Benchmarking
Redis vs Memcached

Vonteri Harshith Reddy - 2019CS50450

Memcached

- Open source, In-memory data store
- Only Key value pairs
- Client-server architecture
- Multi-threaded
- Allows only bit strings as values
- No support for Persistence

REmote DIctionary Server - Redis

- Open source, In-memory data store
- Primarily key value store but can act as multi-model(document-store)
- Client-server architecture
- Single threaded
- Complex Data Structures
- On-disk Persistence

Why Redis and Memcached?

- Redis and Memcached are in-memory key value stores
- In DB ranking of Key value stores Redis stands 1st and Memcached stands 4th
- Redis stands 1st and Memcached 2nd when only in-premise data stores are considered as Azure and DynamoDB can only be deployed on cloud
- Redis and Memcached are top competitors in this domain
- Why is this domain important? Gaming leaderboards, carts, user sessions etc..,

RANK(NOV2023)	DBMS	SCORE(NOV2023)
1	Redis	160.02
2	Microsoft Azure	83.24
3	Amazon DynamoDB	34.11
4	Memcached	19.8

Experimental Setup

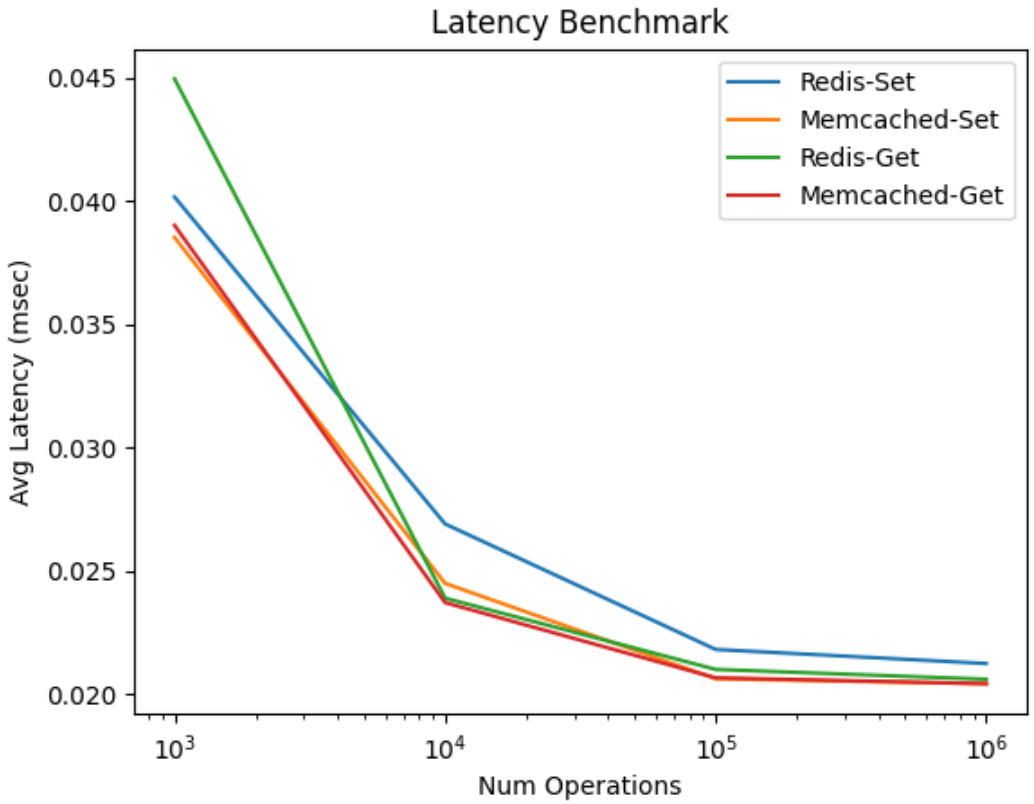
- Benchmarking tool: Mem-tier
  - Supports both testing and performance measure
  - Opensource, developed by Redis
  - Compatible with Memcached and Redis
  - Multithreaded
  - Customizable parameters
  - Statistical Analysis
- Redis version - 7.0.12
- Memcached version – 1.6.22
- 8 Core CPU, 8GB RAM

Experiments

- Benchmarked Redis, Memcached for:
  - Latency
  - Throughput
  - Scalability
  - Fault tolerance

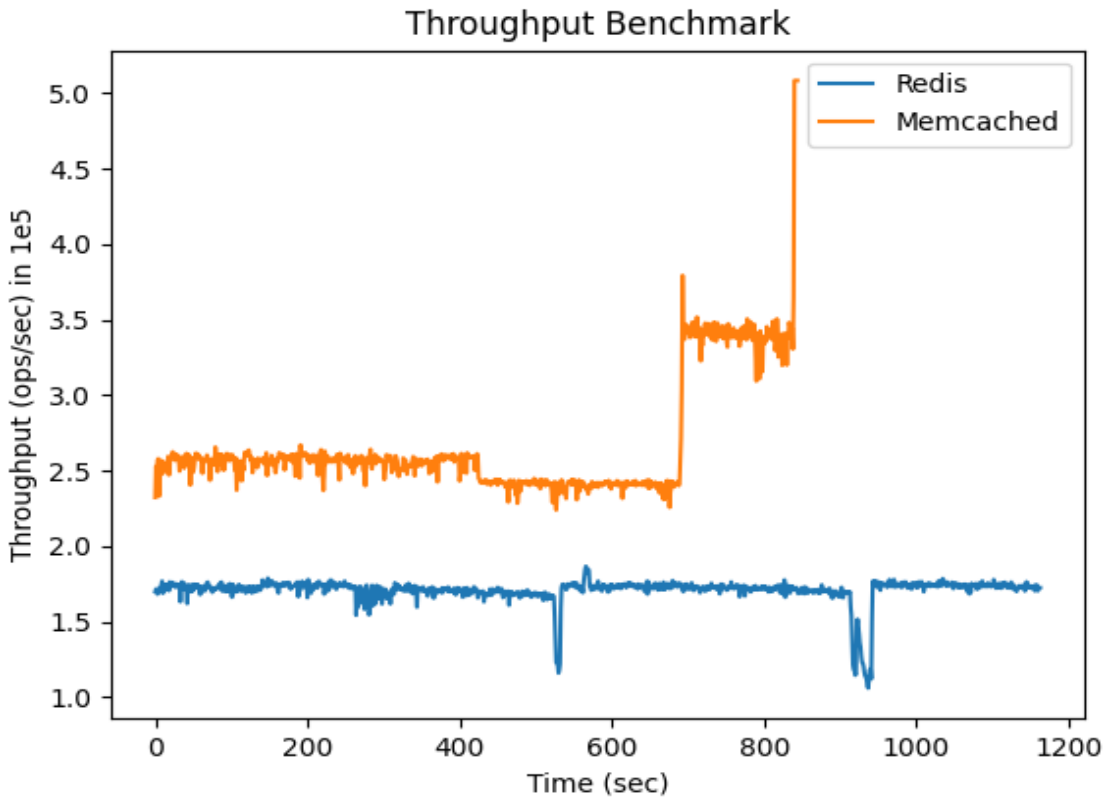
Latency Benchmark

- Mem-tier Benchmark
  - Single thread, 1 client per thread
  - Data size: 32 bytes
- Memcached exhibits slightly better performance than Redis on Set and Get operations
- Trends show that Get operation is slightly faster than Set operation
- Redis-Set operation is 5.5% slower than Memcached-Set, Redis-Get operation is 6% slower than Memcached-Get



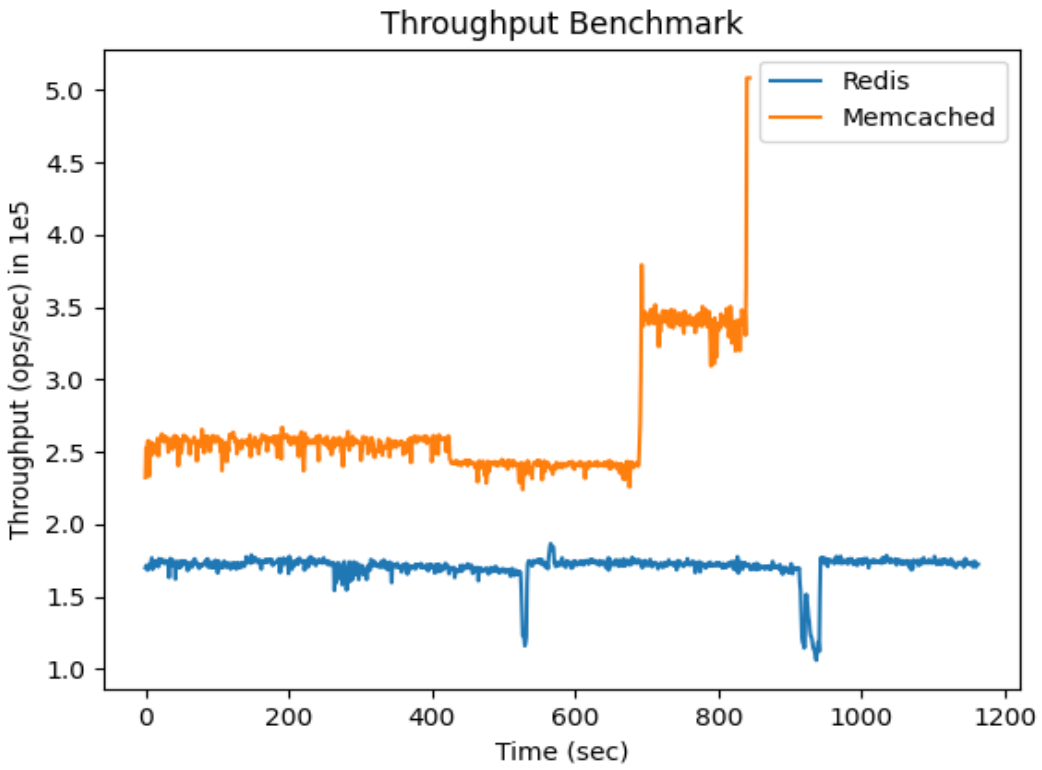
Throughput Benchmark

- Memtier Benchmark
  - 4 threads, 50 clients per thread
  - Each client sends 1000000 requests
  - Data size: 10 bytes
- Memcached clearly has higher throughput than Redis, as it uses multi-threading for request serving



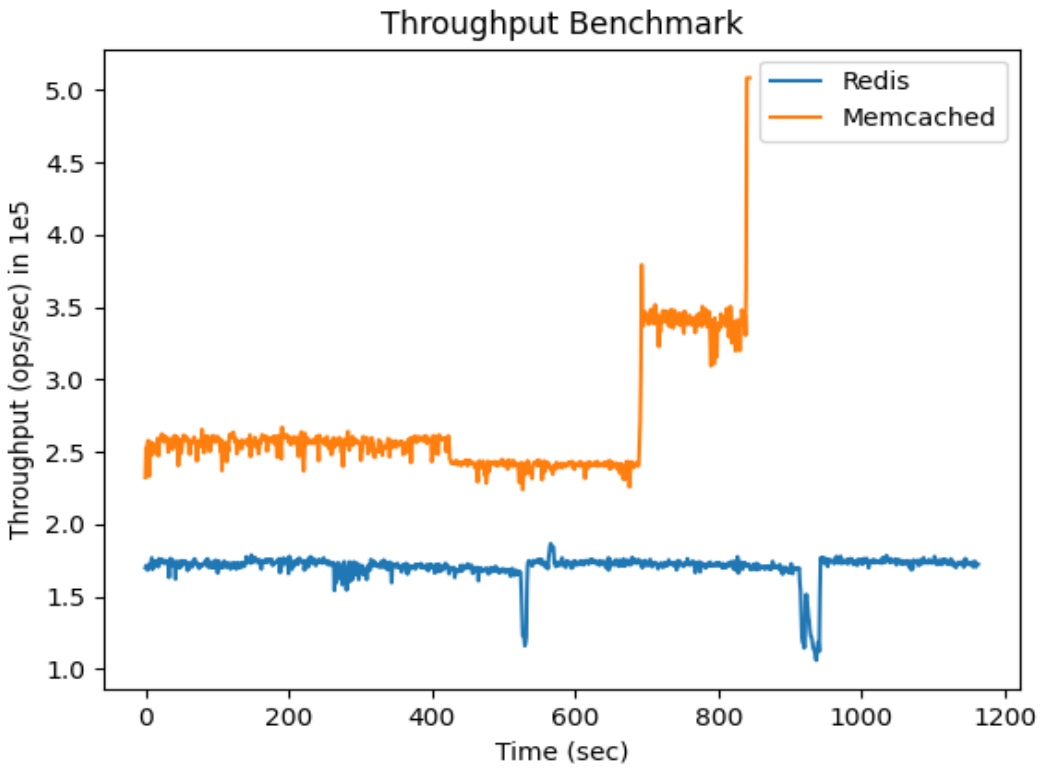
Throughput Benchmark – Dictionary Expansion

- A threshold is set in Redis and Memcached for controlling when dictionary expansion is necessary
- Once the threshold is exceeded, a double-size hash bucket is created, and data gradually moves from existing bucket to the new one
- 2 dents are observed in the plot for Redis, which implies that the server experienced dictionary expansion twice
- Throughput degradation in Redis due to dictionary expansion is ~38%



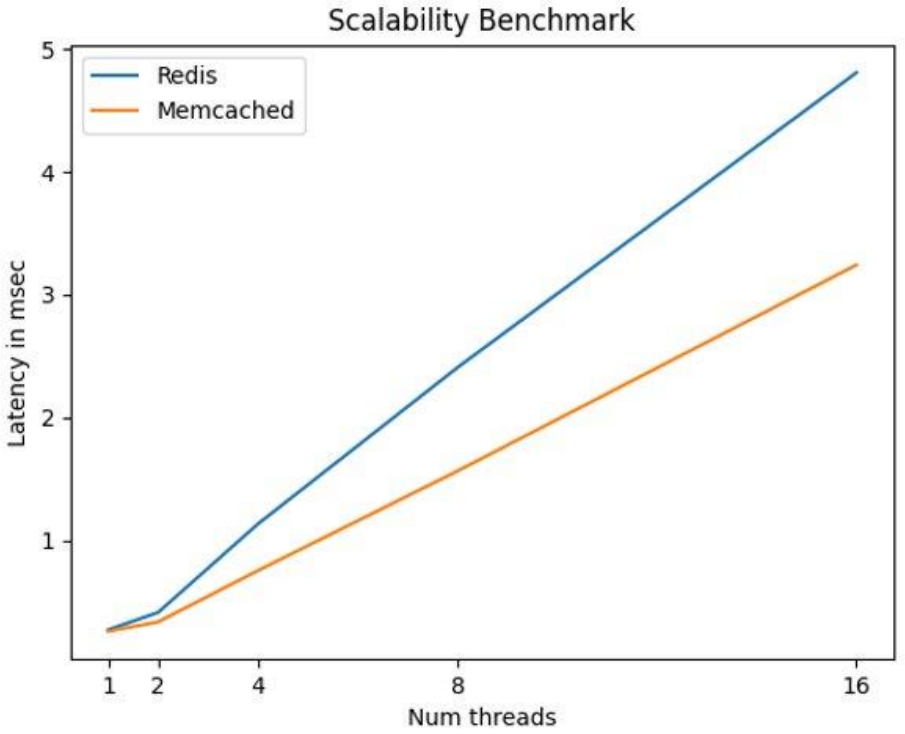
Throughput Benchmark – Dictionary Expansion

- Memcached didn't experience much dent in throughput even with dictionary expansion
- Memcached uses multithreading, dedicates one exclusive expanding thread, always ready to perform dictionary expansion
- In contrast Redis only has single thread which must deal with request serving and dictionary expansion
- The increase in throughput of Memcached may be attributed to a reduction in the number of active clients over time



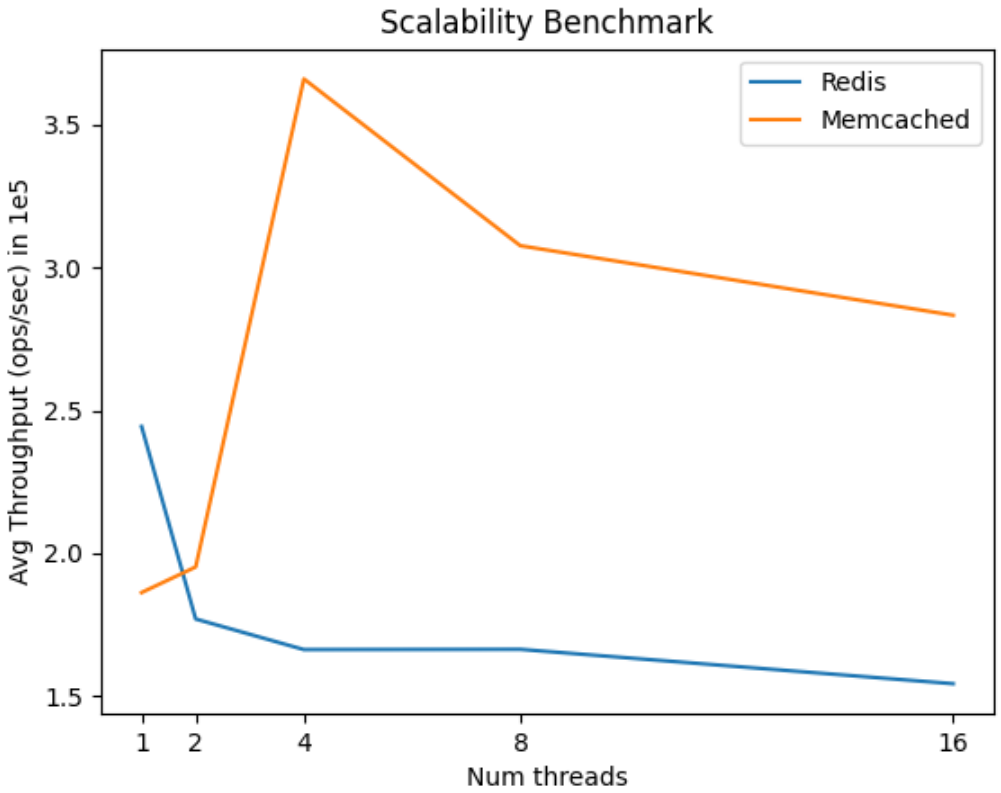
Scalability Benchmark - Latency

- Memtier Benchmark
  - Each client sends 10000 requests
  - Data size: 32 bytes
  - 50 clients per thread
  - Set:Get ratio 1:10
- Memcached clearly exhibits better scalability in Avg. latency with increase in no of threads, thanks to its use of multiple threads to serve requests
- Avg. Latency of Redis is 48.3% more than that of Memcached



Scalability Benchmark - Throughput

- Throughput of Memcached is significantly higher than Redis even with more no of threads
- Redis is single-threaded, hence there is a constant drop in throughput with increase in no of threads
- Because of multi-threaded nature of Memcached, there is an increase in throughput up to 4 threads, a decrease thereafter



Fault Tolerance

- Memcached is not fault-tolerant as it don't support on-disk persistence
- Redis is fault tolerant and has two persistence models:
    1. RDB (Redis Database) Persistence: Performs point-in-time snapshots of your dataset at specified intervals
    2. AOF (Append Only File) Persistence: AOF persistence logs every write operation received by the server. These operations can then be replayed again at server startup, reconstructing the original dataset
        - 3 modes for syncing: Always (Immediate), Every Sec, No (OS flush the output buffer when it wants)
- We can disable persistence completely. This is sometimes used when caching.

Overhead of Persistence Models

- We compare the performance of 3 models of Redis, namely RDB, AOF-Always, No Persistence to understand the overhead of Persistence
- 4 threads, 50 clients per thread, 100000 requests per clients
- Negligible overhead of RDB is that Redis forks a child process by using copy-on-write for the background save purpose, thus loading data and snapshotting can run concurrently
- Significant overhead is observed in AOF-Always (Immediate) sync.

MODEL	AVG. LATENCY (MSEC)	AVG. THROUGHPUT (OPS/SEC)
AOF-Always	7.14429	28021.47
RDB	1.13638	176008.06
No persistence	1.12054	176351.48

Memory Usage

- 1 client, 1 thread
- Only Set operations
- Memory Usage (in MB) of Memcached is better than Memcached with varying sizes of database as memcached uses less metadata to store key, values

Num Keys	10000	100000	1000000
Redis	0.78	7.46	72.4
Memcached	0.67	6.87	49.6

Summary – Redis Vs Memcached

- **Latency:** Both are fast with slight edge for Memcached
- **Throughput:** Memcached has higher throughput due to its use of multiple threads
- **Scalability:** Memcached Scales better than Redis (single instance)
- **Persistence, Replication, Complex Data structures:** Only supported in Redis
- **Memory Usage:** Memcached is better

Applications

Redis:

Gaming leaderboards

Ecommerce cart

user info on social media platforms

Search engines for full text search

Chat applications

Geospatial applications

Memcached:

Transient data caching

Stateless Caching for Web Applications

Future Work

- Perform extensive experiments like analyzing latency in terms of 99percentile
- Benchmark Availability
- Benchmark Consistency

Thank You