Redis-from-Scratch: High-Performance C++ Implementation

Antigravity Technical Report

February 17, 2026

1 Introduction

This report details the implementation of a high-performance Redis-like server written in C++. The project focuses on modern C++ paradigms, memory efficiency through zero-copy architectures, and reliable persistence mechanisms.

2 System Architecture

The server employs a decoupled **Producer-Consumer** architecture, visualized in Figure 1.

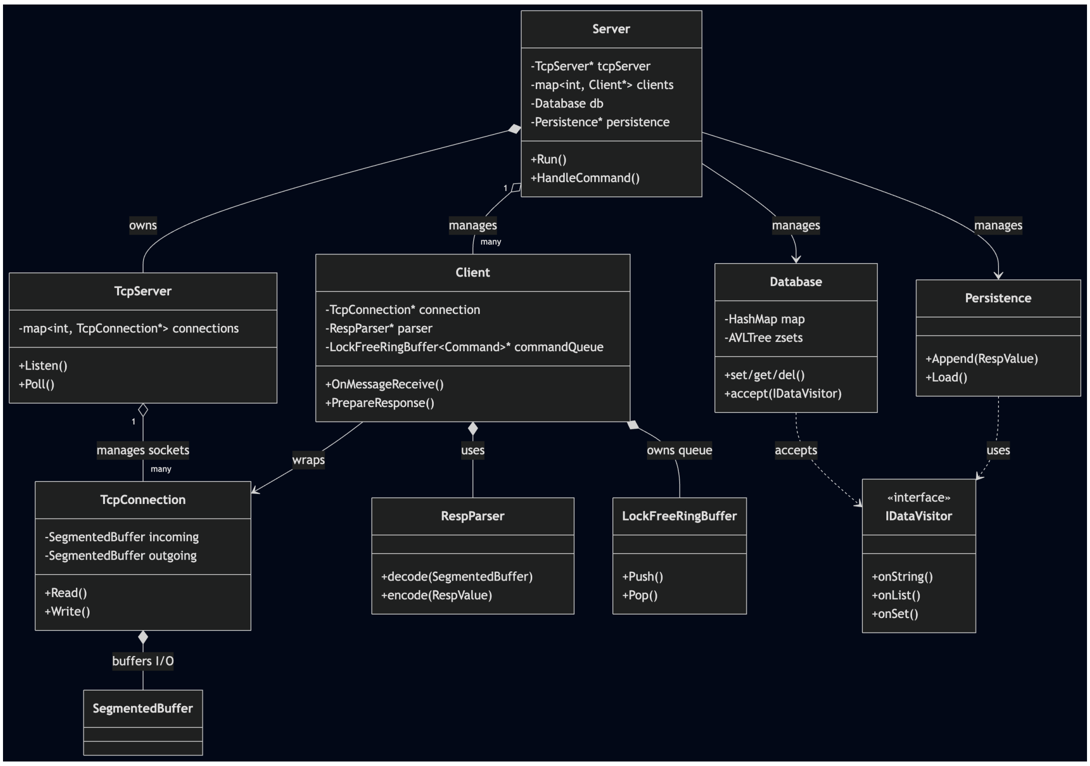

Figure 1: System Architecture Class Diagram

The core components are:

- **Server**: The central coordinator that initializes the networking stack and the database. It manages a collection of `Client` objects.
- **TcpServer & TcpConnection**: Handle the raw socket I/O and event polling (using `poll`).

- **Client**: Encapsulates the state of a connected client, including its protocol parser (`RespParser`) and a command queue.
- **Lock-Free Ring Buffer**: A thread-safe queue used by the `Client` to buffer parsed commands for the main server thread to consume.
- **Database**: The core data store supporting various Redis data types.

3 Core Implementation & Optimizations

3.1 Zero-Copy Data Path

Memory efficiency is achieved by minimizing data copies throughout the stack:

- **SegmentedBuffer**: Custom buffer management that allows direct socket reads into pre-allocated segments.
- **std::string_view**: Used for all key lookups and command parsing, pointing directly into the network segments instead of allocating new heap memory.
- **RespValue Variant**: A memory-optimized `std::variant` that represents all Redis data types with minimal overhead.

3.2 Data Structures

The database utilizes custom intrusive data structures:

- **Intrusive HashMap**: Supports incremental rehashing to prevent latency spikes during resizing.
- **AVL Tree**: Powers Sorted Sets (`ZSET`), maintaining balanced order for efficient range queries.

4 Persistence & The Visitor Pattern

To ensure data durability, the system implements an **Append Only File (AOF)** mechanism.

4.1 Background AOF Rewrite

AOF files can grow indefinitely. Compaction is handled via the `BGREWRITEAOF` command:

- **Forking**: The server forks a child process to perform a point-in-time snapshot of the database state.
- **Visitor Pattern**: The database implements an `accept(IDataVisitor&)` method. This pattern decouples the internal data structure layouts from the persistence serialization logic. The `AofRewriteVisitor` traverses the database and writes a compacted set of RESP commands to a temporary file.

5 Roadmap & Future Work

5.1 Pub/Sub Mechanism

The next major architectural phase will be the implementation of **Pub/Sub**. This will involve:

- A global subscription registry to track client interests in channels.
- Efficient message broadcasting from producers to subscribers.
- Integration with the multi-threaded I/O layer to handle asynchronous notifications.

5.2 Advanced Features

Future developments include RDB binary snapshots, `WATCH/MULTI` transactions, and cluster support.

6 Conclusion

The Redis-from-Scratch project demonstrates that zero-copy lookups and efficient thread-safety patterns can significantly enhance the throughput of networked data stores in C++.