Operating System Project

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

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

30 Septembre 2018

UNI
FR

Sommaire

Multithreaded Server

We selected a multi-threaded 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 not to use Mutex

- No need to prioritize read or write operations
- No dreadlock problems
- All clients have the same right to access the data structure at any time

non-blocking operations

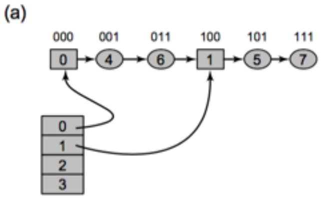
Non-blocking operations use atomic operations allowing to perform several action at each clock cycle
As the actions are perform in one cycle it is not crucial to wait for the operation to be completed
In c atomic operations functionnality such as CompareAndSet can be done by "stealing" a bit from a pointer using bit-wise operators to extract the pointer and the mark from a single word

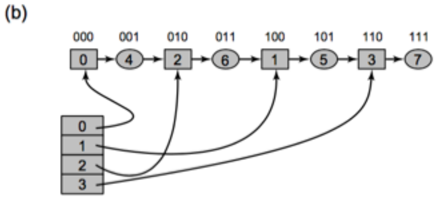
- Swap values exemple

Reversed Split-Ordered Hash-Set

This implementation offers a rapid access to the data but might require slightly more memory that other data-structure

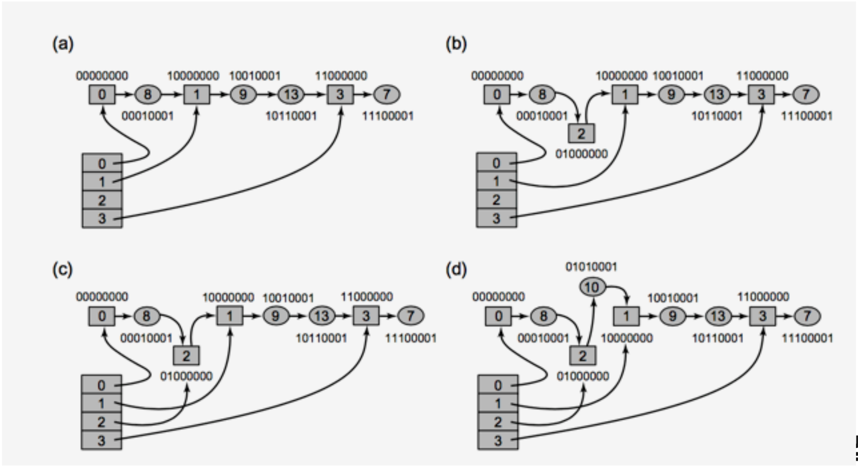
- Buckets are linked to a stack as the list grows supplemental buckets references are added so that buckets is keep small
- Require to set up sentinel bucket in order to avoid "corner case" that occurs when deleting a reference by a bucket reference
- The sentinel bucket is never deleted





Operation add in this Hash-Set

Scheme of the procedure that add 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 Special Starts

- ? -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 GUI

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 scenarii

We tested the following scenarios :

- Collisions scenario : Operations that can collide (a client delete a value before another access it)
- No-Collisions scenario : Operations that are order so that no collision can occurs
- Many client : several client with similar scenario as no-collisions

Collision scenario

We tested 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 scenario

We tested 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 scenario

We tested 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 it has half less operations

Thank you for your attention !