Raft Performance Evaluation

Suman Nakshatri
University of Waterloo

Harsimran Pabla
University of Waterloo

ABSTRACT

We present a exhaustive performance measurements of original Raft open source implementation (cite lograft) under range of workloads and failure scenarios. We tried to justify the performance observed to the design decision made by Raft protocol. In this paper, we describe the set of experiments, methodology and results which we observed.

1. INTRODUCTION

Consensus is a fundamental problem of fault tolerant distributed systems. It is agreement on a single value between multiple servers under any failure model. In this paper, when we refer to failure model, it is non-byzantine failures only. One of the original solutions to consensus problem was Paxos [cite paxos] by Leslie Lamport. However, it is considered to be non-trivial and complicated algorithm especially when it comes to implementing it in real system. Our topic of discussion in this report is performance evaluation of similar consensus protocol, Raft. It must be however noted that classical Paxos itself has many performance flaws. Such flaws were fixed in follow up papers such as Multi Paxos, E Paxos etc. Goal of this exercise is to benchmark Raft protocol and see if we observe any potential performance problem. Before we go into the methodology of our tests, here is quick summary of Raft protocol.

Raft protocol involves strong form of leadership. Log entries only flow from the leader to other servers in the cluster. This simplifies the management of replicated log and presumably makes it easier to understand the protocol. This means we now need to have leader election, which is again a consensus problem. Raft however uses randomized timers to elect leader. Conflicts in leader elections are resolved by randomized timeouts. In practice, it works without having extra overhead of consensus agreement on leader.

Write requests under normal operation works as follows : Once a leader has been elected, it begins servicing client requests. Each client request contains a command to be executed by the replicated state machines. The leader appends the command to its log as a new entry, then issues RPCs in parallel to each of the other servers to replicate the entry. When the entry has been safely replicated, the leader applies the entry to its state machine and returns the result of that execution to the client.

Read only requests can be handled without writing anything to the log. However, before responding to read request, leader needs to make sure that it still is the leader. Raft handles this by having the leader exchange heartbeat messages with a majority of the cluster before responding to read-only requests.

Most of the performance evaluation is based on read and write request time taken by the client, under various workload scenarios. It is important to note that all the requests are serviced by the leader only. This is necessary to maintain safety requirements. Also, clients do not maintain any active information on who is the current leader of Raft cluster. So, there needs to be client communication protocol to resolve leader id.

2. METHODOLOGY

3. EVALUATION

3.1 Experiment Setup

3.2 Results

4. CONCLUSIONS

5. ACKNOWLEDGMENTS

We would like to thank Prof. Samer Al-Kiswany for his guidance during our brainstorming and initial project discussions.

6. REFERENCES

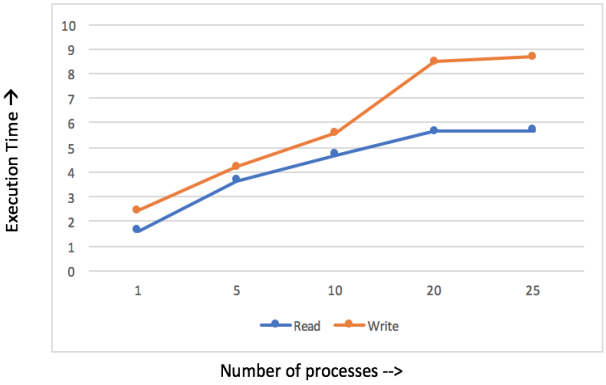

Figure 1: Execution time (in ms) of 1000 READ and WRITE requests generated from each process running in a single client. These requests perform action on the same file.

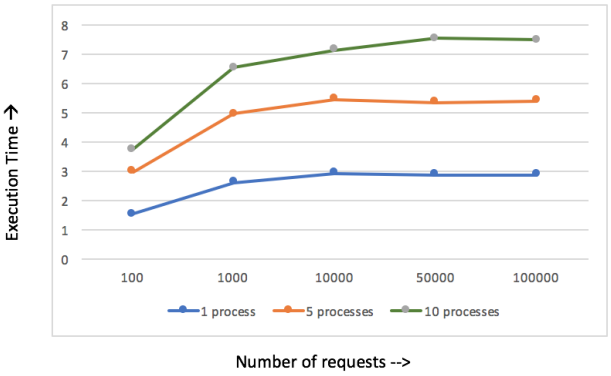

Figure 2: Execution time (in ms) of increasing number of READ requests generated from each process running in 5 clients. These requests perform action on the same file.

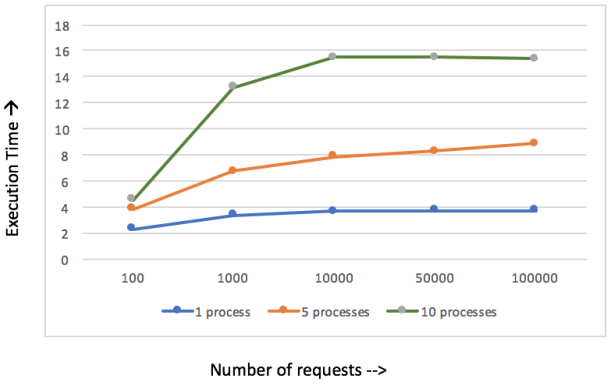

Figure 3: Execution time (in ms) of increasing number of WRITE requests generated from each process running in 5 clients. These requests perform action on the same file.

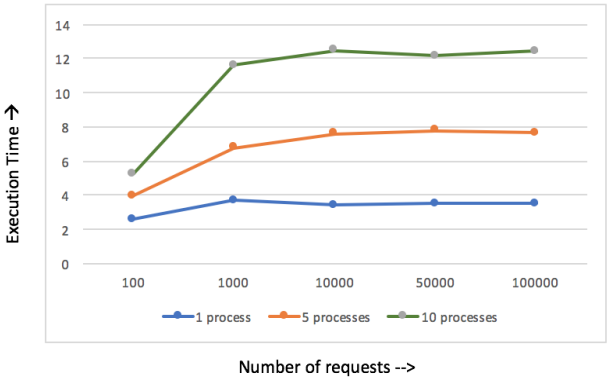

Figure 4: Execution time (in ms) of equal and increasing number of READ and WRITE requests generated from each process running in 5 clients. These requests perform action on the same file.

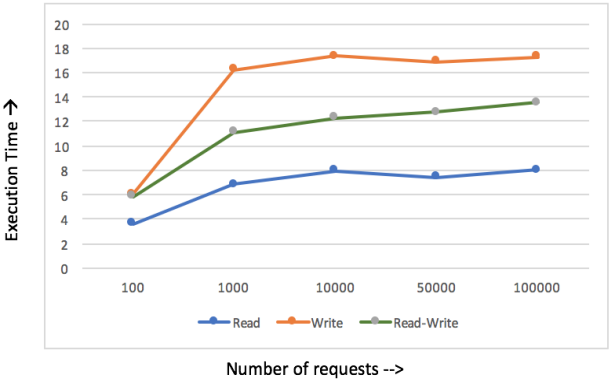

Figure 5: Execution time (in ms) of increasing number of READ and WRITE requests generated from each process running in 5 clients. These requests perform action on different files.