology, and was an assistant professor with the Singapore University of Technology and Design (SUTD) from 2014 to 2017. Her research interest includes crowdsourcing-powered data analytics, massive data integration, spatio-temporal databases, blockchain, and AI. She is a member of the IEEE.

Gang Chen received the BSc, MSc, and the PhD degrees in computer science and engineering from Zhejiang University, in 1993, 1995, and 1998, respectively. He is currently a professor in the College of Computer Science, Zhejiang University. His research interests include databases, information retrieval, information security, and computer-supported cooperative work. He is also the executive director of the Zhejiang University Netease Joint Lab on Internet Technology. He is a member of the IEEE.

Beng Chin Ooi is currently a distinguished professor of computer science with the National University of Singapore. His research interests include database system architectures, performance issues, indexing techniques, and query processing, in the context of multimedia, spatio-temporal, distributed, parallel, peer-to-peer, in memory, and cloud database systems. He has served as a PC member for a number of international conferences (including SIGMOD, VLDB, ICDE, WWW, EDBT, DASFAA, GIS, KDD, CIKM, and SSD). He was an editor of the *Very Large Data Bases Journal* and the *IEEE Transactions on Knowledge and Data Engineering*, editor-in-chief of the *IEEE Transactions on Knowledge and Data Engineering (2009–2012)*, and a co-chair of the ACM SIGMOD Jim Gray Best Thesis Award committee. He is serving as a trustee board member and the president of the VLDB Endowment. He is a fellow of the IEEE and ACM.



Ji Wang received the BSc degree from the Harbin Institute of Technology, China, and the MSc degree from the National University of Singapore, in 2015 and 2017, respectively. He was a research assistant in the Department of Computer Science, National University of Singapore. His research interests include distributed systems and data management.