Mechanised Verification of Paxos-like Consensus Protocols

Anirudh Pillai

January 17, 2018

Thesis advisor: Professor Dr. Ilya Sergey Anirudh Pillai

Mechanised Verification of Paxos-like Consensus Protocols

Abstract

Distributed systems are hard to reason about.

Acknowledgments

Acknowledgements

Contents

1 Introduction

Talk about some stuff for this chapter.

1.1 The Problem

1.2 Aims and Goals

I have highlighted the aims and goals separately. The aims are what I want to achieve out of undertaking this project and the goals are the things that this project tries to achieve.

Aims

1. Learn about distributed system protocols
2. Contribute to open source software

Goals

1. Read about and understand the classical Paxos-like consensus algorithms.
2. Develop state transition systems for the algorithms and identify the invariants that need to be preserved during the operation of the algorithm.
3. Implement a simulation of the protocols in Python.
4. Formulate the implemented protocols in Disel by using the developed state-transition systems.
5. Mechanise the proofs of the identified protocol invariants in Disel/Coq.
6. Add additional communication channels and prove composite invariants.
7. Provide an abstract specification of the protocol, usable by third-party clients.

8. Mechanise a client application of the protocol verified out of the abstract interface.

1.3 Project Overview

1.4 Report Overview

2
Background

This chapter lays down all the previous research which the project builds on. Before going over the design decisions on the project we first need to understand this background information and look at related work to see different approaches used to solve the problem.

2.1 Distributed Systems

A distributed system is a model in which processes running on running different computers, which are connected together in a network, exchange messages to coordinate their action, often resulting in the user thinking of the entire system as one single unified computer.

A computer in the distibuted system is also alternatively referred to as a processor or a node in the system. Each node in a distributed systems has its own memory.

We will now go over a few concepts of distributed systems which will help us understand the characteristics of the protocols that run on these systems. This will lay down the groundwork for us to understand the Paxos protocol on which this project is based.

Asynchronous Environment

An asynchronous distributed system is one where there are no guarantees about the timing and order in which events occur.

The clocks of each of the process in the system can be out of sync and may not be accurate. Therefore, there can be no guarantees about the order in which events occur.

Further, messages sent by one process to another can be delayed for an arbitary period of time.

A protocol running in an asynchronous enviroment has to account for these conditions in its design and try to achieve its goal without the guarantees of timed events. An asynchronous environment is very common for a real world distributed system but it also makes reasoning about the system harder because of the aforementioned properties.

Faults Tolerance

A fault tolerant distributed system is one which can continue to function correctly despite the failure of some of its components. A 'failure' of a node or 'fault' in a node means any unexpected behaviour from that node eg. not responding to messages, sending corrupted messages.

Fault tolerance is one of the main reasons for using a distributed system as it increases the chances of your application continuing to functioning correctly and makes it more dependable. As Netflix mention on their blog 'Fault Tolerance is a Requirement, Not a Feature'. With their Netflix API receiving more than 1 billion requests a day, they expect that it is guaranteed that some of the components of their distributed system will fail. Using a fault tolerant distributed system they are able to ensure that a small failure in some components doesn't hinder the performance of the overall system, hence, enabling them to achieve their uptime metrics.

Fault tolerant distributed system protocols are protocols which achieve their goals despite the failure of some of the components of the distributed system they run on. The protocol accounts for the failures and generally specifies the maximum number of failures and the types of failures it can handle before it stops functioning correctly.

State Machine Replication

In a client server model, the easiest way to implement it is to use one single server which handles all the client request. Obviously this isn't the most robust solution as if the single server fails, so does your service. To overcome the problem you use a collection of servers each of which is a replica of the original single server and ensure that each of these 'replicas' fails independantly, without effecting the other replicas. This adds more fault tolerance.

State Machine Replication is method for creating a fault tolerant distributed system by replicating servers and using protocols to coordinate the interactions of these replicated servers with the client.

A State Machine M can be defined as $M = \{q_0, Q, I, O, \delta, \gamma\}$ where

q_0 is the starting state

Q is the set of all possible states. I is set of all valid inputs O is the set of all valide outputs δ is the state transition function, $\delta : IxQ- > Q$ γ is the output function, $\gamma : IxQ- > O$

The method of modelling a distributed system protocol as state transition system is very common and is a critical component of this project as well.

Consensus Protocols

For handling faults in your distributed system you need to have replication. This leads to the problem of making all these replicas agree with each other to keep them consistent. Consensus protocols try to solve this problem.

Consensus protocols are the family of distibuted systems protocols which aim to make a distributed network of processes agree on one result.

These protocols are of interest because of their numerous real world applications. Let us take the example of a distributed database, which is a critical part of almost all large scale real world applications. This distributed database will run over a network of computers and everytime you use the database you aren't guaranteed to be served by the same computer.

Suppose you add a file to the database. This action is performed by the processor that was serving you 'add' request. Later when you want to retrieve the file from the database you might be served by a different computer that did not perform the 'add' request. Inorder for the new computer to know that the file exists in the database, you will need to use a consensus protocol which helps all the computers in the network (which handle user requests) agree upon the result that the file has been added to the database.

Popular consensus protocols include PageRank used by Google and the Blockchain consensus protocol. George and Ilya verified a subset of the protocol in Coq in their Toychain paper.

2.2 Paxos

Having understood the the main concepts behind distributed system protocols, we can now finally get to the protocol at the heart of this project. Paxos is a family of asynchronous, fault tolerant, consensus protocol which achieves consensus in a network of unrealiable processes as long as a majority of them don't fail.

Paxos is used for state machine replication and helps all the replicas achieve consensus on a result.

2.3 Disel

2.4 Related Work

3
Requirements and Analysis

3.1 Detailed Problem Statement

3.2 Requirements

3.3 Analysis

4
Design and Implementation

4.1 Modelling

4.1.1 The Protocol

4.1.2 State Transitions

4.1.3 Inductive Invariant

4.1.4 Client Implementatin

4.2 Verification

5
Client Implementation

6
Conclusion and Evaluation

6.1 Summary of Achievements

6.2 Critical Evaluation

6.3 Future Work

6.4 Final Thoughts

References

A
Interim Report