Bridging Paxos and Blockchain Consensus

Aleksey Charapko
University at Buffalo, SUNY
acharapk@buffalo.edu

Ailidani Ailijiang
University at Buffalo, SUNY
ailidani@buffalo.edu

Murat Demirbas
University at Buffalo, SUNY
demirbas@buffalo.edu

Abstract—**The distributed consensus problem has been extensively studied in the last four decades as an important problem in distributed systems. Recent advances in decentralized consensus and blockchain technology, however, arose from a disparate model and gave rise to disjoint knowledge-base and techniques than those in the classical consensus research. In this paper we make a case for bridging these two seemingly disparate approaches in order to help transfer the lessons learned from the classical distributed consensus world to the blockchain world and vice versa. To this end, we draw parallels between blockchain consensus and a classical consensus protocol, Paxos. We also survey prominent approaches to improving the throughput and providing instant irreversibility to blockchain consensus and show analogies to the techniques from classical consensus protocols. Finally, inspired by the central role formal methods played in the success of classical consensus research, we suggest more extensive use of formal methods in modeling the blockchains and smartcontracts.**

I. Introduction

The consensus problem has been studied for more than forty years in distributed systems. Distributed consensus is a fundamental problem arising in the context of database transactions, state machine replication, group membership, and leader election. Consensus means processes reach agreement on a single value: The processes provide their candidate values, communicate with one another, and agree/commit on a single consensus value regardless of some faulty processes.

The research on consensus in distributed systems literature has established a rigorous perimeter around the formal safety and liveness/progress guarantees that can be provided. In particular, Paxos [1] protocol for consensus compartmentalizes the safety and progress properties nicely. Even under a fully asynchronous model (where all timing assumptions/expectations break), Paxos preserves safety thanks to its balloting and anchoring system. Paxos also provides progress when the system is out of the realm of the fully asynchronous model (where FLP result [2] holds) and in to the partially synchronous model where weak-complete and eventually-weak-accurate failure detectors are implementable [3]. Formal methods and formally specified assumptions and guarantees played a key role in the success of the classical distributed consensus work. Today Paxos variants have been deployed ubiquitously in many cloud computing and web applications to provide distributed coordination [4].

A shortcoming of distributed consensus work is that it failed to provide a solution that scales to the large number of participants desired in public, permissionless, open participation settings. Paxos deployments are typically limited to 5 nodes. Byzantine tolerant consensus solutions, such as PBFT [5], are limited to less than 10, and the latency of PBFT has been shown to grow quadratically as the number of nodes in the cluster increases [6].

Recently, on the other side of the canyon, blockchain consensus arose as a radical solution to distributed consensus, ignoring both the contributions and baggage of classical distributed consensus work. The blockchain consensus model brings new constraints and requirements to the consensus problem. In this model, participation is open and permissionless. In an open model, it is not sufficient anymore to use 3-5 participants—which was enough for tolerating crashes and ensuring persistency of the data in a datacenter setting. For reasons of attestability and tolerating colluding groups of Byzantine participants, it is preferred to keep the participants at 1000s or higher.

To deal with the challenges of Byzantine participants in open/permissionless settings, many blockchain consensus protocols adopt proof-of-work (PoW) [7], [8] as an identity assignment tool. PoW ensures that the number of identities assigned to the honest and adversarial parties can be made proportional to their aggregate computational power. By employing PoW for blockchain consensus, Sybil attacks [9] that employ impersonation is rendered useless since each sock puppet need to expend significant amount of computing resources. PoW is expensive in terms of time wasted and electricity bills: only a miner which has successfully solved a computationally hard puzzle (finding the right nonce for the block header) can append to the blockchain.

Blockchain solutions are also not immune from shortcomings. Many solutions exhibited low throughput and high latency problems. The approximate probabilistic nature of some blockchain solutions also created issues. Finally, some problems arose from unclear/vague assumptions and guarantees in blockchain protocols.

Contributions. In this paper we make a case for bridging these two seemingly disparate approaches in order to help transfer the lessons learned from the classical distributed consensus world to the blockchain world and in reciprocal. Our paper has the following major contributions.

1) We show parallels between the two approaches by showing how blockchain consensus fits in and compares with the Paxos consensus protocol.
2) We survey prominent blockchain consensus work that achieved improved throughput and instant irreversibility,

and show analogies to techniques from the classic consensus protocols.
3) We suggest the use of formal methods, which has proven its benefits in the classic consensus work, for blockchain consensus for better specifying and model checking the blockchain protocols and smartcontracts.
4) We show analytical evaluation of how blockchain and Paxos compare at the protocol level.

To help address concurrent execution and distributed systems challenges, in our future work, we propose to use TLA+/PlusCal [10] to build a formal framework that facilitates modeling and proving properties for blockchain consensus protocols and smartcontracts. The framework will support ways to test the blockchain and smartcontract protocols under a variety of challenging environments such as strategic partitioning, unfair execution scheduling, overstretched timelines (due to bad/malicious clock synchronization), and inopportune Byzantine and crash failures.

II. Paxos

The original Paxos protocol [1] achieves the fault tolerant consensus among a set of nodes. The classical consensus problem is described by two safety and one liveness properties.

Agreement: No two correct nodes can decide on different values.

Validity: If all initial values are same, nodes must decide that value.

Termination: Correct nodes decide eventually.

Distributed consensus is not an easy problem to solve in the presence of faults, and in some cases a solution that satisfies all three properties above becomes impossible. The "Coordinated Attack" result [11] states that there is no deterministic algorithm for reaching consensus in a model where an arbitrary number of messages can be lost undetectably. The result applies to both asynchronous and synchronous models. Even assuming no message loss, FLP impossibility result [2] states that there is no deterministic algorithm for reaching consensus under the fully asynchronous system model in the presence of just one crash failure. Unlike the coordinated attack result, FLP applies only for asynchronous systems, partially-synchronous, or synchronous systems are safe from this impossibility result. However, it is important to note that even for the most synchronous cluster of computers a heavy load of requests can break all the timeliness assumptions and turn the system into an asynchronous one in effect.

Paxos satisfies the safety properties of consensus even under asynchrony and arbitrary message loss. This does not conflict with the two impossibility results discussed above, because those state that it is impossible to satisfy the safety and liveness properties together, but do not state that the protocol needs to sacrifice safety under those conditions. Paxos preserves safety under any condition and achieves liveness when conditions improve outside the impossibility realm. Paxos comes with a simple formal proof for satisfying its properties.

Paxos runs in three distinct phases: propose (phase-1), accept (phase-2) and commit (phase-3), as shown in Figure 1. During the first phase, a node tries to become the leader by proposing a unique *ballot* number b to its followers with a phase-1a message. The followers acknowledge a leader with the highest ballot seen so far, or reject with a ballot greater than b. Receiving one rejection fails the candidate with this ballot number, because it indicates there is another leader candidate with a higher ballot number reaching to the participants. The node becomes leader and advances to phase-2 after receiving the majority quorum of acknowledgments. In this phase, leader tries to choose a suitable value v for this ballot. The value would be some uncommitted v associated with the highest ballot learned in previous phase, or any new value if none exists. The leader commands followers to accept value v and waits for accepted messages. Once the majority of followers acknowledge the acceptance of the value, it becomes anchored and cannot be revoked. A single rejection received in phase-2b, on the other hand, nullifies the leadership of the node, and sends it back to phase-1 to try with a higher ballot number. Finally, the leader sends a commit message in phase-3 that allows the followers to commit and apply the value to their respected state machines.

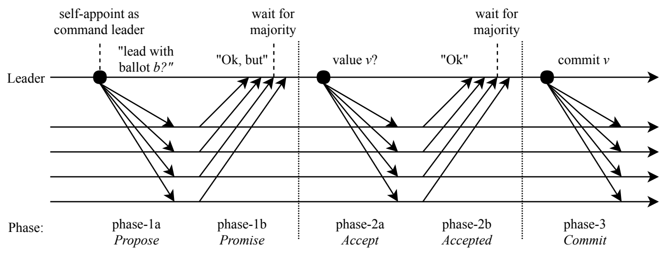

Fig. 1: Overview of Paxos algorithm.

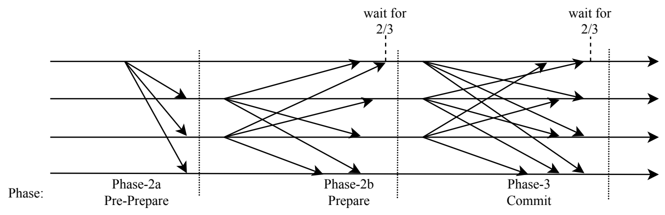

Fig. 2: Overview of PBFT algorithm.

It's important to see that after phase-2, an anchored value cannot be overridden later as it is guaranteed and proven by Paxos that any leader with higher ballot number will learn it before proposing new rounds. Such stability also makes classical consensus definite.

Paxos can only tolerate crash failures. But it is easy to extend phase-2b and phase-3 to tolerate Byzantine failure as shown in PBFT [5]. Figure 2 shows the operation of PBFT algorithm in normal case. When compared to Figure 1, it is easy to see the increase in communication complexity.

III. PoW Blockchain and Relation to Paxos

The proof-of-work (PoW) blockchain approach solves the consensus problem in an open, permissionless setup subject to crashes, byzantine failures, and Sybil attacks. PoW ensures that the number of identities assigned to the honest and

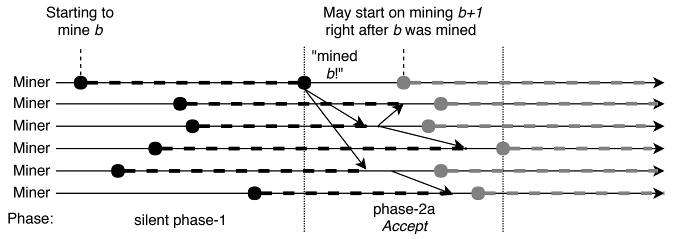

Fig. 3: Proof-of-Work blockchain protocol with silent leader election and one-way phase-2 communication.

adversarial parties is proportional to their aggregate computational power. This makes PoW expensive in terms of wasted computation time and electricity bills, as only a miner successfully solving a computationally hard puzzle appends to the blockchain. However, employing PoW for blockchain consensus renders impersonation or Sybil attacks [9] useless.

Blockchain protocol leverages on a distributed ledger stored at all nodes in the network. The ledger consists of a sequence of blocks, with multiple transactions included in each block. Some nodes periodically add uncommitted transactions to a new block and append it to the ledger. The procedure of appending the block to a ledger is called mining. It involves solving a computationally hard problem, such as finding a nonce that will make a hash of all transactions, previous block and the nonce satisfy some predetermined criteria.

Proof-of-work blockchains rely on Nakomoto [12] consensus, which has somewhat different, yet comparable properties than the classical consensus. The biggest difference between the two is the probabilistic nature of Nakomoto consensus, resulting in slight chance of reversing the transaction. The probability of reversal, however, diminishes quickly with the age of the block.

Probabilistic Agreement: For a blockchain of length n, any two nodes will return the same chain prefix of length $n - r$ with probability greater than p. The probability p is controlled by varying the size of r, with larger r increasing the p. The tailing blocks of length r still have larger chance of reversal.

Transaction Validity: Well-behaved nodes only accept blocks with valid and not spent transactions (no duplicates of same transaction in different blocks). Malicious nodes need to have majority of compute power to put invalid transaction on the chain and maintain it as longest chain in the network.

Probabilistic Termination: The consensus is probabilistic with chance of reversal becoming negligible after sufficient number of blocks r are added to the chain after. As such, any valid node will stabilize on some chain prefix of length $n-r$, with negligible probability of changes to the prefix.

Although blockchain is seemingly a very different solution than the Paxos protocol, next we show how to draw parallels between Paxos and blockchain and highlight the similarities and differences between the protocols.

A. Leader Election: Silent vs. Loud

Both solutions, blockchain and Paxos, require a leader to commit to a ledger or log. The mechanisms of picking a leader, however, are drastically different. The blockchain way of finding a leader to commit a block requires nodes to solve a hard computational puzzle. The first participant to do so appends the block to the ledger and becomes a silent leader for the round, as shown in Figure 3. A node never communicates to the rest of the cluster about the intent to become a leader, since being a miner de-facto represents such intent.

The Paxos way is different, with loud leader election process done via Phase 1 as shown in Figure 1. A leader candidate sends a phase-1a message with its ballot number, or counter, and tries to get "OKs" from a majority number of participants in the quorum. Receiving one rejection message is a deal breaker, because it indicates there is another leader candidate with a higher ballot number reaching to the participants.

B. Multiple Leaders: to fork or not to fork

The loud election process makes it unlikely (but not impossible say due to bad failure detectors) for two nodes to think they should lead the protocol. However, the ballot and quorum system prevents the two leaders from making progress at the same time and ensures that only a single leader can append to the log. This is done via comparing ballots both in phase-1 and phase-2 of Paxos. If a leader enters a phase-2 with a ballot lesser than some other node, it will receive a rejection message causing it revert back to phase-1 and try to become a leader with an even higher ballot. Accept phase responsible for appending to the log successfully completes only when a majority quorum is reached with all "OK" messages.

Silent blockchain election may result in multiple nodes successfully solving the puzzle and appending their blocks to the ledger at roughly the same time, creating a divergence, or fork, in the blockchain. The chances of forking decrease when a puzzle is sufficiently difficult and the incentive mechanism resolves the fork within a few block additions. One forked chain is likely to grow faster, as participants prefer to mine on the longest chain for having a chance to receive the mining fee if they become the leader for that round.

C. Accept Phase Communication: 1-way vs. 2-way

The blockchain protocol employs communication only once and only in one way, from the leader to the participants. This corresponds to Paxos phase-2a: the accept message phase.

As discussed above, phase-2b allows Paxos to catch multiple leaders and guarantee progress only for one leader. Since the blockchain consensus protocol skips phase-2b and phase-3, it provides unavoidably a probabilistic consensus. Therefore, in order to commit, you need to wait to see more consecutive blocks and ascertain that the block sticks in the longest chain. At the point of commit, the work required to rewrite history by creating an earlier fork is so huge that you can conclude the block is finally committed. In other words, blockchain provides an eventually-consistent/stabilizing consensus protocol.

For phase-2 in Paxos, the leader has a two-way acknowledged communication. This works OK for the leader because the number of the participants is typically less than half a dozen. On the other hand, if thousands of nodes participated

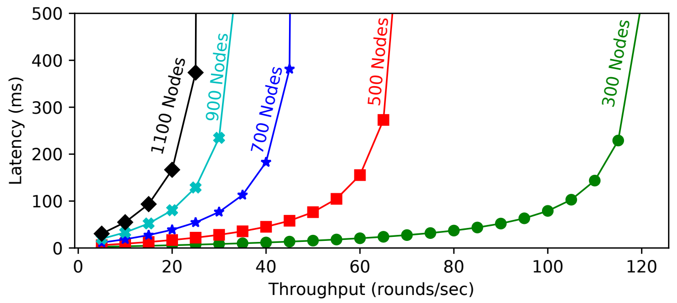

Fig. 4: Paxos throughput drastically reduces for larger cluster.

in a Paxos protocol, the leader would struggle to maintain any reasonable commit throughput while receiving that many ack messages in each round. To avoid the problem with thousands of participants, the blockchain has only one way broadcast communication, which is propagated from the leader to the participants via a gradual gossip protocol (Figure 3).

We illustrate the problem with Paxos leader struggling to sustain high throughput for large clusters in Figure 4, showing the results of performance model [13] with parameters to roughly match single region EC2 deployment on small nodes. In particular, we modeled a FIFO processing pipeline at the leader node with a G/G/1 queue. We approximated message processing and communication delays with real Paxos implementation over Amazon AWS EC2 t2-small instances. We assumed normally distributed network latency. Our final average latency formula is as follows: $L = 2m_o + Nm_i + l_m + w + l$, where m_o and m_i are processing latencies for outbound and inbound messages, l_m is the round-trip time for the message making a majority quorum obtained with Monte-Carlo method approximation of k-order statistics, w is the G/G/1 queue wait time [14] and l is network RTT between leader and a client issuing a request.

D. The Environment: Public vs. Private

The typical use case of Paxos is to replicate state to make it consistently and safely survive crash failures in a private environment. As a result, keeping the number of participants small, around 5 nodes, works well for this case.

The blockchain applications bring new constraints and requirements to the consensus problem. In blockchains, the participants can now be Byzantine, motivated by financial gains. So it is not sufficient to limit the consensus participants to be 3-5 nodes, because the rest of the network does not necessarily trust these nodes. In blockchains, for reasons of attestability and tolerating colluding groups of Byzantine participants, it is preferred to have many thousands of participants.

Comparing popular proof-of-work Nakamoto [12] consensus with Byzantine consensus solutions, such as byzantine Paxos [15] reveals major differences. Whereas Nakamoto consensus has probabilistic guarantees for termination, agreement, and validity, the classic Byzantine Consensus has deterministic guarantees for them. However, the performance of Nakamoto consensus is far less sensitive to the number of participants than that of byzantized Paxos, making it more suitable for public environments without further changes.

IV. Blockchain Extensions and Relation to Paxos

In this section we discuss some recent blockchain optimization and enhancements and show how they relate to classical consensus protocols, such as Paxos and practical byzantine consensus protocol (PBFT).

A. Leader Election: Stable Leader

Paxos deployments typically continue with the same leader for many rounds. So to avoid paying the cost of phase-1 repeatedly, MultiPaxos, or multi-decree Paxos, skips phase-1 and continues with iterating over phase-2 messages for the consecutive rounds. If an incumbent leader arises, the phase-1 leader election may need to happen again. An orthogonal mechanism, such as a failure detection service, is used to ensure that there are not many candidates running to become a leader at a given time, as that may hinder the progress of the protocol. That situation is called the “dueling leaders” situation. As we explained earlier, even when there are multiple leader candidates, Paxos is safe thanks to the ballot and majority mechanisms employed in phase-1 and phase-2.

Bitcoin-NG [16] introduces a similar optimization to the blockchain, where a single leader takes control over the ledger for a duration of several *microblocks*. In the blockchain setting, however, we do not want to have one stable leader for too long, as to incentivize other miners and make sure no single miner can control the network for long enough to gain any advantage. To that case, Bitcoin-NG uses *key blocks* to carry out the leader election. Similar to the original Bitcoin [12], miners try to find a nonce that will generate a certain predetermined hash value for the *key block*. Once the leader is chosen, it can append some limited number of *microblocks* to the ledger. The leader is prohibited from appending too many *microblocks* or from appending them too often to prevent malicious or greedy leader from swamping the system with transactions.

In Bitcoin-NG *microblocks* do not require any proof-of-work, as the leader is already chosen in the *key block*. This allows the leader to process transactions and add them to the ledger more efficiently, given that is stays within the rate limits on issuing the *microblocks*.

B. Irreversibility of Blockchain

Byzcoin [17] is a recent blockchain variant that uses practical byzantine consensus protocol (PBFT) to make the blockchain consensus instantly irreversible when a block is added to the ledger, rather than probabilistically irreversible after adding some number of consecutive blocks to the ledger.

PBFT cannot reasonably scale over to thousands of nodes needed for public blockchain networks, therefore Byzcoin forms a small subset or council of nodes to run PBFT. Unlike regular blockchain with a single miner leading a block, in Byzcoin the council is responsible on agreeing on what the next block should be. Once such agreement has been reached, and thee council collectively endorses the block with Schnorr signatures and the block becomes irreversible.

Byzcoin must choose the council with great care to prevent large colluding groups of nodes from entering the council. A PoW approach is used to elect the council members on a rolling basis: the miner or miners successfully solving the most recent block join the council, while some old members leave. The council consists of 144 members, roughly representing a day worth of miners in a chain producing a new block every 10 minutes. The PoW prevents any colluding party from overpowering the council, given the total number of Byzantine nodes in the network is less than one-third.

C. Federated Blockchains

Stellar Consensus Protocol (SCP) [18] gives a different take on using classical byzantine consensus, such as PBFT. Unlike Byzcoin that must carefully select the council to run PBFT, SCP breaks the network into federation or groups of nodes. Upon entering the network, a node specifies some *quorum slices*. Each *quorum slice* represents a consortium of other participants a node trusts. However, the node does not need to trust every node in the *quorum slice* individually. SCP uses *quorum slices* to construct quorums for different nodes in such a way that every node in the quorum must have at least one of its slices be entirely in the quorum as well.

The protocol runs in a two phases. It starts with a nomination phase, which if run long enough, eventually produces the same set of candidate values at every intact node. Upon predicted/approximated convergence of nomination phase, the nodes start the ballot phase to perform federated voting (PBFT) to commit and abort ballots associated with composite values. The ballot phase runs on some quorum in the system, and for safety any two quorums in the network need to intersect with at least one non-faulty, non-byzantine node. PBFT also makes the committed blocks instantly irreversible.

SCP does not rely on PoW for any of its operations. The quorum intersection requirement also means that quorums must get larger with more nodes participating, potentially reducing the performance of the system.

V. Formal Methods and Blockchain

Formal methods have been an important part of distributed systems research. Since concurrency introduces an explosion of possible executions, operational reasoning about distributed algorithms is inapplicable. Clearly/unambiguously specifying the protocol and environment behavior is of paramount importance for distributed systems. Any assumptions made while building a distributed system will bite back. For instance, it may not be safe to assume that one network hop is faster than two, or even that no hops is faster than one. Or what was assumed to be an atomic execution block may be violated by some rogue process executing concurrently and changing the system state in an unanticipated manner. When faults enter the picture, things get even more convoluted, because fault-actions will collude with program actions to complicate things and create many new corner cases.

A practical companion to formal methods is the model checking tools to support them. Model checking frameworks provide a language to specify/model a distributed algorithm, and also support testing this model exhaustively to detect violations of user provided safety and progress properties. A prominent example is the TLA+ [10] framework, by Leslie Lamport, who is also the inventor for Paxos. TLA+ provides a math/logic based language for specifying distributed algorithms/protocols and model checking them. Many major distributed consensus and coordination protocols, including Paxos and several of its variants [19], [20], [21], have been modeled and verified with TLA+.

Even on the industry side of distributed systems, formal methods have been adapted to deal with the challenges of distributed systems. AWS successfully used invariant-based reasoning and formal methods (in particular TLA+) for building robust distributed systems. AWS used TLA+ in many key projects: S3, DynamoDB, EBS, and a distributed lock manager. The paper "*Use of formal methods at Amazon Web Services, 2014*" [22] writes: "Before launching any complex service, we need to reach extremely high confidence that the core of the system is correct. We have found that the standard verification techniques in industry (deep design reviews, code reviews, static code analysis, stress testing, fault-injection testing, etc.) are necessary but not sufficient. Human fallibility means that some of the more subtle, dangerous bugs turn out to be errors in design; the code faithfully implements the intended design, but the design fails to correctly handle a particular 'rare' scenario. We have found that testing the code is inadequate as a method to find subtle errors in design."

Based on the significant role formal methods played in the success of the classical distributed consensus and coordination work, we suggest the use of formal methods for blockchain consensus for better specifying and model checking the blockchain protocols and smartcontracts.

A. Using formal methods for blockchain consensus

TLA+ can help precisely specify the assumptions made by blockchain protocols, such as timing and partition assumptions. Several protocols have been shown vulnerable to timing attacks because they depended on some reasonable time synchronization assumptions. While PoW consensus protocols make several reasonable timing assumptions, when an attacker manages to violate tem, the safety of the protocol may be violated as well. Using TLA+ it is possible to model these assumptions and identify vulnerabilities.

Another benefit of using formal methods and TLA+ for modeling blockchain protocols is in the design process. TLA+ specification language supports stepwise refinement, as it is easy to check if one model implements/refines the other: this is specified via an implication sign. This can be leveraged to identify new implementations and abstractions as well as designing and verifying new features to blockchain protocols, such as sharding.

Of course modeling blockchain protocols brings new challenges for TLA+ framework. Blockchain protocols have a

```
1  variable attack = 3,
2    bankBalance = BALANCE,
3    malloryBalance = 0;

5  define {
6    SafeWithdrawal ≜
7         ∨ bankBalance = BALANCE ∧ malloryBalance = 0
8         ∨ bankBalance = BALANCE ∧ malloryBalance = AMOUNT
9         ∨ bankBalance = BALANCE − AMOUNT ∧ malloryBalance = AMOUNT }

11 procedure BankWithdraw( amount ) {
12   CheckBalance:   check if Mallory has sufficient balance
13     if ( bankBalance < amount ) return;
14   DispenseAmount:   dispense Mallory the amount
15     call MallorySendMoney(amount);
16   FinishWithdraw:   update Mallory's bankBalance
17     bankBalance := bankBalance − amount;
18   return; }

20 procedure MallorySendMoney( amount ) {
21   Receive:
22     malloryBalance := malloryBalance + amount;
23     if ( attack > 0 ) {
24       attack := attack − 1;   avoid infinite stack; don't run out of gas
25       call BankWithdraw(amount); } ;   cheating! doublecalling withdraw
26 FC: return; }

28 fair process ( blockchain = "blockchain" ) {
29   Transact:   Mallory calls Bank to withdraw AMOUNT from her bankBalance
30     call BankWithdraw(AMOUNT); }
```

(a) Faulty model

```
1  variable attack = 3,
2    bankBalance = BALANCE,
3    malloryBalance = 0;

5  define {
6    SafeWithdrawal ≜
7         ∨ bankBalance = BALANCE ∧ malloryBalance = 0
8         ∨ bankBalance = BALANCE − AMOUNT ∧ malloryBalance = 0
9         ∨ bankBalance = BALANCE − AMOUNT ∧ malloryBalance = AMOUNT
10   Invariant ≜ bankBalance + malloryBalance ≤ BALANCE
11   EndState ≜ ◇(bankBalance + malloryBalance = BALANCE
12                  ∧ bankBalance ≤ BALANCE − AMOUNT) }

14 procedure BankWithdraw( amount ) {
15   CheckBalance:   check if Mallory has sufficient balance
16     if ( bankBalance < amount ) return;
17   UpdateBalance:   update Mallory's bankBalance
18     bankBalance := bankBalance − amount;
19   DispenseAmount:   dispense Mallory the amount
20     call MallorySendMoney(amount);
21   return; }

23 procedure MallorySendMoney( amount ) {
24   Receive:
25     malloryBalance := malloryBalance + amount;
26     if ( attack > 0 ) {
27       attack := attack − 1;   avoid infinite stack; don't run out of gas
28       call BankWithdraw(amount); } ;   cheating! doublecalling withdraw
29 FC: return; }

31 fair process ( blockchain = "blockchain" ) {
32   Transact:   Mallory calls Bank to withdraw AMOUNT from her bankBalance
33     call BankWithdraw(AMOUNT); }
```

(b) Fixed model

Fig. 5: TLA+ model for DAO attack

probabilistic component, and it is important to model it in TLA+. It is also important to develop libraries and high level primitives for specifying consensus algorithms and blockchain consensus protocols (with models for basic math primitives like hashes, public/private signatures).

B. Using formal methods for smartcontracts

Smartcontracts [23], [24] represent an important application of blockchain technology. Smartcontracts help users establish and enforce the common rules for all involved parties, avoiding the need in lawyers and courts as reward and/or failure clauses execute automatically in the contract. However, smart contracts are not devoid of challenges. Since a contract is non-mutable once in place, all parties must be sure of contract's correctness before joining. Formal methods and model checking allow to simplify the contract development, ensure its correctness under various conditions and provide an extra level of confidence to the parties joining the contract.

When money is involved, attackers get smart quickly: "Never underestimate the time and expense your opponent will take to break your code"[25]. It is easy to have vulnerabilities in concurrent code due to the many corner cases. Smartcontracts are especially at risk due to money on the line and the high speed of transfers. TLA+ can be used for identifying those cornercases and help design correct protocols.

Modeling the DAO hack.

Incorrect or buggy smart contract can have large repercussion on the parties involved. For instance, the DAO attack [23] resulted in unauthorized drainage of large amounts of money from the DAO, an investment fund managed entirely by smart contracts with no central authority or human control. The contract protocol allowed the participants to withdraw some of their money from the fund, however, the withdraw part of the contract was flawed. The bug was ultimately exploited, resulting in the fund's money stolen by a malicious participant.

To illustrate the usefulness of formal methods and how they can save contract developers from major problems, we have modeled the DAO withdrawal protocol with the bug ultimately exploited by the hackers as shown in Figure 5a. We then performed model checking to find the concurrency problem with the contract's protocol and corrected it in Figure 5b.

TLA+ facilitates invariant-based reasoning, so we first define an invariant for safe withdrawal: *SafeWithdrawal == (bankBalance=BALANCE ∧ malloryBalance=0) ∨ (bankBalance=BALANCE-AMOUNT ∧ malloryBalance=AMOUNT)*. This invariant checks that Mallory's withdrawal affects the bank balance only once. However, we can see the invariant is too tight for our protocol, since the bank balance updates before Mallory receives the money in non-atomic way, i.e. the money can be in-flight from bank account to Mallory. That is how the invariant-based thinking helps us immediately: we can see that the withdrawal is a non-atomic operation, and realize that we should be more careful with the updates. Nevertheless, we relax the invariant as shown in Figure 5a on lines 5-9.

Procedure *BankWithdraw* on lines 11-18 in Figure 5a models the bank withdrawal contract, while procedure *mallorySendMoney* (lines 20-26) governs how Mallory receives the money after the bank withdrawal. Note that in line 25 she can attempt to withdraw the money again. Also note that *BankWithdraw* line 15 gives money to Mallory before updating the bank balance on line 17. If Mallory is cheating, she may attempt to withdraw the money before the bank balance is updated from her previous withdrawal.

TLA+ model checking catches the malicious behavior easily, since some execution causes the violation of the invariant, resulting in a double spending. The model checker also produces the error trace, outlining all steps that must happen to cause the invariant violation. In this case, the error trace contains 8 steps: initially *BankWithdraw* is called, which then calls the *MallorySendMoney* to complete withdrawal. However, Mallory's *SendMoney* implementation includes another call to *BankWithdraw* and the balance check on line 13 passes because *bankBalance* is not decremented by amount (that comes in line 17). So the second *BankWithdraw* executes **concurrently** and Mallory manages to do double withdrawal.

In Figure 5b we illustrate the fixed version of the protocol. We move the updating bank balance before invoking *MallorySendMoney*, making sure the bank balance updates before Mallory receives the funds. Of course, for that we change *SafeWithdrawal* to accommodate the new way of updating *bankBalance*. But it turns out that the invariant is still too tight. For instance, if initially BALANCE=10 and we call *BankWihdraw(AMOUNT=4)*, it is still valid to have two withdrawals concurrently provided that in the final state no new money is produced: *Invariant* == *bankBalance+malloryBalance* $\leq$ *BALANCE*. We also model check for progress and write an EndState temporal formula for it on lines 11-12.

VI. Paxos and Blockchain Performance

Paxos and PoW blockchain protocols have very different performance characteristics, making it rather hard to perform a direct and fair comparison. Paxos is designed for small clusters, relatively quick operation latency and high throughput. Blockchain, on the other hand, runs on many thousands of participants and trades small operation latency for predictable throughput in large deployments.

Paxos-style protocols rely on inter-node communication, making the sheer number of message exchanges to be a bottleneck for a Paxos round leader. In typical small deployments, the message requirement for a round is manageable, however as cluster size increases, the same leader needs to handle more traffic, ultimately degrading the performance.

The maximum throughput of Paxos largely depends on the network and processing capacity of the leader server and the number of followers. If an incoming and outgoing message and outgoing broadcast take an average ofμ_i and μ_o and μ_b milliseconds to process respectively, we can estimate the maximum throughput of a Paxos system as $R_{max} = \frac{1}{N\mu_i + \mu_b + \mu_o}$, where N is the number of nodes in the cluster.

On the blockchain scale with thousands of nodes, network bandwidth becomes the limiting factor, since sending an accept message to all participants is problematic even with smallest possible blocks. For instance, in the absence of UDP multicast on the Internet scale, 1 Kb block requires about 0.82 seconds to be send over to all Paxos followers in a cluster of 10,000 nodes with 100 mbps connection at the leader. This drives the broadcast cost μ_b very high for large clusters. Orthogonal solutions are then required to make Paxos overcome the network bottlenecks, such as using peer-to-peer networks or pub/sub systems, such as Apache Kafka or BookKeeper [26], [27], to deliver signed messages from the leader and back.

Blockchain, on the other hand, puts more effort in the proof-of-work and delay it introduces. PoW requires only a one-way communication from the successful miner to its peers,

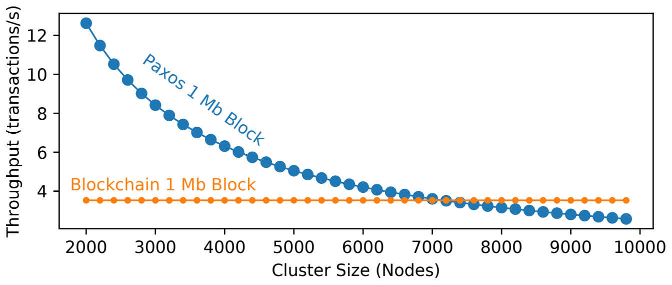

Fig. 6: Estimated maximum throughput of paxos and blockchain, both using 1 Mb blocks with 495 byte transactions.

while the difficulty of the puzzle both reduces the rate at which that one-way communication happens and reduces the chance of multiple miners arriving to the solution at roughly the same time. Additionally, the peer-to-peer communication further reduces miner bottlenecks by shedding some of the network load to the nearby peer nodes.

All of these techniques make sure that the network performance does not become the bottleneck in the system, allowing blockchains to provide stable throughput regardless of the number of participants. One can easily estimate the maximum throughput of the system just by knowing how many transactions can go to a block and the mining rate of the chain: $R_{max} = \frac{b_s R_{blocks}}{\mu_{tx}}$, where R_{blocks} is the block mining rate in blocks per some unit of time, b_s is the block's storage capacity, and μ_{tx} is average transaction size.

Figure 6 shows our estimated maximum throughput for Paxos and blockchain. Both protocols use 1 Mb blocks filled with transactions of 495 bytes. For Paxos, we assumed the leader is responsible for sending blocks to each follower over a 100 Mbit connection. Expectedly, the number of messages greatly hinders the performance of Paxos as we increase the number of nodes due to the longer block transmission times. At the blockchain scale, a lower baseline throughput is offset by its stability and better performance in large deployments. Note that we did not use byzantized Paxos in this comparison, and protocols like PBFT require even more communication and will degrade quicker than non-byzantine Paxos.

Reasoning about operation latencies involves accounting for all possible delays between client issuing a request and receiving the ack from the system. For very large clusters, the network bandwidth at the leader will determine these latencies, and for smaller deployments the latencies are largely driven by the compute capacity. For instance, in a network-bound estimate, the time to run a Paxos round on 1 Mb block at a leader with 100 mbps connection may exceed 800 seconds, if no additional methods to relieve network bottlenecks are used.

PoW blockchain latency largely depends on the probabilistic nature of Nokamoto consensus. Thus, the latency of PoW blockchain depends on the time required to mine sufficient number of blocks to make reversing the consensus highly improbable. In Bitcoin this is typically 6 blocks, each taking roughly 10 minutes to mine. However, transactions may spend some additional time in the pool before being picked up by the miners and added to the block, and the queuing effect of the pool may drive transaction commit latency above the 60-minute minimum. More recent blockchains that offer irreversibility of transactions once the block is appended to the chain can greatly reduce the minimum latency to the time of mining/adding one block.

VII. Future Work and Concluding Remarks

In this paper we drew parallels between classical consensus protocols and blockchains. Although the algorithms have originated in disparate communities with different background and expertise, both share many similarities, such as leader election phase, and transaction accept phase. Recently there has been several successful examples of transferring techniques from classical consensus protocols to blockchain domain to offer better throughput and latency. Aside from borrowing on the protocol-level techniques, we suggest that blockchain work would benefit from the formal methods commonly adopted in distributed systems. For instance, the DAO hack case study in Section V shows how formal specification and model checking can help find bugs in smartcontracts.

In our current work, we work towards a more direct performance comparison of blockchain and classical consensus at the protocol level in order to illustrate the benefits and drawbacks of protocol-level techniques independent of implementations and deployment environments.

We have developed the Paxi framework [28], which allows consensus protocols to be evaluated in a common environment under varying workloads, deployment size, and failure recovery scenarios. Paxi framework already ships with a variety of Paxos protocols including MultiPaxos [29], Flexible Paxos [30], EPaxos [19], Paxos Groups, WPaxos [31], WanKeeper [32], within the framework. We plan to add Blockchain implementations to Paxi in the near future to enable direct comparison between Paxos and Blockchain solutions.

In our future work, we propose to use TLA+/PlusCal to develop a formal framework and a domain specific language in order to facilitate modeling and proving properties for blockchain consensus protocols and systems. The framework will be used for model checking blockchain and smartcontracts in the presence of inopportune crash and byzantine failures, stretched timelines and schedules, and network partitions. Finally, the stepwise refinement that TLA+ formalism supports can help us explore the blockchain design state-space for hybridized/hierarchical versions of blockchains protocols that provide efficiency/performance benefits.

References

[1] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 18–25, 2001.

[2] M. J. Fischer, N. A. Lynch, and M. S. Peterson, "Impossibility of distributed consensus with one faulty processor," *Journal of the ACM*, vol. 32, no. 2, pp. 373–382, 1985.

[3] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 225–267, 1996.

[4] A. Ailijiang, A. Charapko, and M. Demirbas, "Consensus in the cloud: Paxos systems demystified," in *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*. IEEE, 2016, pp. 1–10.

[5] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, November 2002.

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

[7] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Annual International Cryptology Conference*. Springer, 1992, pp. 139–147.

[8] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Secure Information Networks*. Springer, 1999, pp. 258–272.

[9] M. Demirbas and Y. Song, "An rssi-based scheme for sybil attack detection in wireless sensor networks," *Advanced EXPerimental activities ON WIRELESS networks and systems (EXPONWIRELESS) Workshop (as part of WOWMOM)*, pp. 564–570, 2006.

[10] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 3, pp. 872–923, May 1994.

[11] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[12] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system."

[13] "Paxos performance model," https://github.com/acharapko/ppm/, 2018.

[14] W. G. Marchal, "An approximate formula for waiting time in single server queues," *AIIE transactions*, vol. 8, no. 4, pp. 473–474, 1976.

[15] L. Lamport, "Byzantizing paxos by refinement," in *International Symposium on Distributed Computing*. Springer, 2011, pp. 211–224.

[16] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.

[17] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 279–296.

[18] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," *Stellar Development Foundation*, 2015.

[19] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is more consensus in egalitarian parliaments," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 358–372.

[20] S. Peluso, A. Turcu, R. Palmieri, G. Losa, and B. Ravindran, "Making fast consensus generally faster," *Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016*, pp. 156–167, 2016.

[21] L. Lamport, D. Malkhi, and L. Zhou, "Vertical paxos and primary-backup replication," in *Proceedings of the 28th ACM symposium on Principles of distributed computing*. ACM, 2009, pp. 312–313.

[22] C. Newcombe, "Why amazon chose tla+," in *Abstract State Machines, Alloy, B, TLA, VDM, and Z*. Springer, 2014, pp. 25–39.

[23] M. HERLIHY, "Blockchains from a distributed computing perspective," 2018.

[24] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 79–94.

[25] R. Morris, "Ways of losing information," 1995.

[26] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing." NetDB, 2011.

[27] F. P. Junqueira, I. Kelly, and B. Reed, "Durability with bookkeeper," *ACM SIGOPS Operating Systems Review*, vol. 47, no. 1, pp. 9–15, 2013.

[28] A. Ailijiang, A. Charapko, and M. Demirbas, "Paxi - paxos protocol framework," https://github.com/ailidani/paxi, 2018.

[29] R. Van Renesse and D. Altinbuken, "Paxos made moderately complex," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p. 42, 2015.

[30] H. Howard, D. Malkhi, and A. Spiegelman, "Flexible Paxos: Quorum intersection revisited," 2016. [Online]. Available: http://arxiv.org/abs/1608.06696

[31] A. Ailijiang, A. Charapko, M. Demirbas, and T. Kosar, "Multileader wan paxos: Ruling the archipelago with fast consensus," *arXiv preprint arXiv:1703.08905*, 2017.

[32] A. Ailijiang, A. Charapko, M. Demirbas, B. O. Turkkan, and T. Kosar, "Efficient distributed coordination at wan-scale," in *Distributed Computing Systems (ICDCS), 2017 37th International Conference on*. IEEE, 2017.