Announcements

- Office hours on 11/29 and 12/06 open for project wrap-up discussion
  - Or, send me an email to make an appointment for other time that works for you

- Final project presentation
  - When: 4:30—7:30pm, 12/14
  - Where: Room 4201, Engineering Building
An In-Memory Object Caching Framework with Adaptive Load Balancing

Yue Cheng (George Mason)
Aayush Gupta (IBM Research – Almaden)
Ali R. Butt (Virginia Tech)

ACM EuroSys 2015
Memcached is an essential component in datacenters

Local deployment

Cloud deployment

Amazon ElastiCache

airbnb
Memcached is an essential component in datacenters
Memcached is an essential component in datacenters

Local deployment

Cloud deployment

In-memory caching tier e.g., Memcached

Network

Web app servers

Client library

set(key)

get(key)

Cache miss

set(key)

Persistent storage tier e.g., MySQL

Amazon ElastiCache

airbnb
Memcached is desirable

- Offers high performance
- Enables quick deployment
- Provides ease of use

- **Problem:** Load imbalance impacts performance
Access load imbalance

**Workload skewness (Zipfian constant)**

- Ideal balance
- High imbalance

95% GET, 5% SET, Zipfian, 20 cache servers
Access load imbalance

Per-client throughput (QPS in thousands)

Throughput

Workload skewness (Zipfian constant)

Ideal balance → high imbalance

95% GET, 5% SET, Zipfian, 20 cache servers
Access load imbalance

Per-client throughput (QPS in thousands)

Workload skewness (Zipfian constant)

Ideal balance  high imbalance

95% GET, 5% SET, Zipfian, 20 cache servers
Access load imbalance

![Graph showing throughput across different workload skewness (Zipfian constant)]

**Workload skewness (Zipfian constant):**
- **Ideal balance**
- **High imbalance**

**Per-client throughput (QPS in thousands):**
- unif
- 0.4
- 0.8
- 0.9
- 0.99
- 1.01
- 1.1

**Throughput:**
- > 60%

**Access load imbalance**

95% GET, 5% SET, Zipfian, 20 cache servers
Access load imbalance

**Per-client throughput (QPS in thousands)**

- unif
- 0.4
- 0.8
- 0.9
- 0.99
- 1.01
- 1.1

**Workload skewness (Zipfian constant)**

- Ideal balance
- high imbalance

**99th %ile latency (ms)**

- 3.2x

**95% GET, 5% SET, Zipfian, 20 cache servers**
Access load imbalance

Great opportunity for performance improvement

Workload skewness (Zipfian constant):
- Ideal balance
- High imbalance

95% GET, 5% SET, Zipfian, 20 cache servers
Our contribution: **MBal**

Revisiting in-memory cache design

A novel holistic in-memory caching framework with adaptive **Multi-phase load Balancing**

- Synthesizes different load balancing techniques
  - Key replication
  - Server-local cachelet migration
  - Coordinated cachelet migration

- Improves scale-up gains
- Mitigates load imbalance
Outline

MBal cache design
MBal load balancer design
Evaluation
Related work
Outline

**MBal Cache Design**
MBal load balancer design
Evaluation
Related work
In-memory data structure in Memcached

Shared in-memory data structure

In-memory data
Fine-grained data structures in MBal
MBal cachelet: a partition with associated resources

- Cachelet encapsulates resources
- Avoid lock contention

Indexing metadata (e.g., chained hash table)
Key-to-thread mapping

Query

Client side
Server side

MBal cache
Key-to-thread mapping

1. **Compute VN # with hash**

Client side
Server side

**Query**

**Key ring**

VN_1
VN_N
VN_N-1
VN_2

MBal client

**MBal cache**

header

key, (value*)

hash(key)
Key-to-thread mapping

1. Compute VN # with hash
2. Map VN # to Cachelet ID
Key-to-thread mapping

1. Compute VN # with hash
2. Map VN # to Cachelet ID
3. Map Cachelet ID to the worker thread
Outline

MBal cache design

MBal Multi-Phase Load Balancer

Evaluation

Related work
Phase 1: key replication

**TRIGGER?**
- EWMA access > threshold

**ACTION?**
- Randomly pick a shadow server
- replicate hot keys
- Proportional sampling

**FEATURES?**
- Fine-grained
- Temporary
Phase 1: key replication

- **TRIGGER?**
  - EWMA access > threshold

- **ACTION?**
  - Randomly pick a shadow server
  - Replicate hot keys
  - Proportional sampling

- **FEATURES?**
  - Fine-grained
  - Temporary

Key replication
Phase 2: server-local cachelet migration

- **TRIGGER?**
  - # hot keys > REPL\textsubscript{HIGH}
  - Enough local headroom

- **ACTION?**
  - Migrate/swap cachelet(s) within a server
  - ILP

- **FEATURES?**
  - Coarse-grained
  - Temporary
Phase 2: server-local cachelet migration

- **TRIGGER?**
  - # hot keys > $\text{REPL}_{\text{HIGH}}$
  - Enough local headroom

- **ACTION?**
  - Migrate/swap cachelet(s) within a server
  - ILP

- **FEATURES?**
  - Coarse-grained
  - Temporary

Server-local migration
Phase 3: coordinated cachelet migration

- **TRIGGER?**
  - # hot keys > REPL$_{HIGH}$
  - Not enough local headroom

- **ACTION?**
  - Migrate/swap cachelet(s) across servers
  - ILP

- **FEATURES?**
  - Coarse-grained
  - Permanent
Phase 3: coordinated cachelet migration

- **TRIGGER?**
  - \# hot keys > REPL_{HIGH}
  - Not enough local headroom

- **ACTION?**
  - Migrate/swap cachelet(s) across servers
  - ILP

- **FEATURES?**
  - Coarse-grained
  - Permanent
Phase 3: coordinated cachelet migration

• TRIGGER?
  – # hot keys > $\text{REPL}_{\text{HIGH}}$
  – Not enough local headroom

• ACTION?
  – Migrate/swap cachelet(s) across servers
  – ILP

• FEATURES?
  – Coarse-grained
  – Permanent
Cost/benefit tradeoffs in MBal

Cost: metadata; space; n/w transfer
Benefit: fast fix for hot keys

PI: Key replication
Cost/benefit tradeoffs in MBal

<table>
<thead>
<tr>
<th>Benefit</th>
<th>Cost</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>Cost: metadata; space; n/w transfer</td>
<td>Benefit: fast fix for hot keys</td>
</tr>
<tr>
<td>high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>low</td>
<td>Cost: metadata</td>
<td>Benefit: fast fix for hot partitions</td>
</tr>
<tr>
<td>high</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P1: Key replication

P2: Server-local cachelet migration
Cost/benefit tradeoffs in MBal

**Benefit**

- **Cost:** metadata; space; n/w transfer
  **Benefit:** fast fix for hot keys

- **Cost:** metadata; bulk transfer n/w
  **Benefit:** global load balancing

- **Cost:** metadata
  **Benefit:** fast fix for hot partitions

- **P3:** Coordinated cachelet migration

- **P2:** Server-local cachelet migration

- **P1:** Key replication
Outline

MBal cache design
MBal load balancer design

Evaluation

Related work
Methodology

• Scale-up cache performance tests
  – Local testbed
  – Single instance (8-core and 32-core server)

• End-to-end load balancer evaluation
  – 20-VM cluster (EC2, c3.large)
MBal evaluation – micro-benchmark

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Uniform workload, 100% GET, 10B key 20B value
- Without network

![Bar chart showing throughput (QPS in millions) vs number of threads for MBal, MBal no NUMA, Mercury, and Memcached.](image)
MBal evaluation – micro-benchmark

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Uniform workload, 100% GET, 10B key 20B value
- Without network
MBal evaluation – micro-benchmark

✓ MBal eliminates bucket-level lock contention!
MBal evaluation – micro-benchmark

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Uniform workload, 100% SET, 10B key 20B value
- Without network

![Throughput Graph](image)
MBal evaluation – micro-benchmark

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Uniform workload, 100% SET, 10B key 20B value
- Without network

![Graph showing throughput (QPS in millions) versus number of threads for MBal, MBal no NUMA, Mercury, and Memcached. The graph indicates a throughput increase of 62x for MBal no NUMA compared to MBal.]
MBal evaluation – micro-benchmark

✓ MBal eliminates global cache lock contention!

Throughput (QPS in millions)

Number of threads

- MBal
- MBal no NUMA
- Mercury
- Memcached

1.7 1.6 1.4 1.3 3.5 3.2 1.9 0.9 6.7 6.2 2.9 0.6 8.1 7.4 1.3 0.3 13.7 11.6 1.2 0.2

62×
MBal evaluation – complete system

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Zipfian workload, 75% GET, 10B key 20B value
- 10Gb Ethernet, MultiGET
MBal evaluation – complete system

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Zipfian workload, 75% GET, 10B key 20B value
- 10Gb Ethernet, MultiGET
MBal evaluation – complete system

MBal uses lightweight CPU cache-aligned bucket locks!
### End-to-end load balancer evaluation

<table>
<thead>
<tr>
<th>Workload</th>
<th>Characteristics</th>
<th>Application scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>100% read, Zipfian</td>
<td>User account status info</td>
</tr>
<tr>
<td>Workload B</td>
<td>95% read, 5% update, hotspot (95% ops on 5% data)</td>
<td>Photo tagging</td>
</tr>
<tr>
<td>Workload C</td>
<td>50% read, 50% update, Zipfian</td>
<td>Session store recording actions</td>
</tr>
</tbody>
</table>

Amazon EC2, us-west-2b, Clients on 36 instances (c3.2xlarge), 20-node VM cluster (c3.large)
Load balancer evaluation

Memcached is unable to sustain write-intensive workload

<table>
<thead>
<tr>
<th>Workload</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>100% read, Zipfian</td>
</tr>
<tr>
<td>Workload B</td>
<td>95% read, 5% update, hotspot</td>
</tr>
<tr>
<td>Workload C</td>
<td>50% read, 50% update, Zipfian</td>
</tr>
</tbody>
</table>

![Chart showing 90th percentile latency (ms) vs Runtime (seconds) for different workloads. Workload A, B, and C, with Memcached highlighted where it fails.]
Load balancer evaluation

<table>
<thead>
<tr>
<th>Workload</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>100% read, Zipfian</td>
</tr>
<tr>
<td>Workload B</td>
<td>95% read, 5% update, hotspot</td>
</tr>
<tr>
<td>Workload C</td>
<td>50% read, 50% update, Zipfian</td>
</tr>
</tbody>
</table>

Memcached is unable to sustain write-intensive workload

![Graph showing the performance comparison between Memcached and MBal across different workloads. Memcached struggles with write-intensive workloads, while MBal maintains stability.](image-url)
Load balancer evaluation

<table>
<thead>
<tr>
<th>Workload</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>100% read, Zipfian</td>
</tr>
<tr>
<td>Workload B</td>
<td>95% read, 5% update, hotspot</td>
</tr>
<tr>
<td>Workload C</td>
<td>50% read, 50% update, Zipfian</td>
</tr>
</tbody>
</table>

Memcached is unable to sustain write-intensive workload

![Graph showing performance of Memcached and MBal under different workloads.](image)
Load balancer evaluation

<table>
<thead>
<tr>
<th>Workload</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>100% read, Zipfian</td>
</tr>
<tr>
<td>Workload B</td>
<td>95% read, 5% update, hotspot</td>
</tr>
<tr>
<td>Workload C</td>
<td>50% read, 50% update, Zipfian</td>
</tr>
</tbody>
</table>

Memcached is unable to sustain write-intensive workload

---

Graph showing runtime (seconds) vs. 90th percentile latency (ms) for different workloads and load balancers.
Load balancer evaluation

<table>
<thead>
<tr>
<th>Workload</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>100% read, Zipfian</td>
</tr>
<tr>
<td>Workload B</td>
<td>95% read, 5% update, hotspot</td>
</tr>
<tr>
<td>Workload C</td>
<td>50% read, 50% update, Zipfian</td>
</tr>
</tbody>
</table>
Load balancer evaluation

<table>
<thead>
<tr>
<th>Workload</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>100% read, <strong>Zipfian</strong></td>
</tr>
<tr>
<td>Workload B</td>
<td>95% read, 5% update, hotspot</td>
</tr>
<tr>
<td>Workload C</td>
<td>50% read, 50% update, <strong>Zipfian</strong></td>
</tr>
</tbody>
</table>

**Workload A**: 100% read, **Zipfian**

- **90th %ile latency (ms)**
- **Runtime (seconds)**

**all 3 phases are triggered**

- **Memcached**
- **35%**
- **MBal**
- **MBal, all phases**
- **MBal, ideal balance**
Load balancer evaluation

<table>
<thead>
<tr>
<th>Workload</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>100% read, Zipfian</td>
</tr>
<tr>
<td>Workload B</td>
<td>95% read, 5% update, hotspot</td>
</tr>
<tr>
<td>Workload C</td>
<td>50% read, 50% update, Zipfian</td>
</tr>
</tbody>
</table>

For Workload B, only Phase 2 is needed.

Memcached
MBal
MBal, all phases
MBal, ideal balance
Load balancer evaluation

<table>
<thead>
<tr>
<th>Workload</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A</td>
<td>100% read, Zipfian</td>
</tr>
<tr>
<td>Workload B</td>
<td>95% read, 5% update, hotspot</td>
</tr>
<tr>
<td>Workload C</td>
<td>50% read, 50% update, Zipfian</td>
</tr>
</tbody>
</table>

Workload Characteristics

- **Workload A**: 100% read, Zipfian
- **Workload B**: 95% read, 5% update, hotspot
- **Workload C**: 50% read, 50% update, Zipfian

A combination of phase 2 & 3 is triggered

- **MBal**, all phases
- **MBal**, ideal balance

![Graph showing workload characteristics and latency](image)

- **Workload C**
- **Runtime (seconds)**
- **90th %ile latency (ms)**
Summary of results

• MBal fine-grained partitioning design
  – $2\times$ more QPS for GETs
  – $62\times$ more QPS for SETs

• MBal multi-phase load balancer
  – $35\%$ lower tail latency
  – $20\%$ higher throughput
  $\rightarrow$ Effectively improves QPS/$
Outline

MBal cache design
MBal load balancer design
Evaluation
Related work
Related work

• High performance in-memory KV store
  – Masstree [EuroSys’12], MemC3 [NSDI’12], MICA [NSDI’14]

• Storage load balancing
  – DHT (Pastry [Middleware’01], CFS [SOSP’01], Chord [SIGCOMM’01]), Proteus [ICDCS’13]

• Access load balancing
  – SmallCache [SoCC’11], Chronos [SoCC’12], SPORE [SoCC’13], Streaming Analytics [Feedback’14]
Conclusions

• Fine-grained, horizontal partitioning of in-memory data structure
  – eliminates sync overhead
  – enables load balancing

• MBal synthesizes three replication and migration techniques into a holistic system
  – reduces load imbalance
  – Improves tail latency