B

Blockchain Transaction Processing

Suyash Gupta and Mohammad Sadoghi
Department of Computer Science, University of California, Davis, Davis, CA, USA

Synonyms

Blockchain consensus; Blockchain data management; Cryptocurrency

Definitions

A blockchain is a linked list of immutable tamper-proof blocks, which is stored at each participating node. Each block records a set of transactions and the associated metadata. Blockchain transactions act on the identical ledger data stored at each node. Blockchain was first perceived by Satoshi Nakamoto (Satoshi 2008), as a peer-to-peer money exchange system. Nakamoto referred to the transactional tokens exchanged among clients in his system as Bitcoins.

Overview

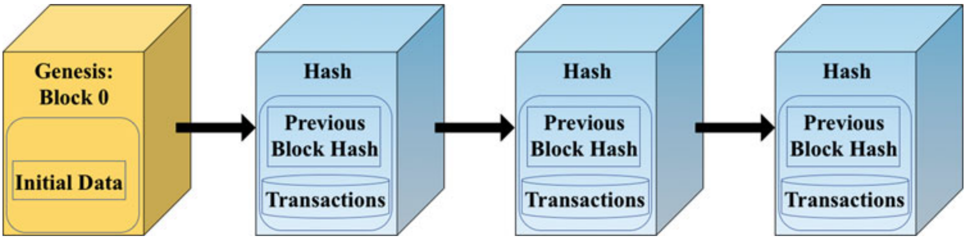
In 2008, Satoshi Nakamoto (Satoshi 2008) came up with the design of an unanticipated technology that revolutionized the research in the distributed systems community. Nakamoto presented the design of a peer-to-peer money exchange process that was shared yet distributed. Nakamoto named his transactional token as *Bitcoin* and brought forth design of a new paradigm *blockchain*.

The key element of any blockchain system is the existence of an immutable tamper-proof *block*. In its simplest form, a block is an encrypted aggregation of a set of transactions. The existence of a block acts as a guarantee that the transactions have been executed and verified.

A newly created block is appended to an existing chain of blocks. This chain of blocks is predominantly a *linked list* which associates one block with the other. The initial block of any such list is a *genesis block* (Decker and Wattenhofer 2013). Genesis block is a special block that is numbered zero and is hard-coded in the blockchain application. Each other block links to some previously existing block. Hence, a blockchain grows by appending new blocks to the existing chain.

A transaction in a blockchain system is identical to any distributed or OLTP transaction (TPP Council 2010) that acts on some data.

S. Sakr, A. Zomaya (eds.), *Encyclopedia of Big Data Technologies*,
https://doi.org/10.1007/978-3-319-63962-8_333-1



Blockchain Transaction Processing, Fig. 1 Basic blockchain representations

Traditional blockchain applications (such as Bitcoin) consist of transactions that represent an exchange of money between two entities (or users). Each valid transaction is recorded in a block, which can contain multiple transactions, for efficiency. Immutability is achieved by leveraging strong cryptographic properties such as hashing (Katz and Lindell 2007). Figure 1 presents the structure of a simple blockchain.

A blockchain is a *linked list* in a true sense, as each block stores the hash of the previous block in its chain. Each block also digitally signs its contents by storing the hash of its contents inside the block. These hashes provide cryptographic integrity, as any adversary intending to modify a block needs to also modify all the previous blocks in a chain, which makes the attack cryptographically infeasible. A key design strategy is to construct a Merkle tree (Katz and Lindell 2007) to efficiently store and verify the hashes. Thus, each block only stores the *root* of the Merkle tree, as given the root, it is easy to verify the immutability.

The preceding discussion allows us to summarize that a blockchain aims at securely storing a set of transactions. In the succeeding sections, we discuss in detail the transaction processing in a blockchain system. We also study mechanisms to validate these transactions and analyze some blockchain applications that employ the same.

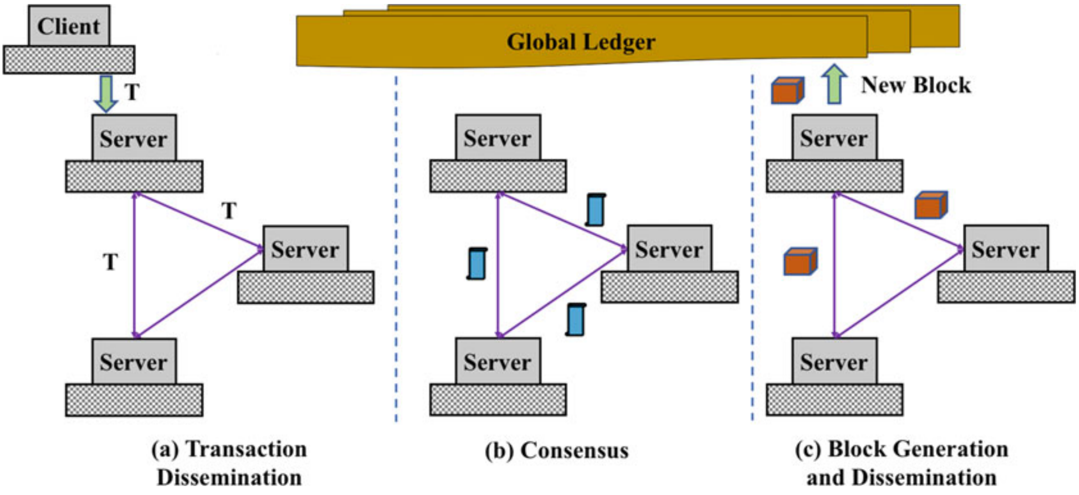
Key Research Findings

Transactions in a blockchain system are identical to their traditional database counterparts. These transactions are issued by the clients to the servers of the blockchain system (Nawab 2018). These transactions act on the data stored on all the participating servers. In its vanilla form, a blockchain transaction could be visualized as a set of *read/write* operations performed on each node of a replicated distributed database. To determine an ordering for all the incoming transactions, each blockchain application employs a consensus (Steen and Tanenbaum 2017) protocol.

A distributed consensus algorithm (Lamport 1998; Gray and Lamport 2006) allows a system to reach a common decision, respected by the majority of nodes. Recent blockchain technologies present several strategies for establishing consensus: Proof-of-Work (Jakobsson and Juels 1999; Satoshi 2008), Proof-of-Stake (King and Nadal 2012), Proof-of-Authority (Parity Technologies 2018), Practical Byzantine Fault Tolerance (Castro and Liskov 1999; Cachin 2016), and so on. To quantify the allowed set of nodes that can create a block (or participate in the consensus process), it is also necessary to characterize the topologies for blockchain systems.

Blockchain Execution

Figure 2 illustrates the three main phases required by any blockchain application to create a new block. The client transmits a transactional request to one of the servers. This server multicasts the request to all other servers. We term this phase as *transaction dissemination*. Once all the servers have a copy of client request, they initiate a *consensus* protocol. The choice of underlying consensus protocol affects the time complexity and resource consumption. The winner of the consensus phase generates the next block and transmits it to all other servers. This transmission



Blockchain Transaction Processing, Fig. 2 Blockchain flow: Three main phases in any blockchain application are represented. (**a**) Client sends a transaction to one of the servers, which it disseminates to all the other servers. (**b**) Servers run the underlying consensus protocol, to determine the block creator. (**c**) New block is created and transmitted to each node, which also implies adding to global ledger

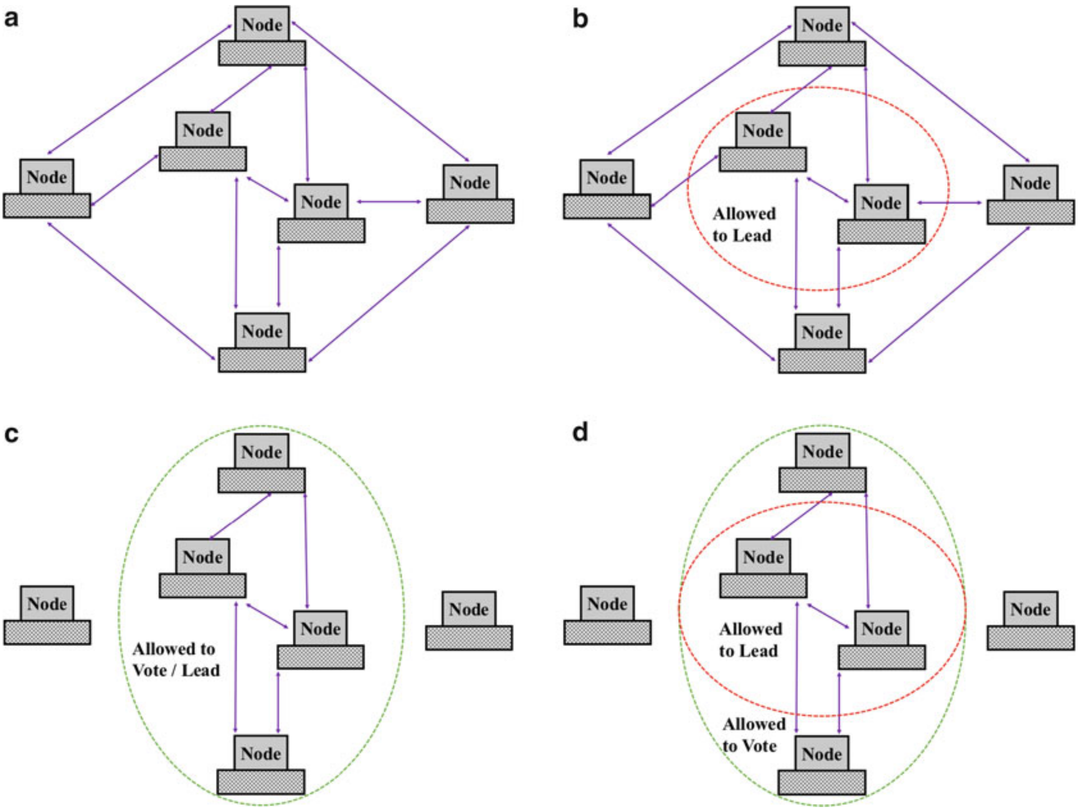
process is equivalent to adding an entry (block) to the global distributed ledger.

Blockchain Topologies

A key parameter that renders the design of a blockchain application is the categorization of nodes to be part of the application. Recent works (Pilkington 2015; Cachin and Vukolic 2017) categorize blockchain systems as either public, private, permissioned, or hybrid. Although the characteristics of a public blockchain is clearly stated in the community, there is no common consensus on the terminology pertaining to the other terms.

We categorize blockchain systems under four heads: public, private, permissioned, and hybrid. Figure 3 presents a pictorial representation of the different categories. In these figures nodes that are not allowed to participate in the activities pertaining to the network lack connections to any node in the network. We use different circles to demarcate different zones of operation; certain nodes are allowed to lead (or create the block), and some are allowed to participate in the consensus protocol. *Public blockchain* systems such as Bitcoin (Satoshi 2008) and Ethereum (Wood 2015) allow any node to participate in the consensus process, and any node can generate the next valid block. Hence, if all the nodes have same resources, then each node has equal probability of creating a block (We are assuming fair and ideal conditions, where each node works independently.). *Private blockchain* systems are at the other end of the spectrum, as they allow only some nodes to be part of the consensus process, and only a subset of these nodes can generate the next block. These systems could be utilized at a banking organization which only allows its customers to participate in the consensus, while its employees are only permitted to commit the results by creating a block.

Hybrid blockchain systems attain a middle ground between the two extremes. These systems allow any node to be part of the consensus process, but only some designated nodes are allowed to form the next block. Cryptocurrency Ripple (Schwartz et al. 2014) supports a variant of the hybrid model, where some public institutions can act as transaction validators. *Permissioned blockchain* systems are the restricted variant of public blockchain systems. These systems enlist few nodes as the members with equal rights, that is, each node as part of the system can form a block. These systems allow us to represent internal working of any organization, social media groups, and an early stage startup.



Blockchain Transaction Processing, Fig. 3 Topologies for blockchain systems. (**a**) Public blockchain. (**b**) Hybrid blockchain. (**c**) Permissioned blockchain. (**d**) Private blockchain

Blockchain Consensus

The common denominator for all the blockchain applications is the underlying distributed consensus algorithm. Nakamoto suggested **Proof-of-Work** (Satoshi 2008) (henceforth referred as PoW) for achieving consensus among the participating nodes. PoW requires each node to demonstrate its ability to solve a nontrivial task. The participating nodes compete among themselves to solve a complex puzzle (such as computation of a 256-bit SHA value). The node which computes the solution gets the opportunity to generate the next *block*. It also disseminates the block to all the nodes, which marks as the *proof* that it performed some nontrivial computation. Hence, all other nodes respect the winner's ability and reach the consensus by continuing to chain ahead of this block.

In PoW the accessibility to larger computational resources determines the winner node(s). However, it is possible that multiple nodes could lay claim for the next block to be added to the chain. Based on the dissemination of the new block, this could lead to *branching* out of the single chain. Interestingly, these branches are often short-lived, as all the nodes tend to align themselves to the longest chain, which in turn leads to pruning of the branches. It is important to understand that a node receives incentive when it is able to append a block to the longest chain. Hence, it is to their advantage to always align with the longest chain; otherwise they would end up spending their computational resources for zero gains. Note: PoW algorithm can theoretically be compromised by an adversary controlling at least 51% of computational resources in the network. This theoretical possibility has practical implications as a group of miners can share resources to generate blocks faster, skewing the decentralized nature of the network.

Proof-of-Stake (King and Nadal 2012) (henceforth referred at PoS) aims at preserving the decentralized nature of the blockchain network. In PoS algorithm a node with $n\%$ resources gets $n\%$ time opportunity to create a block. Hence, the underlying principle in PoS is that the node with higher stake lays claim to the generation of the next block. To determine a node's stake, a combination of one or more factors, such as wealth, resources, and so on, could be utilized. PoS algorithm requires a set of nodes to act as validators. Any node that wants to act as a validator needs to *lock out* its resources, as a proof of its stake.

PoS only permits a validator to create new blocks. To create a new block, a set of validators participate in the consensus algorithm. PoS consensus algorithm has two variants: (i) chain-based and (ii) BFT-style. *Chain-based* PoS algorithm uses a pseudorandom algorithm to select a validator, which then creates a new block and adds it to the existing chain of blocks. The frequency of selecting the validator is set to some predefined time interval. *BFT-style* algorithm runs a byzantine fault tolerance algorithm to select the next valid block. Here, validators are given the right to propose the next block, at random. Another key difference between these algorithms is the synchrony requirement; *chain-based* PoS algorithms are inherently synchronous, while *BFT-style* PoS is partially synchronous.

Early implementations of PoS such as Peercoin (King and Nadal 2012) suffer from nonconvergence, that is, lack of single chain (also referred as "nothing-at-stake" attack). Such a situation happens when the longest chain branches into multiple forks. Intuitively, the nodes should aim at converging the branches into the single longest chain. However, if the participating nodes are incentive driven, then they could create blocks in both the chains, to maximize their gains. Such an attack is not possible on PoW as it would require a node to have double the number of computational resources, but as PoS does not require solving a complex problem, nodes can easily create multiple blocks.

Proof-of-Authority (Parity Technologies 2018) (henceforth referred as PoA) is designed to be used alongside nonpublic blockchain systems. The key idea is to designate a set of nodes as the authority. These nodes are entrusted with the task of creating new blocks and validating the transactions. PoA marks a block as part of the blockchain if it is signed by majority of the authorized nodes. The incentive model in PoA highlights that it is in the interest of an authority node to maintain its reputation, to receive periodic incentives. Hence, PoA does not select nodes based on their claimed stakes.

Proof-of-Space (Ateniese et al. 2014; Dziembowski et al. 2015) also known as *Proof-of-Capacity* (henceforth referred as PoC) is a consensus algorithm orthogonal to PoW. It expects nodes to provide a proof that they have sufficient "storage" to solve a computational problem. PoC algorithm targets computational problems such as *hard-to-pebble graphs* (Dziembowski et al. 2015) that need large amount of memory storage to solve the problem. In the PoC algorithm, the verifier first expects a prover to commit to a labeling of the graph, and then he queries the prover for random locations in the committed graph. The key intuition behind this approach is that unless the prover has sufficient storage, he would not pass the verification. PoC-based cryptocurrency such as SpaceMint (Park et al. 2015) claims PoC-based approaches more resource efficient to PoW as storage consumes less energy.

A decade prior to the first blockchain application, distributed consensus problem had excited the researchers. Paxos (Lamport 1998) and Viewstamped Replication (Oki and Liskov 1988) presented key solutions to distributed consensus when the node failures were restricted to *fail-stop*. Castro and Liskov (1999) presented their novel *Practical Byzantine Fault Tolerance* (henceforth referred as PBFT) consensus algorithm to handle byzantine failures. Interestingly, all of previously discussed consensus algorithms provide similar guarantees as PBFT. Garay et al. (2015) helped in establishing an equivalence between the PoW and general consensus algorithms. This equivalence motivated the community to design efficient alternatives to PoW.

PBFT runs a three-phase protocol to reach a consensus among all the non-byzantine nodes. PBFT requires a bound on the number of byzantine nodes. If “n” represents the total number of nodes in the system, then PBFT bounds the number of byzantine nodes “f” as $f \leq \lfloor \frac{n+1}{3} \rfloor$; PBFT-based algorithms have an advantage of reduced resource consumption but suffer from large message complexity (order $O(n^2)$). Hence, these algorithms are preferred for restricted or small blockchain systems, in order to have less message overhead.

The algorithm starts with a client sending a *request* to the primary node. The primary node verifies the request, assigns a sequence number, and sends a *pre-prepare* message to all the replicas. Each *pre-prepare* message also contains the client’s *request* and the current *view* number. When a replica receives a *pre-prepare* message, it first verifies the request and then transmits a *prepare* message to all the nodes. The *prepare* message contains the digest of client *request*, sequence number, and *view* number. Once a node receives $2f$ *prepare* messages, matching to the *pre-prepare* message, it multicasts a *commit* message. The *commit* message also contains the digest of client *request*, sequence number, and *view* number. Each replica waits for receiving identical $2f+1$ *commit* messages, before executing the request and then transmits the solution to the client.

The client needs $f+1$ matching responses from different replicas, to mark the request as complete. If the client *timeouts* while waiting for $f+1$ responses, then it multicasts the request to all the replicas. Each replica on receiving a request from the client, which it has not executed, starts a timer and queries the primary. If the primary does not reply until *timeout*, then the replicas proceed to change the primary. Note: PBFT heavily relies on strong cryptographic guarantees, and each message is digitally signed by the sender using his private key.

Zyzzyva (Kotla et al. 2007) is another interesting byzantine fault tolerance protocol aimed at reducing the costs associated with BFT-style protocols. Zyzzyva (Kotla et al. 2007) allows the replicas to reach *early* consensus by permitting speculative execution. In Zyzzyva, the replicas are relieved from the task of ensuring a global order for the client requests. This task is delegated to the client, which in turn informs the replicas if they are not in sync.

The protocol initiates with a client sending a request to the primary, which in turn sends a *pre-prepare* message to all the replicas. The structure of the *prepare* message is similar to its PBFT counterpart except that it includes a digest summarizing the *history*. Each replica, on receiving a request from the primary, executes the request and transmits the response to the client. Each response, in addition to the fields in the *pre-prepare* message, contains the digest of history and a signed copy of message from the primary. If the client receives $3f+1$ matching responses, then it assumes the response is stable and considers the request completed. In case the number of matching responses received by the client are in the range $2f+1$ and $3f$, then it creates a *commit certificate* and transmits this certificate to all the nodes. The *commit certificate* includes the history which proves to a receiving replica that it can safely commit the history and start processing the next request. All the replicas acknowledge the client for the *commit certificate*. If the client receives less than $2f+1$ responses, then it starts a timer and informs all the replicas again about the impending request. Now, either the request would be safely completed, or a new primary would be elected.

Although Zyzzyva achieves higher throughput than PBFT, its *client-based* speculative execution is far from ideal. Zyzzyva places a lot of faith on the correctness of the client, which is quite discomforting, as a malicious client can prevent the replicas from maintaining linearizability. **Aardvark** (Clement et al. 2009) studies the ill-effects of various fast, byzantine fault-tolerant protocols and presents the design of a *robust* BFT protocol. In Aadvark, the messages are signed by the client using both digital signatures and message authentication codes (Katz and Lindell 2007). This prevents malicious clients from performing a *denial-of-service* attack, as it is costly for the client to sign each message two times. Aadvark uses *point-to-point* communication, instead of multicast communication. The key intuition behind such a choice is to disallow a faulty client of replica from blocking the complete network. Aadvark also periodically changes the primary replica. Each replica tracks the throughput of the current primary and suggests replacing the primary, when there is a decrease in its throughput. The authors use a simple timer to track the rate of primary response.

RBFT (Aublin et al. 2013) is a simple extension to Aardvark. RBFT aims at detecting *smartly* malicious primary replica, which could avoid being detected malicious by other replicas. The key intuition behind RBFT is to prevent the primary from insinuating delays that are within the stated threshold. Hence, the primary could reduce the throughput without being ever replaced. To tackle this problem, RBFT insists running $f + 1$ independent instances of the Aardvark protocol on each node. One of the instances is designated as the "master instance," and it executes the requests. The rest of the instances are labeled as the "backup instances," and they order the requests and monitor the performance of the master instance. If any backup instance observes a degradation of the performance of the master instance, then it transmits a message to elect a new primary. Note: RBFT does not allow more than one primary on any node. Hence, each node can have at most one primary instance. RBFT protocol has an additional step in comparison to the three phases in PBFT and Aardvark. The client node initiates the protocol by transmitting a request to all the nodes. Next, each node propagates this request to all other nodes, and then the three-phase protocol begins. This extra round of redundancy ensures that the client request reaches all the instances.

Blockchain Systems

Bitcoin (Satoshi 2008) is regarded as the first blockchain application. It is a cryptographically secure digital currency with the aim of disrupting the traditional, institutionalized monetary exchange. Bitcoin acts as the token of transfer between two parties undergoing a monetary transaction. The underlying blockchain system is a network of nodes (also known as *miners*) that take a set of client transactions and validate the same by demonstrating a *Proof-of-Work*, that is, generating a block. The process of generating the next block is nontrivial and requires large computational resources. Hence, the miners are given incentives (such as Bitcoins) for dedicating their resources and generating the block. Each miner maintains locally an updated copy of the complete blockchain and the associated ledgers for every Bitcoin user.

To ensure Bitcoin system remains fair toward all the machines, the difficulty of *Proof-of-Work* challenge is periodically increased. We also know that Bitcoin is vulnerable to 51% attack, which can lead to *double spending* (Rosenfeld 2014). The intensity of such attacks increases when multiple forks of the longest chain are created. To avoid these attacks, Bitcoin developers suggest the clients to wait for their block to be confirmed before they mark the Bitcoins as transferred. This wait ensures that the specific block is a little deep (nearly six blocks) in the longest chain (Rosenfeld 2014). Bitcoin critics also argue that its *Proof-of-Work* consumes huge energy (As per some claims, one Bitcoin transaction consumes power equivalent to that required by 1.5 American homes per day.) and may not be a viable solution for the future.

Bitcoin-NG (Eyal et al. 2016) is a scalable variant to Bitcoin's protocol. Our preceding discussion highlights that Bitcoin suffers from throughput degradation. This can be reduced by either increasing the block size or decreasing the block interval. However, increasing the former increases the propagation delays, and the latter can lead to incorrect consensus. Bitcoin-NG solves this problem by dividing the time into a set of intervals and selecting one leader for each time interval. The selected leader autonomously orders the transactions. Bitcoin-NG also uses two different block structures: *key blocks* for facilitating leader election and *microblocks* for maintaining the ledger.

Ethereum (Wood 2015) is a blockchain framework that permits users to create their own applications on top of the Ethereum Virtual Machine (EVM). Ethereum utilizes the notion of smart contracts to facilitate development of new operations. It also supports a digital cryptocurrency, *ether*, which is used to incentivize the developers to create correct applications. One of the key advantages of Ethereum is that it supports a Turing complete language to generate new applications on top of EVM. Initially, Ethereum's consensus algorithm used the key elements of both PoW and PoC algorithms. Ethereum made nodes solve challenges that were not only compute intensive but also memory intensive. This design prevented existence of miners who utilized specially designed hardware for compute-intensive applications.

Recently, Ethereum modified its consensus protocol to include some notions of PoS algorithm. The modified protocol is referred as *Casper* (Buterin and Griffith 2017). Casper introduces the notion of *finality*, that is, it ensures that one chain becomes permanent in time. It also introduces the notion of *accountability*, that is, to penalize a validator, who performs the "nothing-at-stake" attack in PoS-based systems. The penalty leveraged on such a validator is equivalent to negating all his stakes.

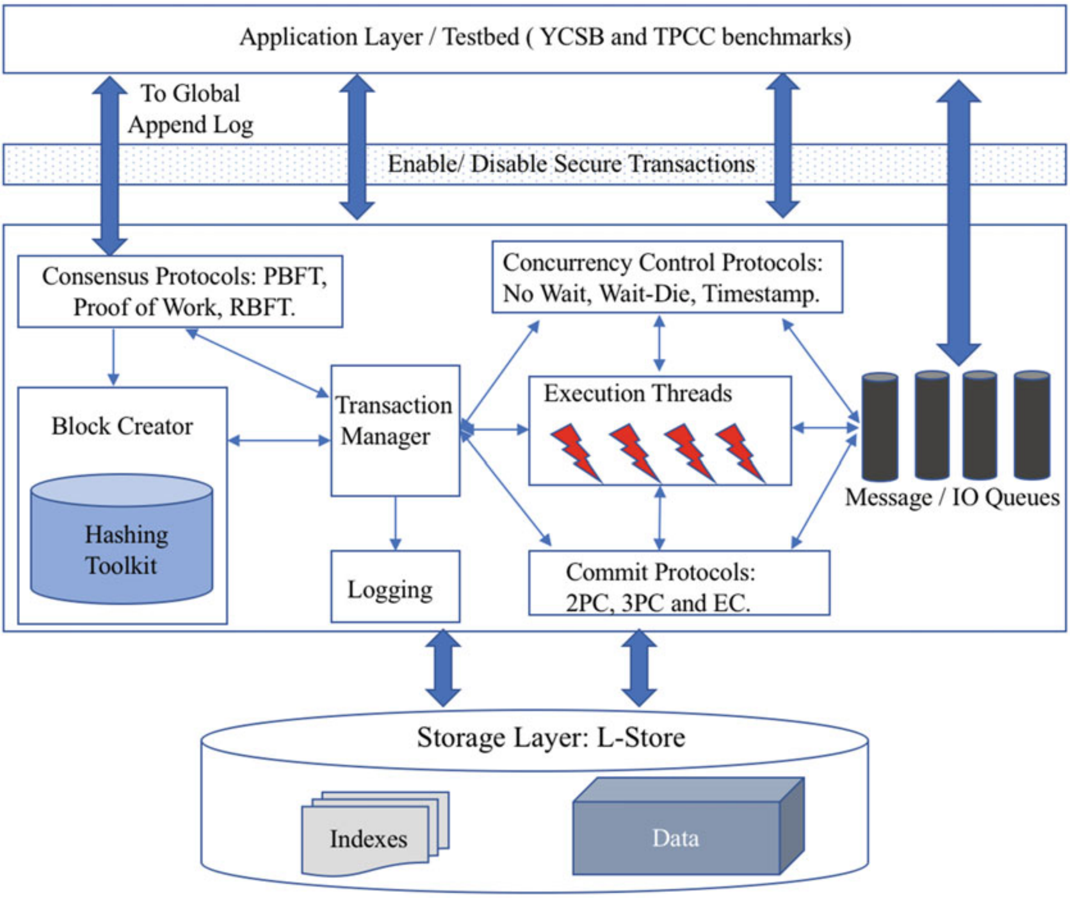
Parity (Parity Technologies 2018) is an application designed on top of Ethereum. It provides an interface for its users to interact with the Ethereum blockchain. Parity can be regarded as an interesting application for the blockchain community, as it provides support for both *Proof-of-Work* and *Proof-of-Authority* consensus algorithms. Hence, they allow mechanism for users of their application to set up "authority" nodes and resort to non-compute-intensive, POA algorithm.

Ripple (Schwartz et al. 2014) is considered as the third largest cryptocurrency after Bitcoin and Ethereum, in terms of market cap. It uses a consensus algorithm which is a simple variant of BFT algorithms. Ripple requires a number of failures f to be bounded as follows: $\leq (n-1)/5+1$, where n represents the total number of nodes. Ripple's consensus algorithm introduces the notion of a *Unified Node List* (UNL), which is a subset of the network. Each server communicates with the nodes in its UNL for reaching a consensus. The servers exchange the set of transactions they received from the clients and propose those transactions to their respective UNL for vote. If a transaction receives 80% of the votes, it is marked permanent. It is important to understand that if the generated UNL groups are a clique, then forks of the longest chain could coexist. Hence, UNLs are created in a manner that they share some set of nodes. Another noteworthy observation about Ripple protocol is that each client needs to select a set of validators or unique nodes that they trust. These validators next utilize Ripple consensus algorithm to verify the transactions.

Hyperledger (Cachin 2016) is a suite of resources aimed at modeling industry standard blockchain applications. It provides a series of application programming interfaces (APIs) for developers to create their own nonpublic blockchain applications. Hyperledger provides implementations of blockchain systems that uses RBFT and other variants of the PBFT consensus algorithm. It also facilitates the use and development of smart contracts. It is important to understand that the design philosophy of Hyperledger leans toward blockchain applications that require existence of nonpublic networks, and so, they do not need a compute-intensive consensus.

ExpoDB (Sadoghi 2017; Gupta and Sadoghi 2018) is an experimental research platform that facilitates design and testing of emerging database technologies. ExpoDB consists of a set of five layers providing distinct functionalities (refer to Fig. 4). *Application layer* acts as the test bed for evaluating the underlying database protocols (Gupta and Sadoghi 2018). It allows the system to access OLTP benchmarks such as YCSB Cooper et al. (2010), TPC-C TPP Council (2010), and PPS Harding et al. (2017). Application layer acts as an interface for the client-server interaction. *Transport layer* allows communication of messages between the client and server nodes.

Execution layer facilitates seamless execution of a transaction with the help of a set of threads. These threads lie at the core of the execution layer as they run the transaction, abide by the rules of stated concurrency protocol, and achieve agreement among different transactional


Application Layer / Testbed (YCSB and TPCC benchmarks)
To Global Append Log
Enable/ Disable Secure Transactions
Consensus Protocols: PBFT, Proof of Work, RBFT.
Concurrency Control Protocols: No Wait, Wait-Die, Timestamp.
Block Creator
Hashing Toolkit
Transaction Manager
Execution Threads
Message / IO Queues
Logging
Commit Protocols: 2PC, 3PC and EC.
Storage Layer: L-Store
Indexes
Data

Blockchain Transaction Processing, Fig. 4 Architecture of ExpoDB

partitions. ExpoDB provides implementation for *eight* concurrency controls and *three* commit protocols. ExpoDB also characterizes a *storage layer* (Sadoghi et al. 2018) for storing the transactional data, messages, and relevant metadata.

ExpoDB extends blockchain functionality to the traditional distributed systems through a *secure layer*. To facilitate secure transactions, ExpoDB provides a cryptographic variant to YCSB – *Secure YCSB* benchmark. ExpoDB also contains implementations for a variety of consensus protocols such as PoW, PBFT, RBFT, and Bitcoin-NG.

Future Directions for Research

Although blockchain technology is just a decade old, it gained majority of its momentum in the last 5 years. This allows us to render different elements of the blockchain systems and achieve higher performance and throughput. Some of the plausible directions to develop efficient blockchain systems are as follows: (i) reducing the communication messages, (ii) defining efficient block structure, (iii) improving the consensus algorithm, and (iv) designing secure lightweight cryptographic functions

Statistical and machine learning approaches have presented interesting solutions to automate key processes such as face recognition (Zhao et al. 2003), image classification (Krizhevsky et al. 2012), speech recognition (Graves et al. 2013), and so on. The tools can be leveraged to facilitate easy and efficient consensus. The intuition behind this approach is to allow learning algorithms to select nodes, which are fit to act as a block creator and prune the rest from the list of possible creators. The key observation behind such a design is that the nodes selected by the algorithm are predicted to be non-malicious. Machine learning techniques can play an important role in eliminating the human bias and inexperience. To learn which nodes can act as block creators, a feature set, representative of the nodes, needs to be defined. Some interesting features can be geographical distance, cost of communication, available computational resources, available memory storage, and so on. These features would help in generating the dataset that would help to train and test the underlying machine learning model. This model would be ran against new nodes that wish to join the associated blockchain application.

The programming languages and software engineering communities have developed several works that provide semantic guarantees to a language or an application (Wilcox et al. 2015; Leroy 2009; Kumar et al. 2014). These works have tried to formally verify (Keller 1976; Leroy 2009) the system using the principles of programming languages and techniques such as finite state automata, temporal logic, and model checking (Grumberg and Long 1994; Baier and Katoen 2008). We believe similar analysis can be performed in the context of blockchain applications. Theorem provers (such as Z3 De Moura and Bjørner 2008) and proof assistants (such as COQ Bertot 2006) could prove useful to define a certified blockchain application. A certified blockchain application can help in stating theoretical bounds on the resources required to generate a block. Similarly, some of the blockchain consensus has been shown to suffer from denial-of-service attacks (Bonneau et al. 2015), and a formally verified blockchain application can help realize such guarantees, if the underlying application provides such a claim.

References

Ateniese G, Bonacina I, Faonio A, Galesi N (2014) Proofs of space: when space is of the essence. In: Abdalla M, De Prisco R (eds) Security and cryptography for networks. Springer International Publishing, pp 538–557

Aublin PL, Mokhtar SB, Quéma V (2013) RBFT: redundant byzantine fault tolerance. In: Proceedings of the 2013 IEEE 33rd international conference on distributed computing systems, ICDCS '13. IEEE Computer Society, pp 297–306

Baier C, Katoen JP (2008) Principles of model checking (representation and mind series). The MIT Press, Cambridge/London

Bertot Y (2006) Coq in a hurry. CoRR abs/cs/0603118, http://arxiv.org/abs/cs/0603118, cs/0603118

Bonneau J, Miller A, Clark J, Narayanan A, Kroll JA, Felten EW (2015) SoK: research perspectives and challenges for bitcoin and cryptocurrencies. In: Proceedings of the 2015 IEEE symposium on security and privacy, SP '15. IEEE Computer Society, Washington, DC, pp 104–121

Buterin V, Griffith V (2017) Casper the friendly finality gadget. CoRR abs/1710.09437, http://arxiv.org/abs/1710.09437, 1710.09437

Cachin C (2016) Architecture of the hyperledger blockchain fabric. In: Workshop on distributed cryptocurrencies and consensus ledgers, DCCL 2016

Cachin C, Vukolic M (2017) Blockchain consensus protocols in the wild. CoRR abs/1707.01873

Castro M, Liskov B (1999) Practical byzantine fault tolerance. In: Proceedings of the third symposium on operating systems design and implementation, OSDI '99. USENIX Association, Berkeley, pp 173–186

Clement A, Wong E, Alvisi L, Dahlin M, Marchetti M (2009) Making byzantine fault tolerant systems tolerate byzantine faults. In: Proceedings of the 6th USENIX symposium on networked systems design and implementation, NSDI'09. USENIX Association, Berkeley, pp 153–168

Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on cloud computing. ACM, pp 143–154

Decker C, Wattenhofer R (2013) Information propagation in the bitcoin network. In: 13th IEEE international conference on peer-to-peer computing (P2P), Trento

De Moura L, Bjørner N (2008) Z3: an efficient SMT solver. Springer, Berlin/Heidelberg/Berlin, pp 337–340

Dziembowski S, Faust S, Kolmogorov V, Pietrzak K (2015) Proofs of space. In: Gennaro R, Robshaw M (eds) Advances in cryptology – CRYPTO 2015. Springer, Berlin/Heidelberg, pp 585–605

Eyal I, Gencer AE, Sirer EG, Van Renesse R (2016) Bitcoin-NG: a scalable blockchain protocol. In: Proceedings of the 13th USENIX conference on networked systems design and implementation, NSDI'16. USENIX Association, Berkeley, pp 45–59

Garay J, Kiayias A, Leonardos N (2015) The bitcoin backbone protocol: analysis and applications. Springer, Berlin/Heidelberg/Berlin, pp 281–310

Graves A, Mohamed A, Hinton GE (2013) Speech recognition with deep recurrent neural networks. CoRR abs/1303.5778, http://arxiv.org/abs/1303.5778, 1303.5778

Gray J, Lamport L (2006) Consensus on transaction commit. ACM TODS 31(1):133–160

Grumberg O, Long DE (1994) Model checking and modular verification. ACM Trans Program Lang Syst 16(3):843–871

Gupta S, Sadoghi M (2018) EasyCommit: a non-blocking two-phase commit protocol. In: Proceedings of the 21st international conference on extending database technology, Open Proceedings, EDBT

Harding R, Van Aken D, Pavlo A, Stonebraker M (2017) An evaluation of distributed concurrency control. Proc VLDB Endow 10(5):553–564

Jakobsson M, Juels A (1999) Proofs of work and bread pudding protocols. In: Proceedings of the IFIP TC6/TC11 joint working conference on secure information networks: communications and multimedia security, CMS '99. Kluwer, B.V., pp 258–272

Katz J, Lindell Y (2007) Introduction to modern cryptography. Chapman & Hall/CRC, Boca Raton

Keller RM (1976) Formal verification of parallel programs. Commun ACM 19(7):371–384

King S, Nadal S (2012) PPCoin: peer-to-peer crypto currency with proof-of-stake. peercoin.net

Kotla R, Alvisi L, Dahlin M, Clement A, Wong E (2007) Zyzzyva: speculative byzantine fault tolerance. In: Proceedings of twenty-first ACM SIGOPS symposium on operating systems principles, SOSP '07. ACM, New York, pp 45–58. https://doi.org/10.1145/1294261.1294267

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th international conference on neural information processing systems, NIPS'12, vol 1. Curran Associates Inc., pp 1097–1105

Kumar R, Myreen MO, Norrish M, Owens S (2014) CakeML: a verified implementation of ML. In: Proceedings of the 41st ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL '14. ACM, New York, pp 179–191

Lamport L (1998) The part-time parliament. ACM Trans Comput Syst 16(2):133–169

Leroy X (2009) A formally verified compiler back-end. J Autom Reason 43(4):363–446

Nawab F (2018) Geo-scale transaction processing. Springer International Publishing, pp 1–7. https://doi.org/10.1007/978-3-319-63962-8_180-1

Oki BM, Liskov BH (1988) Viewstamped replication: a new primary copy method to support highly-available distributed systems. In: Proceedings of the seventh annual ACM symposium on principles of distributed computing, PODC '88. ACM, New York, pp 8–17

Parity Technologies (2018) Parity ethereum blockchain. https://www.parity.io/

Park S, Kwon A, Fuchsbauer G, Gaži P, Alwen J, Pietrzak K (2015) SpaceMint: a cryptocurrency based on proofs of space. https://eprint.iacr.org/2015/528

Pilkington M (2015) Blockchain technology: principles and applications. In: Research handbook on digital transformations. SSRN

Rosenfeld M (2014) Analysis of hashrate-based double spending. CoRR abs/1402.2009, http://arxiv.org/abs/1402.2009, 1402.2009

Sadoghi M (2017) Expodb: an exploratory data science platform. In: Proceedings of the eighth biennial conference on innovative data systems research, CIDR

Sadoghi M, Bhattacherjee S, Bhattacharjee B, Canim M (2018) L-store: a real-time OLTP and OLAP system. OpenProceeding.org, EDBT

Satoshi N (2008) Bitcoin: a peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf

Schwartz D, Youngs N, Britto A (2014) The ripple protocol consensus algorithm. https://www.ripple.com/

Steen Mv, Tanenbaum AS (2017) Distributed systems, 3rd edn, version 3.01. distributed-systems.net

TPP Council (2010) TPC benchmark C (Revision 5.11)

Wilcox JR, Woos D, Panchekha P, Tatlock Z, Wang X, Ernst MD, Anderson T (2015) Verdi: a framework for implementing and formally verifying distributed systems. In: Proceedings of the 36th ACM SIGPLAN conference on programming language design and implementation, PLDI '15. ACM, New York, pp 357–368

Wood G (2015) Ethereum: a secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf

Zhao W, Chellappa R, Phillips PJ, Rosenfeld A (2003) Face recognition: a literature survey. ACM Comput Surv 35(4):399–458. https://doi.org/10.1145/954339.954342