MEMC3: MEMCACHE W/CLOCK AND CONCURRENT CUCKOO HASHING
Bin Fan, David G. Andersen (Carnegie Mellon University), Michael Kaminsky (Intel Labs Pittsburgh)

MEMCACHED BACKGROUND

DRAM-based object cache
- Speed up web applications
- Alleviate database load.

OPTIMIZATIONS

COMPACT DATA STRUCTURES
- Replace chaining with "optimistic cuckoo hashing" [2]
  - Max hash table occupancy → 93%
  - Support single-writer/multi-reader access
  - Change linked list-based LRU to CLOCK
  - Replace two pointers with a reference bit for each object

ARCHITECTURE-AWARE OPTIMIZATIONS
- CPU cache locality
- Instruction/memory level parallelism

WORKLOAD-AWARE OPTIMIZATIONS
- Read mostly (e.g., > 98% [1])
- Small objects dominate

SYSTEM TUNING
- Pin thread on one core
- Increase page size: fewer TLB misses
- Purpose-built memcmp for small keys: compare 32bit at a time

CONCLUSIONS
- Memcached is bottlenecked by its core data structures when scaling to multiple cores
- Compact and concurrent algorithms improve significantly memcached throughput and space efficiency.


EVALUATION

HASHTABLE PERFORMANCE

- 6 threads accessing a hashtable (~1GB)
- Results are independent of key-value size

END-TO-END PERFORMANCE
- 16 B key + 32 B value, 95% get + 5% set
- 50 remote clients, one 10Gb NIC on server
- Store 25% more objects
- Scales to 12 cores with 4.3 MOPS or ~3x improvement

(Optimistic Cuckoo Hash Table)

- Map each key to two buckets
- Lookup: 3 memory reads
- Insert: O(1) w/ recursive displacement
- Increment key version before/after each displacement
- Compare key version snapshots before/after read

(CORE DATA STRUCTURES)

- Hash table w/ chaining
- Doubly-linked-list (for each slab)
- LRU header

(Compact Data Structures)

- Replace chaining with "optimistic cuckoo hashing" [2]
  - Max hash table occupancy → 93%
  - Support single-writer/multi-reader access
  - Change linked list-based LRU to CLOCK
  - Replace two pointers with a reference bit for each object

(Architecture-Aware Optimizations)

- CPU cache locality
- Instruction/memory level parallelism

(Workload-Aware Optimizations)

- Read mostly (e.g., > 98% [1])
- Small objects dominate

(System Tuning)

- Pin thread on one core
- Increase page size: fewer TLB misses
- Purpose-built memcmp for small keys: compare 32bit at a time

(Conclusions)

- Memcached is bottlenecked by its core data structures when scaling to multiple cores
- Compact and concurrent algorithms improve significantly memcached throughput and space efficiency.


(Dram-based object cache)

- Speed up web applications
- Alleviate database load.

(Webserver)

- get(x)
- sety,"123")
- get(z)

(Database)

(Memcached server)

- on miss

(Webserver)

- get(x)

(Memcached server)

- on miss

(Doubly-linked-list (for each slab))

(LRU header)

(Hash table w/ chaining)

(Key version counters)

(Cuckoo hash table)

(Log)

(Key value)

(Metadata)