Instructors' Guide to Raft

Posted on Mar 16, 2016 — shared on Hacker News, Twitter, and Lobsters

For the past few months, I have been a Teaching Assistant for MIT’s 6.824 Distributed Systems class. In the past, the class has had several labs building on the Paxos consensus algorithm. This year, we decided to make the move to Raft. Raft was designed with the explicit goal of being easy to understand, in the hope that this would make the students’ lives easier.

This post, and the accompanying Students’ Guide to Raft, chronicles our journey with Raft this past semester. Our hope in sharing this is that it might be useful to other educators looking to add Raft to their curriculum. If you want to build or understand Raft, you should look at the Students’ Guide linked to above instead.

Before we dive into Raft, some context may be useful. 6.824 used to have a set of Paxos-based labs that were built in Go. Go was chosen because it is a simple language, and because it is well-suited for writing concurrent, distributed applications. Over the course of several labs, students built a fault-tolerant, sharded key-value store. The first lab had them build a consensus-based log library. The second added a key/value store replicated state machine (RSM) on top of that. The third sharded the key space among many fault-tolerant clusters, with a fault-tolerant shard master handling configuration changes. This article will discuss our experiences with rewriting the first lab, as it is the one most related to Raft.

Explaining Raft

The Raft protocol is a fairly straightforward algorithm to explain at a high level. Visualizations like this one give a good overview of the principal components of the protocol, and the paper gives good intuition for why the various pieces are needed. If you haven’t already read the extended Raft paper, you should go read that before continuing this article, as I will assume a decent familiarity with Raft.

In the steady state where there are no failures, Raft’s behavior is easy to understand, and can be explained in an intuitive manner. For example, it is simple to see from the visualizations that, assuming no failures, a leader will eventually be elected, and eventually all operations sent to the leader will be applied by the followers in the right order.

As with all distributed consensus protocols, the devil is in the details. When networks delay RPCs, networks are partitioned, and servers fail, sophisticated reasoning about the exact rules dictated by the specification (i.e., the paper) is required to explain why Raft behaves correctly. The ultimate guide to Raft is in Figure 2 of the Raft paper, which specifies the behavior of every RPC exchanged between Raft servers, gives invariants that servers should maintain, and dictates when certain actions should occur. Both your teaching, the students attention, and the rest of this article will be centered around Figure 2.

Raft and Paxos

An important difference between our Paxos and Raft labs is that our Paxos labs were based on Paxos, the single-value consensus algorithm, not Multi-Paxos. The latter adds features such a single-round soft leader commits, and is also commonly referred to as just Paxos.

The basic idea when building a replicate state machine on top of Paxos is that you run multiple instances of Paxos consensus. The messages for each instance are then "tagged" with the index in the global log they belong to. Multi-Paxos adds several optimizations on top of this, which also adds to the protocol's complexity. For 6.824, we decided the increased performance was not worth the added complexity for the purposes of teaching. Instead, we had the students design their own simple protocol for keeping a replicated log on top of Paxos, and from that, a replicated state machine.

In contrast, Raft provides a full protocol for building a distributed, consistent log, including a number of optimizations such as persistence, leader election, single-round agreement, and snapshotting. Raft is thus more similar to Multi-Paxos than Paxos, both in terms of feature set, performance, and complexity. Paxos consensus alone (i.e., not Multi-Paxos) is conceptually simpler than Raft.

Implementing Raft

The go-to guide for implementing Raft is Figure 2 of the extended Raft paper. At first, both you and the students will be tempted to treat Figure 2 as an informal guide; you read it once, and then start coding up an implementation that follows roughly what it says to do. Doing this, you will quickly get up and running with a mostly working Raft implementation. However, Figure 2 is, in reality, a formal specification, where every clause is a **MUST**, not a **SHOULD**. The Students' Guide to Raft goes into a great deal of depth about this. Failure to follow Figure 2 to the letter often leads to complex bugs, and errors in one part of the algorithm (e.g., snapshotting) can often adversely impact seemingly unrelated parts of the protocol (e.g., leader election).

In and of itself, this is not a problem. A consensus algorithm specification must necessarily be precise, and it is reasonable that things break if you don't follow it exactly. However, since Raft bakes in a number of optimizations into the consensus algorithm, there are many more things that can go wrong than for a "simple" RSM implementation on top of Paxos.

If you want to assign a stripped-down, low-performance RSM to students to teach them the basics of consensus-based distributed systems, we know how to do that starting with Paxos. If you forego the optimizations implemented by Multi-Paxos, and instead use simple multi-round Paxos agreement for each value in the log, you get a design that seems to us to be simpler than both Raft and current practice for Paxos-derived RSM.

We don’ t know how to do that starting with Raft. This is because Raft has a fair amount of sophistication and optimization build into its core protocol, and it is not clear how to cleanly separate them out. The Raft paper does a good job of separately describing each of the protocol’ s main components, which helps gradually build the students’ understanding of Raft at a high level, but this decomposition does not easily translate into strict isolation in the implementation.

What happened when we switched to Raft?

We originally switched to Raft because we believed that it would be easier for the students to follow a complete design than fiddling with how to construct their own Paxos RSM protocol out of Paxos’ single-value agreement. Distributed consensus is complicated, and the Raft authors have tried hard to give a complete description of a protocol that is relatively easily digestible and understandable.

Despite having a useful, complete write-up of the algorithm (which we did not have for Paxos), it seemed considerably harder for students to complete our Raft lab than the previous Paxos lab. We suspect that, as previously discussed, this is because Raft is a more sophisticated protocol than our previous, naïve, Paxos-based RSM. In addition, since we did not anticipate the additional work required due to the increased complexity ahead of time, we did not give the students additional time to complete the new labs, which added to the lab difficulty.

The story is similar when it comes to the students understanding what they are building – understanding the naïve Paxos RSM design in the old labs was easy; you did not have to reason about things like rolling back logs on leader re-election, or what guarantees that committed entries aren’ t lost when doing so. Because each log entry is determined by a completely separate Paxos instance, and single-agreement Paxos is dramatically simpler than Raft, the overall complexity was moderately low.

Raft is “easy” in the sense that you can translate Figure 2 directly into code, and get a working system without having to think too deeply about why everything works. However, it is not easy to understand exactly why it works, and thus an intuitive implementation approach (i.e., not slavishly following Figure 2) is unlikely to work. In many ways, the key advantage of Raft is that, while it may be complex and hard to fully understand, it is written down and explained in a paper.

Going forward

Despite the increased perceived difficulty of our labs after switching to Raft, we are going to continue using Raft for 6.824 labs in the coming years. This is for a couple of reasons. First, there exists a paper that contains a good, complete description of Raft (namely, the Raft paper). This was not the case for the old lab. Second, Raft eliminates annoying corner cases that students were forced to deal with in the Paxos-based labs, such as holes in the log or entries committing out of order. Third, Raft has persistence built-in, which we believe is important for students to understand, and which allows for more rigorous testing (by crashing servers).

One might also argue that the stripped-down Paxos RSM we had the students build in the previous set of labs was overly simplistic. Exposing the students to a well thought-through, optimized, end-to-end algorithm like Raft arguably gives them more experience with what building useful, production-ready consensus-based distributed systems is like.

In an attempt to improve the situation for future years, and to help other students of Raft (academic or otherwise), we have also written the Students' Guide to Raft. It gives a more implementation-oriented description of the Raft protocol, discusses common questions and pitfalls, and gives(/will eventually give) pseudocode for a correct implementation.

(revision history)