1 Evaluation

Name of system? RepQueue-RSS, ReqQueue-Strict
Our experimental evaluation answers:

- How does our system, built specifically to achieve MD-RSS, compare with multi-dispatch safe usage of Spanner-RSS in terms of end-to-end latency? Target is single data-center evaluation, and geo-replicated setting is a stretch.
- How does our system throughput scale as the number of shards increases, and how does this compare with Spanner-RSS with multi-dispatch interface.

Due to time constraint, we do not aim for a true apples-to-apples comparison. In particular, we reuse the implementation of the Spanner-RSS with MD baseline and do not recreate it in the same framework as our system. We anticipate that the protocol-level difference should be significant enough such that any implementation induced performance differences will not affect the anticipated results.

1.1 Implementation

Spanner-RSS implementation is taken as-is from paper. Additional client interface also written in the same framework. RepQueue is written from scratch in Rust. Both are in-memory transactional key-value stores.

1.2 Workloads

We reuse Retwis workload setup from the Spanner-RSS evaluation. We can adjust the Zipfian skew to simulate more or less contention. The notion of client sessions carry over very naturally to RepQueue, since it also has a notion of client session. Will not include neither dependent transactions nor conditional writes for simplicity.

1.3 Configuration

AWS EC2? Emulab?
Single datacenter, all within single availability zone.
Geo-replicated, spread across three availability zones (all within US).
Replication factor is 3. Try to have the optimal case where all leaders and tail all in the same datacenter, although this may not be possible depending on implementation.
Each experiment done 5 times at a minimum.

1.4 Experiments

The overall key point is that in order to guarantee multi-dispatch correctness, Spanner-RSS clients must create a unique sequence number key/value that they must validate and increment on every transaction. The 2PL read $\rightarrow$ write transition causes aborts and effectively serializes transactions on a per-client basis. On the other hand, RepQueue-RSS has this validation as a part of its protocol and is thus able to "pipeline" client operations internally.

1.4.1 End-to-End Latency Experiments

Single client, write-only workload:
Take the median latency over the trials. Error bars can be variance, min/max, or percentile.

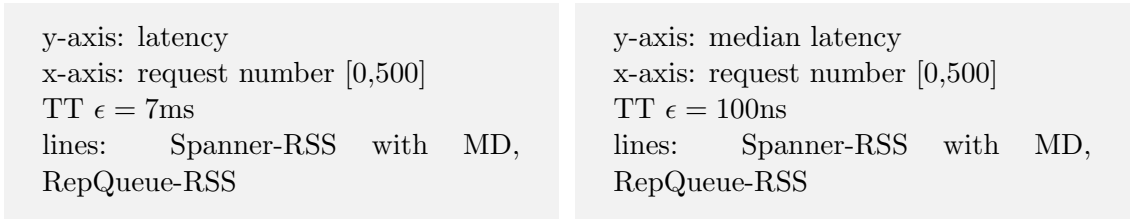

Figure 1: Write-only latency in single data center with moderate contention

- Key data point on this graph is the difference between first and last request. This is the total end-to-end latency.
- Here, we expect linear behavior for both systems. Due to the overhead of commit wait, both the slope and intercept of the line corresponding to Spanner-RSS with MD will be greater than that of RepQueue-RSS.
- Test with ϵ given in original paper (left) and in Sundial (right).
- Should discuss the fact that even with highly synchronized clocks, the serialization caused by the MD interface results in a larger slope than RepQueue-RSS. Thus, given enough concurrent requests, end-to-end latency for RepQueue-RSS is less than that of Spanner-RSS with MD.

Multiple clients, write-only workload
03/17/2023: Replace this with end-to-end latency/throughput graph with #clients as independent variable

Take median for each client. Then, use error bars over clients to give a better idea of distribution. Assume each node capable of ≈ 100000 req/s throughput.

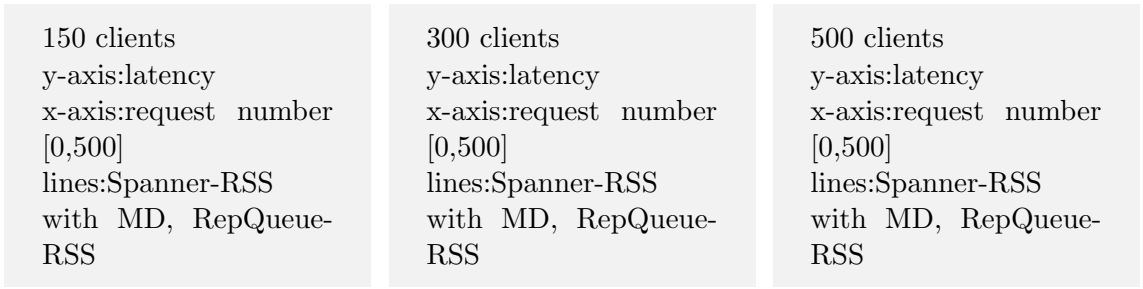

Figure 2: Write-only latency in single data center, moderate contention, and multiple clients

- Given a sufficiently large amount of clients and associated concurrent requests, we would expect Spanner-RSS with MD to be more performant. The reason is that client writes are spread out amongst the shards in Spanner-RSS, whereas RepQueue uses a single chain of nodes for serialization.
- Expect a breakeven point where the overhead due to per-client serialization is less than the time spent in the RepQueue-RSS chain behind other client's requests.
- Breakeven point should be somewhere above the max throughput for a single node. The figures should illustrate this phenomenon.

Single client, read-write workload
Want to show comparable read performance. Disparity in write performance causes Spanner-RSS with MD to perform worse in this case.

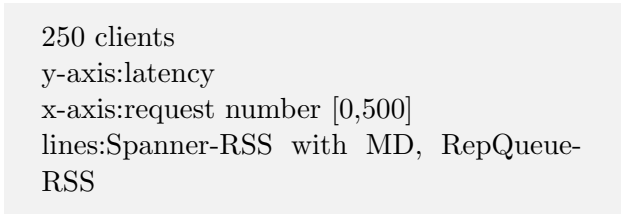

Figure 3: Read-write latency in single data center, moderate contention, single client

- Both lines should be "step" functions. Batches of reads need to wait for any preceding writes to complete.
- For clear comparison, can constrain workload to deterministically have 1 write every 50 or so reads.
- Both Spanner-RSS with MD and RepQueue-RSS admit optimization where prefix of reads not interfering with write can be immediately serviced. Result should stay the same if optimization implemented on both systems.

Multiple clients, read-write workload
Mirrors write-only case. Graph and experiment configuration should be the same.

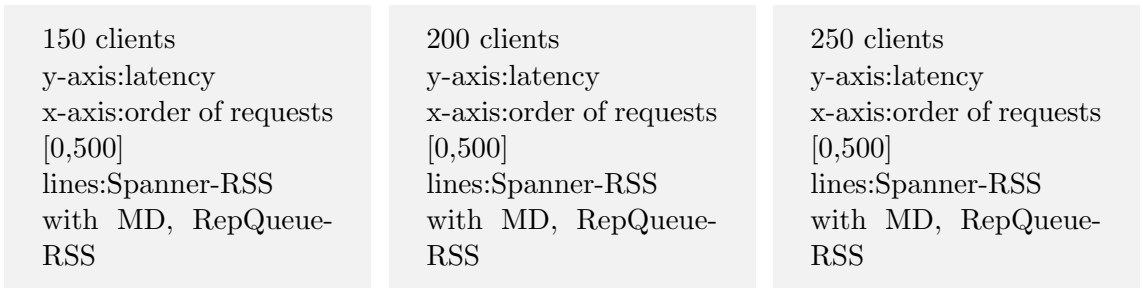

Figure 4: Read-write latency in single data center, moderate contention, and multiple clients

1.4.2 Throughput and Scale Out Experiments

Contention
Want to show that per-client throughput for Spanner-RSS with MD will always be bounded above by one transaction per client-system RTT.

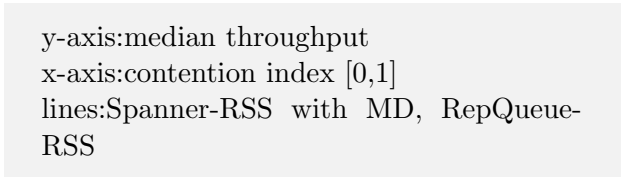

Figure 5: Throughput versus contention index in single data center, single client

- Echoes the Calvin argument for determinizing transactions – do not have to worry about lock contention.

- RepQueue-RSS gradually curves down to meet Spanner-RSS with MD line at contention index 1.

Throughput-latency
The number of clients determines the maximum total system throughput for Spanner-RSS with MD. In particular, adding more shards to Spanner-RSS with MD will not bring about an increase in throughput.

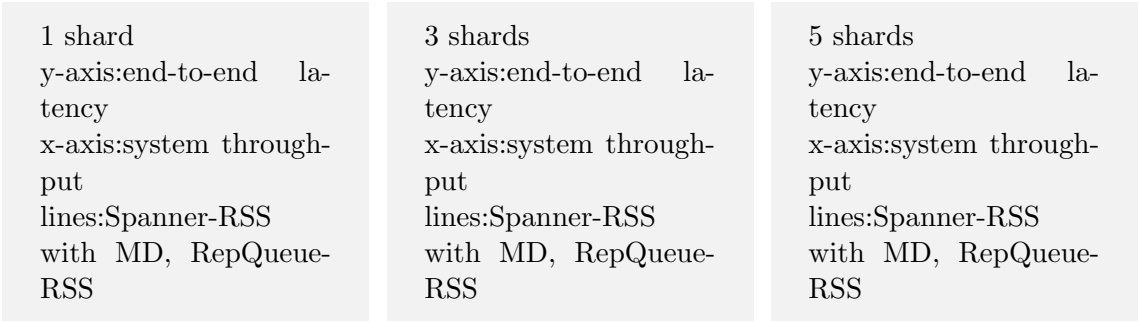

Figure 6: Throughput versus latency in single data center, 100 clients

- Adding more shards also does not increase throughput for RepQueue-RSS (actually can decrease throughput).
- Possible future direction to explore new design or modification of RepQueue-RSS that allows easy scalability.