Revisiting the Paxos Foundations: A Look at Summer Internship Work at VMware Research

Heidi Howard
University of Cambridge, and VMware OCTO

Dahlia Malkhi
VMware OCTO

Sasha Spiegelman
Technion, Israel, and VMware OCTO

1 Introduction

The summer of 2016 was buzzing with intern activity at the VMware Research Group (VRG), working with all the research team and with David Tennenhouse, Chief Research Officer of VMware. In this paper, we give a brief introduction to Flexible Paxos [4], one of the internship results. There were several other exciting outcomes; internships are a great way to participate in driving innovation at VMware!

Flexible Paxos introduces a surprising observation concerning the foundations distributed computing. The observation revisits the basic requisites of Paxos [7, 8], Lamport's widely adopted algorithmic foundation for fault tolerance and replication, and a pinnacle of his Turing award [1]. Since its publication, Paxos has been widely built upon in teaching, research and production systems.

Paxos implements a fault tolerant state-machine among a group of *nodes*. At its core, Paxos uses two phases, each requires agreement from a subset of nodes (known as a quorum) to proceed. Throughout this manuscript, we will refer to the first phase as *the leader election phase*, and the second as *the replication phase*. The safety and liveness of Paxos is based on the guarantee that any two quorums will intersect. To satisfy this requirement, quorums are typically composed of any majority from a fixed set of nodes, although other quorum schemes have been proposed.

In practice, we usually wish to reach agreement over a sequence of commands, not one. This is often referred to as the Multi-Paxos problem [3]. In Multi-Paxos, we use the leader election phase of Paxos to establish one node as a leader for all future commands, until it is replaced by another leader. We use the replication phase of Paxos to agree on a series of commands, one at a time. To commit a command, the leader must always communicate with at least a quorum of nodes and wait for them to accept the value.

In the Flexible Paxos work, we observe that Paxos is conservative:

> Each of the phases of Paxos may use non-intersecting quorums. Only quorums from different phases are required to intersect. Majority quorums are not necessary as intersection is required only across phases.

Everyone in the field of distributed systems knows that quorums in Paxos must intersect. So, what gives?

> *"The most useful piece of learning for the uses of life is to unlearn what is untrue. "* Antisthenes

Flexible Paxos (FPaxos) shows that within each of the phases of Paxos, it is safe to use disjoint quorums and majority quorums are not necessary. FPaxos captures a weaker requirement for quorum intersection than the original algorithm, requiring only that quorums from different phases intersect. Using this weakening of the requirements made in the original formulation of Paxos, FPaxos generalizes over the Paxos algorithm to provide flexible quorums.

There are far reaching implications of this result. Since the replication phase of Paxos is far more common than the leader election phase, we can use FPaxos to reduce the size of commonly used replication quorums. For example, in a system of 10 nodes, we can safely allow any set of only 3 nodes to participate in replication, provided that we require 8 nodes to participate when recovering from leader failure. This strategy, decreasing replication quorums at the cost of increasing leader election quorums, is referred to in the body of the FPaxos paper as *Counting Quorums*.

The new strategy reduces latency, as leaders will no longer be required to wait for a majority of participants to accept proposals. Likewise, it improves steady state

throughput as disjoint sets of participants can now accept proposals, enabling better utilization of participants and decreased network load.

In Counting Quorums, the price we pay for this is reduced availability as the system can tolerate fewer failures whilst recovering from leader failure. Surprisingly, there is a specific case of counting quorums where there is no cost at all: reducing the size of replication quorums by one when the number of acceptors is even. This improves the performance and availability of steady-state replication phase, without hurting availability of the leader election phase. In the above system of 10 nodes, you would use sets of size 5 to form replication quorums, and sets of 6 to form leader elections quorums.

Below, we additionally illustrate that there are quorum systems such as *Group Quorums*, in which FPaxos allows us to decrease the quorum sizes of both phases.

In summary, by no longer requiring quorums from the same phase to intersect, FPaxos has removed an important limit on the scalability and performance of distributed consensus.

2 Background

Lamport's Paxos alogrithm [7, 8] solves distributed consensus, the problem of reaching agreement in the face of failures. It is a common problem in modern distributed systems and its applications range from distributed locking and atomic broadcast to strongly consistent key value stores and state machine replication.

Mulit-Paxos has been widely utilized within production systems such as Google's Chubby [2] and Apache Zookeeper [5] and shares its underlying approach with other consensus algorithms such as Raft [12], Zab [6] and Viewstamped Replication [11, 9].

We will now look at the approach that Mulit-Paxos (and similar algorithms) take to solve distributed consensus. Readers should have some familiarity with at least one consensus algorithm. Van-Renesse and Altinbuken [14] or Ongaro and Ousterhout [12] are good starting points.

For this discussion, we will consider Multi-Paxos in the context of appending commands into a log, which is replicated across a system of nodes. These commands are then applied to replicas of an application's state machine. This technique is known as *state machine replication* [13].

The underlying approach behind Multi-Paxos is that we need the following two phases to commit commands:

- **Leader election phase** In this phase, one node essentially takes charge of the system, we call this node the leader. When this node fails, then the system detects this and uses this phase to choose another node to take the lead.
- **Replication phase** In this phase, the leader replicates commands into the logs of other nodes and decides when it is safe to call the commands committed.

The purpose of the leader election phase is twofold. Firstly, we need to stop past leaders from changing the state of the system. Typically, this is done by getting nodes to promise to stop listening to old leaders. For example, in Raft consensus a follower updates its term when it receives a *RequestVote* RPC for a higher term. The second purpose of the leader election phase is for the leader to learn all of the commands that have been committed in the past. For example, in Raft the leader election mechanism ensures that only nodes with all committed entries can be elected to lead.

The purpose of the replication phase is to copy commands onto other nodes. When sufficient copies of a command have been made, the leader considers the request to be committed and notifies the interested parties. For example, in Raft the leader applies the command locally and updates its commit index to notify the followers in the next AppendEntries.

We refer to the nodes who are required to participate in each phase as the *quorum*. Paxos uses majority quorum for both leader election and replication phases.

3 Flexible Paxos

The guarantee that is made by Multi-Paxos is that once a command is committed, it will never be overwritten by another command. To satisfy this, we must require that the quorum used by the leader election phase will overlap with the quorums used by previous replication phase(s). This is important as it ensures that the leader will learn all past commands and past leaders will not be able to commit new ones.

Paxos uses majorities but there are many other ways to form quorums for these two phase and still meet this requirement. Previously, it was believed that all quorums (regardless of which phase they are from) needed to intersect to guarantee safety.

Flexible Paxos (FPaxos) observes that this need not be the case. It is sufficient to ensure that the leader election quorums will overlap with replication phase quorums.

4 Implications

We will now explore a few of the wide-ranging implication of this result.

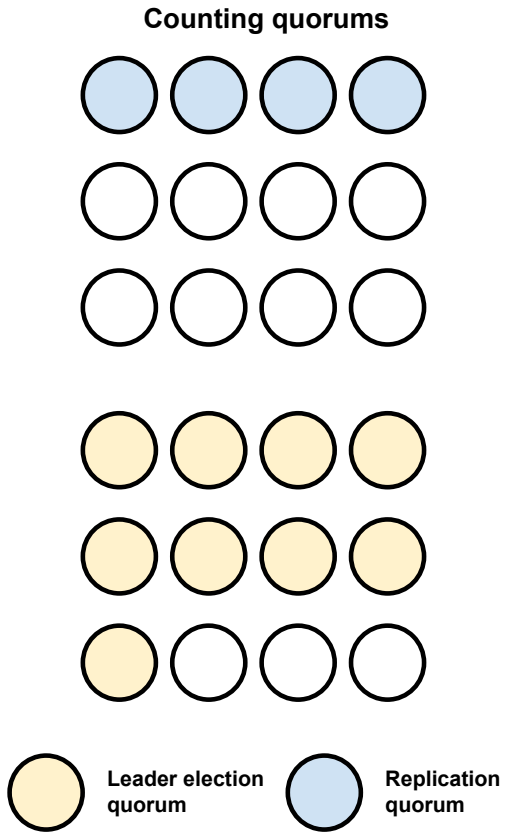

Figure 1: Example of counting quorums in a system of 12 nodes. The replication quorum size is 4 and the leader election quorum size is 9

Firstly, we can now safely trade-off the quorum size in the leader election phase for the quorum size in the replication phase. We refer to this as *counting quorums* and can express this as follows, where N is the number of nodes, Q_{leader} is the size of the leader election quorum and $Q_{replication}$ is the size of the replication quorum.

$$Q_{leader} + Q_{replication} > N$$

For example, in a system of 6 nodes, it is sufficient to get agreement from only 3 nodes in the replication phase when using majorities for leader election. Likewise, it is sufficient to get agreement from only 2 nodes in the replication phase if we require that 5 nodes participate in leader election. An example of this approach is shown in Figure 4.

By reducing the number of nodes required to participate in the replication phase, we have improved performance in the steady state as we have fewer nodes to wait upon and to communicate with. But has this improved performance come at the cost of reduced availability?

We consider two types of availability: The ability to learn committed commands and elect a new leader (leader election availability) and the ability for the current leader to replicate commands (replication availability). Both types of availability are useful in their own right. If the current leader can commit commands but we cannot elect a new leader then the system is available until the current leader fails. If we can elect a new leader but not commit new commands then we can still safely learn all previously committed values and then use reconfiguration to get the system up and running again.

In our first example, we used replication quorums of size $N/2$ for a system of N nodes when N was even. This both improves performance and is more fault-tolerant than Paxos's majority quorums. If exactly $N/2$ failures occur, we can now continue to make progress in the replication phase until the current leader fails.

For different quorum sizes, we have a trade-off to make. By decreasing the number of nodes in the replication phase, we are making it more likely that a quorum for the replication phase will be available. However, if the current leader fails then it is less likely that we will be able to elect a new one.

However, the story does not end here.

We can be more specific about which nodes can form replication quorums so that it is easier to intersect with them. For example, if we have 12 nodes we can split them into 4 groups of 3. We could then say that a replication quorum must have one node from each group. Then when electing a new leader, we need only require any one group to agree. This is shown in the Figure 4.

It is the case in both examples that 4 failures could be sufficient to make the system unavailable if the leader also fails. However, with groups is not the case that any 4 failures would suffice, now only some combinations of node failures are sufficient (e.g. one failure per group).

There is a host of variants on this idea. There are also many other possible constructions [10]. The key idea is that if we have more information about which nodes have participated in the replication phase then it is easier for the leader election quorums to intersect with replication quorums.

We can take this idea of being more specific about which nodes participate in replication quorums even further. We could extend the consensus protocol to have the leader notify the system of its choice of replication quorum(s). Then, if the leader fails, the new leader need only get agreement from one node in each possible replication quorum from the last leader to continue.

No matter which scheme we use for constructing our quorums, we always have a fundamental limit on availability. If all nodes in the replication quorum fail and so does the leader then the system will be unavailable (until

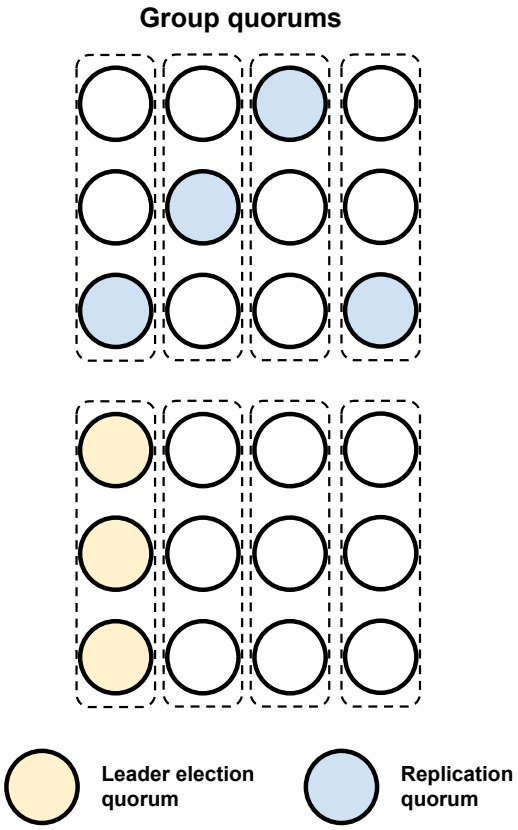

Figure 2: Example of *Group Quorums* in a system of 12 nodes. The replication quorums include at least one node from each group and the leader election quorums include at least all nodes from one group.

a node recovers) as no one will know for sure what the committed command was.

5 Conclusion

In this paper, we have given a brief introduction to Flexible Paxos (FPaxos). FPaxos observes that each of the two phases of Paxos may use non-intersecting quorums. This generalisation enables the use of many interesting quorum constructions, removing an important limit on the scalability and performance of distributed consensus. By utilizing this result and with pragmatic system design, we believe a new breed of scalable, resilient and performant consensus algorithms is now possible.

References

[1] ACM. A.M. Turing Award - Award winner Leslie Lamport. `http://amturing.acm.org/award_winners/lamport_1205376.cfm` [Online; accessed 6-June-2017].

[2] BURROWS, M. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2006), OSDI '06, USENIX Association, pp. 335–350.

[3] CHANDRA, T. D., GRIESEMER, R., AND REDSTONE, J. Paxos made live: An engineering perspective. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2007), PODC '07, ACM, pp. 398–407.

[4] HOWARD, H., MALKHI, D., AND SPIEGELMAN, A. Flexible Paxos: Quorum Intersection Revisited. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016)* (2017).

[5] HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2010), USENIXATC'10, USENIX Association, pp. 11–11.

[6] JUNQUEIRA, F. P., REED, B. C., AND SERAFINI, M. Zab: High-performance broadcast for primary-backup systems. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on* (2011), IEEE, pp. 245–256.

[7] LAMPORT, L. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2 (May 1998), 133–169.

[8] LAMPORT, L. Paxos made simple. *ACM SIGACT News (Distributed Computing Column)* (2001).

[9] LISKOV, B., AND COWLING, J. Viewstamped replication revisited. Tech. Rep. MIT-CSAIL-TR-2012-021, MIT, 2012.

[10] NAOR, M., AND WOOL, A. The load, capacity, and availability of quorum systems. *SIAM J. Comput. 27*, 2 (Apr. 1998), 423–447.

[11] OKI, B. M., AND LISKOV, B. H. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 1988), PODC '88, ACM, pp. 8–17.

[12] ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In *Proc. USENIX Annual Technical Conference* (2014), pp. 305–320.

[13] SCHNEIDER, F. B. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv. 22*, 4 (Dec. 1990), 299–319.

[14] VAN RENESSE, R., AND ALTINBUKEN, D. Paxos made moderately complex. *ACM Comput. Surv. 47*, 3 (Feb. 2015), 42:1–42:36.