Kyle Fauerbach

CS7610 Fall 2018

Final Project Report

Why Paxos?

Prior to this class I had zero experience working with distributed systems and I also didn't have any research that I thought would benefit from adding distributed features. So when I was looking for projects I was looking for something that would both reinforce things that had been discussed in class and be applicable to different scenarios. I ended up choosing Paxos because it is a pretty fundamental protocol. Paxos was originally proposed by Lamport in *The Part-Time Parliament* and Kirsch and Amir also wrote *Paxos for System Builders,* which provides explanation of their implementation of the protocol.

A lot of other systems use Paxos, or similar operation, at different points. This seemed like a good opportunity to get deeper exposure and it might help better understand other systems. Another motivating factor in selecting Paxos over another protocol is the documentation that exists for the protocol. *Paxos for System Builders* provides a lot of information on steps that are required to create a practical, functioning implementation of Paxos. Given my lack of experience in building distributed systems, this seemed like a good starting point.

Project Scope

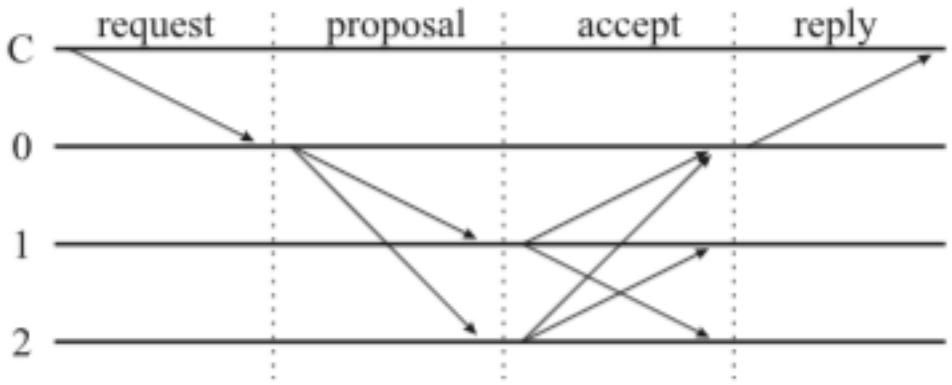
When selecting the scope for the project it was important to consider that I would be working alone, the timeframe for the project was one month, and there had to a be working implementation at the end of that time. The most straightforward goal would be to build a 100% functional implementation that was ready to be deployed. Given the previous constraints, this was unrealistic for several reasons. The reconciliation process is one of the most important parts of the protocol. It is how severs that fall behind are brought up to date, allowing ordering to continue with all servers. It is also one of the hardest parts of the process. Lamport's original paper does not describe a reconciliation method and Kirsch and Amir only provide a high level overview of their implementation. Given the work still needed for the rest of the protocol we chose to hold off on reconciliation.

Unfortunately, without a functioning reconciliation process we cannot completely implement the view change procedure and recovery portions because we cannot bring the server up to date with its peers.

Problem Statement

Given these limitations it was decided that we would focus on the normal case operation. A leader would be elected and the initial view installed. Then a client update would be received and accepted by the servers.

The below image is from *Paxos for System Builders* and depicts the normal case (no failure) operation of Paxos upon receiving a message from the client.



Current Project Status

At the completion of the project timeframe we have successfully implemented the normal case operation of Paxos. The servers complete leader election and install the same view. Upon receiving a client update, the leader proposes the new global ordering, which is accepted by a quorum of the other servers, at which point the update is executed. The next section will go into greater detail on how the implementation was completed.

Implementation Details

The implementation was written in C. This was chosen because the point of the project was to gain a better understanding of the protocol and doing this made sure that we knew exactly what was happening with the data structures. There are some cases where this made things a little more complicated especially without including other libraries, for example the Update Queue of client updates. Because this data structure can grow and shrink we had to create the queue so it was able to support those operations. If we had been using a higher level language, or using other libraries, we wouldn't have had to do that.

All networking is done using UDP datagrams. Because of the assumptions that Paxos makes about the network we did not have to implement any sort of reliable UDP transmission. Paxos is designed to work in the asynchronous model so it is able to tolerate messages that are lost or delayed. When transmitting messages between servers we prepended a basic header that contained the type and size of the message that followed.

We also assume a list of peers is known at startup. The focus of the project was on the consensus algorithm not on peer discovery.

Because the project was only focused on the normal case operation, we did not implement any snapshotting. We were not planning for servers to crash and even if they did they would not be brought back into the network.

We have implemented a main loop that checks if the Progress Timer has expired and if it has triggers leader election. After that any Update Timers for pending Client Updates are checked. Then we check the listening socket for any messages, and apply conflict checks and handle the messages as necessary. Then we loop back to the Progress Timer.

For the most part our implementation followed *Paxos for System Builders,* the biggest problem that we encountered was that Paxos was never designed to run on its own. The whole point is to assign a global order to a series of client updates. But without another application relying on Paxos, there were no client updates. There also isn't much description on the initial state when the protocol begins. Most of the individual choices were made in this area.

We begin in the Leader Election phase, wherein one of the servers is designated to start the election by nominating themselves are leader. Once a majority of the servers are informed of the new view we move directly into the Prepare phase. The Prepare phase goes smoothly because nothing has been ordered yet so there is no Global History and there are no updates to Propose.

After the view has been installed by the majority of the servers, server 1, the leader in this view, triggers its Client Update Handler by sending a manufactured Client Update to itself. After the Proposed update is Accepted by the majority of the other servers, the Update is Executed by printing a message to the terminal.

Conclusion

We were able to implement a version of Paxos that functions correctly under the normal case. We were also able to create a demonstration video that shows this in action using 4 peers. While this implementation is admittedly limited in its capabilities, we were able to achieve the functionality we specified and the current code base is a good starting point for future work.