BVP: Byzantine Vertical Paxos

Ittai Abraham Dahlia Malkhi

May 28, 2016

Abstract

In this paper we introduce Byzantine Vertical Paxos. BVP is a framework whose goal is to provide a method to produce high throughput Byzantine fault tolerant state machine replication that is tailored for a permissioned blockchain environment. We present the framework and show its manifestation in several models: synchronous, asynchronous, and asynchronous while assuming servers have access to a Trusted Platform Module (TPM).

1 Introduction

BitCoin's blockchain demonstrated how to share information securely among a wide peer-to-peer network, ordering blocks in a totally ordered sequence, and guaranteeing that information is immutable. An alternative approach is being considered today by many serious institutions such as financial organizations and banks. The approach relies on Byzantine agreement among a federated network of participants, jointly storing an agreed sequence of such blocks.

In this paper, we consider the challenge of driving a serious, industrial-grade infrastructure for Byzantine agreement and state machine replication. We focus on two aspects, elasticity (dynamic reconfiguration) and throughput. We introduce Byzantine Vertical Paxos (BVP), a family of protocols designed to address both goals.

BVP is built of the foundations of Vertical Paxos (VP) [LMZ09], which separates a solution into two modes. The first mode is a simple steady state protocol, and the second is a reconfiguration protocol. Often, the steady state protocol is just primary-backup or its generalization to multiple nodes via Chain Replication (CR) [vRS04] or Two Phase Commit (2PC). The second mode is a Paxos based reconfiguration engine. The advantage of VP is that the steady state can be optimized for high throughput and the heavy lifting on Paxos is used only when reconfiguration is needed.

Here we introduce BVP, a Byzantine variant of VP. As in VP, in BVP the steady state mode can be simple and highly optimized despite the Byzantine settings. In some settings, as we will see below, a non-faulty leader can provide safe progress with as few as $f + 1$ replicas. A separate consensus engine is used for reconfiguration.

Previously, a VP-oriented approach was used in an asynchronous Byzantine setting as follows. The Byzantine Chain Replication (BCR) protocol of van Renesse et al. [RHS12] implements state-machine-replication using $2f + 1$ replicas for steady-state, and using a separate $3f + 1$ consensus engine for reconfiguration management. Zyzzyva [KAD+10], while not providing the full elasticity that we aim for, separates between an optimistic, steady-state mode, and a recovery mode. Its optimistic faultless mode requires unanimity among an entire set of $3f + 1$ replicas, and exhibits 3 message delays. Zyzzyva falls back for recovery to using $(2f + 1)$-of-$(3f + 1)$ replicas, and exhibits 5 message delays.

Our work considers BVP in its full generality, extending these works into several new, practical settings. First, for all settings, including the classical asynchronous Byzantine settings, our approach provides a full-fledged reconfiguration mechanism.

Additionally, in many realistic settings the network is synchronous. In this case, our steady-state solution requires only $f + 1$ replicas, and $2f + 1$ replicas for reconfiguration. In fact, we don't really require the system to be synchronous in steady state, only to have some out-of-band synchronous control channel for reconfiguration purposes. Yet other settings may have secure hardware devices such as TPMs. Again we provide an $f + 1$ steady-state solution here.

In all of these settings, the core of the VP-oriented technique is to "wedge" a replicated state-machine and capture its "closing state". For every consensus decision, this requires the wedging coordinator to

access one non-Byzantine replica participating in the decision. For example, in Zyzzyva this is done by wedging $2f + 1$ out of $3f + 1$ replicas. In a synchronous settings, the coordinator waits for a known latency bound. And so on.

The following table summarizes the results we provide and compares against previously known ones. Note that in BCR and all BVP variants, only $f + 1$ replicas need to store state; the rest of the replicas may be utilized as active witnesses for agreement purposes only.

model	steady state			reconfiguration	
	msg delays	msgs	replicas	full reconfig	replicas
async (PBFT)	4	quadratic	$(2f + 1)$-of-$(3f + 1)$	yes	$(2f + 1)$-of-$(3f + 1)$
async (Zyzzyva)	3	quadratic	$3f + 1$	no	$(2f + 1)$-of-$(3f + 1)$
async (Zyzzyva+BVP)	3	quadratic	$3f + 1$	yes	$(2f + 1)$-of-$(3f + 1)$
async (BCR)	$2f + 2$	linear	$2f + 1$	yes	$3f + 1$
sync (BVP)	4	quadratic	$f + 1$	yes	$2f + 1$
sync (BVP)	3	quadratic	$2f + 1$	yes	$2f + 1$
async+TPM (BVP)	3	linear	$f + 1$	yes	$2f + 1$

2 Wedging a Replicated State-Machine

State-Machine Replication (SMR) is the task of forming agreement on a sequence of state-machine *commands*. Consensus on each command is formed independently of other commands. In steady state, there is a fixed set of replicas and a fixed algorithm driving decisions, one after another.

A *reconfiguration* mechanism changes the steady-state mode of the system. For example, it can be employed to replace a leader in a leader-based scheme. Although this is a minor change, conceptually we think of it as a configuration change. Likewise, reconfiguration may be employed to change the entire set of replicas.

The core mechanism employed for reconfiguration is a *wedging* scheme. A wedging coordinator obtains validation from a *wedge*, which is a subset of the replicas. At the same time, the coordinator obtains the latest state the wedge stores (including all the proposals it stores in every sequence position).

Then the coordinator drives a reconfiguration consensus decision. This consensus decision is implemented by a separate Byzantine consensus engine called a *reconfiguration engine*.

Importantly, the reconfiguration consensus decision itself has two components, (i) *next configuration*, and (ii) *closing state*. Whereas the first may be obvious, the second component deserves explanation.

When wedging starts, some consensus decisions may be only partially completed. This is inevitable in a distributed system, and consequently, there is going to be uncertainty about the status of ongoing decisions. For example, in a $(2f + 1)$-of-$(3f + 1)$ scheme, a wedging coordinator collects information from $2f + 1$, leaving the remaining f unknown. If it hears that $f + 1$ (of the $2f + 1$) voted for a certain state-machine command, say command number 1, the only safe course for the coordinator is to adopt the command into the closing state. Note that it is unknown whether the remaining f replicas ever vote for this command, past or future. This uncertainty is inherent in all of the settings and schemes we discuss below, and determining a safe closing-state is the heart and the core of correct wedging.

After the wedging procedure is complete and reconfiguration a consensus decision is reached, the SMR implementation switches to a conceptually new system (although the configuration change itself may be minimal, e.g., a leader change). The new system adopts as its starting state the closing state of the current. For example, if the reconfiguration decision contains a decision on some state-machine command number at slot number 1, then in the new system, command number 1 is already decided.

Below, we discuss different solutions for wedging and for the reconfiguration engine. The solutions differ in their model assumptions, their wedges, and their implementation of the separate reconfiguration engine.

2.1 Asynchronous Model

We start with a brief recap of the classic asynchronous Byzantine setting. In this model, PBFT [CL99] gives 4 rounds and $2f + 1$ for the steady state and Zyzzyva [KAD$^+$10] uses 3 rounds in the optimistic case but requires $3f + 1$ replicas for the steady state. Using $3f + 1$ for reconfiguration is also standard in both cases.

2.2 Synchronous-Reconfiguration Model

In the synchronous model, we provide two options, a 4-round (4 message delays) solution with $f + 1$ replicas, and a 3-round (3 message delays) solution with $2f + 1$ replicas.

4**-Round:** The first option is a steady-state mode with one (trivial) quorum of $f + 1$ replicas. We can run a standard 4-round protocol:

round 1	Client sends to Primary
round 2	Primary signs and sends to all $f + 1$
round 3	all $f + 1$ send signed-echoes of the Primary's message to each other
round 4	each of $f + 1$ sends a composite message containing all signed-echoes to Client
Client proceeds when all $f + 1$ composites arrive	
closing state	every composite containing $f + 1$ signed echoes

Note that we do not require a synchronous model for this interaction, only for reconfiguration. Namely, if at any stage the delay gets too large we can run a reconfiguration. Each round incurs one message delay, for a total of 4. The third round is an all-to-all exchange, and the fourth has linear size messages, each incurring quadratic single-message complexity.

Reconfiguration is handled as follows. The wedging coordinator must contact all surviving, non-faulty replicas in order to prevent Byzantine replicas from truncating the history of validated Primary commands. The coordinator relies on a known bound on communication delays and waits for it to expire. This guarantees that any surviving, non-faulty replica responds to the coordinator. Every command for which the wedging coordinator receives a composite message with $f + 1$ signed echoes is adopted in the closing state.

On a practical note, in our experience, it is possible to engineer systems to switch to synchronous mode when needed, e.g., by employing during these periods certain networking infrastructure or a special authority which has network priority. Alternatively, we could simply assume that a system has synchrony.

As for the consensus reconfiguration engine itself, it is well known that this can be done using $2f + 1$ replicas in the synchronous model.

3**-Round:** The second option is a steady-state mode with one (trivial) quorum of $2f + 1$ replicas, and a a Zyzzyva-like 3-round protocol:

round 1	Client sends to Primary
round 2	Primary signs and sends to all $2f + 1$
round 3	all $2f + 1$ send signed-echoes of the Primary's message to Client
Client proceeds when all $2f + 1$ echoes arrive	
closing state	every command which has $f + 1$ signed echoes

Again note that we do not require synchronous model for this interaction. If at any stage the delay gets too large we can run a synchronous reconfiguration protocol as follows. The wedging coordinator waits for the known communication upper bound in order to hear from all surviving, non-faulty replicas. Every command for which the wedging coordinator receives $f + 1$ signed echoes is adopted in the closing state.

2.3 Asynchronous Model with a TPM

In this setting we assume servers are equipped with Trusted Platform Module. Formally we assume a weak sequential broadcast (WScast), defined in [CMSK07, AAM10]. Roughly speaking, WScast is a

broadcast that ensures that (a) messages from a given sender are delivered by correct processes in the same order; this is an ordering per-sender, similar to FIFO broadcast, and (b) if the sender is correct then eventually all processes will receive all its messages.

We provide an SMR solution in this model with a steady state 3-round protocol, using $f+1$ replicas, and incurring a linear message complexity:

round 1	Client sends to Primary
round 2	Primary WScast to all $f+1$
round 3	all $f+1$ WScast echoes of the Primary's message to Client
Client proceeds when $f+1$ echoes arrive	
closing state	every command whose echo was (provably) WScast

Reconfiguration can be done in constant time using $2f+1$ nodes as in [AAM10]. The wedging coordinator asks all replicas to prove the history of messages it has WScast, and waits for one valid reply. Every command for which a valid reply indicate it was WScast is adopted in the closing state.

Note that there are several performance advantages of using a TPM. In steady state, we need only $f+1$ replicas, and at the same time, communication is linear in number of replicas (unlike the quadratic number of messages in the standard models).

Another advantage of using secure hardware, which is left outside the scope of discussion here, is the possibility to leverage "proof of elapsed time" to simplify and improve the efficiency of the leader election part inside the reconfiguration service.

References

[AAM10] Ittai Abraham, Marcos K. Aguilera, and Dahlia Malkhi. Fast asynchronous consensus with optimal resilience. In *Proceedings of the 24th International Conference on Distributed Computing*, DISC'10, pages 4–19, Berlin, Heidelberg, 2010. Springer-Verlag.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[CMSK07] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 189–204, New York, NY, USA, 2007. ACM.

[KAD+10] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27(4), January 2010.

[LMZ09] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical paxos and primary-backup replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC '09, pages 312–313, New York, NY, USA, 2009. ACM.

[RHS12] Robbert Renesse, Chi Ho, and Nicolas Schiper. Byzantine chain replication. In Roberto Baldoni, Paola Flocchini, and Ravindran Binoy, editors, *Principles of Distributed Systems*, volume 7702 of *Lecture Notes in Computer Science*, pages 345–359. Springer Berlin Heidelberg, 2012.

[vRS04] Robbert van Renesse and Fred B. Schneider. Chain replication for supporting high throughput and availability. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 7–7, Berkeley, CA, USA, 2004. USENIX Association.