Concurrent Tree

Sharanjeet Singh Mago

The concurrent tree implements a key-value store using fine-grained synchronizations (hand-over-hand locking). The concurrent tree supports 3 functionalities

1. Put - To insert a key-value pair in the concurrent tree
2. Get - To get the value for a given key
3. Range - To get the key-value pairs between the given range of keys

All three functions are implemented using recursive calls. The hand-over-hand locking is achieved by locking the next node before unlocking the current node. For example, if we are adding a key-value pair in a tree. At the first iteration, we check if we are at the root node of the tree. If yes, we lock the tree and check if the tree is empty. If the tree is empty we insert a new node, unlock the tree and return from the function. If the tree is not empty, we lock the first node and unlock the tree and proceed forward. Next, we check if the key to be entered is towards the left of the root node or the right of the root node. Accordingly, we check if the next node exists or not. If the next node doesn't exist we create a new node and place it at the respective position. If the next node exists, we lock the next node and unlock the root node and then make a recursive call. At the second iteration, we are not at the root node and thus the compiler skips the first if condition and directly check if the new node belongs to the left or right of the current node.

The program takes input from the user for the number of threads to run and the number of iterations each thread should run. For instance, if the user enters 10 threads and 20 iterations. A total of 200 key-value pairs will be attempted to be inserted in the tree. If a key-value pair with the same key already exists in the tree, the program keeps the original value and ignores the new key-value pair.

Each thread runs a for loop for the number of iterations entered by the user and generates random keys and values. A put function call then attempts to insert the key-value pair in the tree. Each put function call is followed a get function call to get the key-value pair at the same random key.

The code writes all the key-value pairs within the provided range in an output file.

A bash script is provided which runs the code with different threads and iterations and in different settings: normal settings, high contention settings, and low contention settings

High Contention Settings

To run the code in high contention settings add “-c 1” during runtime.
In the high contention settings, before spawning of the threads, a tree is already built and then in the threads, keys are only generated between 0 and 10. And since the total iterations are 100x1000. The program tries to enter the same key again and again.

Command -> ./contree -t 100 -i 1000 -c 1

Low Contention Settings

To run the code in low contention settings add “-c 0” during runtime.
In the low contention settings also, a tree is built before the spawning of threads so that the results of the two are comparable. But in the low contention settings, the keys generated are not limited between 0 and 10.

Command -> ./contree -t 100 -i 1000 -c 0

Compilation instructions

make : to compile the project - generates contree named executable
make test : to compile the unit testing file - generates utest named executable
make clean : to remove all the executables generated

Execution Instructions

./contree --name : prints name
./contree -t <number of threads> -i <number of iterations> -r <range lower bound> <range upper bound>

./script.sh : to run the test script with different settings

Files Submitted

trees_lock.h - header file containing all the necessary functions
trees_lock.cpp - file containing all the function definitions for put, get, range, etc.
main.cpp - Main project source file
unit_test.cpp - Unit testing source file
Makefile - Makefile to compile the project
script.sh - script to run and test the project with different settings
out.txt - output file generated by the code
A separate folder is included for implementation with reader-writer locks