Memcached Experiment Results: Linux Kernel Stack

Samuel Grossman

Stanford University

Experiment Description

- Run memcached on maverick-17 as follows:

```
memcached -t 24 -c 65535 -m 8192 -T -d
```

- Run mutilate using all hotbox machines plus maverick-16 as agents

- Use maverick-14 as the master which gathers latency samples
    - Sends requests at 1000 QPS

Framework Description

- Experiments are captured in files with “.experiment” extension
    - These simply contain a list of mutilate parameters with comments

- Agents used in each experiment are captured in “agent profiles”
    - A profile is two shell scripts, one to list and one to generate command lines
    - See examples, which are commented, for more information on each

- Typically experiments use 1,000,000 memcached records
    - Database is way too large to fit in cache

Experiment Set-up Guidelines

- Set up the memcached host per Jacob’s paper
  - Kill unneeded processes
  - Assign NIC queue interrupt affinity to be fixed to a single core per queue
  - Tweak some kernel stack TCP/IP parameters
  - These tasks are captured in “prep.sh” for machines with Intel 82599 NICs

- Run memcached that supports pinning threads to CPU cores
  - Uses the “-T” option to do so

Summary of Experiments

1. Constant, balanced connections from memcached clients
2. Constant, unbalanced connections from memcached clients
3. Variable key and value sizes (ex. Facebook ETC)
4. Scaling total number of client connections
5. Creating and destroying connections during the test

Graph Notes

- Currently some graphs are quite noisy
  - Changing some experiment parameters and rerunning to address this issue
  - Experiments 1 (basic_tuned only) and experiment 4 have been fixed
  - Working on experiment 5

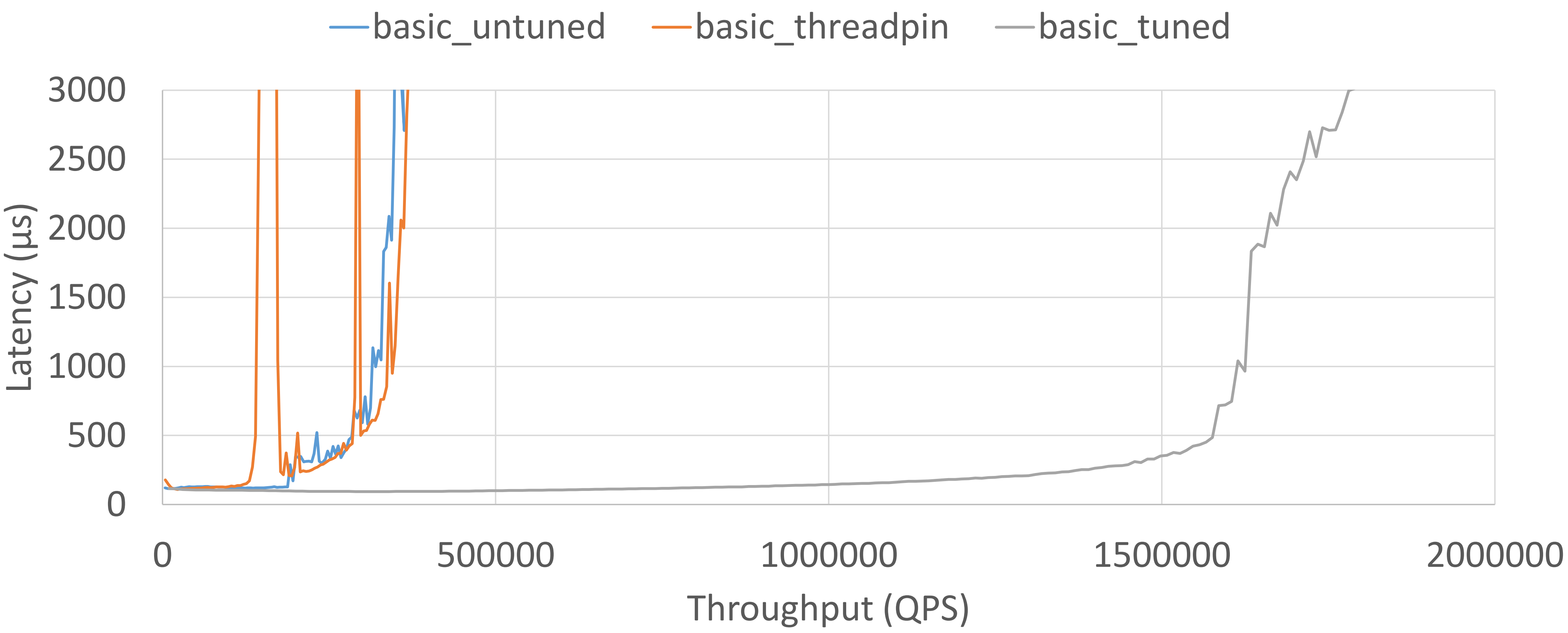
- Graphs shown in this presentation report the **95th percentile** of latency samples
  - Also captured were graphs for the average (arithmetic mean)

Experiment 1 Results

- Ran with three different configurations
  - basic_untuned: out-of-the-box Linux and memcached, no prep steps taken
  - basic_threadpin: only prep step is to run memcached with “-T”
  - basic_tuned: all documented prep steps taken (including “prep.sh”)

- Shows significant improvements when the memcached server is properly tuned
  - Jacob was right!

Experiment 1 Results

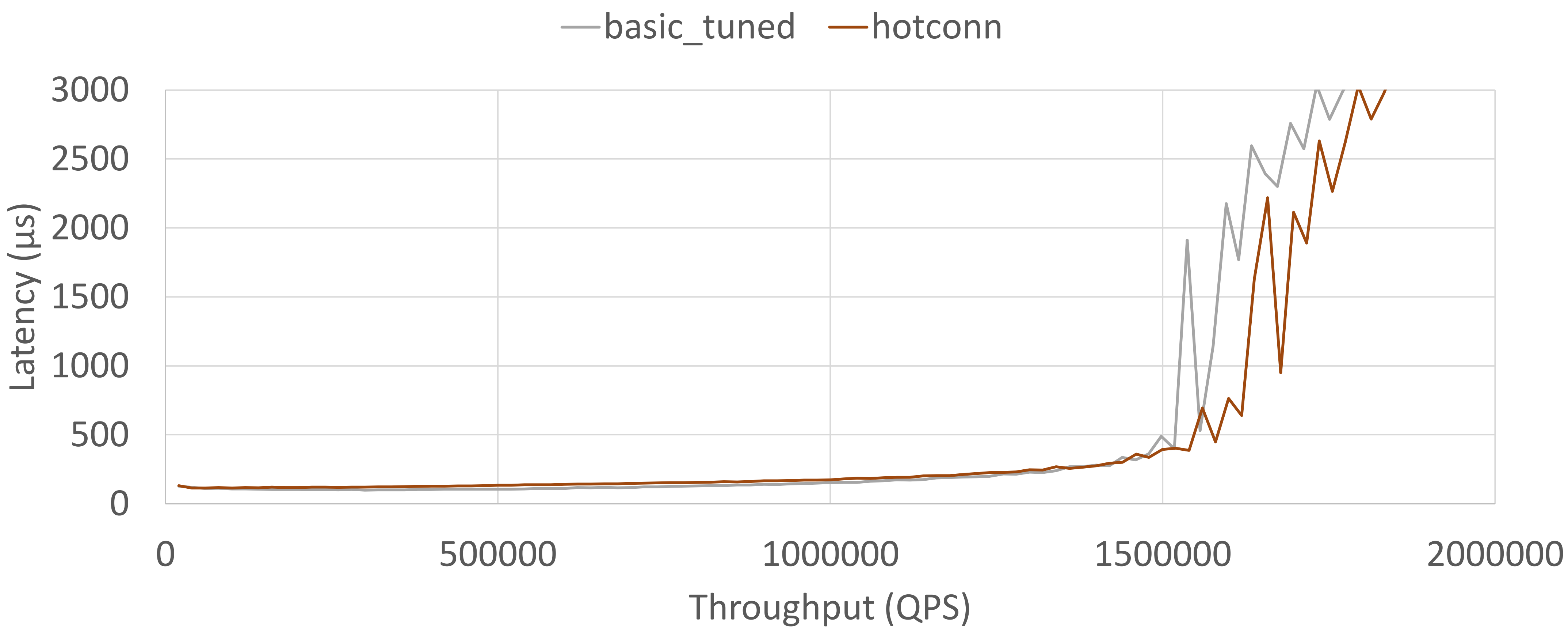


Experiment 2 Results

- Largely similar to experiment 1

- Need to create even more imbalance using a larger pipeline depth and a higher QPS share multiplier for the “hot” connections
    - Insufficient load is likely why it appears to perform better with load imbalance than with load balance

Experiment 2 Results

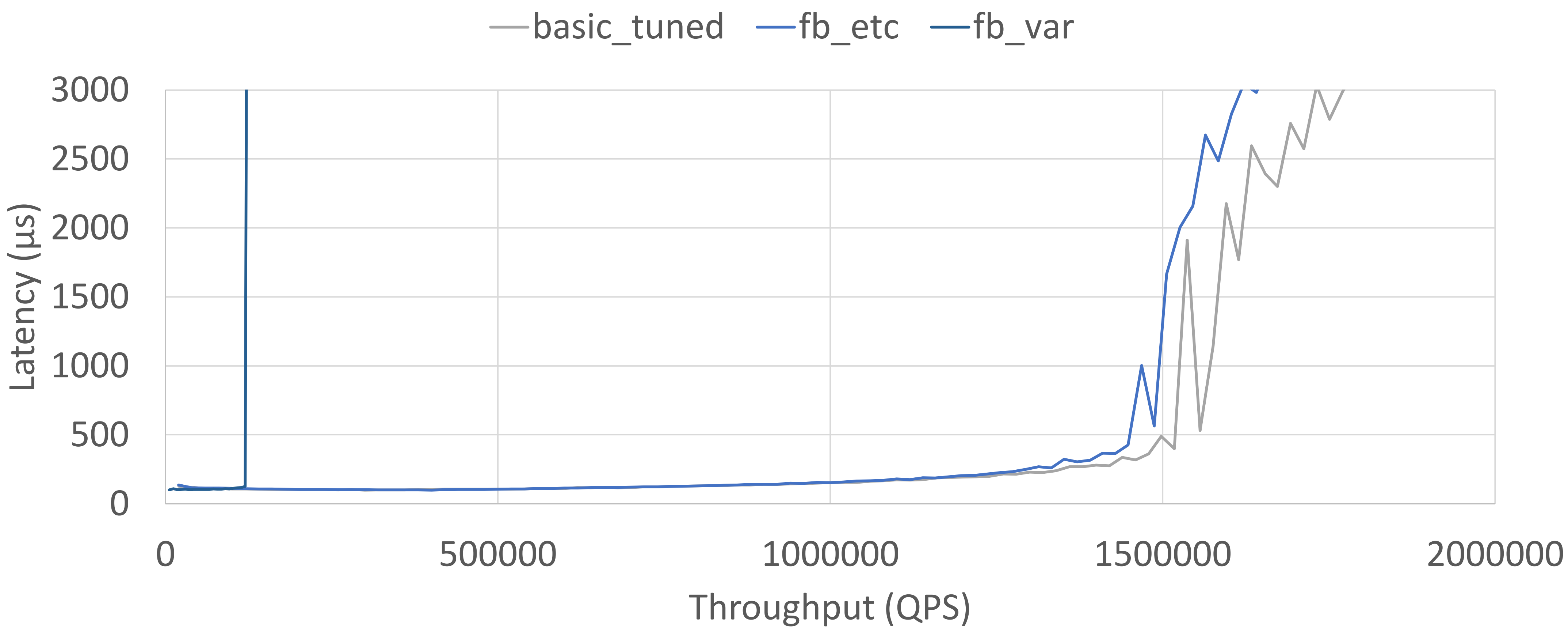


Experiment 3 Results

- Facebook ETC distribution is similar to, but slightly worse than, the basic distribution
  - Variable key size, value size, and inter-arrival times for requests

- Facebook VAR distribution is very write-heavy
  - 82% of requests are writes
  - Memcached locking behavior slows down this test quite a bit
  - Re-running with fewer memcached threads does help to alleviate some lock contention and improve performance

Experiment 3 Results

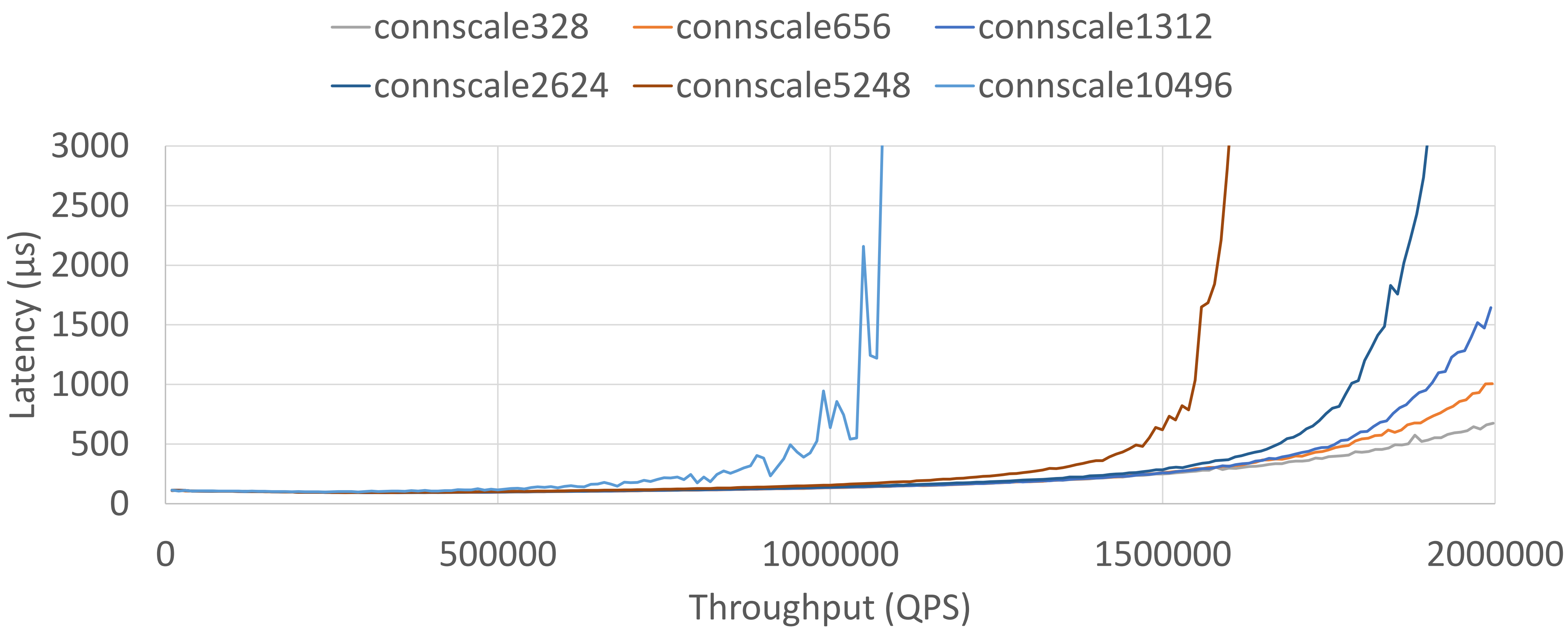


Experiment 4 Results

- Results are as one would expect
  - Knee moves to the left for higher numbers of connections

- Memcached starts dropping client connections once that number reaches between 10,000 and 15,000 total

Experiment 4 Results



Experiment 5 Results

- Currently very noisy, need to re-run these

- Results coming soon!