Unacademy

# REDIS IMPLEMENTATION

## Overview

The project contains all the functionalities mentioned in the *GOALS* from the problem statement given. It contains **REDIS** implementation, with few of them having same time complexity as that in REDIS. The whole Code is written in **python.**

**Steps**:

- First parse the input and get the arguments.
- Execute commands according to input and present the corresponding output.
- Upon exiting, we save the output in terms of **pickle** file (storage.pickle).
- Upon starting the code, we first check if a pickle file already exists. If yes, load it. Otherwise create a new file for storing.

## Language

| | | |
|---|---|---|
| Language used | : | **Python** |
| Reason | : | Lot of input needs to be processed. Storing cache and logs into a file would be convenient. Python has inbuilt many inbuilt modules to read inputs and to store data as **Pickle**. |

## Improvements

- Implement Clients and Servers setting by using multiple threads and locks. This allows **multiple** Servers with **multiple** clients accessing the data.
- Can try to improve time complexity of one function ZRANGE using some distributed methods for faster retrieval of information.

## Data Structures Used

- *Dictionary* : To tackle commands like **GET** and **SET** and **EXPIRE.**
  Though lookup time for a key in python set in the worst case is O(n) (due to collisions), ours runs in O(1). We take advantage of **try-catch** in python. Hence, if an element is not found, it goes into the catch function. Else it is just printed directly.
- *SortedSet* : To handle **ZADD, ZRANK** and **ZRANGE.**
  *Sortedset* though not in basic python libraries, is a very simple and effective implementation using lists and binary trees.
- *Pickle* : To store intermediate states of the data structures.

### Explanation and reason for using Sorted Sets

Sets in python are not sorted. They are just a hashmap*. Sortedsets* are a simple implementation of Lists and Binary trees. Brief overview of SortedSets is as follows:

1. Sorted sets use multiple sublists(Each of length L < max_lenght_of_sublist, generally 1000).
2. Each of these sublists are sorted. Hence while adding a new element, the exact sublist for insertion is searched so that it is still sorted.
3. The maximum value in each sublist is stored in another list, hence extra memory is used. The exact sublist for insertion of a new element can be searched in this list reducing search complexity to O(logN), if N sublists are present.
4. The element is inserted in this sublist we found in O(K), if K is the length of the sublist.
   **NOTE** : Though 1000 seems to be an increase of order of magnitude three times, the documentation says it is faster than many tree based methods and 100 performs better on extremely large data.
5. It also maintains a pairwise number of elements in the sublist and forms a binary tree. This can be used to traverse to find the value at an index. Traversing the tree takes O(logN) time incase N elements exist.

   For further implementation details, click here

**Our time complexities compared to REDIS original complexities**

| Method | Our Time complexity | REDIS Time complexity |
|---|---|---|
| GET | O(1) | O(1) |
| SET | O(1) | O(1) |
| EXPIRY | O(1) | O(1) |
| ZADD | O(logN) | O(logN) |
| ZRANK | O(logN) | O(logN) |
| ZRANGE | O(MlogN) | O(M + logN) |

Difference lies only in ZRANGE. All other functions are on par with the REDIS implementation.

## Multi-threaded operations

Our implementation **allows threads** for **EXPIRE** command to set time for expiry. This is done using the threading library in python. We set a particular time for the thread and once it reaches the limit, the thread ends. We also used threads to handle multiple users and a single server. Changes made by one client are reflected across all clients through the server. We have used a threading library for this.