A large scale analysis of hundreds of in-memory cache clusters at Twitter

Juncheng Yang, Yao Yue, Rashmi Vinayak
Carnegie Mellon University, Twitter, Carnegie Mellon University
Background

In-memory caching is ubiquitous in the modern web services.
To reduce latency, increase throughput, reduce backend load.
How are in-memory caches used? Do existing assumptions still hold?

Cache use cases
- Write-heavy workloads
- Object size distribution and evolution
- Time-to-live (TTL) and working set
In-memory caches at Twitter

- Single tenant, single layer
  - Container-based deployment

- Large scale deployment
  - 100s cache clusters
  - 1s billion QPS
  - 100s TB DRAM
  - 100,000s CPU cores
Trace collection and open source

- Week-long **unsampled** traces from one instance of each Twemcache cluster
  - 700 billion requests, 80 TB in size
  - Focus on 54 representative clusters

- Traces are open source
  - [https://github.com/twitter/cache-trace](https://github.com/twitter/cache-trace)
  - [https://github.com/Thesys-lab/cacheWorkloadAnalysisOSDI20](https://github.com/Thesys-lab/cacheWorkloadAnalysisOSDI20)
Cache use cases

● Caching for storage
  ○ Most common and use most resources

● Caching for computation
  ○ Increasingly popular
  ○ Machine learning, stream processing

● Transient data with no backing store
  ○ Rate limiters
  ○ Negative caches
Write-heavy workloads

35% of clusters are write-heavy (more than 30% writes)

Implication for future research:
- Optimization needed for write-heavy workloads
  - Challenges: scalability, tail latency
Object size

Object sizes are small
- 24% cluster mean object size < 100 bytes
- Median 230 bytes

Metadata size is large
- Memcached uses 56 bytes per-obj metadata
- Research systems often add more metadata
- -> Reduce effective cache size

Implication for future research:
- Minimizing object metadata to increase effective cache size
Object size

Value/key size ratio can be small
- 15% cluster value size <= key size
- 50% cluster value size <= 5 x key size

Small value/key size ratio
- Name spaces are part of keys
  - $\text{Ns1:ns2:obj}$ or $\text{obj/ns1/ns2}$

Implication for future research:
- A robust and lightweight key compression algorithm can increase effective cache size
Dynamic size distribution

Size distribution can be static
Bright color: more requests are for objects of that size in the time window

Most of the time, it is not static
The workload below shows a diurnal pattern
Size distribution over time

Sudden changes are not rare

Implication for future research:
- Size distribution changes pose challenges to memory management
- Innovations needed on better memory management techniques
Time-to-live (TTL)

- How long an object can be used for serving requests
- Set during object writes
- Expired objects cannot be served
TTL use cases and usages

- **Bounding inconsistency**
  - Cache updates are best-effort

- **Periodic refresh**
  - Caches for computation store computation based on dynamic features

- **Implicit deletion**
  - Rate limiter
  - GDPR compliant

**TTLs are usually short**

![Graph showing the distribution of Mean TTL (s) with CDF]
Short TTLs lead to bounded working set sizes

There is no need for a huge cache size if expired objects can be removed in time.

Implication for future research:
- Efficient proactive expiration techniques are more important than evictions
- Innovation needed on efficient TTL expiration
More in the paper

Production statistics
- Small miss ratio and small variations
- Request spikes are not always caused by hot keys

Object popularity
- Mostly Zipfian with large parameter alpha
- Small deviations

Eviction algorithms
- Highly workload dependent
- Four types of results
- FIFO achieves similar miss ratios as LRU
Summary

- Key observations and implications
  - Non-trivial fraction of write-heavy workloads
  - Small objects -> expensive metadata
  - Dynamic object size distribution
  - Short TTLs -> proactive expiration > eviction

- Traces open sourced for the community

Traces are available at
https://github.com/twitter/cache-trace
https://github.com/Thesys-lab/cacheWorkloadAnalysisOSDI20

Contact: juncheny@cs.cmu.edu