Asymmetric Linearizable Local Reads

Myles Thiessen

Department of Computer Science
University of Toronto
mthiessen@cs.toronto.edu

1 Introduction

Linearizability is the gold standard for distributed databases because it creates the illusion that the database is run on a single machine that performs client operations one at a time. To provide linearizability, databases use a state machine replication algorithm such as Paxos or Raft. These algorithms totally order all client operations by electing a distinguished process known as the *leader* to assign each operation a unique index number. They also ensure that the database remains available despite process crashes by replicating each operation across multiple followers, i.e., processes other than the leader.

To improve the performance of these algorithms many production databases do not replicate read operations [1]. Instead, each read operation is forwarded to the leader who performs it immediately against its local copy of the database. While this is faster than replicating it, read operation latency can still be as high as a few hundred milliseconds in geo-distributed networks.

To further reduce read operation latency in geo-distributed networks, many algorithms have been proposed that enable *all* processes to perform read operations *locally*: in the best case, each process executes read operations immediately against its local copy of the database [2, 3, 5, 4]. This is a significant improvement compared to performing all read operations at the leader because no inter-process communication is required. However, in the presence of concurrent read-modify-write operations, processes may block read operations for some time. This is to ensure that processes are never performing read operations against different versions of the database simultaneously. Clearly, it is desirable to minimize the worst-case read blockage time.

2 Presentation Outline

In this presentation, we will first show that with *all* existing algorithms when deployed on commodity hardware, the worst-case read blockage time at *every* follower is approximately the *network's diameter* in terms of the maximum message delay between any two processes. We will then show that the worst-case read blockage time can be smaller than the network's diameter at *well-located* followers, i.e., those that are close to the leader or those that are close to the network's center. We do so by presenting two algorithms named *Pairwise-Leader (PL)* and *Pairwise-All (PA)*: with PL, the worst-case read blockage time at a process is approximately the round-trip message delay between it and the leader; with PA, the worst-case read blockage time at a process is approximately its *eccentricity*, i.e., the maximum message delay between it and all other processes.

To illustrate the above, consider a simple three-region AWS deployment spanning the Montreal (M), North Virginia (NV), and North California (NC) regions where the leader is located in Montreal. The diameter of this network is 40 ms and so, with all existing algorithms, the worst-case read blockage time at both NV and NC is 40 ms. In contrast, with PL, the worst-case read blockage time is 16 ms at NV and 80 ms at NC, because the round-trip message delays from NV and NC to the leader located in M are 16 ms and 80 ms, respectively. With PA, the worst-case read blockage time is 32 ms at NV and 40 ms at NC since these are the eccentricities of NV and NC, respectively. This simple example showcases the fundamental difference between existing algorithms and our Pairwise algorithms: *while existing algorithms do not leverage network asymmetry, our Pairwise algorithms do.*

References

[1] Chandra et al. Paxos made live: an engineering perspective. In *PODC*, 2007.

[2] Chandra et al. An algorithm for replicated objects with efficient reads. In *PODC*, 2016.

[3] Katsarakis et al. Hermes: A fast, fault-tolerant and linearizable replication protocol. In *ASPLOS*, 2020.

[4] Moraru et al. Paxos quorum leases: Fast reads without sacrificing writes. In *SOCC*, 2014.

[5] VanBenschoten et al. Enabling the next generation of multi-region applications with cockroachdb. In *SIGMOD*, 2022.