Elastic Replication for Scalable Consistent Services

Hussam Abu-Libdeh, Haoyan Geng, Robbert van Renesse
Cornell University
{hussam, geng, rvr}@.cs.cornell.edu

Abstract

Most of the scalable and high-performance services used in datacenters today provide relaxed consistency guarantees in order to achieve good responsiveness. One reason for this is that it is believed that expensive majority-based consensus protocols are needed in order to provide strong consistency in asynchronous and partially synchronous environments such as a datacenter or the Internet.

In this extended abstract, we briefly describe our research into building a new lightweight replication protocol that does not use majority voting and yet provides strong consistency in the presence of crash faults and imperfect failure detectors.

1 Motivation and related work

Systems are replicated in order to improve availability (by having multiple independently failing copies of a system, the failure of some subset of the copies can be tolerated), and performance (multiple copies can divide load among them). However, with replication comes the question of consistency. A *strongly consistent* replicated system behaves, externally, identical to its unreplicated counterpart. But making a replicated system strongly consistent can compromise both availability and performance as the replicas need to coordinate operations. Consequently, many replicated systems have embraced relaxed consistency in which the replicated system sometimes behaves differently from the unreplicated counterpart.

Cloud computing services are often built using replication protcols such as Primary-Backup [1], or Quorum Intersection [3]. These services often rely on a centralized configuration manager (CCM) [2]. The CCM itself is a system that is replicated using a strongly consistent state machine replication protocol such as Paxos [4].

In the case of Primary-Backup, a primary replica receives all updates, orders them, and forwards them in FIFO order to the non-faulty backup replicas. Clients can read any of the non-faulty replicas. In case of a primary failure, one of the backups becomes the new primary. This is coordinated using the CCM.

Quorum Intersection protocols are useful for `put`/`get`-type systems such as Key-Value Stores. A `put` operation, accompanied by a timestamp, is sent to a "put-quorum," while a `get` operation reads from a "get-quorum." By guaranteeing that any put-quorum and get-quorum intersect, a `get` can be guaranteed to see the latest put operation. By making quorums smaller than the entire set of replicas, availability and performance are achieved. The CCM is responsible for keeping track of the replicas and the quorum sizes.

Neither of these common replication approaches guarantees strong consistency in the absence of accurate failure detection (*aka* fail-stop failures [5]). Both can provide stale data: a client might read one replica (or quorum of replicas) to get version n of the data, and afterward another client may read from a replica (or quorum of replicas) that has not yet been updated and has only seen version $n-1$. But stale data is not the worst problem. In Primary-Backup, if the primary is mistakenly suspected of having failed and another backup is designate primary by the CCM, a client may read the result of some operation from the original primary that is not applied, and never will be, to the second replica. In the case of Quorum protocols, reconfiguration of the set of replicas could temporarily result in non-intersecting quorums.

2 Elastic Replication

Elastic replication is a new light-weight crash-tolerant replication protocol. It supports strong consistency semantics, fast reconfiguration, flexible replica placement policies, and it does not rely on a CCM or accurate failure detection.

	PB	CR	Paxos	QI	ER
minimum # replicas needed	$f+1$	$f+1$	$2f+1$	$2f+1$	$f+1$
strongest consistency provided	linearizable	linearizable	linearizable	atomic R/W	linearizable
requires accurate fault detection	yes	yes	no	no	no
uses timeouts for liveness	yes	yes	yes	no	yes

Table 1: Comparison of Primary-Backup (PB), Chain Replication (CR), Paxos, Quorum Intersection (QI), and Elastic Replication (ER) protocols.

In Elastic Replication, state is replicated among a group of replicas specified by some *configuration*. For the purpose of this discussion, we model the state at each replica as a finite sequence of commands and we call it the replica's *history*. One of the replicas in a configuration is designated as an *orderer* to serialize all incoming commands. Additionally, a configuration must contain at least one replica that is not suspected faulty in order to guarantee safety and liveness. To execute a command on the replicated state, the command is first serialized by the orderer and propagated to all the replicas in the configuration. If all replicas receive that command, it is added to the local history by each of them. After that, the command executed and a response is generated if needed.

Two techniques are central to the operation of our protocol: *wedging* and *reconfiguration*. If any replica of the current configuration is suspected faulty, then a *wedge* command is broadcast to all replicas in the current configuration. When a replica receives a wedge command, it becomes wedged and it stops adding new commands to its history. Since each configuration contains at least one correct replica, at least one replica will become wedged, and thus the replicated history won't be extended any further. Wedging prevents the replicated history from diverging due to network partitioning or inaccurate failure detection.

Wedging guarantees the safety of our protocol. To ensure liveness, wedged configurations must be reconfigured. To reconfigure, a new set of replica processes are created with one of them designated as an orderer. The new replicas then proceed to *inherit* the history from any of the replicas of the prior configuration. Once all of the new replicas have obtained their histories, the new configuration becomes operational and it can start accepting new commands.

A key insight in elastic replication is that we use horizontal partitioning (sharding) to cut down the cost of reconfiguration and failure tolerance. Many existing datacenter and cloud services already rely on horizontal partitioning for scalability. We treat each partition as a separate replicated object and rely on partitions with no faulty replicas to quickly reconfigure partitions containing faulty ones.

A partition's state is separated from its configuration. Each partition acts as the *configuration sequencer* of another partition. When a partition needs to be reconfigured, its sequencer wedges the old configuration and issues a new configuration according to some local policy. This technique has two implications: First, a CCM is no longer needed. Second, each partition requires $f+1$ replicas to tolerate f crash failures as long as there always exists at least one partition composed entirely of non-faulty replicas. The reduced number of required replicas is similar to that of ZZ [6] with the exception that Elastic Replication only tolerates crash failures and does not require an additional f "sleeping replicas". Table 1 compares the properties of Elastic Replication to previously discussed replication techniques.

References

[1] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg. The primary-backup approach. In S. Mullender, editor, *Distributed systems (2nd Ed.)*. ACM Press/Addison-Wesley, New York, NY, 1993.

[2] M. Burrows. The Chubby Lock Service for loosely-coupled distributed systems. In *7th Symposium on Operating System Design and Implementation*, November 2006.

[3] M.P. Herlihy. A quorum consensus replication method for abstract data types. *Trans. on Computer Systems*, 4(1):32–53, February 1986.

[4] L. Lamport. The part-time parliament. *Trans. on Computer Systems*, 16(2):133–169, 1998.

[5] R.D. Schlichting and F.B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. *Trans. on Computer Systems*, 1(3):222–238, August 1983.

[6] T. Wood, R. Singh, A. Venkataramani, P. Shenoy, and E. Cecchet. ZZ and the art of practical BFT execution. In *Proceedings of the 6th conference on Computer systems*, EuroSys '11, pages 123–138, 2011.