Homework 4 Design

Michael Tang

May 30, 2019

This is really for my own personal usage so we can take a chill pill

1 Design

Need: hash table implementation - see hashtable.c! Still requires testing.
Program flow: $main() \rightarrow opcontrol() \rightarrow dispatch()\&worker()$. Worker flow: $worker() \rightarrow tryLock() \rightarrow deq() \rightarrow fingerprint() \rightarrow$hash table op. Lock-free hash seems hard. Optimistic locking should bring good results and be simpler. Also ask if open-address methods are allowed. Unsure how to do concurrent contains() and add()/remove() operations, pay attention to that. Perfect hashing promising as well, maybe combined with optimistic locking. Parallel no-load: completely drop the packet, no fingerprint check or hash operations.
Good idea to keep track of which packet came from which source.
Hash table is to track what's new data and what we've seen before.
Remove() packets are request to remove the packet from the hash table. No guarantee it is preceded by an add() of the same packet; so allow remove() to fail.
Consider efficient hash table sizing, bucket sizing, and resize policies.

1.1 Load Imbalance

- Would emerge from potentially "hot" hash spaces
- Resize handling

2 Testing

2.1 Correctness

Hash table invariants:

- Correct hash indexing
- Correct collision handling
- Up-to-date information (correct removal)
- FIFO? Not necessarily with certain methods

Actual tests to do:

- 1: Perform a sequence of ops, write hash table state to a file, check states
- Warn: many possible linearizations
- Enforce sequencing with sync barriers to have a bunch of adds, then a bunch of removes of the same, in order to ensure unique and proper add/remove (also force all these bunches of calls to be concurrent
- Really just try to fuck up that hash table as much as possible
- White-box testing for resize()
- Parallel no-load: ensure hash packet gen works for the whole process
- Sum of checksums to ensure packets that go in equal packets that go out/remain

2.2 Performance Hypotheses

- Dispatcher rate: idk should be the same, slightly slower? see nature of hashpcks
- Parallel overhead: slower than before, hash table overhead, awesome overhead
- Speedup:
  - Mostly reads: optimistic locking should be faster here, open addr also has lower reading overhead than other hashing methods
  - Mostly writes: optimistic locking would not make a difference, other methods would cause faster writing
  - Speedup near 14x when we reach 14 cores, no higher (speedup should be linear)
  - Pthread is a yielding lock so if contention between threads is only based on lock, then should be consistent at 28 core (but collision or communication-based speedup collapse can still happen in other sections, analyze your program)

that's all folks im gonna get dinner now