Lock-Free Priority Queue Based on Multi-Dimensional Linked Lists

Juntao Luo Yumin Chen

SUMMARY

We are going to investigate the performance of a concurrent lock-free priority queue based on multi-dimensional linked lists, which guarantees the worst-case sequential search time of O(logN) .

URL

https://juntaoluo.github.io/618-Project/Proposal

BACKGROUND

Scalable concurrent priority queues, which are pivotal to topics such as the realization of parallelizing search algorithms, priority task scheduling and discrete event simulation, has been a research topic for many years. INSERT and DELELETEMIN are two canonical operations of a priority queue. In sequential execution scenarios, priority queues can be implemented on top of balanced search trees or array-based binary heaps. However, these two implementations both encounter bottlenecks in concurrent situations for maintaining a consistent global data structure. Recently, researchers attempted to solve this problem by applying skiplists and attained a great breakthrough. The skiplist solution organized several linked lists into different levels to eliminate the need of rebalancing and at the same time separate the memory use within different parts of the structure, which allows concurrent access to the data structure. However, the solution is still deficient in some regards. Both of the operations are restricted by the dependency of data within the structure and limits the overall throughput.

As a performance baseline, we will implement a fine-grained priority queue[2]. Inspired by the ideas of [1], we will investigate the performance of a concurrent lock-free priority queue based on MDList that guarantees a worst-case search time of O(logN). Based on MDList, we are going to implement the concurrent insertion with two steps: node splicing and child adoption[Figure 1], which only updates at most two consecutive nodes. For the deleteMin operation, we will apply both logical and physical deletion while maintaining a deletion stack to provide the operation the information about the position of the next smallest node to reduce node traversal[Figure 2]. Finally, we will analyze the performance of the MDList concurrent lock-free priority queue given variations in parameters such as the number of threads and the number of dimensions of the linked list using various traffic loads. Finally we will benchmark the performance against some

well-established concurrent lock-free priority queue implementation, if applicable, such as TBBPQ by intel and LJPQ by Herlihy and Shavit[3] on NUMA systems.

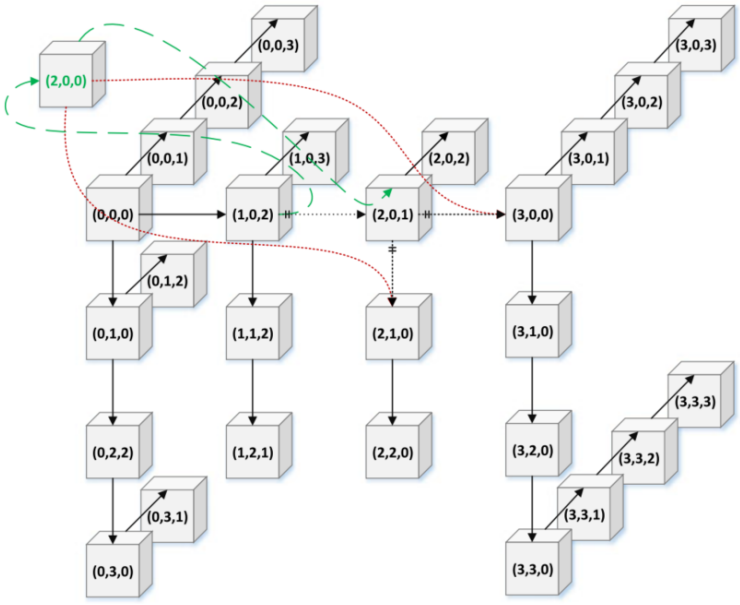

Figure 1: Insert operation in a 3DList

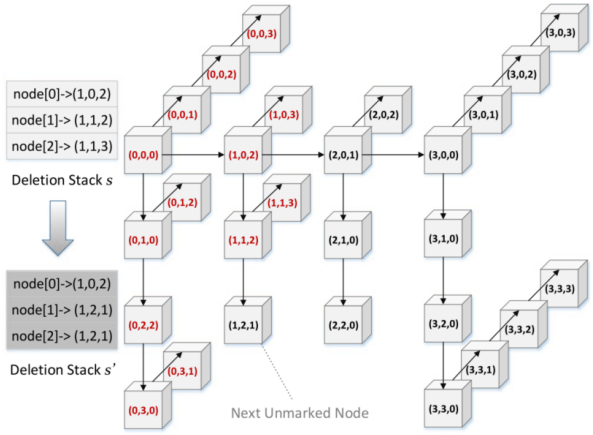

Figure 2a: Logical Deletion

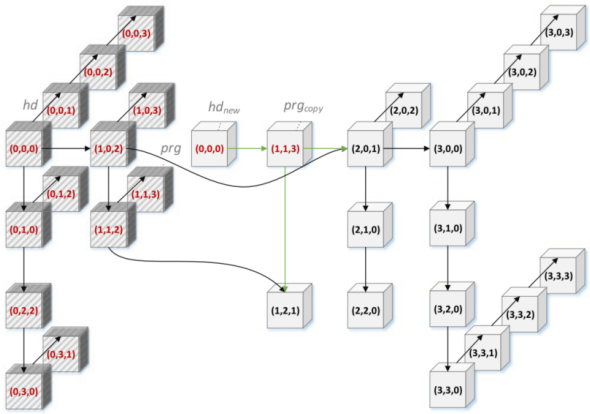

Figure 2b: Physical Deletion

THE CHALLENGE

- Firstly, to decouple different parts of the priority queue to make parallelism possible, we need to maintain a complicated MDList structure and establish a delicate mechanism for the INSERT and DELETE operations. The efforts include conducting lazy modification, preserving multiple flags to indicate the current status of the nodes and building extra data structures, such as stacks, to reduce duplicative traversal over the nodes.
- For the correctness and performance evaluation part, we need to develop a benchmarking script and even implement our own version of sequential and skiplist

priority queue to compare the operation efficiency of INSERT and DELETEMIN operations for different versions of priority queue.
- There is also tuning work involved. The overall performance of the priority queue is highly related to the number of dimensions we chose and the number of threads. We need to explore as much as possible to find the relationship between these two parameters.
- We would like to use the concepts we learned in lecture to see if we can identify additional improvements in the MDList implementation of priority queue.

RESOURCES

For the code base, we will build the data structure from scratch with the help of pseudo code provided in [1]. For parallel machine resources, we plan to gather performance data on the PSC Bridges-2 RM.

GOALS AND DELIVERABLES

- PLAN TO ACHIEVE
    - Implement a fine-grained concurrent priority queue
    - Implement a concurrent lock-free priority queue based on multi-dimensional linked lists in C++, which theoretically can achieve an average of 50% speedup over other versions of priority queue[1].
    - Implement a benchmarking script to evaluate the performance of the concurrent lock-free priority queue and come up with an analysis report regarding the influence of both the dimensions and number of threads on the throughput
- HOPE TO ACHIEVE
    - Compare the performance of different versions of concurrent priority queue(e.g. TBBPQ and LJPQ) with the MDList priority queue
    - Improve the concurrent lock-free priority queue based on profiling statistics, such as memory access and cache management using concepts learned in the course
- Demo plan to show at poster session
    - Graphs displaying the comparison of the performance of fine-grained concurrent priority_queue and the MDList lock-free concurrent priority_queue across different threads, and different INSERT / DELETEMIN ratios.
    - Graphs displaying the lock-free concurrent priority queue's performance across different workload ratio(e.g. 50% of INSERT, 75% of INSERT), different number of threads and different number of dimensions
    - The comparison of skiplist lock-free concurrent priority queue and MDList lock-free concurrent priority queue across different number of threads

PLATFORM CHOICE

We will implement the priority_queue in C++ and develop the benchmark scripts in Python. Experiments will be conducted on PSC machines because it can easily scale from 1 core to 256 cores allowing for more comprehensive study into the performance of the data structure when scaling to larger systems. We also chose this platform due to our familiarity and the reliability of performance statistics since we can guarantee exclusive access to a node for benchmarking.

SCHEDULE

Week	Task
Nov. 7 - Nov. 13	Finish project proposal and study related research paper thoroughly
Nov. 14 - Nov. 20	Implement the fine-grained concurrent priority queue, build Data Structures, including Node, Stack, AdoptDesc and PriorityQueue
Nov. 21 - Nov. 27	Implement concurrent INSERT and start to work on project milestone report
Nov. 28 - Dec. 4	Implement concurrent DELETEMIN, including Logical deletion and batch physical deletion,test correctness of the MDList priority queue and finish the project milestone report
Dec. 5 - Dec. 11	Profile the MDList priority queue, conduct experiments, create graphs showing performance statistics and make comparisons with other mature lock-free concurrent priority-queue if applicable
Dec.11 - Dec.18	Finish up the project and finish the final report

Reference

[1] Zhang, D., & Dechev, D. (2016). A lock-free priority queue design based on multi-dimensional linked lists. *IEEE Transactions on Parallel and Distributed Systems*, *27*(3), 613–626. https://doi.org/10.1109/tpds.2015.2419651
[2] Hunt, G. C., Michael, M. M., Parthasarathy, S., & Scott, M. L. (1996). An efficient algorithm for concurrent priority queue heaps. *Information Processing Letters*, *60*(3), 151–157. https://doi.org/10.1016/s0020-0190(96)00148-2
[3] M. Herlihy and N. Shavit, The Art of Multiprocessor Programming, Revised Reprint. Amsterdam, The Netherlands: Elsevier, 2012.