ASSIGNMENT 2: A Key-value Store Server (EXTENSION)

SHUAI KE - 1609628

April 24, 2018

1 Introduction

The extension part implements the skip list [1] in *skiplist.h* to fix the memory problems in *kv.h* and improve the complexity in operating key-value store from $o(n)$ to $o(logn)$.

The implementation of kv-store in *kv.h* has some tiny memory problems, **memory leak**. There are three points which are fixed in *skiplist.c* as follows:

- In the update function (updateItem), it overrides the pointer of the value and does not release the memory of the previous value [line 64 in kv.c].
- In the delete function (deleteItem), it releases the memory of the value [line 73 in kv.c], but it does not release the memory of the key.
- There is no function for release the memory of the whole kv-store after the server is closed.

Skip list [1] provides an efficient strategy for the key-value store. The complexity of the operation in the skip list, like search, insertion, and deletion, is $o(logn)$.

2 Implementation

2.1 Overall

The extensional code is using the APIs in *skiplist.h* to operate the kv-store, and the memory management for the key and value is doing in the skip list, which replaces doing in the out of kv-store. The memory of the whole kv-store also is deallocated [lines 617-634].

2.2 Global variables

There is a added global variable Items which is the skip list instance for key-value store.

2.3 The structure of skip list

A skip list is a variant linked list which the node has different levels. The bottom layer is an ordinary ordered linked list. Furthermore, the level of each node is depending on the *random_level* function [lines 22-28 in skiplist.c] during the insert operation. The *Figure 1* is shown the structure of skip list:

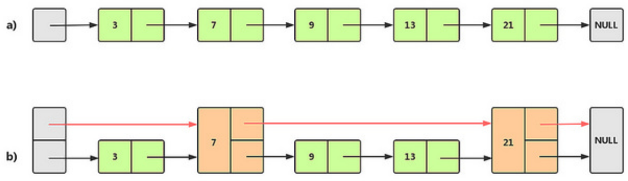

Figure 1: The Structure of Skip List

$a)$ is an ordinary ordered linked list and $b)$ shows the skip list with maximum level is 2. The nodes of level 2 form a new linked list, which as the quick path. When we need to search $Key = 13$, it searches the nodes of level 2 firstly to determine the scope, then finds it in the level 1. The *Figure 2* shows the procedure.

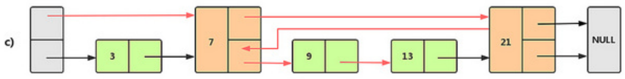

Figure 2: The search path

2.4 APIs

The implementation of the key APIs in the skip list is shown as follows (the code is in skiplist.c):

- init_skiplist: initialize the skip list instance, the return value is a pointer of skip list or NULL [lines 30-43].
- search_skiplist: search for the value stored under the key [lines 46-68].
- insert_skiplist: insert a new item under the given key, the memory for storing the key and value are allocated in this function [lines 94-139].
- update_skiplist: update a new item under the given key, the old value will be overridden by the new value [lines 142-168].
- delete_skiplist: delete an item, including to free memory of the key and value [lines 171-203].
- free_skiplist: free the memory of the whole skip list instance [lines 212-225].

3 The reasons for using skip list

The skip list is a realizable and efficient data structure. The average complexity is $o(logn)$, the worst situation is $o(n)$ [1]. Redis, an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker, which also use the skip list to implement the sorted set.

4 References

1. W. Pugh. Skip lists: a probabilistic alternative to balanced trees. In *WADS*, 1989.