Concurrent Binary Search Tree

Developer: Steve Kenedy

Description of Algorithm:

Concurrent unbalanced binary search tree is implemented using hand over locking mechanism. There are two versions of handover locking used here. 1. Mutex and 2. Reader-writer lock

Low contention test case:
The root of the BST is selected to be the mid point of the range of key which are planned to be inserted by the threads. The threads insert unique values into the tree and as the root is chosen as mid value, it is made sure that the constructed tree is a balanced BST. As the elements are inserted uniquely over a range in a BST, there would be low contention as the insertion would take **varying paths from the root**.

High Contention Test case:
A skewed BST is constructed with elements only on one side of BST.
Say a skewed tree is constructed with elements from 0 to NO_OF_NODES (passed as command line argument). Root being the value given for NO_OF_NODES.

Now all threads always insert same elements 0 to 10 which would be in the bottom of the BST on one side. This will ultimately create high contention on the nodes as all threads traverse through the **same path from root** on one side as they insert the same elements.

Analysis:

Test cases for low and high contention are performed with the same number of threads, the same number of nodes in the BST

Test Cases with varying number of threads
Handover locking:
Low Contention:

No of threads	No of nodes	Execution time (Only insertion)	Execution time (insertion, range query, get)
5	10000	1.30	1.30
10	10000	2.82	3.38
20	10000	8.20	6.14
50	10000	26.57	22.45

High Contention

No of threads	No of nodes	Execution time (Only insertion)	Execution time (insertion, range query, get)
5	10000	4.13	5.29
10	10000	12.23	22.45
20	10000	28.93	47.22
50	10000	82.84	216.41

Read write locking:
Low Contention

No of threads	No of nodes	Execution time (Only insertion)	Execution time (insertion, range query, get)
5	10000	2.54	2.34
10	10000	4.74	4.94
20	10000	14.07	12.23
50	10000	34.93	36.79

High Contention

No of threads	No of nodes	Execution time (Only insertion)	Execution time (insertion, range query, get)
5	10000	7.26	26.79
10	10000	15.29	58.55
20	10000	41.56	104.99
50	10000	186.83	470.00

Test Cases with varying number of nodes in BST
Mutex lock
Low Contention:

No of threads	No of nodes	Execution time (Only insertion)	Execution time (insertion, range query, get)
5	5000	0.317	0.312
5	10000	1.233	1.24

High Contention

No of threads	No of nodes	Execution time (Only insertion)	Execution time (insertion, range query, get)
5	5000	0.79	1.01
5	10000	2.90	7.13

Read write lock
Low Contention:

No of threads	No of nodes	Execution time (Only insertion)	Execution time (insertion, range query, get)
5	5000	0.46	0.43
5	10000	1.73	1.92

High Contention

No of threads	No of nodes	Execution time (Only insertion)	Execution time (insertion, range query, get)
5	5000	1.12	7.57
5	10000	7.26	27.20

Result Interpretation:
It can be found that the time taken for high contention test cases are higher when compared to low contention test cases for mutex/read write handover locking.

It is found that the time taken for read write lock is higher than mutex lock and it is because of the fact that insert operation dominates when compared to other operations.
It was also found that read write lock takes less time when there are multiple threads performing get or range query in a BST as it can allow multiple readers

Code organization and Files submitted:

Bst.c
This file contains the functions for BST operation namely get, insert, range query for mutex handover locking and reader writing handover locking.

Bst.h
This file contains the structure definition relating to bst tree

Main.c
This file helps in parsing the command line parameters and spawns the threads
Test_script.sh
This bash script runs a shell script with many test cases with command line various input

Compilation instructions:

Make mutex
Builds the executable with mutex handover locking

Make rwlock
Builds the executable with reader writer handover locking

Execution instruction:

./bst [Test case type] [No of threads] [No of BST nodes] [Test case]

Test case type - can pass **HIGH_CONTENTION or LOW_CONTENTION or UNIT_TEST**
UNIT_TEST does unit test for BST operations in a single threaded environment (test case parameter not required when UNIT_TEST is passed)

No of threads - Can pass a maximum of 100 threads
No of BST nodes - Pass the number of nodes to be inserted by each thread
Test case - 1 or 2
1 - many threads insert nodes into the bst and gives the execution time
2 - Threads does insert, get and range query simultaneously

Eg:

./bst LOW_CONTENTION 5 10000 1

./bst HIGH_CONTENTION 10 1000 1

./bst UNIT_TEST 1 10000
Run the Automated shell script to test
Bash test_script.sh

Further notes:
Range query is called for all test cases with the min and max values of int64_t.
Also the values obtained from range query is not printed on the console. It is just copied to a vector and can be printed if needed.