Simplified Memcached

-In memory Key value store

Linsen Ma, Yuyang Zhang

What is Memcached

- Open source, high-performance, distributed memory object caching system.
  - Intend to be used in speeding up dynamic web applications by alleviating database load.
  - Cache data in RAM to reduce the calling times for external data source like database.
- An in-memory key-value store for small chunks of arbitrary data from results of databases, API calls or page rendering.
  - client-server architecture:
    - Server maintains key-value associative array.
    - Clients populate and query the array.
  - Use least Recently Used replacement to purge old data.
  - O(1)
    - All commands are fast and lock-friendly.
    - Queries time on low-end machine < 1ms.
    - High-end servers can serve millions of keys per second.

Memcached utilization case

Straight database queries without memcached:

```
function get_foo(int userid)
    data = db_select("SELECT * FROM users WHERE userid = ?", userid)
    return data
```

Database queries with memcached:

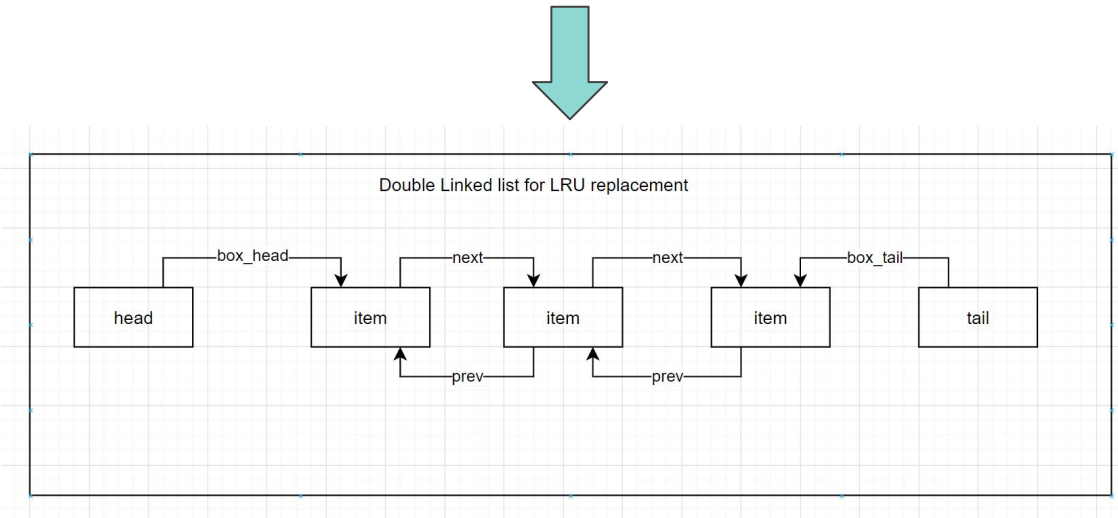
```
function get_foo(int userid)
    /* first try the cache */
    data = memcached_fetch("userrow:" + userid)
    if not data
        /* not found : request database */
        data = db_select("SELECT * FROM users WHERE userid = ?", userid)
        /* then store in cache until next get */
        memcached_add("userrow:" + userid, data)
    end

    return data
```

Our simplified memcached

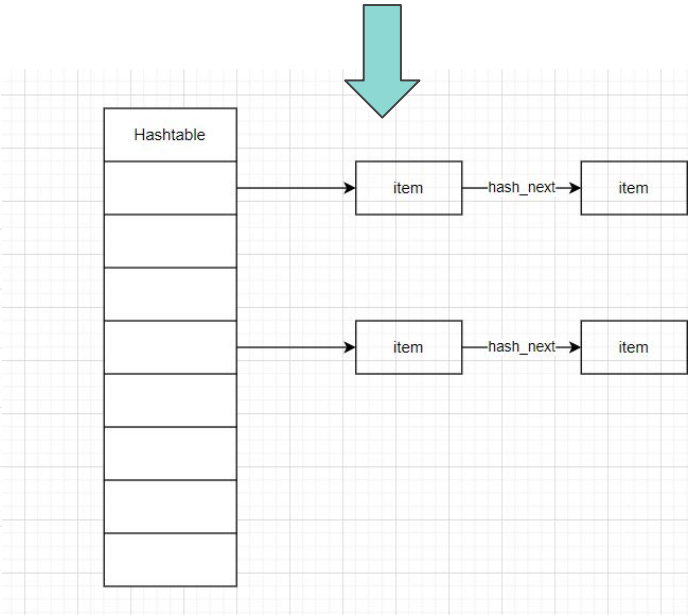
- A in memory key-value store
- Support multiple operations:
  - Get, Put, Set, Delete, Container Status
  - Use mutex for concurrency control
- Autorehashing is supported
  - Automatically rehash the hashtable to balance the hash table size.
- LRU replacement is used to maintain container size
  - Double linked list is used to support LRU
  - Evict oldest item when the container size is full
- Easily further extension
  - Slab
  - Timing mechanism

Basic structure

The Double linked list



The hashtable



Basic structure

```c
typedef struct item_
{

    struct item_ *hash_next;
    struct item_ *next;
    struct item_ *prev;

    int itemsize;

    char * key;
    char * value;

}item;
```

The Basic item structure

The Container structure

```c
typedef struct box_
{

    item** hashtable;
    item* head;
    item* tail;


    int hashpower;
    unsigned long long hashmask;
    unsigned long long maxsize;
    unsigned long long cursize;
    unsigned long long tablesize;


    int item_num;
    int rehash_time;

    int get_hit;
    int get_miss;

    int put_hit;
    int put_miss;

    int set_hit;
    int set_miss;

    int delete_hit;
    int delete_miss;



}box;
```

Operation realization

Get Operation:

1. Calculate the hash value.
2. Get the Table Index.
3. Check the items pointed by the calculated pointer.
4. Go through the items in the linked list until finding matched item or NULL.
5. Update the container status.
6. Return the Find item.

Put Operation:

1. Check if the container if full.
    a. Iteratively evict the last item in the container until the container size will not exceed the max size after adding the new item
2. Calculate the hash value.
3. Get the Table Index.
4. Put the item to be the first item in the specified linked list.
5. Put the item to be the head of the LRU list.
6. Check if the rehashing is needed
    a. If the item number is > 150% of the table size
7. Update the container status.

Set Operation:

1. Check if the item with given key is existing.
2. If exist, check if the newly added data value will make the container full.
    a. If full, do eviction.
    b. If not, replace the old value with new value.
3. If not exist, do Put operation.
4. Put the item to be the head of the LRU list.
5. Update the container status.

Operation realization

Delete Operation:

1. Check if the item with given key is existing.
    a. If exist, delete it in the hashtable and the double linked list
    b. If not, update the container status and return
2. Update the container status.

Status Operation:

1. Print the information of the container
    a. Current_size
    b. Item_num
    c. Command hit/miss num

Testing results

```
----------The BOX----------
Maxsize    is: 64MB
Cursize    is: 0MB
Tablesize  is: 65536
Rehash num is: 0
Item num   is: 0

Get hit     : 100
Get miss    : 0

Put hit     : 100
Put miss    : 0

Set hit     : 100
Set miss    : 0

Delete hit  : 100
Delete miss : 0
----------The BOX----------
```

Small test:
100 Put
100 Get
100 Set
100 Delete
No rehashing
No eviction

Large test:
1000000 Put
20000 Get
4 rehashing
10472 eviction

```
----------The BOX----------
Maxsize    is: 64MB
Cursize    is: 63MB
Tablesize  is: 1048576
Rehash num is: 4
Item num   is: 989528

Get hit     : 9528
Get miss    : 10472

Put hit     : 1000000
Put miss    : 0

Set hit     : 0
Set miss    : 0

Delete hit  : 0
Delete miss : 0
----------The BOX----------
```

Further Enhancement

1. Slab operation
    a. Item with different data size will be assigned to different slab.
    b. Reduce the problem of fragmentation.
2. Timing mechanism
    a. Record the Enter/Modify time for each item.
    b. Evict item with expired time ticket.

Thank you!