Mandatory Assignment 1
INF-3203: Advanced Distributed Systems: Paxos

January 26, 2022

Introduction

Observing the behavior and being able to understand the key characteristics of a computer program is an important part of being a systems researcher. The non-functional requirements of a system are often difficult to measure, and it's important that we construct good experiments that accurately portray these. Paxos [1] is arguably the most famous consensus algorithm, and its' creator Leslie Lamport one of the most important computer scientists in our field. For this assignment, you are expected to implement and evaluate a version of the Paxos algorithm published by Robbert Van Renesse and Deniz Altinbüken called *multi-Paxos* [2].

The Paxos Algorithm

The Paxos algorithm allows a set of processes to reach consensus on a value. In each run of the algorithm, processes take one of three roles; *proposer, learner* or *acceptor*. The total number of acceptors is assumed known by all proposers.

A Paxos run starts with a proposer sending a *prepare* request with a proposal number n (where n is part of some ordered set) to a majority of acceptors. Acceptors respond with a promise to ignore messages with a lover proposal number than n, or the proposal with the highest proposal number n' lover than n that it has accepted, if any. If the proposer gains a promise from a majority of acceptors, it sends out a *proposal* message to a majority of acceptors with proposal number n", where n" is the highest proposal number among the responses, along with a value v. Here, v is the value from the highest numbered proposal accepted by any of the acceptors. Finally, the acceptors or proposer broadcast the decision to the learners, who are passive participants in this process.

It should be noted that each process in Paxos can take multiple roles, even within the same Paxos run, although this is not required.

Multi-Paxos

Multi-Paxos is a variant of the base Paxos algorithm that uses slightly different roles. In multi-Paxos, processes can have three roles; replica, leader or acceptor.

Replicas have *slots* which need to be filled with values. The goal of multi-Paxos can be seen as reaching a consensus between all replicas on what value should be stored in each slot in order to maintain a consistent state between them so that they appear to be the same to an outside observer. Reaching a consensus on what value to store for each slot takes the form of a single Paxos run. The replicas accept requests from clients and forward them to one or more leaders, requesting that it starts a proposal for the lowest unused slot of the replica. The leaders then act as proposers towards the acceptors as with simple Paxos. The replicas wait until they get a confirmation response from the leaders before updating its internal state.

Requirements

Code

You can implement your solution in any programming language you like. You are also not required to deploy it to the uvcluster - local only testing is fine.

There is no need to implement the *perform* part of multi-Paxos. The replicas must reach a consensus on the value for each slot, but these values can be simple integers if you wish. It is also acceptable to have your replicas act as acceptors instead of running them as separate processes or servers. Finally, you do not need to consider replicas failing and recovering, but feel free to test how your system handles them if you want to.

Additionally, you are expected to write test scripts that does any required setup and runs your experiments, and these scripts should preferable be easy to configure. At a minimum, your scripts should:

- Initialize a Paxos cluster of a specified size. Size must be a command-line argument, e.g. ./paxoseval size=3.
- Measures the accept throughput by issuing a series of concurrent proposals to the cluster. The upper threshold of concurrent clients must be a command-line argument, and the program has to perform individual measurements for each concurrency level.
- Outputs a graph of system throughput in accepted proposals per second based on the number of concurrent clients.
- Gracefully shuts down the Paxos cluster and releases any resources used by your program.

Instruction on how to run your scripts should be included in a ReadMe file alongside the rest of your code.

Report

Your report should be 2-6 pages long, preferably in the IEEE conference style[1]. Apart from the usual parts of a scientific paper, it should include:

- A short introduction to Paxos and Multi-Paxos.
- A description of the architecture and design of your solution. This should cover the details about the language and libraries you used, what decisions you made while making it and so on.
- A thorough description of your experimental methodology. What experiments did your run, and on what hardware? In what way did you take measurements, and what did you actually measure? Why did you think those were worthwhile measurements?
- Plots and data from your experiments.
- Discussion of the results.

Resources

Reading both the Paxos papers referenced in the introduction are good places to start this assignment. A Python implementation of Multi-Paxos is available online[2] alongside other useful materials. The Python implementation can be used as a basis for your own implementation, or you can implement it from scratch if you want a challenge. Two good libraries for plotting your data are Mathplotlib[3] and Gnuplot[4].

Deliverables

You are required to hand in your code, scripts and report in as a zip folder or tarball on Canvas. Hand ins can be done as a group, with up to 3 students to a group. You can self organize using the group function in Canvas. Larger groups will have higher expectations in both code and report quality.

You *can* work individually, but you are strongly encouraged to find a group so you can discuss the assignment with fellow students.

The deadline for handing in the assignment is **February 23, 2022 at 23:59**. As always, start early, fail early. Learning and understanding Paxos takes time, so don't worry if it seems hard at first. Good luck, and happy coding!

[1] https://www.ieee.org/conferences/publishing/templates.html
[2] http://paxos.systems/
[3] http://matplotlib.org/api/pyplot$_api.html$
[4] http://www.gnuplot.info/

References

[1] Leslie Lamport *Paxos made simple.* ACM Sigact News, 32(4), 18-25.
[2] Robbert Van Renesse and Deniz Altinbüken *Paxos made moderately complex.* ACM Computing Surveys (CSUR), 47(3):42, 2015.