Evaluation of Paxos Cluster Performance

Aakash Sharma
Department of Computer Science
UiT – The Arctic University of Norway
Tromsø, Norway
aakash.sharma@uit.no

***Abstract*—This report provides an evaluation study of Paxos algorithm from a system's perspective. The evaluation is based on an implementation of Paxos [3] in python. Various parameters were modified to understand different behaviors of reaching consensus using the Paxos algorithm. Various evaluation parameters are presented and the results are discussed in this report. The evaluation was performed using a single machine running the *replica* nodes and the *leader* nodes.**

***Index Terms*—Distributed System, Consensus, Fault Tolerance**

I. Introduction

Paxos was proposed by Leslie Lamport in his paper "Part-time Parliament" [1]. The algorithm provides a way to reach consensus in a system with multiple participating nodes. The working principle behind the Paxos algorithm is that the proposer node receives a *propose*, it sends a prepare message to all of the *acceptor* nodes which contains the Paxos iteration number and the proposal number. The acceptor node on receiving such message can reply with a rejection or with a *promise*. A *promise* is a guarantee that the *acceptor* node will not accept another proposal from any proposer node with a lower proposal number. If the proposer receives promise messages from a majority of the acceptor nodes, it can move forward with the actual proposal message. The proposer node, then sends a message containing Paxos iteration number, the proposal number and the proposal value to the acceptors. If the proposer node receives Accept from a majority of acceptor nodes, the iteration will end resulting in the network reaching a consensus on the value.

Although the concept seems easy, it has been considered complex to implement. In this report, we are not focused on explaining much of Paxos algorithm. We are interested in evaluating the performance of Paxos cluster, in a changing setup. The goal is to evaluate and understand the throughput of a Paxos cluster. We do this by varying few of the key components in a Paxos cluster, which is explained in the next part.

II. Experiments and Results

We will now describe the experimental setup for evaluating performance of Paxos cluster. For the evaluation, we used the Paxos implementation by van Renesse & Altinbuken [3]. The code is available online![1]. The evaluation is designed to determine the scalability of the Paxos algorithm by increasing the amount of participating nodes in the consensus. The throughput is measured as the amount of consensus reached per second. It can also be seen as the commands that are performed after reaching consensus. For the evaluation, multiple sets of the test runs are evaluated in order to achieve average performance indicators. For this study, we have used 4 test runs.

A. Hardware Platform

The evaluation is done using a commodity hardware, Apple MacBook Pro 2017. The system used for evaluation is powered by an Intel 7^{th} generation 7700HQ processor clocked at a frequency of 2.8 GHz. The system has 16 GB of RAM and running MacOS version 10.14.3. The implementation is running on Python 2.7.10 interpreter and using matplotlib [4] fir generating plots. We have written a running script for evaluating, measuring and plotting throughput of the cluster.

```
python paxostest.py --size=11 --threshold=2
```

where, *size* defines the ending parameter for how bigger the cluster should get & *threshold* defines the number of messages to be exchanged. The system was only running basic OS services and no other heavy user application at the time of evaluation. We run a Paxos cluster which includes set of replicas, acceptor and leader nodes based on our configuration. To further limit any suspended iteration of the Paxos code, we ensure proper cleanup of the Paxos cluster at shutdown.

B. Evaluation

The existing code for Paxos provided by van Renesse & Altinbuken [3] using a number of variables for different configurations. Replicas maintain application states and receive interact with clients. Acceptor maintains the fault tolerant memory of Paxos cluster. Leader serializes requests and responds to replicas. For tolerating f faults, there is a minimum requirement for the number of *Replica*, *Acceptor* and *Leader* nodes in a Paxos cluster. There must be $f + 1$ Replicas, along with $2f + 1$ Acceptor and $f+1$ Leader nodes to tolerate f failures. For example, when we configure our

[1] https://github.com/denizalti/paxosmmc

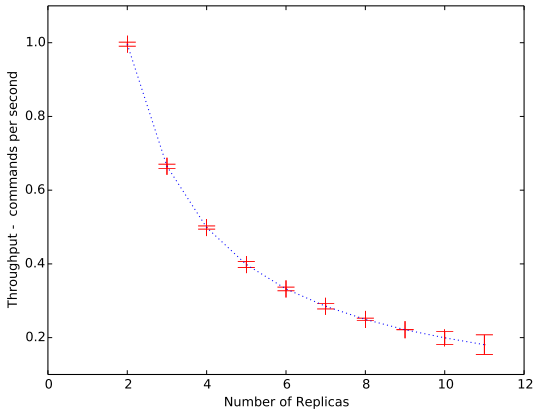


Fig. 1: Throughput of Paxos with 3 Acceptor and 2 Leader nodes for tolerating 1 failure.

system to test the initial configuration with only one failure, we provide 3 Acceptor nodes along with 2 Leader nodes. We vary the number of replica nodes from 2 until 11 and measure throughput of the system. This is shown in Figure 1.

While keeping this initial configuration, we varied the number of acceptor (NACCEPTORS) nodes. We increased the acceptor nodes from 3 to 6 and plotted the throughput of the system. Figure 2 shows the results that we obtained. As evident from earlier Figure 1, the throughput of the cluster falls as the number of replica nodes increase in a Paxos cluster. The effects of varying amount of acceptor nodes are negligible on the system. However, around the tail of the graph, we can observe that there's a slight decrease in performance with higher replicas and high number of acceptor nodes. Even though there is a very small change, it can be attributed to the system's varying load.

Furthermore, we then varied the number of leaders in the configuration to obtain the changes in Paxos cluster. Similar to earlier steps, we kept the initial configuration for tolerating one failure and increased the number of leader nodes (NLEADERS) in the Paxos cluster. The results of this evaluation are plotted in Figure 3. As expected, with the increasing number of consensus nodes in the network, the throughput declines. The results are very similar for both experiments. There are variations in the throughput shown by the error bars, but the difference is insignificant.

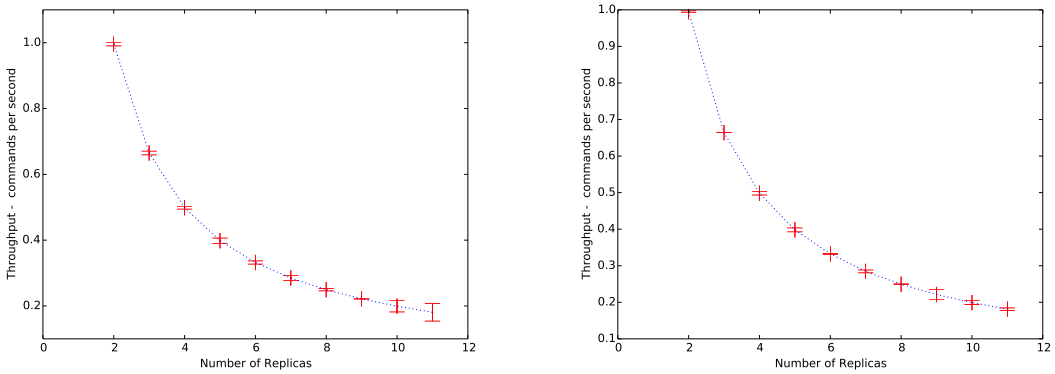
The results are in line with the expectation that the proposer node has to wait for sufficient promise messages to form a majority in the system. Similarly, before committing the proposer again waits for enough Accept messages from the participating nodes to obtain a majority. The amount of messages induces increased time to reach consensus and thereby reducing the overall throughput of the system. Work by Santos & Schiper [2] has suggested batching and pipelining to increase the overall throughput of a Paxos based system. However, this implementation is not using that. Additionally, this evaluation was done on a laptop and on a synthetic load. Significantly higher throughput can be reached in a cluster environment with multiple clients submitting their messages. However, that is not in the scope of this report.

III. Conclusion

In this work, we have studied the scalability of the Paxos algorithm. The result shows reduced throughput with an increasing number of participating nodes. Also, for the simple configuration setup for one failure tolerance, the varying number of acceptor or leader nodes has almost no effect in our setup of a single machine.

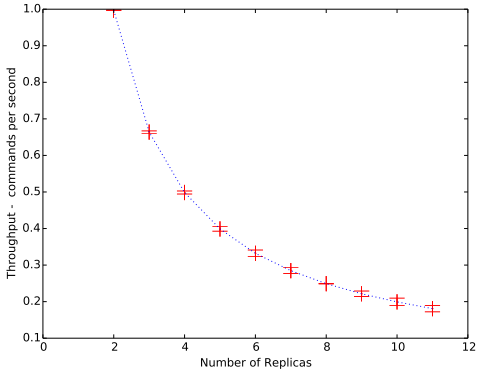
References

[1] Lamport, Leslie. "The part-time parliament." ACM Transactions on Computer systems 16.2 (1998): 133-169.
[2] Santos, Nuno, and Andr Schiper. "Tuning paxos for high-throughput with batching and pipelining." International Conference on Distributed Computing and Networking. Springer, Berlin, Heidelberg, 2012.
[3] Van Renesse, Robbert, and Deniz Altinbuken. "Paxos Made Moderately Complex." ACM Comput. Surv. 47.3 (2015): 42-1.
[4] Hunter, John D. "Matplotlib: A 2D graphics environment." Computing in science & engineering 9.3 (2007): 90.

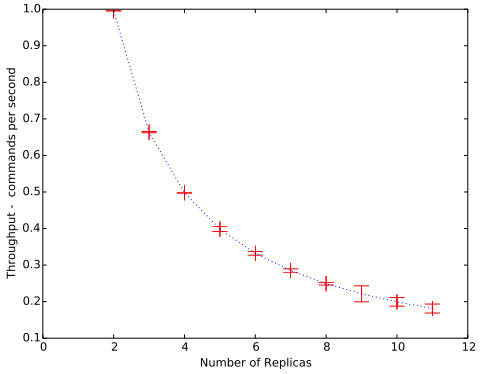


(a)

(b)

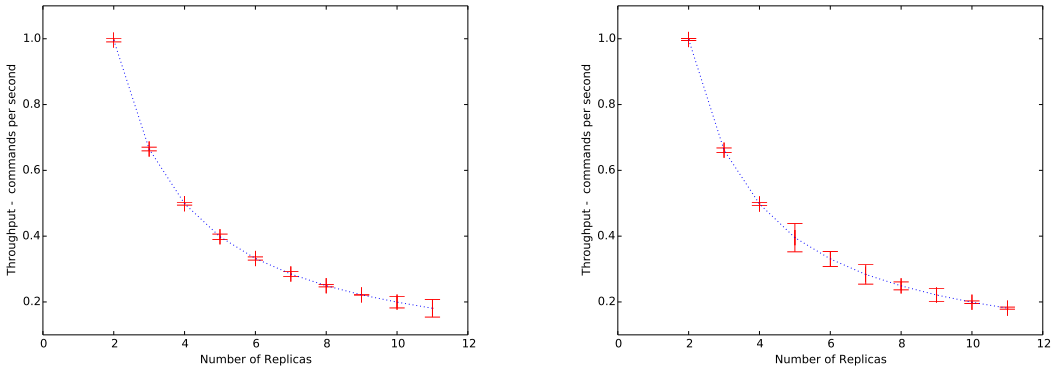


(c)



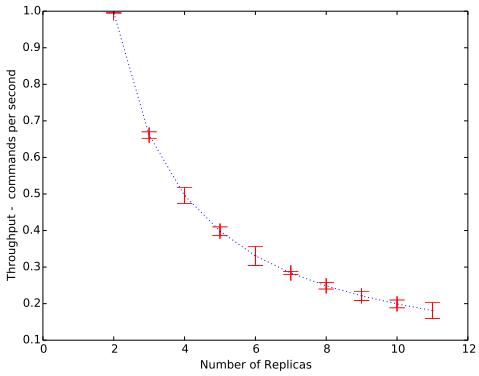
(d)

Fig. 2: Throughput of the Paxos cluster with NREQUESTS = 2, NLEADERS = 2 with varying NACCEPTORS = 3 (a), 4 (b), 5 (c) & 6 (d) and number of replicas.

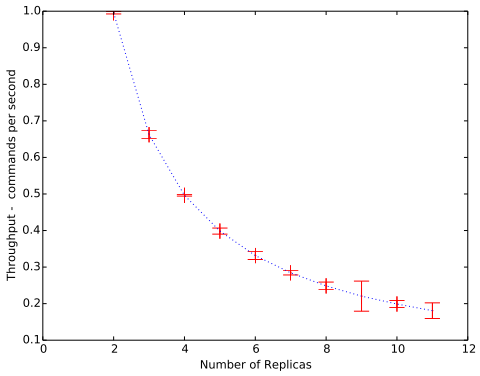


(a)

(b)



(c)



(d)

Fig. 3: Throughput of the Paxos cluster with NREQUESTS = 2, NACCEPTORS = 3 with varying NLEADERS = 2 (a), 3 (b), 4 (c) & 5 (d) and number of replicas.