Visualizing Raft

Brugnara, Martin
#182904

Hoxha, Fatbardha

Nardelli, Andrea
#171723

Abstract

Raft is a consensus algorithm for managing a replicated log by Ontaro and Ousterhout. One of its key aspect is to be designed for understandability. The protocol author provides a simulator to visualize/understand/present it. Such tool implements the protocol only in its basics. In this work we extend its capabilities to support all the features and we enhance the GUI to clarify the more complex protocol's aspects.

1 Introduction

Raft is a consensus algorithm for managing a replicated log proposed by Ontaro and Ousterhout [1]. It is powerful as the state of the art (multi-)Paxos [2] and aims at replacing it thanks to its understandability. Consensus algorithm tend to be complex due to the problem they try to solve, it is not worthless saying that leaving space for interpretation, like Lamport did when proposed Paxos via a tale [2], does not help. The combination of Paxos' complexity and the way it was presented required indeed Lamport to release an alternative description [3]. Even if the author claims that "The Paxos algorithm, when presented in plain English, is very simple." it seams that the level of abstraction with which Paxos was designed and its complex architecture which does not map to the real world applications complicate its implementations, which sometimes require to relax the model leading to unproved protocols [4]. The same argument holds for the students and researches.
Ongaro, thus, identified in the *understandability* its "most important goal" [5].

What is the simplest way to present an try to understand and interaction based protocol? *Visualizing it!*

Ongaro uses and provides a Raft simulator, Raft Scope [6], to introduce the protocol. Such tool is very useful for easy scenarios simulation and slides creation, unfortunately it does not implement all the features of the protocol and lacks some graphics functionalities which can be key in the presentation of many corner cases.

In our project we aim at complete the protocol implementation, integrate the new functionalities with the existing one and improve the graphics where needed to enhance transparency and thus enable even more understanding.

2 Raft Scope

Raft Scope is a visualized (Figure 1) developed by the very same protocol author Diego Ontaro (@ongardie). It provides raft basic functionalities simulation, fast forward and run playback.

2.1 Protocol

The tools simulates the *leader election* and the *log replication* protocols as described in Ongaro's Ph.D. thesis [5] Chapter 3, except for Section 3.10 which is not implemented.

2.2 Interaction

The user can send *Client request*, start/stop/restart a server, manage the simulation speed and finally *time travel* (play/rewind/rollback/playback).

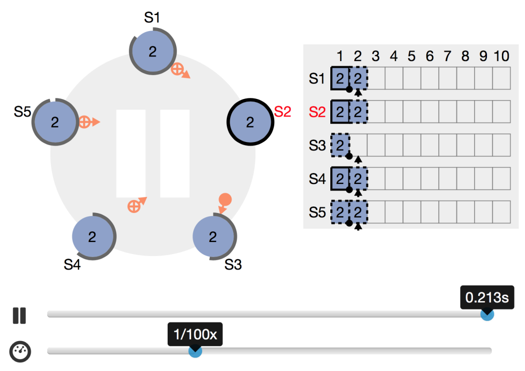


Figure 1: Original

2.3 Internals

It is worth nothing that JavaScript it self, language of the simulator, is single threaded, Ontaro implemented a sort of virtual machine (VM) where the concurrency is only simulated playing with the concept of time: each action (*e.g.* receiveMessage) is set to be taken effect at a specific point in time. The simulation take place in a main loop (`raft.update`) where each server is give the opportunity to make an action (*e.g.* start a new election); the available action are checked in defined order. The VM has an internal clock, that at each simulation cycle is advanced by an amount of time computed *w.r.t* the actual simulation speed.

Due to this design decision many operation are executed in *bulk*, the fact that they take place in order of due is not worthless but does not guarantee correctness: Suppose the Leader server (SL) send an `AppendEntries` message (M1) to server 2 (S2), and that the S2 `ElectionTimeout` is due to expire just after the M1 arrive, and that the two events take place in the same simulation slice time: in the real-world S2 would receive M1, reset its `ElectionTimeout`, and answer with and `ACK` messager. In the simulation the S2 check for *start new election* is preformed before the read of the due message, this lead to S2 actually start a new election.

Fixing this behaviour is not straight forward: swapping the two checks would break other scenarios (`e.g.` M1 is due just after the `ElectionTimeout`). This without considering what would happen if the simulation speed is in the range $[ELECTION\_TIMEOUT/2 - ELECTION\_TIMEOUT]$. The only definitive fix would have been rewriting the entire simulation system. Given the application of such tool: in-class demonstration/explanation and visual simulation we preferred to simply limit the speed, to a speed that is anyhow to fast to understand anything.

Moreover the current design allows to easily take snapshot, action that is actually performed after each serialization point enabling simulation rewind, rollback and playback.

3 Extending Raft Scope

We extended the current official Raft visualizer to emulate the full protocol and we integrated the latest published variation and fixes (Figure 2).

3.1 Protocol

Now the simulator support *Cluster membership changes*. Addition and removal of a follower and removal of the leader from the cluster via the single-server mutation algorithm proposed by Ongaro in his Ph.D. thesis [5].

The fix for single-server algorithm was also backported; as discussed at the #raft-dev mailing list [7] with this algorithm could happen double leadership and committed history rewrite. The fix we implemented is to write in the log a `NOOP` message every time a server becomes a *leader*.

We also implemented the safety check for Disruptive servers: Follower servers ignore `voteRequest` if it has heard from the current limit within `MIN_ELECTION_TIMEOUT`; prescribed in the section 4.2.3 of the Ongaro's thesis.

3.2 Simulation

We added the concept of `network noise`; the system simulate the noise on the network dropping each message with a probability set by the user.

A *cluster configuration changes queue* was added to allow the user to submit multiple configuration change requests. In the thesis it is not clear whether it is the leader to have to buffer for the changes request or the client trying up to succeed. We decided that it is duty of the leader; this means that when the leader changes pending operations are lost.

3.3 Internals

To support the new protocol functionalities we had to modify the `checkpoint` snapshot format and the main loop. In order to allow transparently replay/rollback/playback we have also to modify the rendering phase to take in account the possibility that servers can join and leave, particularly challenging was to deterministically decide when it is safe to remove a server from the graphic and thus from the simulation. We decided to remove it when no one server has got it in the `peers` list.

3.4 Interface

For the timing issue discussed in Section 2.3 we limited the simulation speed into a reasonable range. The log table has been rewritten to support an infinite log and now features also the auto scroll. To avoid confusion the "leader only" fields in the state modal of the follower servers have been hidden and a legend was added to explain the otherwise cryptic symbols in the log table. Moreover, now, each log entry is represented differently on the basis of its nature (`NOOP`, Config change message, user value). We obviously added controls for the feature we introduced in the Protocoll, we thus added a slider to set the *network noise*, a *server add* button and a voice to the context menu for server removal and finally a new table to visualize the cluster configuration changes queue. Besides that few minor bug were fixes.

3.5 Non implemented functionalities

The *leadership transfer extension* described in Section 3.10 of the thesis was not implemented since it was never implemented or evaluated by the authors.

We did not implemented *joint consensus* since the one implemented is as powerful as it is, but it is also simpler, more understandable and it is the one the author indicated as preferred.

4 Conclusions

With this work we completed a great and useful simulator for the Raft protocol. In particular we implemented the cluster changing functionalities; we also simplified and completed the GUI to allow the understanding of all the algorithm aspects and corner case. We (re-)learned the importance of questioning ourselves about the correctness of what we are reading. Modify the simulation model required us to master the Raft algorithm to fully comprehend the model itself.

We hope our work will easy the learning curve for the others to come, as the original Raft Scope did partially for us.

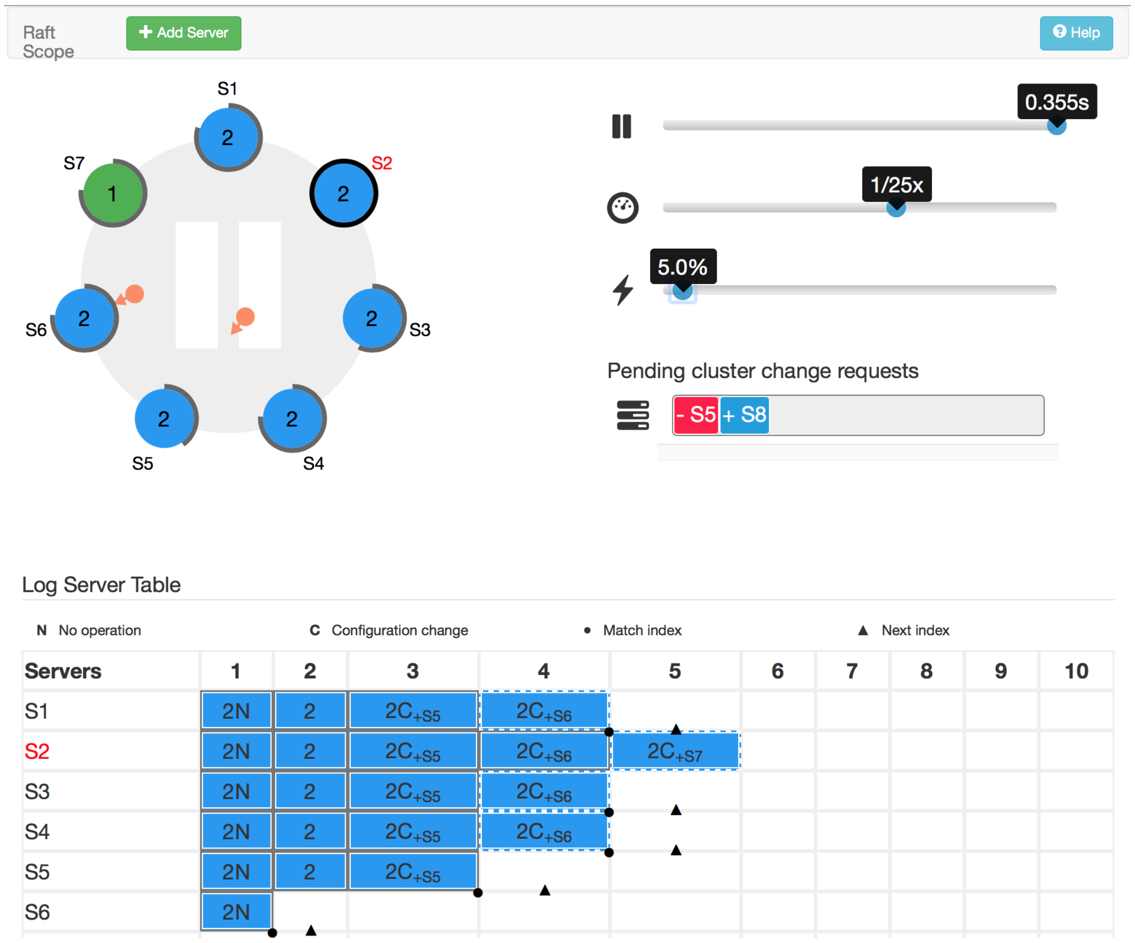
References

[1] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, 2014.

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

[3] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[4] Tushar D Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on*



0.355s

1/25x

5.0%

Pending cluster change requests

- S5 + S8

Log Server Table

N No operation · C Configuration change · ● Match index · ▲ Next index

Servers	1	2	3	4	5	6	7	8	9	10
S1	2N	2	2C$_{+S5}$	2C$_{+S6}$ ●	▲					
S2	2N	2	2C$_{+S5}$	2C$_{+S6}$	2C$_{+S7}$					
S3	2N	2	2C$_{+S5}$	2C$_{+S6}$ ●	▲					
S4	2N	2	2C$_{+S5}$	2C$_{+S6}$ ●	▲					
S5	2N	2	2C$_{+S5}$ ●	▲						
S6	2N ●	▲								

Figure 2: Extended

Principles of distributed computing, pages 398–407. ACM, 2007.

[5] Diego Ongaro. *Consensus: Bridging theory and practice*. PhD thesis, STANFORD UNIVERSITY, 2014.

[6] Diego ongardie Ongaro. Super hacky visualization of raft, 2014.

[7] Ongaro. Bug in single-server membership changes, 2015.