A Scalable Cryptocurrency Based on A Byzantine-Safe Key-Value Store

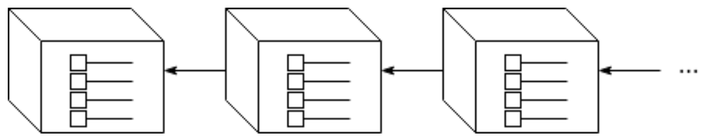
Amy Yang

DUKE COMPUTER SCIENCE

Research

Blockchain

- Blockchain is widely used as a storage system for cryptocurrency, smart contract, and asset management applications.
- Blockchain systems maintain a public, tamper-evident, totally ordered log in a "permissionless" network with no trust among participating principals.
- Full replication of a totally ordered log makes blockchain a platform for implementing replicated state machine applications.
- State machine replication approach has scalability limits as every server executes every operation in the same order.

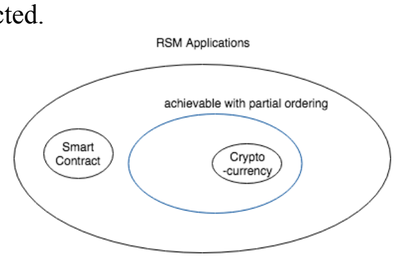


The Case Against Blockchain

- **Low throughput**. Current throughput for Bitcoin is 7 transactions/ sec, with a 10 minutes block interval.
- **Probabilistic commitment**. Blocks included in the public ledger still have nonzero probabilities of being reverted. Longest chain wins protocol makes it possible for any block to be superseded by a deep fork.
- **Long confirmation latency**. Inter-block interval must be sufficiently high to prevent frequent forks. Each block has to wait for more blocks to be appended to ensure its position in the blockchain.
- **Redundant execution**. Due to full replication of a totally ordered log, every node executes every operation in the blockchain, making it difficult to scale throughput by adding more servers.

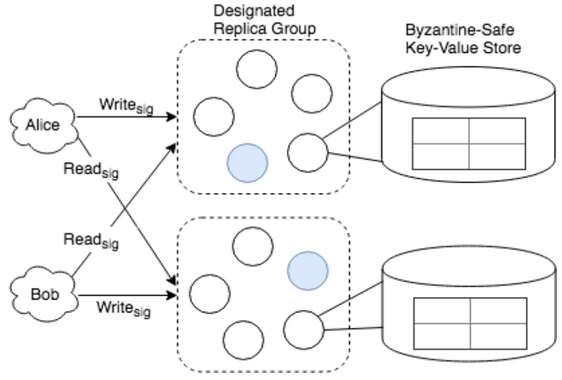
Our Approach / Results

- Totally ordered log is not necessary for asset management applications. We show that cryptocurrency, the canonical example of asset management applications, can be implemented with a partially ordered key-value store.
- Transactions can be executed in parallel as long as input and output dependencies are respected.
- Enforce deterministic transaction confirmation and mutually exclusive ownership with atomic writes guaranteed by the Byzantine-safe key-value store.



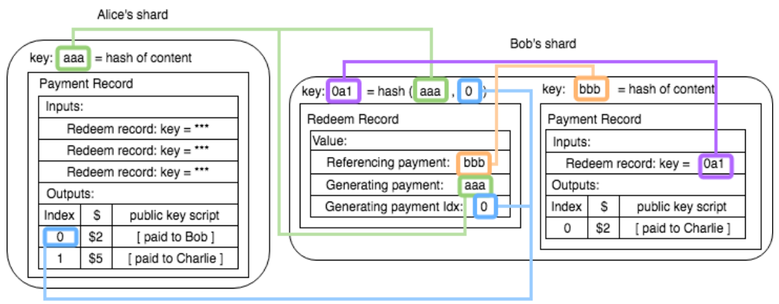
Architecture

- Principals are represented by public key pairs. Messages are signed under public keys such that they cannot be forged.
- Replicas are permissioned to join by a set of security policies
- Each principal has a designated replica group chosen randomly from the permissioned replicas.



Transaction Records

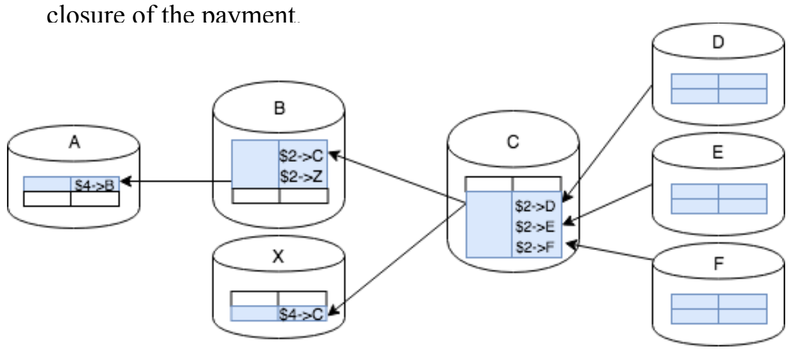
- Each transaction is decomposed into a set of redeem records and a payment record.



- **Payment records** reference a list of redeem records as input, and generate a list of outputs, each paid to a public key. In the above example, Bob pays Charlie $2 with a payment record written to Bob's shard. The payment record references the key of the redeem record **0a1**, and specifies that $2 goes to Charlie by providing a public key script that only Charlie's key pair satisfies. The payment record is written to the key which is the hash over its content.
- **Redeem records** reference transaction outputs. In the above example, Alice pays Bob $2 with a payment record written to Alice's shard. Bob is named as the receiver of one of the outputs for Alice's payment. To spend the output from Alice's payment, Bob writes a redeem record for the output he receives from Alice. The redeem record is written to the key which is the hash over the key of Alice's payment record, and the index of Bob's output in the output list of Alice's payment.
- Transaction records are **immutable**: once a record is committed by the shard, the record is available to all future readers and cannot be modified. Immutability is guaranteed if at most 1/3 shard replicas fail.

Generalized Transaction

- Transactions may have multiple inputs and outputs. In particular, inputs may reference outputs of a payment record stored in a different shard.
- Transaction records are linked into DAGs (with implicit back pointers) by key references. Readers validate a payment by verifying all records are duly constructed in the DAG that is the closure of the payment.



Security Properties

- **Immutable Key**. If a write *w* is committed by a quorum for an immutable key *k*, then no other write may commit on a quorum for *k*, and the value committed by *w* cannot be changed.
- *Proof*. If two values are returned for the same key, then the two quorums must overlap in at least f+1 nodes. Given that at most f replicas may fail for each shard, there is at least one replica in the intersection of any two quorums. The honest replica cannot agree with both values.
- **Liveness**. If a write *w* committed on an immutable key *k*, then *w* is always returned for future reads on *k*.
- *Proof*. Among the replies for a read Q-RPC on *k*, there is at least one honest nodes who participated in the quorum for *w*. Its signatures will be accepted by a quorum of replicas, thus *w* will be returned as the value for the read.

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[2] Dahlia Malkhi and Michael K Reiter. Secure and scalable replication in phalanx. In Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on , pages 51-58. IEEE, 1998.

[3] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. Distributed Computing , 11(4):203-213, 1998.

[4] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S Wallach. Secure routing for structured peer-to peer overlay networks. ACM SIGOPS Operating Systems Review , 36(SI):299-314, 2002.