Blockchain Consensus

Tyler Crain[1], Vincent Gramoli[1,2], Mikel Larrea[1,3] et Michel Raynal[4]

[1]*University of Sydney, Australia*
[2]*Data61-CSIRO, Australia*
[3]*University of the Basque Country, Spain*
[4]*Institut Universitaire de France, IRISA and Université de Rennes, France*

In this paper, we present a new definition of Byzantine consensus that applies to blockchains, applications that allow to transfer digital assets through transactions. More precisely, a blockchain is a distributed abstraction where participants must reach a consensus on a unique block of transactions to be appended to the chain. This new consensus definition requires a validity property based on an application-specific predicate but does not require a decided value to be proposed as long as it is valid. Its novelty is to allow to reach consensus on a value even if this value was proposed only by Byzantine processes, provided that it is considered valid by the application. The advantage of differentiating the validation of a proposed value and the nature of the proposing participant is to make the system progress by appending blocks with transactions, where other algorithms would typically abort.

Mots-clefs : asynchrony, consortium blockchain, Byzantine

1 Introduction

Blockchain, as originally coined in the seminal Bitcoin paper [13], is a promising technology to track ownerships of digital assets within a distributed ledger. This technology aims at allowing processes to agree on a series of consecutive blocks of transactions that may invoke contract functions to exchange these assets. While the first instances of these distributed ledgers were accessed by Internet users with no specific permissions, companies have since then successfully deployed other instances in a *consortium* context, restricting the task of deciding blocks to a set of n (among others) carefully selected institutions with appropriate permissions.

For the distributed computing community, a blockchain may seem like the application of classical state machine replication [14], where processes can be Byzantine [12], to the cryptocurrency context. A major distinction between blockchain and state machine replication is, however, the relation between consecutive consensus instances. A blockchain requires each of its consensus instances to be explicitly related to the previous one. To be decided, a block proposed in consensus instance number x must embed the hash of the block decided at instance number $(x-1)$. By contrast, the classical state machine replication simply concatenates consensus instances one after the other without relating the input of a consensus instance to the previous consensus instance : the result of a command may depend on the previous commands, but not the fact that it can be applied.

This relation between instances is interesting as it entails a natural mechanism during a consensus instance for discarding fake proposals or, instead, considering that a proposal is *valid* and could potentially be decided. Provided that processes have a copy of the blockchain and the hashing function, they can locally evaluate whether each new block they receive is a valid candidate for a consensus instance : they simply have to re-hash a block and compare the result to the hash embedded in the new proposed block. If the two hashes differ, the new block is considered an invalid proposal and is simply ignored. If the hashes are identical, then the block could potentially be decided. (Whether this block is eventually decided depends on additional well-formedness properties of the block and the execution of the consensus instance.) This validity generalizes common definitions of Byzantine consensus, that either assume that no value proposed only by Byzantine processes can be decided [6], or, in the case where not all non-faulty processes propose the same value, that any value can be decided (i.e., possibly a value proposed by a Byzantine process) [7].

As it is impossible to solve consensus in asynchronous message-passing systems where even a single process may crash (unexpected premature stop) [8], it follows that it is also impossible to solve this generalized definition of consensus. We propose a reduction of the problem of blockchain consensus to the Byzantine binary consensus that works in an asynchronous model. This reduction is similar to the reduction from asynchronous secure computation [2] with explicit validation. When combined with our binary consensus [5], it offers a blockchain consensus with no randomization, no leader, and no signatures. The reduction spawns concurrent instances of racing reliable broadcast followed by racing binary consensus : It only waits for the fastest of the concurrent reliable broadcast instances to terminate before spawning binary consensus instances. As it assumes $t < n/3$, where n is the number of processes and t is an upper bound on the number of faulty processes, this reduction is resilience optimal. While our consensus is presented for n permitted processes, its blockchain could be publicly readable by more processes.

Section 2 presents the model. Section 3 presents the blockchain consensus problem. Section 4 presents the reduction of blockchain consensus to binary consensus.

2 Model

The system is made up of a set Π of n asynchronous sequential processes, namely $\Pi = \{p_1, \ldots, p_n\}$; i is called the "index" of p_i. "Asynchronous" means that each process proceeds at its own speed, which can vary with time and remains unknown to the other processes. "Sequential" means that a process executes one step at a time. This does not prevent it from executing several threads with an appropriate multiplexing. Both notations $i \in Y$ and $p_i \in Y$ are used to say that p_i belongs to the set Y.

The processes communicate by exchanging messages through an asynchronous reliable point-to-point network. "Asynchronous" means that there is no bound on message transfer delays, but these delays are finite. "Reliable" means that the network does not lose, duplicate, modify, or create messages. "Point-to-point" means that any pair of processes is connected by a bidirectional channel. Hence, when a process receives a message, it can identify its sender. A process p_i sends a message to a process p_j by invoking the primitive "send $\text{TAG}(m)$ to p_j", where TAG is the type of the message and m its content. To simplify the presentation, it is assumed that a process can send messages to itself. A process p_i receives a message by executing the primitive "receive()". The macro-operation broadcast $\text{TAG}(m)$ is used as a shortcut for "**for each** $p_i \in \Pi$ **do** send $\text{TAG}(m)$ to p_j **end for**".

Up to t processes can exhibit a *Byzantine* behavior. A Byzantine process is a process that behaves arbitrarily : it can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transitions, etc. A process that exhibits a Byzantine behavior is called *faulty*. Otherwise, it is *non-faulty*.

Reliable broadcast in Byzantine systems RB-broadcast [1] is a one-shot one-to-all communication abstraction, which provides processes with two operations denoted RB_broadcast() and RB_deliver(). When p_i invokes the operation RB_broadcast() (resp., RB_deliver()), we say that it "RB-broadcasts" a message (resp., "RB-delivers" a message). An RB-broadcast instance, where process p_x is the sender, is defined by the following properties.

- RB-Validity. If a non-faulty process RB-delivers a message m from a non-faulty process p_x, then p_x RB-broadcast m.
- RB-Unicity. A non-faulty process RB-delivers at most one message from p_x.
- RB-Termination-1. If p_x is non-faulty and RB-broadcasts a message m, all the non-faulty processes eventually RB-deliver m from p_x.
- RB-Termination-2. If a non-faulty process RB-delivers a message m from p_x (possibly faulty) then all the non-faulty processes eventually RB-deliver the same message m from p_x.

3 The Blockchain Byzantine Consensus

As in all message-passing consensus algorithms, it is assumed (in both the multivalued and binary consensus algorithms presented below) that all non-faulty processes propose a value.

Multivalued Byzantine consensus with predicate-based validity In this paper we consider an adaptation of the classical Byzantine consensus problem to Blockchain. Note that an important novelty is that a decided value does not have to be one of the proposed value. As its validity requirement relies on an application-specific valid() predicate to indicate whether a value is *valid*, we call this problem the Validity Predicate-based Byzantine Consensus (denoted VPBC) and define it as follows. Assuming that each non-faulty process proposes a valid value, each of them has to decide on a value in such a way that the following properties are satisfied. [†]

— VPBC-Termination. Every non-faulty process eventually decides on a value.
— VPBC-Agreement. No two non-faulty processes decide on different values.
— VPBC-Validity. A decided value is valid, it satisfies the predefined predicate denoted valid().

This definition generalizes the classical definition of Byzantine consensus, which does not include the predicate valid().

Binary Byzantine consensus validity The validity property of the underlying binary Byzantine consensus is the following : if all non-faulty processes propose the same value, no other value can be decided.

4 From Multivalued Blockchain Consensus to Binary Consensus

This section describes a reduction of multivalued blockchain Byzantine consensus to the binary Byzantine consensus similar to the reduction from asynchronous secure computation [2] but with an explicit validation, and based on the RB-broadcast communication abstraction and underlying instances of binary Byzantine consensus. Let BBC denote the computational power needed to solve binary Byzantine consensus. Hence, the "multivalued to binary" reduction works in the model (Section 2) when $t < n/3$ and BBC is provided, denoted by $\mathcal{BAMP}_{n,t}[t < n/3, \text{BBC}]$, which is resilience optimal.

Binary consensus objects. The processes cooperate with an array of binary Byzantine consensus objects denoted $BINC[1..n]$. The instance $BINC[k]$ allows the non-faulty processes to find an agreement on the value proposed by p_k. This object is implemented with the binary Byzantine consensus algorithm presented in [5]. To simplify the presentation, we consider that a process p_i launches its participation in $BINC[k]$ by invoking $BINC[k]$.bin_propose(v), where $v \in \{0,1\}$. Then, it executes the corresponding code in a specific thread, which eventually returns the value decided by $BINC[k]$.

```
operation mv_propose($v_i$) is
(01) RB_broadcast VAL($v_i$) ;
(02) repeat if ($\exists\, k : proposals_i[k] \neq \perp \wedge (BINC[k]$.bin_propose() not invoked$)$)
(03)           then invoke $BINC[k]$.bin_propose(1) end if ;
(04) until ($\exists \ell :\ bin\_decisions_i[\ell] = 1$) end repeat ;
(05) for each $k$ such that $BINC[k]$.bin_propose() not yet invoked
(06)           do invoke $BINC[k]$.bin_propose(0) end for ;
(07) wait_until ($\bigwedge_{1 \leq x \leq n} bin\_decisions_i[x] \neq \perp$) ;
(08) $j \leftarrow \min\{x$ such that $bin\_decisions_i[x] = 1\}$ ;
(09) wait_until ($proposals_i[j] \neq \perp$) ;
(10) return($proposals_i[j]$).

(11) when VAL($v$) is RB-delivered from $p_j$ do if valid($v$) then $proposals_i[j] \leftarrow v$.

(12) when $BINC[k]$.bin_propose() returns a value $b$ do $bin\_decisions_i[k] \leftarrow b$.
```

Figure 1: From multivalued to binary Byzantine consensus in $\mathcal{BAMP}_{n,t}[t < n/3, \text{BBC}]$

Local variables. Each process p_i manages the following local variables ; $\perp$ denotes a default value that cannot be proposed by a (faulty or non-faulty) process : An array $proposals_i[1..n]$ initialized to $[\perp, \cdots, \perp]$. The aim of $proposals_i[j]$ is to contain the value proposed by p_j. An array $bin\_decisions_i[1..n]$ initialized to $[\perp, \cdots, \perp]$. The aim of $bin\_decisions_i[k]$ is to contain the value (0 or 1) decided by the binary consensus object $BINC[k]$.

The algorithm. The algorithm reducing multivalued Byzantine consensus to binary Byzantine consensus is described in Figure 1. In this algorithm, a process invokes the operation mv_propose(v), where v is the value it proposes to the multivalued consensus. The behavior of a process p_i can be decomposed into four phases.
Phase 1 : p_i disseminates its value (lines 01 and 11). A process p_i first sends its value to all the processes

[†]. Note that our consensus definition decouples the validity of a value from the nature (faulty or non-faulty) of the process proposing it. We assume that every non-faulty process proposes a valid value for the sake of simplicity. However, if we assume that the Byzantine behavior of a process is related only to its code and not to its input value (which is application-dependent), our algorithm remains correct as long as at least one non-faulty process proposes a valid value.

by invoking the RB-broadcast operation (line 01). When a process RB-delivers the value v RB-broadcast by a process p_j, it stores it in $proposals_i[j]$ if v is valid (line 11).
Phase 2 : p_i starts participating in a first set of binary consensus instances (lines 02-04). Then, p_i enters a loop in which it starts participating in the binary consensus instances $BINC[k]$, to which it proposes the value 1, associated with each process p_k from which it has RB-delivered the proposed value (lines 02-03). This loop stops as soon as p_i discovers a binary consensus instance $BINC[\ell]$ in which 1 was decided (line 04).
Phase 3 : p_i starts participating in all other binary consensus instances (lines 05-06). After it knows a binary consensus instance decided 1, p_i invokes $\mathsf{bin\_propose}(0)$ on all the binary consensus instances $BINC[k]$ in which it has not yet participated. Let us notice that it is possible that, for some of these instances $BINC[k]$, no process has RB-delivered a value from the associated process p_k. The aim of these consensus participations is to ensure that all binary consensus instances eventually terminate.
Phase 4 : p_i decides a value (lines 07-10 and 12). Finally p_i considers the first (according to the process index order) among the successful binary consensus objects, i.e., the ones that returned 1 (line 08). [‡] Let $BINC[j]$ be this binary consensus object. As the associated decided value is 1, at least one non-faulty process proposed 1, which means that it RB-delivered a value from the process p_j (lines 02-03). Let us observe that, due to the RB-Termination-2 property, this value is eventually RB-delivered by every non-faulty process. Consequently, p_i decides it (lines 09-10).

5 Related Work

Our validated predicate-based consensus differs from previous definitions in the way validity is defined. The seminal paper on the agreement between Byzantine generals considers a single source ; its validity requires that if the source proposes only one value, then only this value can be decided [12]. In the case where multiple, potentially Byzantine, processes propose values, validity often requires that if all non-faulty processes propose the same value then this value should be decided [7]. The classic validity used in the crash model is sometimes used in the Byzantine model requiring that a decided value is proposed by some process [4] but the notion of valid value proposed by a Byzantine process can be unclear. A predicate was previously used to assess whether a value is valid, however, the resulting predicate-based validity requires that if all non-faulty processes propose the same valid value then this value should be decided [10]. A variant requires the decided value to be value that was proposed [3]. When values are not necessarily binary, the decided value may either be a special value $\perp$ or a value proposed by a non-faulty process [6]. As $\perp$ is a predefined value, deciding this value is similar to aborting [9]. Our validity property allows for a valid value proposed only by Byzantine processes to be decided rather than aborting.

Références

[1] Bracha G., Asynchronous Byzantine agreement protocols. *Information & Computation*, 75(2) :130-143 (1987)

[2] Ben-Or M., Kelmer B., and Rabin T., Asynchronous Secure Computations with Optimal Resilience. *PODC*, pp. 183-192 (1994)

[3] Cachin C., Kursawe K., Petzold, P. and Shoup V., Secure and Efficient Asynchronous Broadcast Protocols. *CRYPTO*, 2001.

[4] Chandra T. and Toueg S., Unreliable failure detectors for reliable distributed systems. *JACM*, 43(2) :225-267 (1996)

[5] Crain T., Gramoli V., Larrea M., and Raynal M., (Leader/Randomization/Signature)-free Byzantine Consensus for Consortium Blockchains. Technical report, `https://arxiv.org/abs/1702.03068` (2017)

[6] Correia M., Ferreira Neves N., and Verissimo P., From consensus to atomic broadcast : time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1) :82-96 (2006)

[7] Dwork C., Lynch N., and Stockmeyer L., Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2) :288-323 (1988)

[8] Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *JACM*, 32(2) :374-382 (1985)

[9] Hadzilacos V. and Toueg S., On deterministic abortable objects. *PODC*, pp.4-12 (2013)

[10] Kursawe K., Optimistic asynchronous Byzantine agreement. Manuscript (2000)

[11] Lamport L., Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7) :558-565 (1978).

[12] Lamport L., Shostack R., and Pease M., The Byzantine generals problem. *ACM TOPLAS*, 4(3)-382-401 (1982)

[13] Nakamoto S., Bitcoin : a peer-to-peer electronic cash system. http ://www.bitcoin.org (2008)

[14] Schneider F.B., Implementing fault-tolerant services using the state machine approach. *ACM Computing Surveys*, 22(4) :299-319 (1990)

‡. One could replace min by a deterministic function suited for blockchain that returns the value that represents the block with the maximum number of transactions to prevent a Byzantine process from exploiting a lack of fairness to its own benefit.