RangeQueue: A Priority Queue algorithm

Abhay Shankar K: cs21btech11001 Kartheek Tammana: cs21btech11028
Rajapu Jayachandra Naidu: cs21btech11050

December 3, 2023

Abstract

This document is a report for the Final Project of the course CS5300: Parallel and Concurrent Programming, regarding a novel implementation of a lock-free data structure.

Base

This is a generalisation of the TSLQueue data structure proposed by Rukundo and Tsigas. Where the nodes in the TSLQueue data structure contain only a single key, RangeQueue has a range of keys within a node. Like all priority queues, the TNTQueue supports the following operations:

- `insert(key)`: Insert a key `k` into the queue.
- `deleteMin()`: Delete the key with the lowest key from the queue.

Optimisations

The RangeQueue is an improvement over the TSLQueue in the following ways:

- Cache locality: Since each node represents a range of keys, the number of nodes in the queue is reduced. This reduces the number of cache misses.
- Fewer CAS operations:
  - The TSLQueue requires 2 CAS operations per insertion, because each insertion modifes a linked list and a tree. RangeQueue requires only fewer CAS operations on average per insertion. This will be elaborated upon in the Algorithm section
  - The TSLQueue requires 3 CAS operations per deletion, one to logically delete the node by marking a bit, and two to modify pointers. RangeQueue again requires fewer CAS's because not every deletion requires pointer rearrangement.
- Lesser memory footprint: We use bitvectors instead of storing the actual value.

Overview

A threaded tree is a tree in which every node has a pointer to its successorin an inorder traversal. Such a concept can be generalised from key-based nodes to range-based nodes trivially, as non-overlapping ranges have a total ordering.

Each node in the RangeQueue contains

- `base`: The value of the the least key that could be inserted into it. Must be divisible by `BUF_SIZE`.
- `next`: A pointer to the next node in the queue.
- `parent`, `left`, `right`: Pointers to the parent, left child, and right child of the node respectively. Standard BST terminology.
- `buf`: A bitvector of size `BUF_SIZE` that indicates whether the key at an index has been inserted.
- `ins`: A flag indicating whether the node is currently undergoing insertion. Identical to TSLQueue.

The RangeQueue has a `head` pointer that points to the first node in the queue. The `head` pointer is a dummy node that has a key of `-1` and a `next` pointer to the first node in the queue. The `head` pointer is used to simplify the implementation of the `deleteMin()` operation and address the ABA problem.

The RangeQueue also has a dummy root. The root has a `next` pointer to the actual root node of the tree.

Both the dummy nodes are used in TSLQueue, for ease of proof of correctness.

Algorithm

Insertion

- A thread descends the BST to find the node that contains the key to be inserted.
  - If such a node exists, it atomically sets the corresponding bit in the `buf` bitvector. If the bit is already set, returns duplication error.
    - If a node is logically deleted (i.e. `buf` is all zeroes), it's existence will not be reported, i.e. `InsertSearch` returns `existingNode = null`.
  - Otherwise, it allocates a new node with the appropriate bit set, and inserts it into the RangeQueue. This is identical to the TSLQueue insert, with an similar `Seek` struct.
    - Set `next` to next node in the linked list.
    - Atomically insert into the linked list by setting `next` pointer of the previous node to the new node.
    - Set `parent` pointer of the new node to the parent node.
    - Atomically insert into the BST by setting appropriate `left` or `right` pointer of the parent node to the new node.

Deletion

- Deletion uses the `head` pointer to access the minimal node, and resets the highest set bit in the `buf` bitvector.
  - This is a single atomic operation - upon failure, the thread simply retries delete.
  - If after deletion, the `buf` bitvector is all zeroes, the node is said to be logically deleted, unlike in TSLQueue where there is a specific tag on the `next` pointer to indicate the same. With some nonzero probability, the thread physically deletes the node. Physical deletion is identical to TSLQueue.
    - Atomically delete from the linked list by setting `next` pointer of `head` to the next node.
    - Atomically delete from the BST by setting `left` pointer of the parent node to the next node.

Pseudocode

Correctness

We will assume TSLQueue is correct, and show that the correctness of RangeQueue follows by showing that the following conditions hold:

1. An active node key is the maximum key of its left-descendant(s) and the minimum key of its right-descendant(s). Also implying, that an active leaf key is the maximum key of all its preceding active leaves and the minimum of all its succeeding active leaves in the list.
2. An active node can only be logically deleted once and after all its left-descendants have been deleted. Also implying, that an active leaf can only be logically deleted once and after all its preceding leaves have been logically deleted. Further implying that DeleteMin() cannot linearize on a logically deleted node.
3. An inserted node is always pointed to by an active node next- pointer. Also implying, that Insert() cannot linearize on a logically deleted node.