Comparative Evaluation of the Performance of In-Memory Key-Value Stores: Redis vs Memcached

By Noshin Nawar Sadat and Ishank Jain

Department of Computer Science

04/02/2019
Comparative Evaluation for the Performance of In-Memory Key-Value Stores: Redis vs Memcached
RESEARCH QUESTION

- When you need to scale an application with a lot of data, how do you decide on a storage solution?
- How can you both safely store and efficiently interact with large data sets?
- Selecting the right Key-Value store is the question that comes up every time when we think to scale database driven application.
MOTIVATION

• Companies such as Facebook, Twitter, Reddit, and Pinterest have adopted Memcached, while Github, Airbnb, Snapchat, Flicker are among the companies that use Redis.

• Many articles mention how Redis is better than Memcached because of all the different features it provides, but did not compare their performances.
Redis

- Primary database for data that requires rapid processing.
- Redis can persist its data to disk and can be made highly available through in-memory replication and auto-failover.
- All the commands in a transaction are serialized and executed sequentially.
- Variety of expiration policy.
MEMCACHED

• Memcached organizes its memory in **Slabs**, this reduces the pain of memory fragmentation.
• Items are made up of a **key**, an expiration time, optional flags, and raw data.
• Servers are Disconnected From Each Other.
• Least Recently Used cache.
Redis

- Redis is single threaded.

Memcached

- Memcached server is multi-threaded
- Memcached being multithreaded, can easily scale up by giving it more computational resources.
Redis

- Horizontally scale out available in Redis by clustering which is built-in.
- Cluster nodes have information about hash slots.

Memcached

- Horizontally scale out available by just adding new nodes.
- Cluster nodes have no information about hash slots.
COMPARISON: CONSISTENCY

Redis

- Redis provides **consistency** using an operation, which provides **optimistic locking**.

Memcached

- Uses **Check and Set operation** to maintain strong consistency.
COMPARISON: SUPPORTED DATA TYPES

Redis

- Redis supports much richer data types, including String, Hash, List, Set and Sorted Set.

Memcached

- Memcached which only supports data records of the simple key-value structure.
COMPARISON: OTHER FEATURES

Redis

• **Redis supports** replication and persistence.

Memcached

• **Memcached does not support** replication and persistence. (sort of)
BENCHMARKS: YCSB

• Yahoo! Cloud Serving Benchmark (YCSB) framework.
• Types of operations in workloads:
  • Read
  • Insert
  • Update
  • Scan
• The workloads can be customized. There can be following customizations:
  • Number of operations
  • Database size
  • Operation ratios
  • Number of clients
EXPERIMENT
COMPARISON METRICS

- **Latency** (in µs)
  - Read
  - Update
  - Insert

- **Throughput** (operations/second)

- **Memory usage** (in MB)

- **Scaling out**
  - Single node vs three node cluster

- Number of **concurrent clients**: 1, 12, 24, 36, 48 (Threads)
MACHINES

- **Server**
  - 15GB RAM
  - 12 cores CPUs
  - 1 Gbps ethernet link

- **Client**
  - 7.76GB RAM
  - 12 core CPUs
  - 1 Gbps ethernet link
SYSTEM CONFIGURATION

- Redis
  - Default configuration
  - Disabled replication and persistence
- Memcached
  - Default configuration
DATABASE

- Record size = 16 (fields) x 255 (bytes) = 4,080 bytes
- Total number of records = 2,500,000
- Total database size = 10.2 GB
## Workloads

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Description</th>
<th>Initial Database Size</th>
<th>Number of operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload A (Balanced RU)</td>
<td>50% Reads &amp; 50% Updates</td>
<td>10.2 GB</td>
<td>2,500,000</td>
</tr>
<tr>
<td>Workload B (Read-Heavy RU)</td>
<td>95% Reads &amp; 5% Updates</td>
<td>10.2 GB</td>
<td>2,500,000</td>
</tr>
<tr>
<td>Workload C (Update-Heavy RU)</td>
<td>5% Reads &amp; 95% Updates</td>
<td>10.2 GB</td>
<td>2,500,000</td>
</tr>
<tr>
<td>Workload D (Read-Heavy RI)</td>
<td>90% Reads &amp; 10% Inserts</td>
<td>5.1 GB</td>
<td>1,250,000</td>
</tr>
<tr>
<td>Workload E (Insert-Heavy RI)</td>
<td>10% Reads &amp; 90% Inserts</td>
<td>5.1 GB</td>
<td>1,250,000</td>
</tr>
</tbody>
</table>

Comparative Evaluation for the Performance of In-Memory Key-Value Stores: Redis vs Memcached
EXECUTING EXPERIMENT

Configure and start Redis/Memcached in server(s)

Note down used memory by the server(s)

Latencies and throughputs are stored automatically in client machine.

Load data into the databases

Run a workload with Zipfian distribution

Clear Database

Note down used memory by the server(s)
Comparative Evaluation for the Performance of In-Memory Key-Value Stores: Redis vs Memcached
**WORKLOAD B [95% Reads & 5% Updates]**

Comparative Evaluation for the Performance of In-Memory Key-Value Stores: Redis vs Memcached

<table>
<thead>
<tr>
<th></th>
<th>Memcached</th>
<th>Redis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Update Latency</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Node</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cluster</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Read Latency</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Node</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cluster</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Node</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cluster</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Update Latency for Single Node**
  - Memcached: 0 µs
  - Redis: 2000 µs

- **Update Latency for Cluster**
  - Memcached: 1000 µs
  - Redis: 1500 µs

- **Read Latency for Single Node**
  - Memcached: 500 µs
  - Redis: 1500 µs

- **Read Latency for Cluster**
  - Memcached: 500 µs
  - Redis: 1500 µs

- **Throughput for Single Node**
  - Memcached: 0 ops/s
  - Redis: 5000 ops/s

- **Throughput for Cluster**
  - Memcached: 5000 ops/s
  - Redis: 7000 ops/s

---

**University of Waterloo**
**Faculty of Mathematics**
WORKLOAD C [5% Reads & 95% Updates]

Comparative Evaluation for the Performance of In-Memory Key-Value Stores: Redis vs Memcached
WORKLOAD D [90% Reads & 10% Inserts]

Comparative Evaluation for the Performance of In-Memory Key-Value Stores: Redis vs Memcached
**WORKLOAD E [10% Reads & 90% Inserts]**

### Comparative Evaluation for the Performance of In-Memory Key-Value Stores: Redis vs Memcached

#### Insert Latency for Single Node
- **Memcached**
- **Redis**

#### Read Latency for Single Node
- **Memcached**
- **Redis**

#### Throughput for Single Node
- **Memcached**
- **Redis**

#### Insert Latency for Cluster
- **Memcached**
- **Redis**

#### Read Latency for Cluster
- **Memcached**
- **Redis**

#### Throughput for Cluster
- **Memcached**
- **Redis**
# Memory Utilization

<table>
<thead>
<tr>
<th>Scale</th>
<th>Memory Usage (in MB)</th>
<th>Memcached</th>
<th>Redis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Node</td>
<td></td>
<td>10,876</td>
<td>14,740</td>
</tr>
<tr>
<td>3-Node Cluster</td>
<td></td>
<td>3,630</td>
<td>4958</td>
</tr>
</tbody>
</table>

Memcached requires **6.6%** extra memory.
Redis requires **44.5%** extra memory!
## Takeaways

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Large Number of Concurrent Clients</th>
<th>Small Number of Concurrent Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single Node</td>
<td>Cluster</td>
</tr>
<tr>
<td>Workload A (50%R-50%U)</td>
<td>Memcached</td>
<td>Memcached</td>
</tr>
<tr>
<td>Workload B (95%R-5%U)</td>
<td>Memcached</td>
<td>Memcached</td>
</tr>
<tr>
<td>Workload C (5%R-95%U)</td>
<td>Redis</td>
<td>Redis</td>
</tr>
<tr>
<td>Workload D (90%R-10%I)</td>
<td>Redis</td>
<td>Redis</td>
</tr>
<tr>
<td>Workload E (10%R-90%I)</td>
<td>Redis</td>
<td>Redis</td>
</tr>
</tbody>
</table>

If not much available memory, use Memcached.
Further Exploration

- Comparison of their eviction policies.
- Performing multiple iterations of the test.
- Test cluster performances with increased database size.
THANK YOU