Detect and terminate long-running queries

Optimizing memory usage in KeyDB is essential for maintaining performance, especially in production environments like Elestio. Without proper memory control, large datasets, long-lived keys, or inefficient operations can lead to high memory pressure, slowdowns, or even server crashes. This guide explains how to optimize memory usage, monitor for memory-related issues, and configure automatic cleanup using Docker Compose environments.

Understanding KeyDB Memory Behavior

KeyDB allocates memory based on data structure usage and background operations like persistence or replication. It is important to monitor key memory indicators such as used memory, memory fragmentation, peak memory, and memory policy to understand how your instance behaves under load.

Monitoring KeyDB Memory in Real Time

To inspect memory statistics from the command line, use the `INFO MEMORY` command:

```
keydb-cli -a <password> INFO MEMORY
```

This command returns a detailed report including `used_memory`, `used_memory_rss`, `mem_fragmentation_ratio`, and `maxmemory`. A high fragmentation ratio may indicate inefficient memory usage or a need to tune your allocator.

If you are running KeyDB in a Docker Compose environment, connect to the container first:

```
docker-compose exec keydb bash
```

Once inside, run:

```
keydb-cli -a $KEYDB_PASSWORD
```

This gives you full access to execute monitoring and configuration commands.

Setting Maximum Memory and Eviction Policy

To avoid out-of-memory errors, it is crucial to set a memory cap and enable eviction. Edit your `keydb.conf` or set these at runtime:

```
CONFIG SET maxmemory 512mb
CONFIG SET maxmemory-policy allkeys-lru
```

The `maxmemory` setting defines the upper limit of memory usage. The `maxmemory-policy` determines how keys are evicted when that limit is reached. Recommended policies include:

- `allkeys-lru`: Evicts the least recently used keys across all keys
- `volatile-lru`: Evicts LRU keys with expiration set
- `noeviction`: Rejects writes when memory is full (not recommended in production)

Analyzing Memory Usage with MEMORY STATS

Use the built-in `MEMORY STATS` command for a high-level breakdown of memory usage by component:

```
MEMORY STATS
```

This provides statistics on memory overhead, allocator efficiency, and usage by data structure types.

Cleaning Up Expired or Unused Keys

Expired keys in KeyDB are removed passively upon access or through background expiration cycles. To force cleanup manually or test expiration behavior:

```
MEMORY PURGE
```

This clears internal allocator caches and triggers background memory cleanup without deleting live keys. Use this cautiously in production environments.

Listing Keys Consuming the Most Memory

You can use the `MEMORY USAGE` command to inspect which keys consume the most memory. For example:

```
MEMORY USAGE mykey
```

To automate finding the top memory-consuming keys, use a loop with `SCAN` and `MEMORY USAGE`:

```
SCAN 0 COUNT 100
```

Then evaluate `MEMORY USAGE` per key manually or using a script.

Best Practices for KeyDB Memory Management

Minimize memory pressure by following these recommendations:

- **Avoid large keys:** Break large values into smaller hashes or lists to reduce memory footprint and allow efficient partial retrieval.
- **Expire non-essential keys:** Always set TTLs on cache data or temporary states using `EXPIRE` or `SETEX`.
- **Avoid full dataset scans:** Replace commands like `KEYS *` with `SCAN` to prevent memory spikes.
- **Limit big lists or sets:** Use commands like `LRANGE mylist 0 99` instead of fetching entire datasets with `LRANGE mylist 0 -1`.
- **Use lazy data loading:** Design applications to load only required data in batches.

Monitoring Memory Growth Over Time

Track historical memory usage using the `INFO MEMORY` and `LATENCY DOCTOR` commands periodically, and export metrics to Prometheus or another monitoring system if needed.

Consider integrating KeyDB with monitoring tools like:

- Grafana with Prometheus Exporter
- Elestio’s built-in monitoring agent

These help you visualize and react to memory growth trends in real time.

Optimizing KeyDB’s memory usage is essential to running reliable, responsive services. By configuring `maxmemory`, choosing an appropriate eviction policy, monitoring key memory metrics, and cleaning up expired data, you can ensure predictable performance under load. Combine these strategies with external monitoring for long-term stability in Docker Compose environments like Elestio.

Revision #1
Created 2025-06-26 05:44:36 UTC
Updated 2025-06-26 06:09:21 UTC