Chirp Project Report

BY

Alexis Kueny
Pierre-Yves Savioz

A Twitter-like Platform Using Redis

1. Introduction and Motivation

Social media applications have become increasingly data-intensive in recent years. With internet access becoming nearly universally accessible worldwide, these platforms must handle massive volumes of information from countless simultaneous users. Twitter (now X), Facebook, and Instagram host billions of daily active users, requiring radical solutions to manage the colossal amounts of data generated through user activity.

Key-value databases offer significant advantages for such applications compared to traditional relational database systems (RDBMS) like SQL:

- Speed: Data is accessed directly through keys, enabling negligible response times
- Horizontal scalability: Easy addition of servers to accommodate growing user bases
- Simplicity: Straightforward data access without complex schemas
- Time-to-live (TTL) capabilities: Automatic data expiration reduces memory usage
- Optimized for frequent reads: Ideal for high-volume access patterns (liking popular posts)

In this lab, we've created Chirp (Compact Hub for Instant Real-time Posting), a simplified Twitter clone that leverages Redis as its backend database to demonstrate the effectiveness of key-value stores for social media applications.

2. Requirements Clarification and Decisions

Core Requirements

According to the assignment, our Chirp application needed to:

1. Store and display user information including followers count
2. Allow users to post short text messages (chirps) in English
3. Track and display the top 5 users with the highest follower counts
4. Maintain a list of the 5 most recent chirps

5. Show the top 5 users with the highest chirp counts
6. Provide a web interface using Streamlit

Assumptions

- Each Twitter user is uniquely identified by their screen_name and id_str
- For educational purposes, we simplified or omitted aspects like authentication, rate limiting, and advanced error handling
- The application is intended for demonstration rather than production deployment
- Only English tweets are processed (filtered by lang == 'en')

Design Decisions

1. Data structure selection:
    a. Redis Hashes for user information and chirps due to their efficiency for storing multiple fields with a single key
    b. Redis Sets for collections of users and chirps
    c. Redis Sorted Sets for rankings (by followers and chirp counts)
    d. Redis Lists for the most recent chirps
2. Key naming conventions:
    a. User data: user:{user_id}
    b. Chirps: chirp:{chirp_id}
    c. User's chirps: user:{user_id}:chirps
    d. Screen name mapping: screen_name:{screen_name}
3. Data processing:
    a. Use of Python's defaultdict to accumulate user statistics
    b. Compressed Twitter data in .bz2 format is decompressed and processed line by line
    c. Only tweets with valid user data are stored

3. Data Modeling as Key-Value

Our Redis data model is designed to optimize for the specific access patterns of a social media application. Here's a detailed breakdown of how we structured the data:

User Model

Key: user:{user_id} Type: Hash Fields: id: "123456789" screen_name: "username" name: "Display Name" followers_count: "1000" chirps_count: "42" created_at: "Wed Mar 13 15:30:12 +0000 2025"

Chirp Model

Key: chirp:{chirp_id} Type: Hash Fields: id: "987654321" user_id: "123456789" text: "This is a sample chirp #keyvalue" created_at: "Sun Mar 30 10:15:23 +0000 2025" timestamp: "1711880123"

Collections

Key: users Type: Set Members: [user_id1, user_id2, ...]

Key: chirps Type: Set Members: [chirp_id1, chirp_id2, ...]

Key: user:{user_id}:chirps Type: Set Members: [chirp_id1, chirp_id2, ...]

Rankings

Key: top_users_by_followers Type: Sorted Set Score: followers_count Members: [user_id1, user_id2, ...]

Key: top_users_by_chirps Type: Sorted Set Score: chirps_count Members: [user_id1, user_id2, ...]

Temporal Data

Key: chirps_by_time Type: Sorted Set Score: timestamp Members: [chirp_id1, chirp_id2, ...]

Key: latest_chirps Type: List Members: [chirp_id1, chirp_id2, ...]

Screen Name Lookup

Key: screen_name:{screen_name} Type: String Value: user_id

This model is optimized for:

- Fast user lookup by ID or screen name
- Efficient chronological retrieval of chirps
- Quick access to rankings for followers and chirp counts
- Atomic incrementing of counters

4. Software Architecture and Functionalities

The Chirp application is built with a modular architecture separating data ingestion, storage, and presentation:

Components

1. Data Ingestion Module (main.py):
    a. Loads compressed Twitter data from .bz2 files
    b. Processes tweets line-by-line using the json library
    c. Filters for English-language tweets with valid user data
    d. Extracts user and chirp information
2. Redis Storage Layer:
    a. Stores user information as Redis Hashes
    b. Manages collections using Sets and Sorted Sets
    c. Maintains temporal data using Lists
    d. Provides lookup capabilities via mapping structures
3. Web Interface (chirp_app.py):
    a. Built with Streamlit for interactive web access
    b. Displays user statistics and recent chirps
    c. Provides tabbed navigation for different views
    d. Allows posting new chirps with user attribution

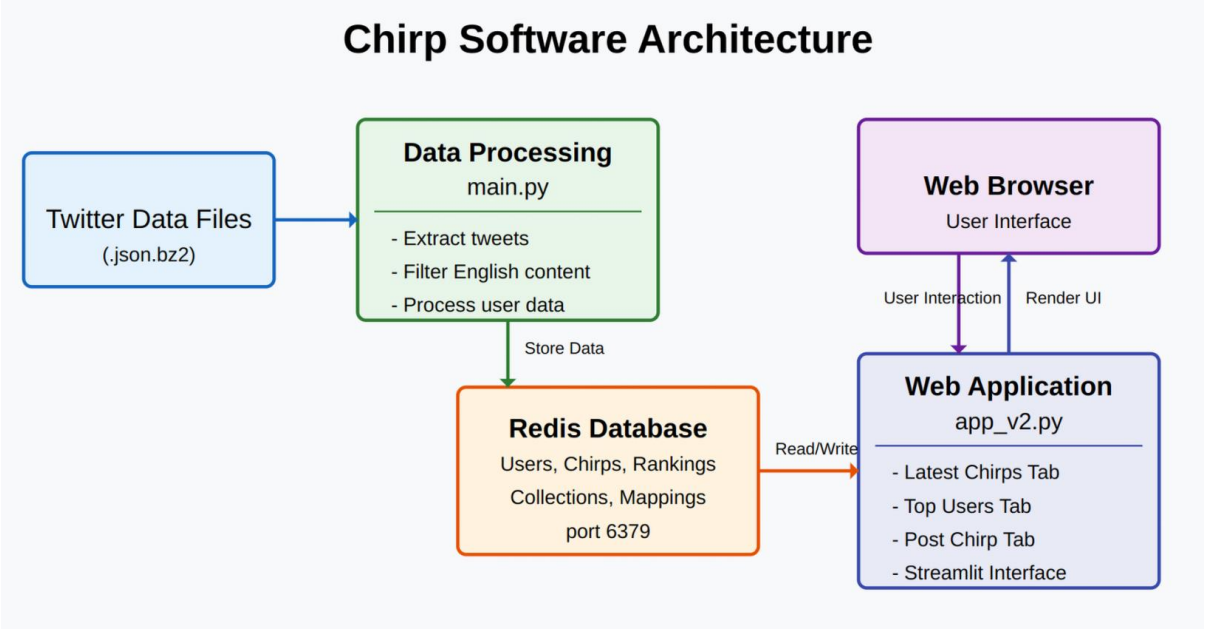
Data Flow

1. Ingestion:
    a. Reads compressed files from the data directory and/or a specified ZIP archive
    b. Extracts and processes JSON tweet data
    c. Filters for English-language tweets
2. Processing:
    a. Extracts user details (ID, screen name, follower count, etc.)
    b. Retrieves chirp content, IDs, and timestamps
    c. Accumulates statistics like chirp counts per user
3. Storage:
    a. Updates Redis with user information using hset
    b. Stores chirp data with relationships to users
    c. Maintains collections and rankings with appropriate Redis data structures

4. Presentation:
    a. Reads from Redis for display in the web interface
    b. Provides tabbed views for latest chirps and top users
    c. Supports new chirp creation with automatic user association

Key Functionalities

1. User Management:
    a. Storage of user profiles with metadata
    b. Screen name to user ID mapping
    c. Ranking by followers and chirp activity
2. Chirp Management:
    a. Chronological chirp storage
    b. Association of chirps with users
    c. Handling of timestamps and sorting
3. Analytics:
    a. Tracking of top users by multiple metrics
    b. Counting of chirps per user
    c. Collection of platform-wide statistics
4. Real-time Interaction:
    a. Creation of new chirps
    b. Automatic updating of user statistics
    c. Refreshing of the UI when data changes



5. Conclusions and Future Work

Conclusions

This project successfully demonstrated how Redis, as a key-value store, can effectively model the core components of a social media application. By structuring user data as Redis Hashes and leveraging various Redis data structures (Sets, Sorted Sets, and Lists), we built a responsive, scalable backend capable of real-time data ingestion and retrieval.

The simplicity of the Redis data model contributed to a clean and efficient implementation, suitable for use cases where low latency and high throughput are critical. Our implementation successfully handled key social media functionalities including user profiles, chirp storage, and basic analytics.

The results included:

- Identification of influential users by follower count
- Measurement of user activity through chirp volume
- Maintenance of a chronological feed of recent chirps
- A functional web interface for viewing and creating content

Future Work

The current implementation could be extended in several ways to more closely resemble a full-featured social media platform like Twitter:

1. Enhanced Redis Data Structures:
    a. Implement Sorted Sets for personalized user timelines
    b. Use Redis Pub/Sub for real-time notifications
    c. Add TTL functionality for session management and temporary data
2. Core Social Features:
    a. Implement the full following/followers system mentioned in the lab requirements
    b. Add hashtag support and trending topics tracking
    c. Create a search functionality for users and chirp content
    d. Enable chirp interactions (likes, rechirps/retweets)
    e. Add threaded conversations and replies
3. Web Application Improvements:
    a. Enhance the Streamlit interface with more interactive elements
    b. Add user profiles with profile pictures and bio information

   c. Implement a more Twitter-like timeline with infinite scrolling
   d. Add engagement buttons (like, reply, rechirp) to each chirp
   e. Create a direct messaging feature
   f. Add media attachment support
   g. Implement a notification system
   h. Include a discover tab for finding new users to follow
4. Authentication and Security:
   a. Add user registration and login functionality
   b. Implement session management
   c. Add privacy settings for user accounts

This project has demonstrated how key-value stores like Redis can effectively power social media applications by prioritizing speed, scalability, and flexibility in data access patterns. By implementing some of these additional features, the Chirp application could evolve into a more comprehensive social platform while continuing to leverage the performance advantages of Redis.