4/29/2020

- Work continuing on Persistent ART
  - Another ART implementation in HOPE if it's better/faster/more tested
  - One could maintain an ART of N second windows to re-use the existing Arena-based one.
  - One could maintain an arena for every N seconds, and then concurrently walk through and remove references into older arenas.
  - If one maintains version history in values rather than keys, then a single-version structure would work. Atomic op-heavy workloads would be the worst case for doing this, though.
  - Entirely different strategy: throw more memory at MVCC buffer?
- Rusty's tree changes likely to go in soon
- Rusty comparing memcpy implementations, to hopefully get an overall speedup
  - It might possibly be worth dusting off the "Align arena memory" PR
- Trevor has another PR for reducing copies with Futures/Promises
  - And has more changes planned for reducing copies in the proxy

4/22/2020

- Allow moving out of futures and actor compiler changes to reduce copies
  - Avoiding unnecessary copies from movable futures
  - R-value references are complicated
- Daniel ran Rusty's ART benchmark
  - Benchmark used random strings, which biased towards making ART work well
  - Result was 7x faster, which looks promising
  - ART code itself relied on area for memory safety
  - Struggled to get refcounting on children working right
- Daniel re-did benchmarks, and found out his target bandwidth is lower than expected.
  - Will look into RocksDB PR after spending a bit of time on ART
  - Better storage server write throughput is needed, something closer to Memory than Redwood
  - We should check with **Markus**, who mentioned someone on RocksDB might have gone peeking into this
  - There's still feelings of uncertainty about RocksDB correctness, and FDB simulation testing wouldn't help find them, as RocksDB would be running on its own threads outside of FDB's control

4/15/2020

- Avoid unnecessary copies in PromiseStream #2915 landed, and mostly

resulted in write bandwidth improvements
- Investigating doing more similar changes
  - Modifying actor compiler to get perfect forwarding in more places #2928
  - Potentially creating a MovableFuture class, so that one can move out of futures.
  - A longer discussion occurred on design choices around futures, actor compiler changes, and resumable functions
  - Eliding copies from reply generation could have similar improvements.
  - Using a better memcpy implementation resulted in a 5%-10% perf increase.
    - **Rusty** to look into this after current work is done. Ask **Kao** for brain dump.
  - make-linux-fast-again.com to disable kernel security patches when comparing perf across versions

4/8/2020

- Storage server profiling and optimizations
  - Daniel and Rusty have both not found low hanging fruit to further optimize
  - **Daniel** to run and profile a test with Rusty's three way comparison merged
  - 5% of CPU time spent allocating std::vector, but unsure if there's a good way to reduce that
- Daniel seeing stalls during performance tests
  - Are there recoveries? OOMs? Seems not.
  - Likely Ratekeeper?
  - **Alex** to connect with Segment folk about getting FoundationDB metrics into Datadog. (Sent!)
- P-Tree improvements/rewrites
  - Daniel chatting with Steve about if there's a better datastructure to use than the existing P-Tree
  - Neelam's previous investigation suggested that there's no low-hanging optimizations to do on the P-Tree itself
  - Persistent Adaptive Radix Tree might be a good candidate
  - (Persistent in the versioned sense, and not persistent as in disk.)
  - HOPE would maybe be useful to reduce in-memory size of data?
    - **Rusty** to bug Pavlo about releasing the code
    - Update: HOPE source was released
- New Slack channel of #write-throughput-discuss made for this project
- Proxy optimizations
  - 550micros spent in batching
  - Via tweaking knobs, this was reduced to 200micros
  - In the process of this work, found Avoid unnecessary copies in

PromiseStream, which might help for more loaded cases.

4/1/2020

- Storage server profiles show 25% CPU time in operator<
  - Rusty's approach: reduce the number of calls to operator< made by lower_bound and upper_bound
  - Daniel's approach: Pass StringRef by value in comparison operators
  - Rusty has a larger change in the works to move from operator< to operator<=> for more gains
  - Taking an `ssd` profile again would be good?
  - Try tweaking knobs to lower the number of versions a storage server keeps in memory to see what effect that has
- Proxy CPU
  - Had used debug transaction to build up stats on where time is going in commits
  - Looked like more time is spent batching transactions on proxy than in TLogs
  - Exploring ways to cheat the commit path, and drop strict serializability if desired
  - Policy engine optimizations might have a large impact on proxy cpu time
- Will continue adding more debug transactions to get better pipeline visualization

3/25/2020

- Rusty's updates
  - Initial results from network loop optimization resulted in a 1%-2% speedup
  - Next focus will be on storage server optimizations
- Daniel's benchmarking run results
  - Tracing of DebugCommit doesn't seem to show large pipelining waits
  - AWS seems to show ~0.1ms for within AZ, and ~0.3ms and ~0.7ms for across different AZs
  - Adding more clients now shows storage server saturation
  - Will continue to run more benchmarks, take and forward profiles to others to examine more closely

3/18/2020

- Pipelining
  - With Chrome Tracing, we now have some tools we can use to investigate commit pipelining.

    - How do we identify issues? What do we think could be issues? What tests should we be running to find issues?
    - Setting `\xff\x02/timeKeeper/disable` will disable timekeeper, which is the every 10s debug transaction.
    - Trevor is doing work in this area also. Will invite him to future meetings.
        * Investigating instances where more time being spent in proxy than waiting for tlog fsync.
    - Resolvers cannot just be turned off for conflict-free workloads due to their involvement in `\xff`
- Other improvements
    - Rusty's event loop 2.0?

To do for next week: * The suspicion is there there should be one proxy that is slower than the others. * Confirm from the trace that resolvers are waiting on pipelining, and preferably that one proxy is indeed slower * Look into or add proxy batching to traces * If there is indeed one slow proxy, then work on single proxy CPU usage and profiling * Sync with Trevor on his current state of proxy profiling