Study Report

Authors

State machine replication (SMR) is a powerful fault-tolerance concept [8]. It runs replicas of same program on multiple nodes. To keep the replicas consistent, it invokes a distributed consensus protocol (typically PAXOS [6, 7, 10]) to ensure that a quorum (typically majority) of the replicas agree on the input request sequence. SMR is proven to tolerate various failure scenarios, like network partition and packet loss.

The fault-tolerant benefit of SMR makes it particularly an attractive high-availability service for general server programs. Unfortunately, despite much effort, existing SMR systems are still hard to deploy, mainly due to two problems.

The first problem is performance. The consensus latency of PAXOS protocols is notoriously high and unscalable [1, 4], incurring high performance overhead for server programs. A main reason is that messages of traditional TCP or UDP-based PAXOS protocols have to go through software network layers in OS kernels. For efficiency, PAXOS protocols typically take the Multi-Paxos approach [6]: it assigns one replica as the "leader" to invoke consensus requests, and the other replicas as "backups" to agree on requests. To agree on an input, at least one round-trip time (RTT) is required between the leader and a backup. Given that a ping RTT in LAN typically takes hundreds of μs, and that the request processing time of key-value store servers (e.g., Redis) is at most hundreds of μs, existing PAXOS protocols incur high overhead in the response time of server programs.

Worse, the consensus latency of extant consensus protocols is often *scale-limited*: it increases drastically when the number of concurrent requests or replicas increases. For instance, Scatter [4] shows that the consensus latency of its PAXOS protocol increases by 1.6X when the number of replicas increases from 3 to 9.

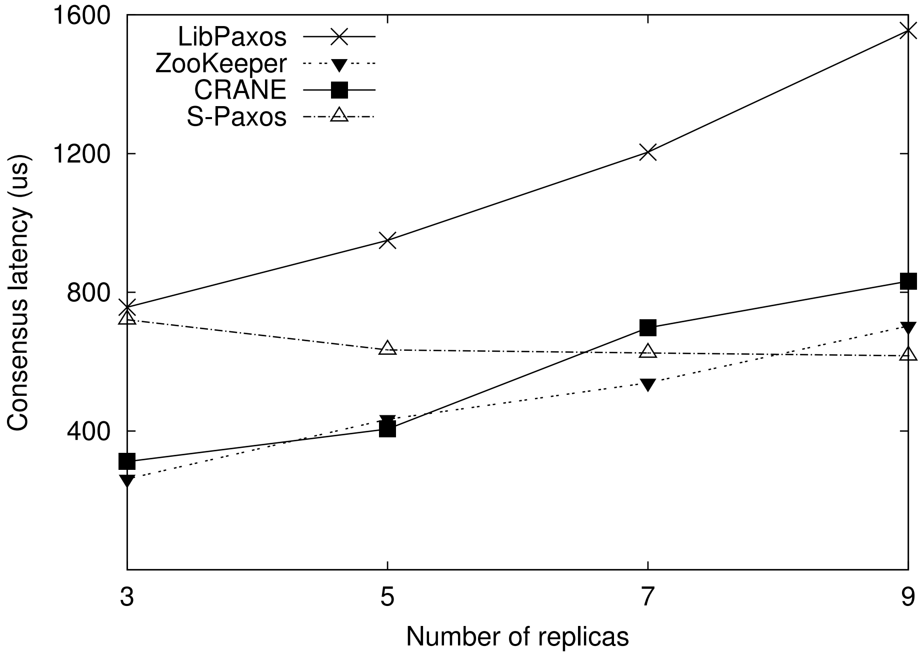

Figure 1: ***Consensus latency of four existing consensus protocols.*** **All four protocols ran a client with 24 concurrent connections. The Y axis is broken to fit in all protocols.**

To quantify this bottleneck, we evaluated four traditional consensus protocols [1, 2, 3, 9] on 24-core hosts with

40Gbps network, and we spawned 24 concurrent consensus connections. As shown in Figure 1, when changing the replica group size from 3 to 9, although network and CPUs were not saturated, the consensus latency of 3 protocols drastically increased by 105.4% to 168.3%, and 36.5% to 63.7% of the increase was in OS kernel. When only one consensus connection was spawned, the latency increase on the number of replicas was more gentle.

This evaluation shows that both the number of concurrent requests and replicas make consensus latency increase drastically. As modern server programs tend to support more concurrent client connections, and advanced SMR systems tend to deploy more replicas (e.g., Azure [5] deploys seven or nine replicas) to support both replica failures and upgrades, the limited scalability in extant consensus protocols becomes even more pronounced.

Due to the poor performance of existing PAXOS protocols, server programs often have to either sacrifice performance or implement weak consistency guarantees instead of PAXOS. For example, for performance, Redis uses a weaker consistency protocol (eventual consistency) for fault tolerance.

The second problem is that most existing SMR systems are not designed to support unmodified server programs. To utilize existing SMR services, developers often have to rewrite their code to orchestrate server programs into the narrowly defined interfaces provided by these SMR systems. For instance, to leverage ZooKeeper [1], developers have to shoehorn their programs into the file IO interface defined by ZooKeeper.

References

[1] ZooKeeper. `https://zookeeper.apache.org/`.

[2] M. Biely, Z. Milosevic, N. Santos, and A. Schiper. S-paxos: Offloading the leader for high throughput state machine replication. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 111–120. IEEE, 2012.

[3] H. Cui, R. Gu, C. Liu, and J. Yang. Paxos made transparent. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, Oct. 2015.

[4] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Scalable consistency in scatter. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 15–28. ACM, 2011.

[5] S. Krishnan. *Programming Windows Azure: Programming the Microsoft Cloud.* " O'Reilly Media, Inc.", 2010.

[6] L. Lamport. Paxos made simple. `http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf`.

[7] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[8] D. Mazieres. Paxos made practical. Technical report, Technical report, 2007. http://www. scs. stanford. edu/dm/home/papers, 2007.

[9] M. Primi. LibPaxos. `http://libpaxos.sourceforge.net/`.

[10] R. Van Renesse and D. Altinbuken. Paxos made moderately complex. *ACM Computing Surveys (CSUR)*, 47(3):42:1–42:36, 2015.