Performance Evaluation

Final project defence

Timon Zimmermann & Jimi Vaubien

February 13, 2019

École Polytechnique Fédéral de Lausanne

Project goal

Evaluate and compare performances of different in-memory databases systems

Overall schema

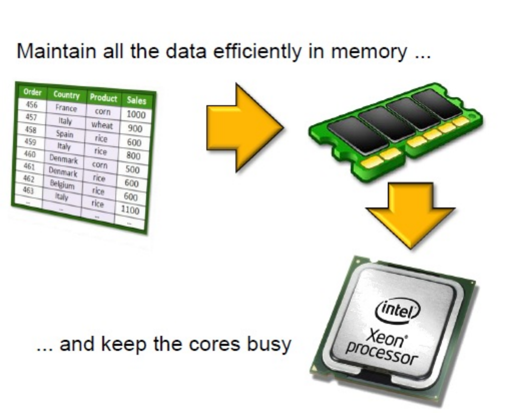


Figure 1: How it works

A few points about such systems

- *In theory*, relies on the main-memory to entirely store the data
- *In practice*, often used in pair with on-disk databases
  - Parts with critical response-time on main-memory
  - Non-critical parts on slower databases (HDD, SSD)
- Thread not idled by CPU because of the speed
- Uses persistent databases as back-up

Operations performed

Compare the atomic operations “set/get” of 2 different in-memory database systems.

Databases systems compared

- Memcached (`http://memcached.org/`)
    - Written in: C
    - Used by Google, Facebook...
- Redis (`https://redis.io/`)
    - Written in: C
    - Used by Twitter, GitHub...

Metrics

We use 3 different metrics:

- Response time (get & set operations)
- CPU usage
- RAM usage

Also investigate effects on the system:

- Similarly to Homework 3
- Congestion collapse

Metrics

We use 3 different metrics:

- Response time (get & set operations)
- CPU usage
- ~~RAM usage~~

Also investigate effects on the system:

- Similarly to Homework 3
- ~~Congestion collapse~~

Where?

- AWS *(Amazon)* EC2 instance
  - Burstable CPU compromising results
  - Limited bandwidth at reasonable pricing
- Own cluster connected through *1 Gb/s Ethernet*
  - “Attacker” with 6 physical cores (12 logical)
  - Database system hosted on 2 physical cores (4 logical) and 1 Gb RAM
  - Isolated in a virtual machine to minimize interaction with other processes

Operations performed

- Get and Set operations
- Scrambled and Contiguous
- 200 clients, each 200 requests/second
  - Less: Too easy to handle
  - More: Bigger load than “attacker” CPUs permit
- Operations distributed following $\mathit{Zipf}(0, 100'000, 0.99)$ [1]

Results: Redis end-to-end

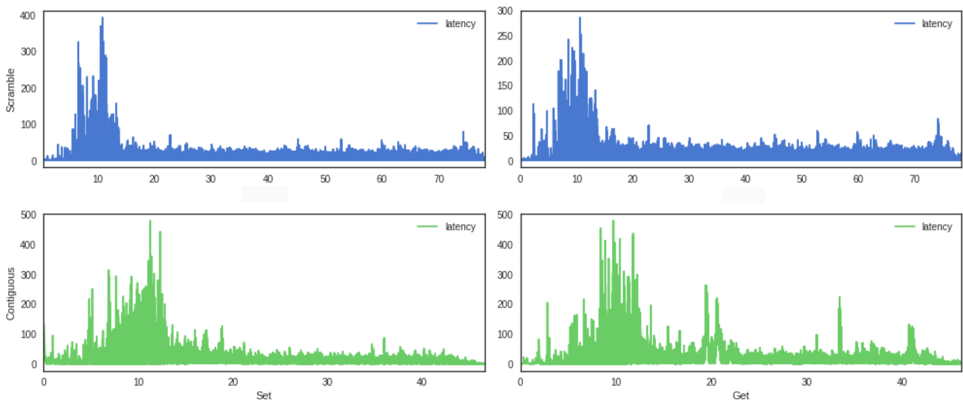


Figure 2: End to end latency time serie

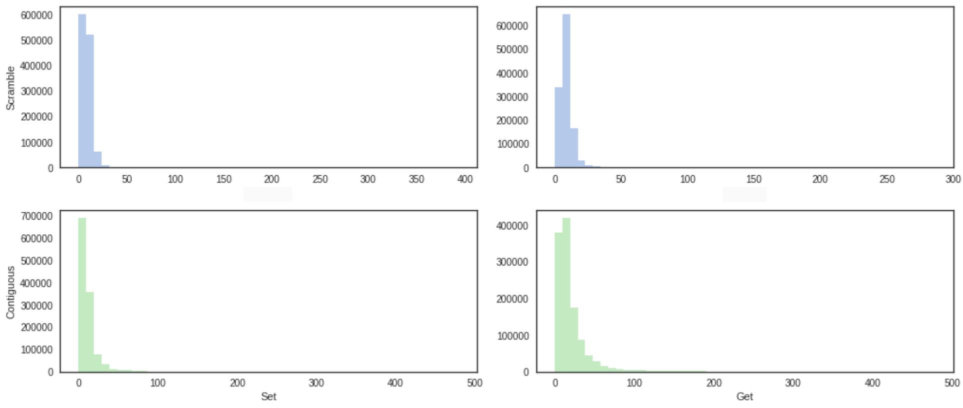


Figure 3: End to end latency distribution

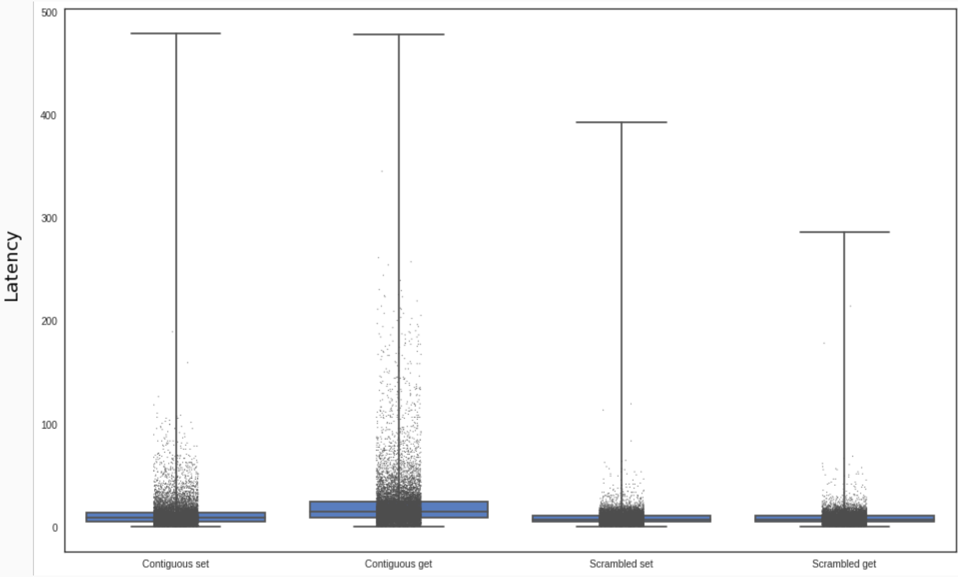


Figure 4: End to end latency box plots

Results: Redis stable regime

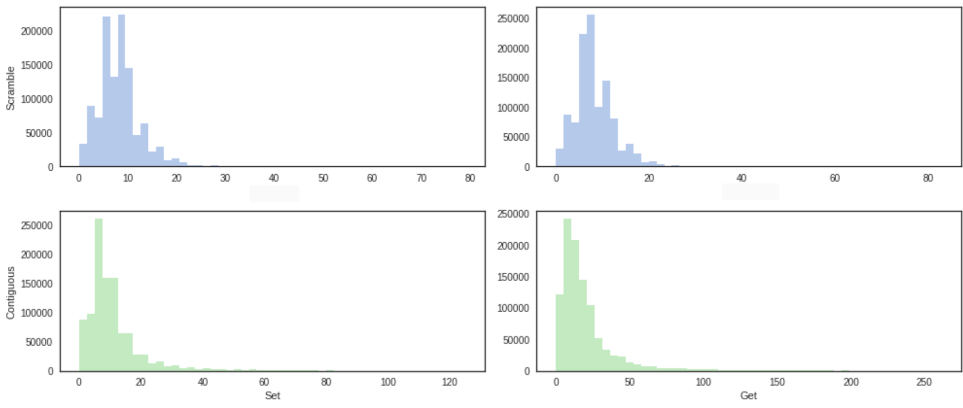


Figure 5: Stable regime latency distribution

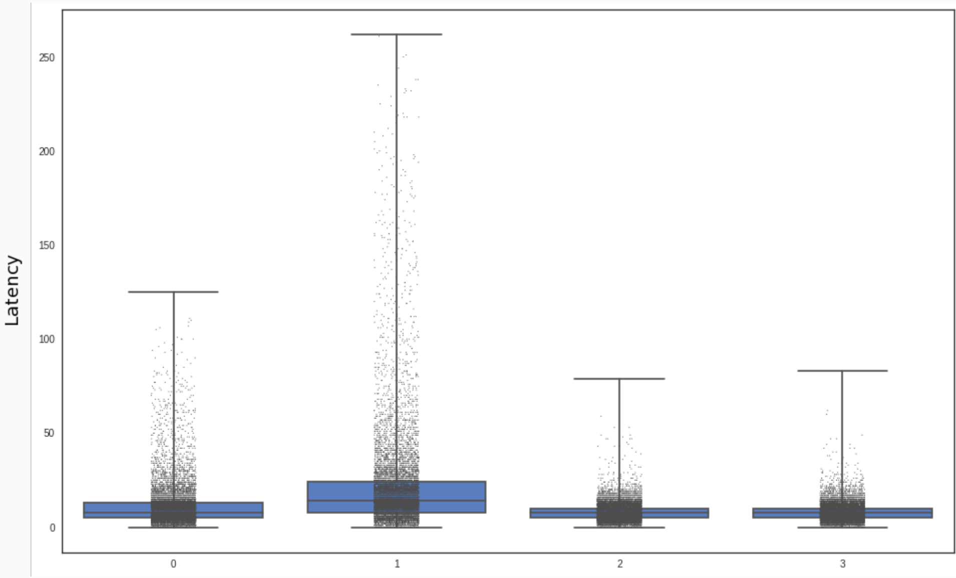


Figure 6: Stable regime latency box plots

Results: Memcached end-to-end

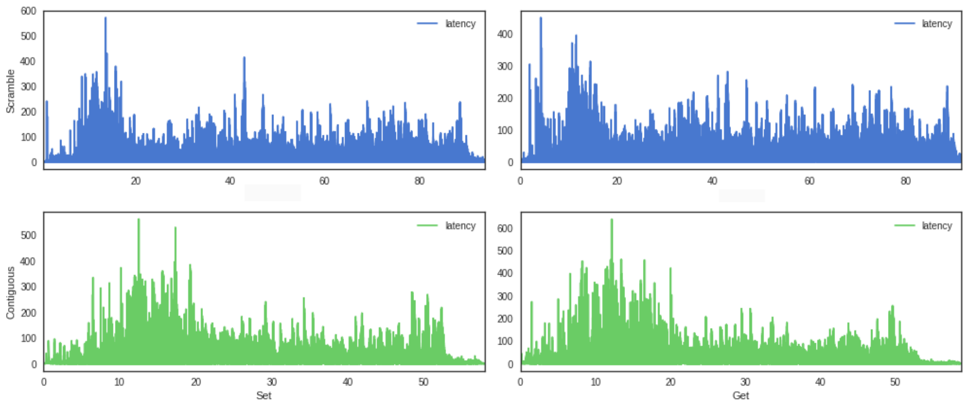


Figure 7: End to end latency time serie

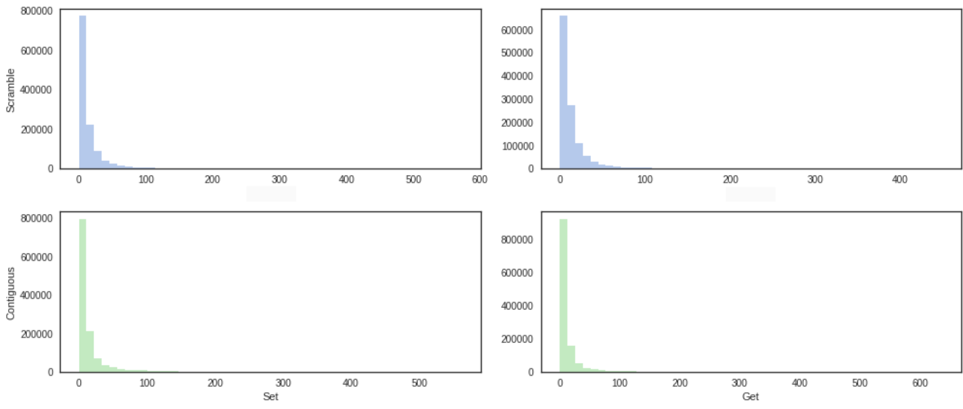


Figure 8: End to end latency distribution

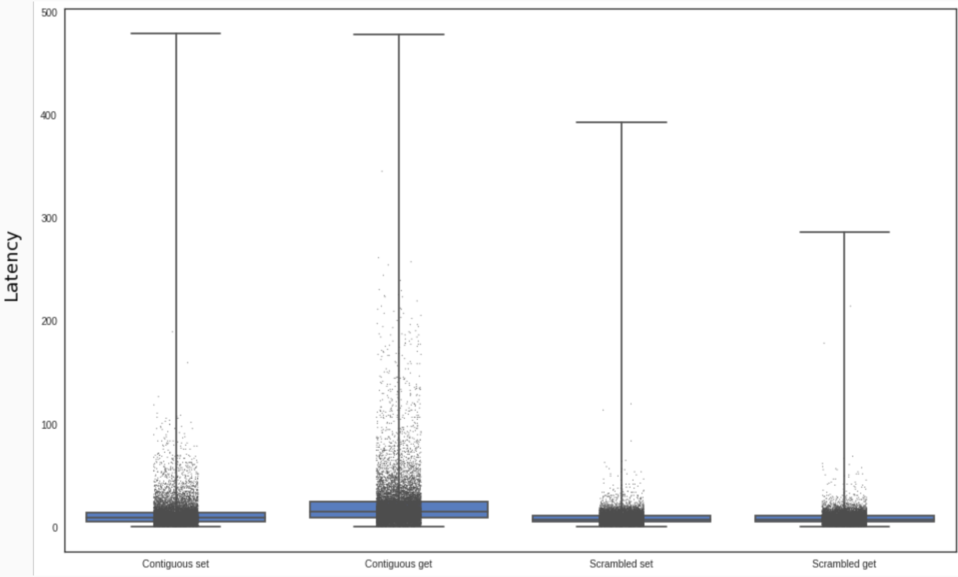


Figure 9: End to end latency box plots

Results: Memcached stable regime

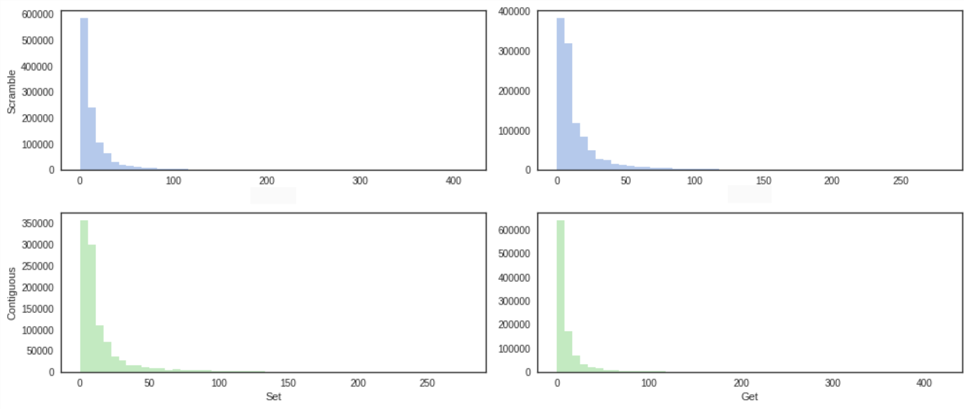


Figure 10: Stable regime latency distribution

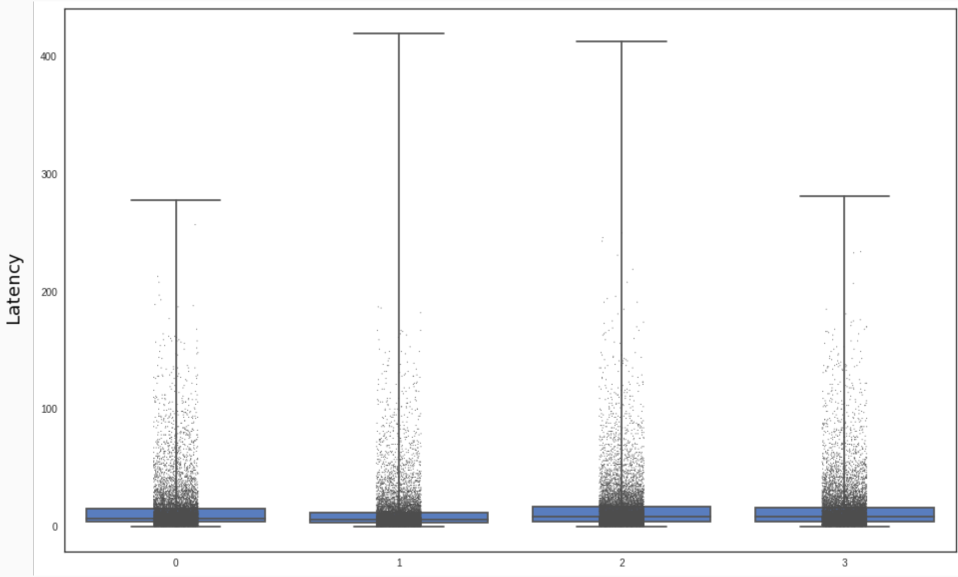


Figure 11: Stable regime latency box plots

Comparisons: Impact on the system

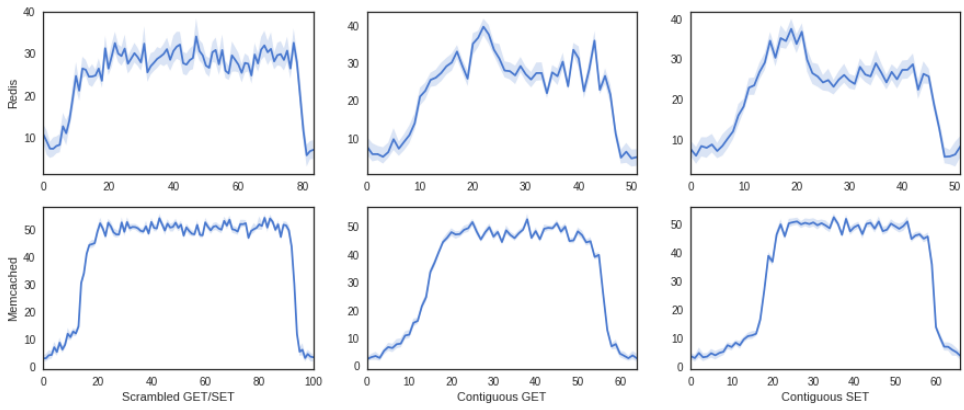


Figure 12: CPU usage of Redis and Memcached for different operations

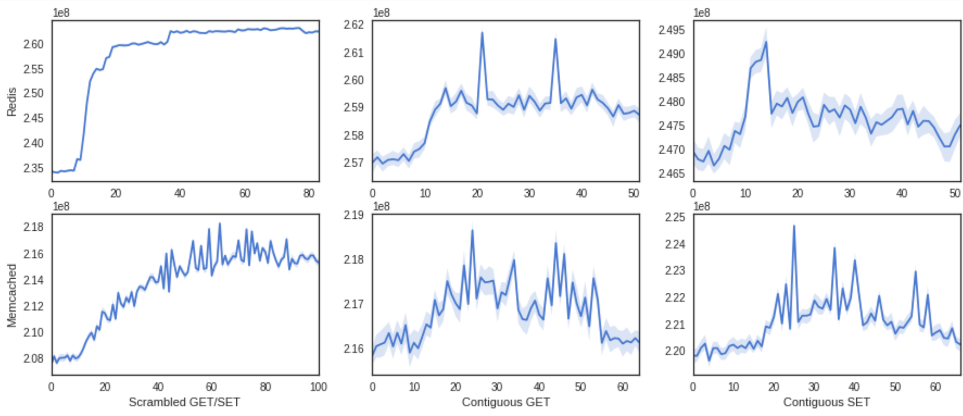
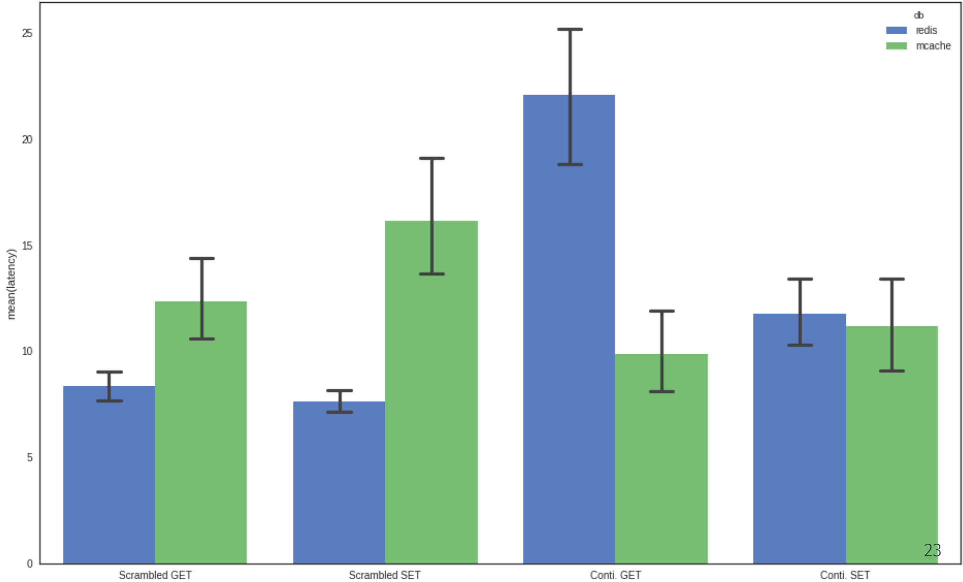


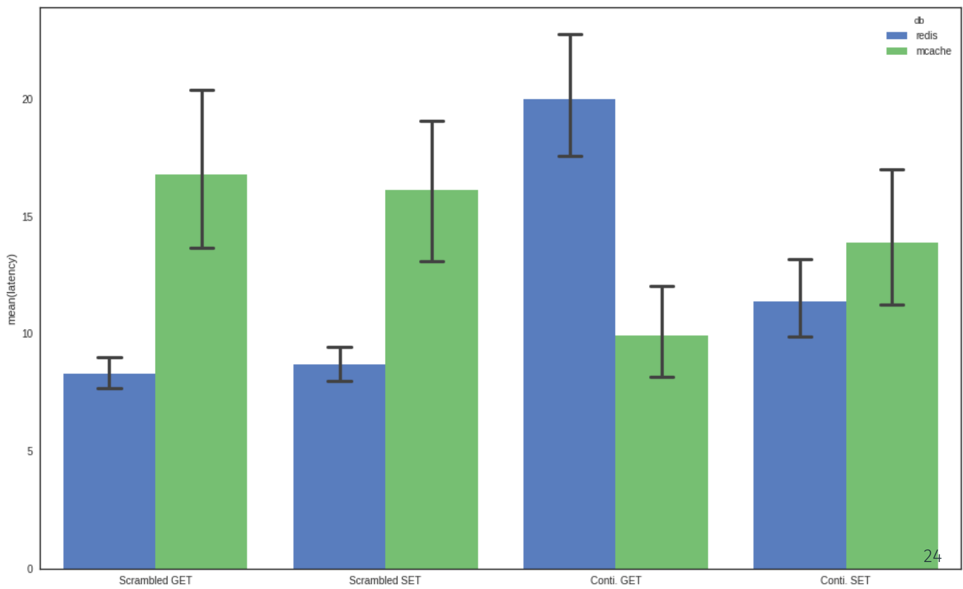
Figure 13: RAM usage of Redis and Memcached for different operations

- Redis uses more RAM than Memcached
- Memcached CPU usage is higher but stays almost the same once in stationary regime
    - Easier to predict power consumption for huge data centers

Comparisons: Latency end-to-end



Comparisons: Latency in “stable” regime



- Redis $\approx$ 1.8x faster than Memcached in 2 categories
  - One category even (contiguous SET)
  - Memcached $\approx$ 2.2x faster than Redis for contiguous GET
  - Memcached widely used for very-high get load (Facebook, Google and so on)
- Redis optimizations out-of-the-box ?
  - Using more RAM to pre-load instruction set
  - Default optimizations (RAM hash tables and so on), where Memcached is "lower-level"
- Confidence intervals validating the "instruction pre-loading" hypothesis
  - Redis latency less "sparse"

Possible improvements and extensions

- Test with optimizations disabled and fully enabled on both
- More cluster power to reach system's limit
  - Current setup bottlenecked by Network interface controller
- Bigger payloads to inspect how RAM is arranged and optimized

Questions?

Bibliography

E. F. S. J. B. Atikoglu, Y. Xu and M. Paleczny, “Workload analysis of a large-scale key-value store,” in *Proceedings of the SIGMETRICS’12*, June 2012.