Assignment #2: Testing Reliability of Quorum Systems

Shuyang Ji (sji15)

1. Select a Quorum System

Q1: What is your quorum system of choice?

TiKV, a distrbuted key-value stroage that relies on the Multi-Raft provided by Raftstore for data replicating and strong consistency, and RocksDB for persistence.

2. Run The Quorum System and Measure The Baseline Performance

Q2: Please describe your configuration.

The experiments are run on a Ubuntu VM within a MacOS with 8 cores CPU (M1 ARM) and 8 GB of memory.

The psedo-distributed setup will contain 1 PD (Placement driver, cluster manager of TiKV) node and 3 TiKV nodes.

(3-node is discouraged but due to limitation of computational resources, it is nearly impossible to accomodate more nodes.)

I use go-ycsb (the Golang port of YCSB, with nature support for TiKV) as the client to measure performance.

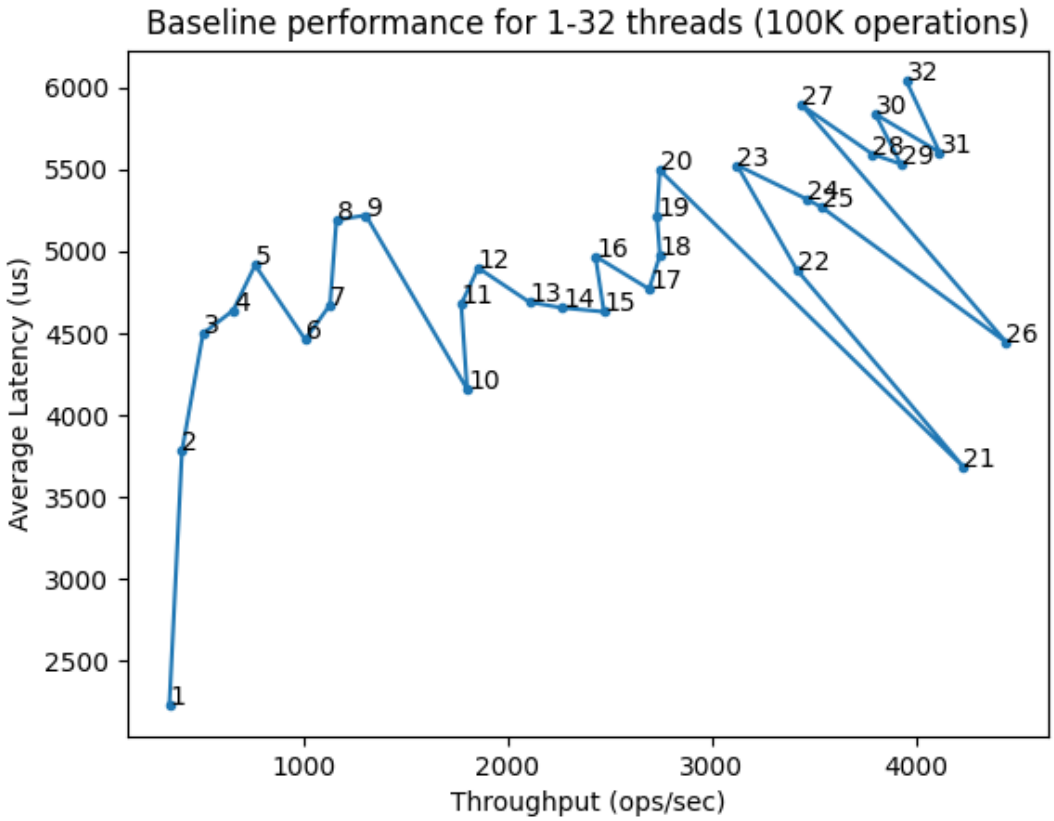
Q3: What is your client workload?

I use a client workload of 10000 record count, 100000 operation count and 0.5/0.5 read/update proportion. Update operations are used for plotting and analysis.

Based on an experiment results of 1-32 threads (see Q4), 26 hits the max throughput. Due to the nondeterminism of concurrency and CPU instruction reordering, the thread number that hits max throughput may not always be the same but we will just use 26 throughout this experiment.

I conclude the bottleneck is the CPU as it exceeded the 100% CPU utilization.

Q4: What is your baseline performance? Plot the throughput-latency figure.



3. Fail-Injection Testing

Q5: How do you simulate crash, slow CPU and memory contention?

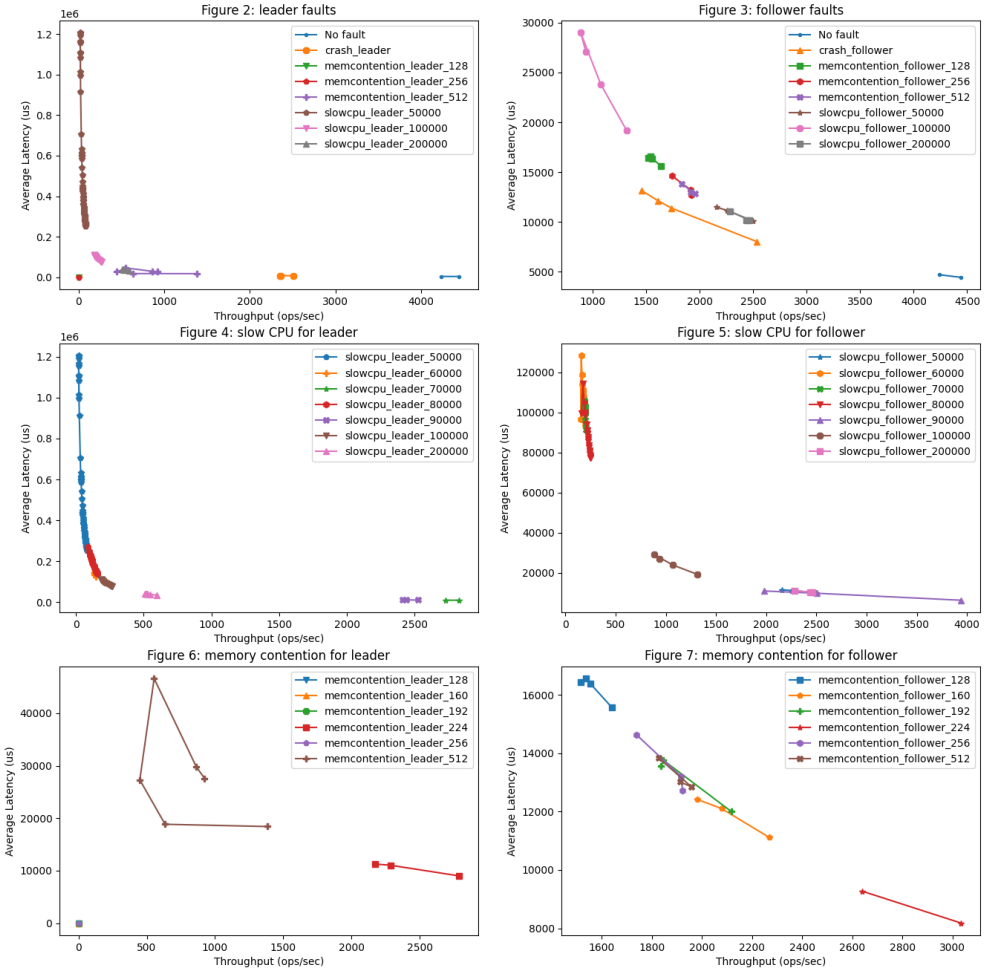
Crash: By killing the process of targeted node.

Slow CPU: By configuring cgroups (`cpu.cfs_quota_us` and `cpu.cfs_period_us`) to limit CPU usage of the process of targeted node in a given amount of time. [Ref 3]

Initially, I limited a node to use CPU for 0.05, 0.1 and 0.2 seconds out of every 1 second.

Memory contention: By configuring cgroups to limit memory usage of the process of targeted node. [Ref 4]

Initially, I limited a node to use 128/256/512 MB memory.



Q6: Please plot the performance with faults on the leader node and compare it with the baseline performance.

See Figure 2.

Q7: Please explain the above results. Is it expected? Why or why not?

It is observed that killing the leader node will only increase the latency (twice as no fault) and reduce the throughput (50% of no fault) of update operations. This is consistent with Raft and TiKV's fault tolerance design.

For slow CPU, initially, I limited a node to use CPU for 0.05, 0.1 and 0.2 seconds out of every 1 second. It is observed that operations are slowed down but can still be processed. When the leader can only use the CPU up to 0.1s out of 1s, the throughput is 16% of the benchmark performance. In particular, when the leader is limited to use CPU 0.05s out of 1s, the throughput is extremely close to 0 and the latency is at the scale of $10^5 - 10^6$ microseconds. Additional quota values between 0.05 and 0.1 seconds are picked in Q10 for further experiments.

For memory contention, initially, I limited a node to use 128/256/512 MB memory. It is observed that when leader can only use 512MB memory, the throughput is 25% of the benchmark performance. However, when the leader can only make use of 128MB or less memory, the update operations will fail and continue to throw exceptions `batchRecvLoop fails when receiving, needs to reconnect`, `mark store's regions need be refill` and `init create streaming fail` (See full log). This is unexpected as fault tolerance does not work. It is expected to elect a new leader when it fails to process operations. Additional quota values between 128MB and 256MB are picked in Q10 for further experiments.

We obtain an initial conclusion that for slow CPU on leader, operations are slowed down but can still be processed but memory contention on the leader could result in operation failures.

Q8: Please plot the performance with faults on the follower node and compare it with the baseline performance.

See Figure 3.

Q9: Please explain the above results. Is it expected? Why or why not?

(Configurations here are the same as Q7)

It is observed that killing the leader node will only increase the latency (4 times as no fault) and reduce the throughput (50% of no fault) of update operations. The overall observation is consistent with Raft and TiKV's fault tolerance design. But the latency being 4 times as no fault is unexpected as I thought killing leader will have a greater impact on latency of update operations.

For slow CPU, it is observed that when a follower can use CPU up to 0.2s out of 1s, the throughput is 60% of the benchmark performance and the latency is twice. The impact seems less compared to slow CPU on the leader.

For memory contention, setting a 128MB memory limit to a follower will result in $\frac{1}{3}$ throughput and 3 times latency compared to benchmark performance. This is different from memory contention on leader and won't cause exceptions.

We obtain an initial conclusion that for killing the follower, the performance degradation is surprisingly greater than leader in terms of latency. For either slow CPU or memory contention on follower, operations are slowed down but can still be processed.

Q10: For the slow CPU and memory contention, could you vary the level of slowness/contention and report the results?

Following Q7 and Q9, further quota values are picked for further experiments. For slow CPU, values between 0.1s and 0.05s with a step decrease of 0.01s (out of 1s) are picked. For memory contention, values between 256MB and 512MB with a step decrease of 32MB are picked.

For slow CPU on leader, from Figure 4 we can further confirm our previous conclusion that operations are slowed down but can still be processed. Generally, the lower CPU quota will lead to higher latency and lower throughput. But we also found some outliers that when leader can use CPU up to 0.09 or 0.07s out of 1s, the throughput is higher than that of 0.2s out of 1s. Repeating the experiment will usually obtain different results and we cannot get a straightly negative correlation due to flakiness.

For memory contention on leader, the result can also be a bit flaky. From Figure 6 we can find result of a "common run". In most runs (4 out of 5 as an rough estimate), 224MB memory limit of a follower will not affect update operations. However, 1 out of 5 run will cause `mark store's regions need be refill` exception and result in very low throughput and high latency. Based on about 20 experiments conducted, 256MB does not produce operation failures. We can unsafely say that 256MB is an "stable minimal memory limit" for the TiKV leader to process update operations without failures, which does not mean the limit will never produce no failures. It may be worthwhile to calculate the successful rate of a given memory quota over a large number of experiments, which can be further investigated.

Similarly, for slow CPU (Figure 5) or memory contention (Figure 7) on follower, we follow the same observation that generally the lower CPU quota and stricter memory contention will only lead to higher latency and lower throughput. But it is not a straightly negative correlation. For both memory contention on leader and follower, usually some quota values such as 224MB (which sometimes even let update operations fail if limit set for leader node) will have a even lower latency and greater throughput than 512MB (for separate experiments on leader and follower). I conjecture that this is due to the small scale of this experiment, which resulted in imbalance of replicas. If there are more nodes, the results may be a more straightly negative correlation.

Metadata

The repository can be found at https://github.com/94rain/tikv-experiment.

References

1. TiUp manual
2. go-ycsb manual
3. CPU Cgroup guide
4. Memory Cgroup guide