Achieving Distributed Consensus with Raft

Heidi Howard, Pembroke College

Originator: Anil Madhavapeddy

14^{th} October 2011

Project Supervisor: Anil Madhavapeddy

Director of Studies: Chris Hadley

Project Overseers: Prof. Jean Bacon & Prof. Ross Anderson

Introduction

Many modern applications rely on the fault tolerance and scalability provided by distributed systems. Nodes within a distributed system need to agree on consistent world-views, leading to the problem of achieving consensus. CAP theorem [1, 3] demonstrates that it is not possible to achieve consistency, availability and partition tolerance in a distributed system. Furthermore, a completely asynchronous consensus protocol cannot guarantee consensus with just a single fail-stop node [2], making comprising liveness or accuracy necessary in addition to comprising partition tolerance and/or availability, from the CAP theorem.

Paxos [4] is at the centre of distributed consensus research and is widely used and taught. But it's famously difficult to understand, reason about and implement because of its non-intuitive approach and the lack of universal agreement on the algorithm for multi-Paxos. This project will look at an alternative to Paxos, which claims to provide the same guarantees whilst being simpler to reason about and more fully specified.

Starting Point

Raft [8] is a newly proposed consensus algorithm (still under submission) for generating a consistent replicated log. Ongaro and Ousterhout (the algorithm's authors) have implemented Raft in ∼2000 lines of C++ and already other implementations exist. My starting points are:

- **Raft Specification:** Raft has been formally specified [10] in the TLA+ specification language and has been partially proven correct using mechanical and informal proofs. Ongaro and Ousterhout have also released the teaching materials [9] used in their study comparing the understandability of Raft and Paxos
- **Literature on Consensus:** To prepare for this project, I've examined some of the key papers, from the extensive literature on Consensus and Paxos. I'm still on the progress of reading about other consensus algorithm which Raft has been likened to such as Viewstamped Replication [7]
- **Previous Programming Experience:** I have previous experience working in OCaml and making use of the Core & Async libraries

Project Description

I plan to implement the Raft algorithm in OCaml, making use of its static type system to limit the reachable erroneous states and eliminate other classes of errors at compile time. I may make use of the following libraries and tools:

- **OPAM** - OCaml package manager for installation of libraries and compiler versions
- **Core & Async** - Alternative to the OCaml standard library and a monadic non-preemptive threading library, designed and maintained by Jane Street
- **SPIN** - Simulator and Model checker for non-deterministic automata described in Promela with properties described in Linear Temporal Logic

- **Statecall Policy Language** [6] - Language for specifying NDFA's which compiles to Promela and OCaml
- **PROMELA** - OCaml library for creating and modifying Promela programs

All of the above are open source and I have already secured copies of each

Resources Required

For this project I shall use my personal laptop running 32 bit Ubuntu 12.10, with an Intel Core i3 Quad core processor. As a backup, I will use my personal desktop running 64 bit Ubuntu 12.10, with an Intel Core i5 Quad core processor. If both of the above were to fail, I will fall back to the university's MCS.

For backup I will use a git repository on Github. I'll work within a Dropbox directory to catch any uncommitted work. I will synchronise regularly across these each of my machines and daily to the MCS.

Work to be done

The project breaks down into the following sub-projects:

1. Build a key-value store with a client, that can perform basic commands such as add, find or remove a key-value pair
2. Implement the core Raft algorithm either directly in OCaml or via SPL (excluding membership changes and log compaction)
3. Manually test the core algorithm with the key-value store application
4. Build a simulator to delay packets, stop/start nodes and run the application faster than real time
5. Run a series of simulations, varying external conditions (such as packet delay and MTBF of nodes) and algorithm parameters (such as election timeout)
6. Model the core algorithm in Promela and verify some basic safety properties

Success Criterion for the Main Result

The project will be a success if I have implemented the Raft consensus protocol and can reason about its safety properties, specifically I would like to be able to demonstrate:

1. The system makes progress whilst the majority of nodes are running and can communicate with each other
2. The system copes with unreliable network communications including network delays, partitions, and packet loss, duplication, and re-ordering
3. The system is able to be run either as a real world implementation or simulation
4. The system operates in an asynchronous environment (with faulty clocks, messages taking arbitrarily long to deliver and nodes operating at arbitrary speeds)
5. Each of the replicated state machines (such as the key-value store) receives commands in the same order therefore implementing a strongly consistent key-value store

Furthermore, I aim to provide the guarantees laid out by Ongaro and Ousterhout [8]:

1. Election Safety: at most one leader can be elected in a given term
2. Leader Append-Only: a leader never overwrites or deletes entries in its log; it only appends new entries.
3. Log Matching: if two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index.
4. Leader Completeness: if a log entry is committed in a given term, then that entry will be present in the logs of the leaders for all higher-numbered terms.
5. State Machine Safety: if a node has applied a log entry at a given index to its state machine, no other server will ever apply a different log entry for the same index.

Possible Extensions

If I achieve my main result early I shall try to additionally perform the following:

1. Implement cluster membership changes, to allow nodes to be removed and added to the system and verify its safety
2. Implement log compaction
3. Optimise performance, possibility inspired by literature on Paxos
4. Modify the protocol to handle a wider class of faults such as Byzantine faults [5]

Timetable

I hope to achieve the following at the end of each time period stated:

1. **24th Oct - 6th Nov (Michaelmas weeks 3-4)** Successful submission of Proposal by 25th October. Comparison of the guarantees claimed to be provided by Raft and those provided by Multi-Paxos. Decision regarding how to implement the core algorithm either directly in OCaml or via SPL
2. **7th Nov - 20th Nov (Michaelmas weeks 5-6)** Completed implementation of the surrounding code i.e the key-value store, client side code, network communication and logging
3. **21th Nov - 4th Dec (Michaelmas weeks 7-8)** Completed implementation of the core algorithm excluding cluster membership changes and log compaction
4. **5th Dec - 15th Jan (Michaelmas vacation)** Completed simulator ready to run simulations and output data for analysis
5. **16th Jan - 29th Jan (Lent weeks 1-2)** Successful submission of Progress Report by Friday 31st January and give Progress Presentation. Run simulations and evaluate

6. **30th Jan - 12th Feb (Lent weeks 3-4)** Completed analysis of my Raft implementation in SPIN.

7. **13th Feb - 26th Feb (Lent weeks 5-6)** Completed any extensions (as listed in the possible extensions section). If time allows then consider any alterations to the algorithm to extend its functionality or optimise it and how they could impact safety

8. **27th Feb - 12th Mar (Lent weeks 7-8)** Plan layout of dissertation and write the main chapters of the dissertation

9. **13th March - 23rd Apr (Easter vacation)** Completed further evaluation, typesetting and completed draft dissertation. Dissertation submitted to supervisor for feedback

10. **24th April - 6th May (Easter weeks 1-2)** Corrected dissertation in response to feedback and submission by 12 noon on Friday 16th May

References

[1] Eric A Brewer. Towards robust distributed systems. In *PODC*, page 7, 2000.

[2] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[3] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002.

[4] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[5] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[6] Anil Madhavapeddy. Combining static model checking with dynamic enforcement using the statecall policy language. In *Proceedings of the 11th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering*, pages 446–465. Springer-Verlag, 2009.

[7] Brian M Oki and Barbara H Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 8–17. ACM, 1988.

[8] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. Draft of October 7, 2013.

[9] Diego Ongaro and John Ousterhout. Raft: A consensus algorithm for replicated logs (user study). http://www.youtube.com/watch?v=YbZ3zDzDnrw.

[10] Diego Ongaro and John Ousterhout. Safety proof and formal specifcation for raft. https://ramcloud.stanford.edu/~ongaro/raftproof.pdf.