mPart: Miss Ratio Curve Guided Partitioning in Key-Value Stores

Daniel Byrne           Nilufer Onder            Zhenlin Wang
djbyrne@mtu.edu        nilufer@mtu.edu         zlwang@mtu.edu

Department of Computer Science
Michigan Technological University

International Symposium on Memory Management
June 2018
Introduction - Multi-Tenant Key-Value Stores

Background - Estimating the Miss Ratio Curve

mPart: Miss Ratio Curve Guided Partitioning

Evaluation

Conclusion
Database Access Times Are Slow

example.com

end user

GET uID

data center

database

time to fetch data from database:

\(~1,000 \mu s\)
Improve Access Time with Key-Value Store

- Key-Value stores reside in server’s main memory
- Data is stored as: key, value
- To retrieve an item: GET user.name
- To store/update an item: SET user.name djbyrne
- Examples: memcached and Redis
Caching Improves End-to-End Latency

database

data center

time to fetch data
from cache: ~10 µs

Byrne, Onder, Wang; ISMM 2018
# Impact of Hit Ratio

## System | Hit Ratio | Access Time (µs) | Cache | Database | End-to-End |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>95%</td>
<td></td>
<td>10</td>
<td>1,000</td>
<td>59.5</td>
</tr>
<tr>
<td>B</td>
<td>96%</td>
<td></td>
<td>10</td>
<td>1,000</td>
<td>49.6</td>
</tr>
</tbody>
</table>

1% increase in hit ratio → 16% increase in end-to-end!

Byrne, Onder, Wang; ISMM 2018
## Impact of Hit Ratio

<table>
<thead>
<tr>
<th>System</th>
<th>Hit Ratio</th>
<th>Cache</th>
<th>Database</th>
<th>End-to-End</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>95%</td>
<td>10</td>
<td>1,000</td>
<td>59.5</td>
</tr>
<tr>
<td>B</td>
<td>96%</td>
<td>10</td>
<td>1,000</td>
<td>49.6</td>
</tr>
</tbody>
</table>

1% increase in hit ratio → 16% increase in end-to-end!
Data Centers Host Multiple Apps

Key-Value Store (cache)

App1

App2

App3

App4

DB 1

DB 2

DB 3

DB 4

Byrne, Onder, Wang; ISMM 2018
Sharing the Cache with Multiple Apps

- Efficient Space Utilization — Cache space is not given to apps that do not benefit
- Near Optimal Miss Ratio — Overall cache miss ratio is minimized
Sharing the Cache with Multiple Apps

- Efficient Space Utilization — Cache space is not given to apps that do not benefit
- Near Optimal Miss Ratio — Overall cache miss ratio is minimized

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Efficient Utilization</th>
<th>Near Optimal Miss Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Pooled (no guidance)</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Pooled (w/ guidance)</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Introduction - Multi-Tenant Key-Value Stores

Background - Estimating the Miss Ratio Curve

mPart: Miss Ratio Curve Guided Partitioning

Evaluation

Conclusion
Miss Ratio Curves

- Miss Ratio Curve (MRC) — Quantifies the utility of increasing/decreasing cache size in terms of miss ratio
- Working Set Size (WSS) — The point that increasing cache size is not significant
Memshare - Estimate the MRC Slope

- *Memshare* by Cidon et al. (USENIX ATC ’17) use Shadow Queues
- Shadow Queues are logical victim caches, keep track of evicted keys

```
cache = 3
GET id1
miss
```

```
cache = 3
GET id1
shadow q = 2
```

When an app hits in the shadow queue, it is assigned 1 credit.
1 credit is taken from another app with lower shadow queue hit rate.
Memshare - Estimate the MRC Slope

- *Memshare* by Cidon et al. (USENIX ATC ’17) use Shadow Queues
- Shadow Queues are logical victim caches, keep track of evicted keys

GET id1

<table>
<thead>
<tr>
<th>cache = 3</th>
<th>miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>id3</td>
<td>id4</td>
</tr>
<tr>
<td>“djb”</td>
<td>“cat”</td>
</tr>
<tr>
<td>id5</td>
<td></td>
</tr>
<tr>
<td>“dog”</td>
<td></td>
</tr>
</tbody>
</table>

id1 id2

shadow q = 2

- When an app hits in the shadow queue, it is assigned 1 credit
- 1 credit is taken from another app with lower shadow queue hit rate
Drawbacks of Shadow Queues

- Critical path overhead — must check on every miss
- Has limited scope — can only estimate gradient for `sizeof(shadow queue)` beyond the current allocation
- No way of quantifying decrease in cache size
Drawbacks of Shadow Queues

- Critical path overhead — must check on every miss
- Has limited scope — can only estimate gradient for `sizeof(shadow queue)` beyond the current allocation
- No way of quantifying decrease in cache size

Our approach: Use app’s complete MRC to guide allocation
Introduction - Multi-Tenant Key-Value Stores

Background - Estimating the Miss Ratio Curve

**mPart: Miss Ratio Curve Guided Partitioning**

Evaluation

Conclusion
Building Miss Ratio Curve

- Track *reuse time* — Logical time between a request’s *last use* and *current use*
- *Average Eviction Time* (Hu et al. USENIX ATC ’16) — constructs accurate MRC by sampling reuse times

```
GET key1
GET key2
GET key2
...
GET key3
GET key1
GET key3
```
Building Miss Ratio Curve

- Track *reuse time* — Logical time between a request’s *last use* and *current use*
- *Average Eviction Time* (Hu et al. USENIX ATC ’16) — constructs accurate MRC by sampling reuse times

![Diagram showing GET key1, GET key2, GET key2, ..., GET key3, GET key1, GET key3, bar chart of reuse times]
Building Miss Ratio Curve

- **Track reuse time** — Logical time between a request’s *last use* and *current use*
- **Average Eviction Time** (Hu et al. USENIX ATC ’16) — constructs accurate MRC by sampling reuse times

![Graph showing the frequency of reuse times and the build MRC process]

- See paper for more details on how build MRC
Example Allocation Using MRCs

- Cache size of 10 million objects
- 20% overall miss ratio
Example Allocation Using MRCs

- Cache size of 10 million objects
- 20% overall miss ratio → 16% overall miss ratio
Minimizing the Number of Misses

\[ \mathbf{m} = [m_1, m_2, \ldots, m_N] \] is a memory assignment to \( N \) apps
Minimizing the Number of Misses

\[ m = [m_1, m_2, ..., m_N] \] is a memory assignment to \( N \) apps

\[
\begin{align*}
\text{minimize} & \quad F(m) = \sum_{i=1}^{N} NR_i \times mrc_i(m_i) \\
\text{subject to} & \quad m_i \text{ is a memory assignment to } N \text{ apps}
\end{align*}
\]

(1)
Minimizing the Number of Misses

\[ m = [m_1, m_2, \ldots, m_N] \] is a memory assignment to \( N \) apps

\[ \text{minimize } F(m) = \sum_{i=1}^{N} NR_i \ast mrc_i(m_i) \] (1)

expected misses for app \( i \)
Minimizing the Number of Misses

\[ m = [m_1, m_2, \ldots, m_N] \] is a memory assignment to \( N \) apps

\[
\text{minimize } F(m) = \sum_{i=1}^{N} NR_i \times mrc_i(m_i)
\]

(1) total misses

Byrne, Onder, Wang; ISMM 2018
Minimizing the Number of Misses

\[ m = [m_1, m_2, \ldots, m_N] \] is a memory assignment to \( N \) apps

\[
\text{minimize } F(m) = \sum_{i=1}^{N} NR_i \times mrc_i(m_i)
\]

subject to \( \sum_{i=1}^{N} m_i \leq M \) (total memory \( M \))
Minimizing the Number of Misses

\( \mathbf{m} = [m_1, m_2, \ldots, m_N] \) is a memory assignment to \( N \) apps

\[
\begin{align*}
\text{minimize} & \quad F(\mathbf{m}) = \sum_{i=1}^{N} NR_i \times mrc_i(m_i) \\
\text{subject to} & \quad \sum_{i=1}^{N} m_i \leq M \\
& \quad \text{(total memory } M) 
\end{align*}
\]

Problem: Brute force search is too costly
Minimizing the Number of Misses

\( m = [m_1, m_2, ..., m_N] \) is a memory assignment to \( N \) apps

\[
\begin{aligned}
\text{minimize } & F(m) = \sum_{i=1}^{N} NR_i \times mrc_i(m_i) \\
\text{subject to } & \sum_{i=1}^{N} m_i \leq M \\
& \text{total memory } M
\end{aligned}
\]

Problem: Brute force search is too costly

Solution: Use Dynamic Programming for polynomial time search
Implementation

- Implemented AET and allocation algorithm in C++ on top of Memshare
  - Special thanks to Cidon, Rushton, and Stutsman for source code
- Sampling for AET is done on incoming GET requests
- Constructing MRC occurs in separate thread
Introduction - Multi-Tenant Key-Value Stores

Background - Estimating the Miss Ratio Curve

mPart: Miss Ratio Curve Guided Partitioning

Evaluation

Conclusion
System Parameters

Memshare

- Increment/decrement 500 objects on a hit in the shadow queue
- Choose the application with lowest shadow queue hit rate to remove from
- Shadow queue represents 10MB of objects

mPart

- Re-allocate every 2 million requests
- Sample rate of 1/10,000 GET requests
Workloads Used

- *etc* and *etc.small* — Modeled after Facebook workload
- *psa* and *psa.small* — Used in prior key-value store works
- *ycsb* and *ycsb.small* — Yahoo Cloud Suite Benchmark workload C
- 100 million requests each
- Have distinct working set sizes
Workload MRCs

Byrne, Onder, Wang; ISMM 2018
Arrive at Stable Allocation Faster

Byrne, Onder, Wang; ISMM 2018
Decrease in Miss Ratio

Over 3 different initial allocation settings
  ▶ WSS Based — Set to 90% of WSS
  ▶ Objects — Set to 75% of unique objects
  ▶ Equal — Each application gets the same number of objects
Decrease in Miss Ratio

Over 3 different initial allocation settings

▶ WSS Based — Set to 90% of WSS
▶ Objects — Set to 75% of unique objects
▶ Equal — Each application gets the same number of objects

<table>
<thead>
<tr>
<th>Initial Setting</th>
<th>Miss Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WSS</td>
</tr>
<tr>
<td>Memshare</td>
<td>4.8%</td>
</tr>
<tr>
<td>mPart</td>
<td>3.6%</td>
</tr>
<tr>
<td><strong>Reduction</strong></td>
<td>-25.0%</td>
</tr>
</tbody>
</table>
Time-Varying Workloads - Hit Ratio

Set workloads to follow the inter-arrival rate distribution seen at Facebook over 24 hour period (Atikoglu, SIGMETRICS ’12)

Overall Miss Ratio: Memshare: 11% → mPart: 8.9%
Time-Varying Workloads - Allocation

Set workloads to follow the inter-arrival rate distribution seen at Facebook over 24 hour period (Atikoglu, SIGMETRICS ’12)
Cache Latency Improvement - GET

GET request latencies measured in µs

- GET may be a miss, cause a check in shadow queue

<table>
<thead>
<tr>
<th>System</th>
<th>95%</th>
<th>99%</th>
<th>99.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memshare</td>
<td>38.27</td>
<td>45.13</td>
<td>55.41</td>
</tr>
<tr>
<td>mPart</td>
<td>37.42</td>
<td>44.48</td>
<td>53.81</td>
</tr>
<tr>
<td>Improvement</td>
<td>2.22%</td>
<td>1.44%</td>
<td>2.89%</td>
</tr>
</tbody>
</table>
Cache Latency Improvement - SET

SET request latencies µs

- SET may cause eviction, need to update shadow queue in Memshare

<table>
<thead>
<tr>
<th>System</th>
<th>95%</th>
<th>99%</th>
<th>99.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memshare</td>
<td>48.69</td>
<td>85.99</td>
<td>126.69</td>
</tr>
<tr>
<td>mPart</td>
<td>47.98</td>
<td>82.92</td>
<td>123.77</td>
</tr>
<tr>
<td>Improvement</td>
<td>1.46%</td>
<td>3.57%</td>
<td>2.30%</td>
</tr>
</tbody>
</table>
## Cache Throughput Improvement

<table>
<thead>
<tr>
<th>Throughput</th>
<th>ycsb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memshare</td>
<td>560.1  kop/s</td>
</tr>
<tr>
<td>mPart</td>
<td>610.8  kop/s</td>
</tr>
<tr>
<td><strong>Improvement</strong></td>
<td><strong>8.8%</strong></td>
</tr>
</tbody>
</table>

No need to check shadow queue; only sample incoming requests.

Byrne, Onder, Wang; ISMM 2018
## Cache Throughput Improvement

<table>
<thead>
<tr>
<th></th>
<th>ycsb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memshare</td>
<td>560.1</td>
</tr>
<tr>
<td>mPart</td>
<td>610.8</td>
</tr>
<tr>
<td><strong>Improvement</strong></td>
<td>8.8%</td>
</tr>
</tbody>
</table>

No need to check shadow queue; only sample incoming requests
Introduction - Multi-Tenant Key-Value Stores

Background - Estimating the Miss Ratio Curve

mPart: Miss Ratio Curve Guided Partitioning

Evaluation

Conclusion
Conclusion

Recent advancements in MRC construction allow them to be used in online allocation

- Improved overall hit ratio
- Increased cache throughput
- Lower tail-latency
Thank you, questions?

mPart: Miss Ratio Curve Guided Partitioning in Key-Value Stores

Daniel Byrne  
djbyrne@mtu.edu

Nilufer Onder  
nilufer@mtu.edu

Zhenlin Wang  
zlwang@mtu.edu

Department of Computer Science  
Michigan Technological University

International Symposium on Memory Management  
June 2018