Multi-Paxos Simulator in Javascript/HTML5

Taylor Purviance, Laura Moss, Eric Vernon

Introduction and Project Goal

The goal of the project was to implement a simulation of the Paxos algorithm, such that it can be easily demonstrated and viewed for the purposes of academia. The simulation is accompanied by a visual representation that assists in the understanding of the algorithm. Our minimum goal was to have the standard and most simple version of the algorithm implemented.  From there, we added interactivity to the system, allowing the user to perturb the system from its base case by making server nodes unresponsive. Additionally, we implemented the Multipaxos variation on standard Paxos, which assumes a stable distinguished proposer, or leader.

It was also important for this project to be easily shared with students to make understanding the algorithm somewhat easier. Hence we chose to implement our project in javascript and HTML5 Canvases so anyone can run it in a browser with little overhead.

Along with the goals of understanding the algorithm and being able to demonstrate it in a way that increases its understandability, the project was also an attempt to learn technologies and brush up on them. As the web is a continuously blossoming field with more and more need for developers, picking up some skills at web technologies was helpful to all of us. Even if we don’t go on to work with web technology, it is still likely helpful to understand how these prevalent technologies work.

Paxos Algorithm Overview

From our research, we determined that basic Paxos works like this:

There are multiple nodes/servers in the system. Each of these can have one or more roles:

* a client that submits a value to proposers
* a proposer that takes the value submitted to it by a client and submits the values to a group of acceptors.  It decides the order in which client commands should be executed.
* a leader that is the special proposer that is elected to be the single proposer to which all the clients submit their values to.
* an acceptor which tries to decide which of the proposed values they are sent should be accepted
* a learner who decides what the final, accepted value is from the acceptors that have locked in on a value.

Each role at each node runs on its own process and interacts with other processes by sending messages. These messages may take a variable amount of time to be sent or received. Furthermore, any of the processes can stop participating by not responding to messages. As long as a majority of the processes do not stop participating, the Paxos algorithm will work to ensure consistency.

To begin with, a leader is elected among the processes. How the election is done is not specified by the algorithm, but assumed to be doable somehow. One possible way to elect a leader might be to select the maximum value of some unique value of each process, like the PID or MAC address.

Once a leader is selected, the processes are all notified of who the leader is. Then, whenever any client submits a value, it submits it to the leader. If the leader becomes unresponsive or fails, a new leader is elected from the remaining processes.

During the prepare phase, the leader/proposer will send a message to a majority of the acceptors with *v* the value that it wants to submit, and *n* some number to associate with the proposal.

Acceptors will receive the message check the number *n*. If *n* is the largest proposal number the acceptor has received yet, it will respond with a promise to not accept any lower-numbered proposals, as well as send the proposal number of the last proposal it had promised to.

Once the leader/proposer receives these promises from a majority of the acceptors it will send the acceptors an accept request for proposal number *n*. The acceptors will accept the proposal if they haven’t promised to a higher numbered proposal yet. when they accept a proposal, the acceptors inform both the proposer and all of the learners of the value of their accepted proposal.

The learners will eventually have a majority of the acceptors accepted values and will note the value that the acceptors formed majority consensus on.

Group Organization

Eric: In charge of mastering and tutoring in Javascript and HTML5, contributes heavily to development.

Laura: Primary effort towards development, secondary towards understanding Paxos.

Taylor: Primary effort towards understanding Paxos, secondary towards development.

Timeline

|  |  |  |
| --- | --- | --- |
| **Date** | **Title** | **Description** |
| 3/29/13 | Milestone 1 - Project Proposal | This document |
| 4/12/13 | Milestone 2 - Literature Review | Read and understand existing literature relating to the Paxos algorithm.  Write a document which summarizes relevant literature, and discusses it in regards to our work. |
| 4/19/13 | Begin to implement Paxos | Start to implement the framework for the Paxos algorithm.  Create nodes and have them send messages to one another. |
| 4/26/13 | Paxos with no perturbations, graphical simulation | Basic Paxos with graphical simulation |
| 5/3/13 | Begin perturbations | Make significant progress on perturbations to Paxos. |
| 5/10/13 | Finish simulation | Finish leader election. |
| 5/17/13 | Milestone 3 - Final Deliverables | Debug issues with leader election and quorum size. |

Implementation

The simulation, again, is written in javascript and the graphical representation of the nodes and messages is drawn on an HTML5 canvas.  The nodes automatically all have the role of acceptor and learner, and during the election phase one node is chosen to also be a leader.  This is a more realistic simulation, as a collection of servers would likely all be running the same script.  The simulation requires little interaction from the user during the actual process: all the user can do is initialize the number of multi-functioning nodes (*n*), how many nodes in a quorum (*n*/2 + 1 by default), pause and unpause the simulation, submit a value to send from the client node, and turn nodes on or off.

One assumption we made but did not implement is that the nodes would “gossip” to be aware which nodes were up.  Since showing the gossip messages would clutter the simulation, other nodes were able to access other nodes’ status through the collection that contained all of them.

Results

asdf

Reference Papers

Lamport, Leslie. "Paxos made simple." *ACM SIGACT News* 32.4 (2001): 18-25. <http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf>

Prisco, Roberto De, Butler Lampson, and Nancy Lynch. "Revisiting the PAXOS algorithm." *Theoretical Computer Science* 243.1 (2000): 35-91. <http://www.sciencedirect.com/science/article/pii/S0304397500000426>

Abraham, Ittai, et al. "Byzantine disk paxos: optimal resilience with byzantine shared memory." *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. ACM, 2004. <http://dl.acm.org/citation.cfm?id=1011801>

Martin, Jean-Philippe, and Lorenzo Alvisi. "Fast Byzantine Paxos." *Proc. Iof the International Conference on Dependable Systems and Networks (DSN’05)*. 2004.

Lamport, Leslie. "Fast paxos." *Distributed Computing* 19.2 (2006): 79-103. <http://research.microsoft.com/apps/pubs/default.aspx?id=64624>

Lamport, Leslie.  “Cheap Paxos”.  *The International Conference on Dependable Systems and Networks.* 2004. <http://research.microsoft.com/en-us/um/people/lamport/pubs/web-dsn-submission.pdf>