15-618 Milestone Report
Concurrent Binary Search Trees

Yifan Cao (Andrew ID: yifanca2), Yahui Liang (Andrew ID: yahuil)

November 2021

1 Project Web Page URL

`https://yahuiliang.github.io/15618-yahuiyifan-team/`

2 Schedule

The up-to-date and revised schedule is shown below:

Week	Date	Tasks	Status
1	11.1 - 11.7	Proposal Literature review Coarse-grained lock version BST	Completed
2	11.8 - 11.14	Fine-grained lock version BST Lock-free version BST	Fine-grained version finished (Memory release problem not solved) Lock-free version started
3	11.15 - 11.21	Lock-free version BST Transactional memory version BST Milestone report	Lock-free version started Transactional memory version not started yet
4	11.22 - 11.24	Hazard pointer implementation (Yahui) Lock-free version BST (Yifan)	
	11.25 - 11.28	Lock-free version BST (Yifan) (Transactional memory version BST) (Yahui)	
5	11.29 - 12.1	Experiment workload generation (Yifan)	
	12.2 - 12.5	Experiments and analysis (Yifan, Yahui)	
6	12.6 - 12.8	Experiments and analysis Final report	
	12.9 - 12.10	Poster	

We are behind schedule for the time being. Part of the reason is that we just had exam, and we did not take it into consideration when making the schedule. However, the main reason is that we met the problem of safe release of memory. When erasing a node, we cannot immediately release the memory of the node because there may be other threads accessing the memory. The paper we referenced did not describe how to do garbage collection in detail. Locking the whole tree and do garbage collection periodically will hurt performance too much. We also tried using smart shared pointer in C++, but it is too expensive, and thus it also hurts the performance a lot. We decided to implement the hazard pointer method, which will be implemented earlier next week. We adjusted our schedule, and implementing the transactional memory version of BST will be a depending task depending on our future progress.

3 Summary

We have finished literature review about fine-grained, lock-free, and transactional memory versions of BST. We have implemented the coarse-grained lock version BST. It is based on the simple single-thread BST and adding a lock for the whole tree. We have implemented the fine-grained lock version BST. It is mainly based on the idea of hand-over-hand lock. For the erase operation, the paper we referenced adopted the method to rotate the node to be erased to leaf position if it is not at the first place so that the tree may be more balanced and benefit future operations on tree. We have finished implementing the basic operations while releasing memory only when destructing the tree due to the safe memory release problem caused by multiple threads. However, this implementation cannot scale to support 100,000 erases. Therefore, we have been seeking ways to resolve the safe memory release problem, and decided to implement hazard pointer.

4 Goal and Deliverable

Plan to achieve

1. Implement the tree with coarse-grained lock supported
2. Implement the tree with fine-grained lock supported
3. Implement the tree without locking mechanism
4. Generate different insertion and deletion workloads
5. Verify the correctness of three implementations
6. Carry out experiments on different implementations of BST with different thread count and workloads
7. Analyze performance, pros and cons of each BST implementation

Hope to achieve

1. Transactional memory based concurrent tree implementation
2. Fine-grained tree balancing
3. Lock-free tree balancing

We have reached the first goal and almost reached the second goal. We also achieved the fifth goal for coarse-grained lock and current fine-grained lock versions of BST. We moved the goal to implement the transactional memory version BST to hope to achieve goal since we spent more time on solving the safe memory release problem. We also achieved the fine-grained tree balancing goal in the hope to achieve goals to some extent. The algorithm of erasing we chose for fine-grained tree is to keep rotating the tree until the node to be erased is a leaf node. Rotating will likely balance the tree, although the balancing may not be perfect since we are not rotating based on height difference.

What to show

We plan to show several speedup graphs during our post session. The speedup graphs will be about Speedupof different BST implementations vs. thread count when using different workloads. There will be one graph foreach workload. These speedup graphs can help people know which BST performs best under different scenarios.

5 Preliminary Results

When deciding whether we can use C++ smart shared pointer to resolve the safe memory release problem, we tested time needed for the fine-grained lock BST implemented by shared pointer to insert and search 100,000 different elements, and the result is 2.42 seconds. We then tested time needed for the coarse-grained lock BST to insert and search 100,000 different elements, and the result is 1.542 seconds. Therefore, we found out that shared pointers are too expensive to use and will hurt the performance so bad that the fine-grained lock BST will even be slower than the coarse-grained lock BST. We think the reason is that shared pointer objects will need to maintain the reference count of the memory location. When different threads are accessing the same memory location, the reference count will be read and written by different threads, causing frequent invalidations to maintain cache coherence and much internal communication traffic, so the overhead of using shared pointers is very high. Therefore, we decided that we cannot use shared pointer, but will implement the hazard pointer method.

6 Concerns

The main concern we have now is how to implement hazard pointer efficiently. The hazard pointer will be used both in fine-grained lock BST and lock-free BST, and after implementing it, we think we can adjust fine-grained lock BST and implement lock-free BST quite quickly.