Performance of the Paxos Algorithm with concurrent clients

Nikolai Magnussen

***Abstract*—Paxos is an algorithm for reaching consensus in a distributed system, providing both guarantees regarding liveness and durability[1]. This paper will explore an implementation of Paxos and it's performance characteristics as the number of concurrent clients increase.**

I. Paxos Consensus Algorithm

Paxos was created by Leslie Lamport in 1989 and submitted to *TOCS* in 1990[2], but the proposed paper was not accepted to *TOCS* until 1998[3]. Today, there are a number of Paxos variants, such as *Disk Paxos*[4] and *Fast Byzantine Paxos*[5].

In its simplest case, it consist of two phases; first a *prepare* phase which, if successful, is followed by the *accept* phase, where the proposal is accepted and subsequently stored for durability.

In this paper, we will explore the performance characteristics of an implementation by Robbert Van Renesse and Deniz Altinbuken[6]. It uses the roles of *Replica*, *Leader*, *Scout*, *Commander* and *Acceptor*, where replicas hold the state of the system, which proposals has been decided, performed and proposed. They will communicate with leaders, which through a scout performs the *prepare* phase. The scout will either get the acceptors to promise not to accept any proposals that are not newer than the one proposed by the scout, or it will get knowledge of a more recent proposal. In both cases, the scout will inform the leader about this. If a leader received the promise via it's scout, it will send a commander to perform the second phase; having the acceptors accept the proposal. Given that the acceptors accept the proposal from the commander, it will inform the replicas of the accepted proposal. Finally, the replicas are responsible for storing the decision in a non-volatile manner, for durability.

II. Experiments

The above described design and implementation of Paxos is benchmarked with an increasing number of concurrent clients. Source code for the implementation by Robbert Van Renesse and Deniz Altinbuken is available online in three versions[7]; *initial*, *backoff* and *state-reduction*. This paper will only run experiments using the *initial* version.

A. Setup

The experiment was run on an Intel i7-4700MQ CPU with 8GB DDR3 RAM under Linux. The Paxos cluster is set up with a single leader, a single replica, and three acceptors. Each of the concurrent clients will send proposals to the replica which will notify the client once its request is decided and performed. By this mechanism, the client can send a new request once the previous one was accepted. All concurrent clients act independent of all other clients.

Experimental data is provided by the replica, by it measuring the number of accepted proposals it receive from the commander. Each client will send 10 sequential requests, and the experiment is run 50 times. The data from 50 runs is ordered before the smallest and greatest values, yielding 48 data points which are used to calculate the mean and standard deviation.

B. Results

The experiment is run with between 1 and 20 concurrent clients.

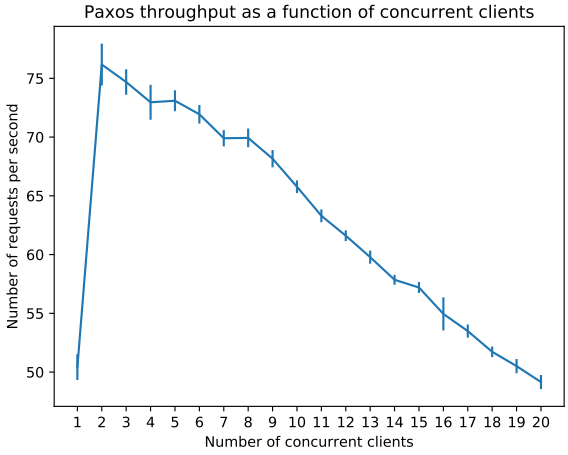

Fig. 1. Paxos throughput as a function of number of concurrent clients

Fig. 1 shows the experimental results, where the vertical lines along the graph are errorbars given by the computed standard deviation. As we can see, the throughput increase drastically from 1 to 2 clients due to them being concurrent. When the number of concurrent clients increase further, the contention also increase which explains why the performance degrades. While that is the case, the throughput with multiple concurrent clients is still higher than a single client until we reach 19 concurrent clients, which seem to yield approximately the same throughput.

III. Conclusion

The Paxos consensus algorithm seem to scale well with concurrent clients, not degrading too quickly. It scales very well from 1 to 2, and then degrades slowly as the number of concurrent clients increase.

References

[1] L. Lamport, "Paxos made simple," Dec. 2001. [Online]. Available: https://www.microsoft.com/en-us/research/publication/paxos-made-simple/.

[2] ——, (2017). The writings of leslie lamport, [Online]. Available: http://lamport.azurewebsites.net/pubs/pubs.html#lamport-paxos (visited on 02/25/2018).

[3] ——, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, May 1998, ISSN: 0734-2071. DOI: 10.1145/279227.279229. [Online]. Available: http://doi.acm.org/10.1145/279227.279229.

[4] E. Gafni and L. Lamport, "Disk paxos," *Distributed Computing 16*, vol. 16/1, May 2003. [Online]. Available: https://www.microsoft.com/en-us/research/publication/disk-paxos/.

[5] J.-P. Martin and L. Alvisi, "Fast byzantine paxos," Mar. 2012.

[6] R. Van Renesse and D. Altinbuken, "Paxos made moderately complex," *ACM Comput. Surv.*, vol. 47, no. 3, 42:1–42:36, Feb. 2015, ISSN: 0360-0300. DOI: 10.1145/2673577. [Online]. Available: http://doi.acm.org/10.1145/2673577.

[7] ——, (2015). Paxos made moderately complex, [Online]. Available: https://paxos.systems (visited on 02/24/2018).