Operating System Project

An implementation of server-client database using non-blocking operations

Julmy, S., Papinutto, M. & Veillard, S.

University of Fribourg

30 Septembre 2018

Table of contents

Multi-threaded Server

We selected a multithreaded server as :

- Allows to access same data structure
- light weight as compared to multi-process server
- easier to implement

To use mutex or not to use mutex

Our lock-free implementation allowed us to omit mutex.

- No need to prioritise read or write operations
- No deadlock problems
- All clients have the same right to access the data structure

Atomic calls

Non-blocking operations use atomic operations in order to perform multiple operation in one clock cycle. With GCC, we can access them with some specific compiler functions [1] :

```
type __sync_fetch_and_add(type *ptr, type value, ...);
bool __sync_bool_compare_and_swap(
  type* ptr,
  type old_v,
  type new_v,
...);
```

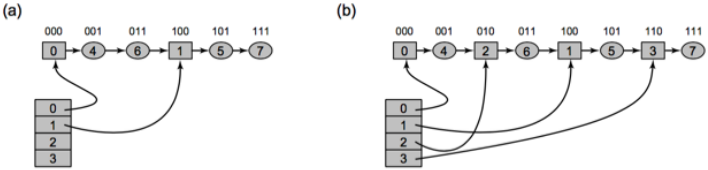
Such operations are in a way an acquire lock - operate - realease lock in only one CPU cycle.

[1] https://gcc.gnu.org/onlinedocs/gcc-4.1.0/gcc/Atomic-Builtins.html

Reversed Split-Ordered Hash-Set

This implementation offers a rapid access to the data but might require slightly more memory than other data structures.

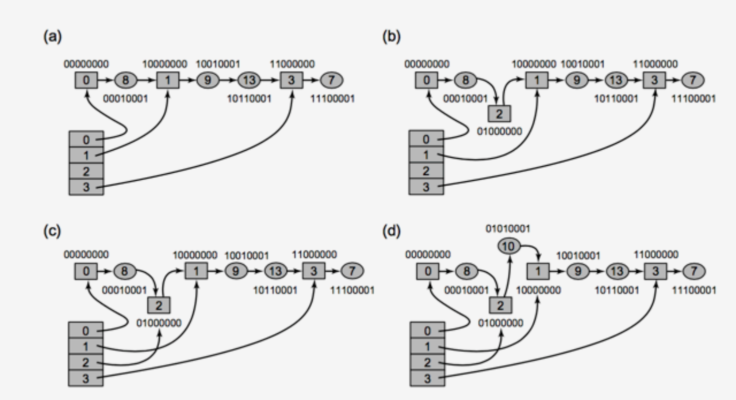
- Buckets are linked to a stack as the list grows supplemental bucket references are added so that buckets is kept small.
- Require to set up a sentinel bucket in order to avoid "corner case" that occurs when deleting a reference by a bucket reference.
- The sentinel bucket is never deleted.



FIGURE

Operation add in this Hash-Set

Scheme of the procedure that adds the key 10 to the lock-free hash-set



Program Basic Usage

Server usage	
`TCP Port`	5000 (can be changed in file)
`./server`	server start

Client Start	
`./ client <server ip address>`	client start
`./ client -option <server ip address>`	client start with options

Client Usage and Commands

Client Start with options	

`-?`
`-h`
`-help`
	client command help

`-f <file>`
`-file <file>`
	client start and execute commands in the file

`-F <file1> ... <fileN>`
`-files <file1> ... <fileN>`
	client start and execute commands in the files

Commands in interactive CLI	
`add <value> or add <key> <value>`	add a value to the database
`ls`	list content (unordered)
`read_v <key>`	read value from key
`read_k <value>`	read key from value
`rm_v <key>`	delete value from key
`rm_k <value>`	delete value from key
`update_kv <value> <newvalue>`	update an entry

Demo

DEMO

Tests scenarios

We tested the following scenarios :

- Scenario with collisions : operations that can collide (a client delete a value before another access it).
- Scenario without collision : operations that are ordered so that no collision can occur.
- Scenario with many clients : several clients with a similar scenario as no-collisions.

Collision scenarios

11 clients and 28 commands (308 operations in total)

	Add	Read	Delete
Number of errors	0	22	0
Percentage of errors	0%	7.14%	0%

No-collision scenarios

8 clients and 2700 commands (21600 operations in total)

	Add	Read	Delete
Number of errors	0	0	0
Percentage of errors	0%	0%	0%

Many clients scenarios

32 clients and 300 commands (9600 operations in total)

	Add	Read	Delete
Number of errors	0	0	0
Percentage of errors	0%	0%	0%

This last test required more time than the previous one despite the fact that it has half fewer operations

Thank’s for your attention !