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 three problems.

High consensus latency. The consensus latency of traditional PAXOS protocols is notoriously high, 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 OS kernel. 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.

Poor scalability. The consensus latency of extant consensus protocols is often *scale-limited*: it increases drastically when the number of concurrent requests or replicas increases [1, 4].

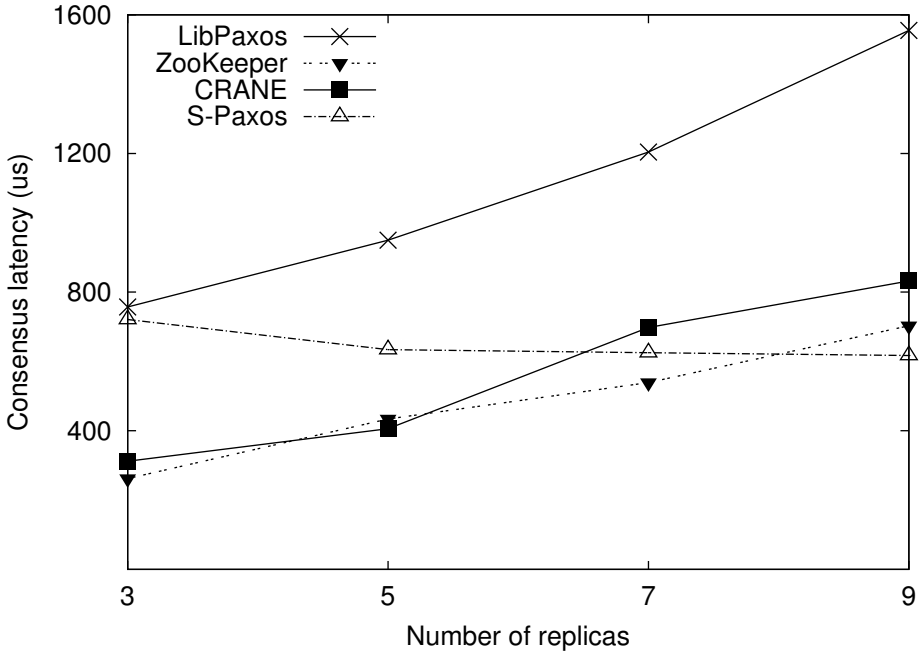

Figure 1: ***Consensus latency of four existing consensus protocols.*** **All four protocols ran a client with 24 concurrent connections.**

To quantify this problem, we evaluated four traditional consensus protocols [1, 2, 3, 9] on 24-core hosts with 40Gbps

network. For each protocol, we spawned 24 concurrent consensus connections. As shown in Figure 1, when changing the replica group size from 3 to 9, the consensus latency of three traditional protocols increased almost linearly to the number of replicas except S-Paxos. S-Paxos batches requests from replicas and invokes consensus when the batch is full. More replicas can take shorter time to form a batch, so S-Paxos incurred a slightly better consensus latency with more replicas. Nevertheless, its latency was always over 600μs.

To find scalability bottlenecks in traditional protocols, we used only one client connection and broke down their consensus latency on leader (Table 1). From 3 to 9 replicas, the consensus latency (the "Latency" column) of these protocols increased more gently than that on 24 concurrent connections. For instance, when the number of replicas increased from three to nine, ZooKeeper latency increased by 30.3% with one connection; this latency increased by 168.3% with 24 connections (Figure 1). This indicates that concurrent consensus requests are the major scalability bottleneck for these protocols.

Table 1: ***Performance breakdown of traditional protocols on leader with only one connection.*** **The "Proto-#Rep" column is the protocol name and replica group size; "Latency" is the consensus latency; "First" is the latency of leader's first received consensus reply; "Major" is the latency of leader's consensus; "Process" is leader's time spent in processing all replies; and "Sys" is leader's time spent in systems (OS kernel, network stacks, and JVM) between the "First" and "Major" reply. Times are in μs.**

Proto-#Rep	Latency	First	Major	Process	Sys
libPaxos-3	81.6	74.0	81.6	2.5	5.1
libPaxos-9	208.3	145.0	208.3	12.0	51.3
ZooKeeper-3	99.0	67.0	99.0	0.84	31.2
ZooKeeper-9	129.0	76.0	128.0	3.6	49.4
CRANE-3	78.0	69.0	69.0	13.0	0
CRANE-9	148.0	83.0	142.0	30.0	35.0
S-Paxos-3	865.1	846.0	846.0	20.0	0
S-Paxos-9	739.1	545.0	731.0	35.0	159.1

Specifically, three protocols had scalable latency on the arrival of their first consensus reply (the "First" column), which implies that network is not saturated. libPaxos is an exception because its two-round protocol consumed much bandwidth. However, on the leader, there is a big gap between the arrival of the first consensus reply and the "majority" reply (the "Major" column). Given that the replies' CPU processing time was small (the "Process" column), we can see that various systems layers, including OS kernel, network libraries, and language runtimes (e.g., JVM), are another major scalable bottleneck (the "Sys" column).

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.

Hard to use. 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.