TITLE

David McLaren DMCLAREN@STANFORD.EDU
Elizabeth Walkup EWALKUP@STANFORD.EDU
Patrick Harvey P1HARVEY@STANFORD.EDU
Computer Science Department, Stanford University

Abstract

ABSTRACT

1. Introduction

basic description of system (maybe 1/2 page)

2. Simulating Partitions

concepts for simulating partitions, sniffing packets - iptables and packet redirection - sniffing with libtins

3. System Overview

discussion of different modules in system, and how we support adaptation for different raft implementations

- test driver (glue that holds everything together) - setup scripting - the monitor/sniffer module - the Client API – API functions to read/write using a specific Raft implementation – system-wide API also getting tucked in here – (The Client API will probably be renamed to something else to better reflect its meaning– something like an "Adaptation Layer") - teardown, if it's worth it

3.1. RaftMonitor

The checker monitors and responds to messages passed internally among the RAFT cluster's nodes by acting as a proxy or middleman to all of the cluster's internal communications. The IP address and port presented to the rest of the cluster for a RAFT node is instead one assigned to the monitor. However, on setup this should be completely transparent to the RAFT cluster, since the default behavior of the monitor is just to log packets, rewrite their destination to the appropriate node based on the address and/or port the monitor received the packet on, and resend the packet. Thus when non-invasive testing is being performed, the cluster should operate in an entirely normal manner unaffected by the monitor's presence.

For this to be the case it is important that any processing the monitor performs on the packet not lengthen too much the time required for the packet to reach the RAFT node it is intended for, lest the monitor interfere with RAFT's heartbeat messages and thus affect the cluster's operation. However, this does seem a manageable constraint: since the checking program simulates a cluster within a single machine, round-trip times for packet transmission are low (on the order of 0.05-0.2 milliseconds). Recommended settings for RAFT election timeout are in the low hundreds of milliseconds (Ontaro & Ousterhout, 2013). Simple logging, mangling and resending of packets, then, is unlikely to cause an unexpected timeout under this sort of mode of operation. However, more expensive actions such as killing one of the RAFT node processes could be concerning given this time constraint; actions that are too expensive are placed in a separate thread. For example, the action to kill a node will consist of creating a separate thread tasked with killing the node's RAFT process, and the monitor will begin dropping all packets to and from that node in the meantime.

3.2. Future Work

- plans for additional checkers for more raft implementations - more functionality in monitors? delaying or blocking packets

4. Tests

discussion of tests supported by the system: - basic sanity check (client does op expected to succeed) - multiple clients writing to file system - failure and restart of raft nodes - partitions

4.1. Future Work

additional tests we plan to do or didn't get to

This document is part of a programming assignment for CS 244B: Distributed Systems, Stanford University (Autumn 2014) and was created on December 4, 2014.

References

Ontaro, Diego and Ousterhout, John. In search of an understandable consensus algorithm (extended version). 2013.