CiCTrie - C Implementation of CTrie

Daniel Solomon
ID:
Email: DanielSolomon94.ds@gmail.com

Or Karni
ID:
Email: orkarni@mail.tau.ac.il

***Abstract*—We ported high-level memory managed data structure *CTrie* to an unmanaged language *C*, using *hazard pointers* mechanism, while keeping the basic operations of the data structure. We compared our implementation against Scala's standard library *TrieMap* (*CTrie*) using similar tests as in the original article.**

***Index Terms*—CTrie, Hazard Pointers.**

I. Introduction

CTrie [1] [2] (or concurrent hash-trie) is a concurrent *thread-safe*, *lock-free* implementation of a hash array mapped trie. This data structure is consists of *key-value* pairs and it supports the following operations:

- **insert**: add a new *(key, value)* pair.
- **remove**: remove a *(key, value)* pair if it exists.
- **lookup**: find the *value* (if any) for a specific *key*.

In addition the **CTrie** data structure has a *snapshot* operation which is used to implement consistent *iterators*. In fact **CTrie** is the first known concurrent data structure that supports *O(1)*, *atomic*, *lock-free* snapshots.

CTrie aspires to preserve the space-efficiency and the expected depth of hash tries by *compressing* after removals, disposing of unnecessary nodes and thus keeping the depth reasonable. The **CTrie** implementation is based on single-word *compare-and-swap* instructions.

II. Implementation

We wanted to give a simple interface for using *CiCTrie* which will be as close as possible to object oriented programming, therefore we used *Object Oriented C*; The user creates a *ctrie_t** struct using a "constructor" *create_ctrie* and use its functions passing itself as a "self object" (such as: `ctrie->insert(`**`ctrie`**`, key, value)`).

This way we could port easily the original *Scala* code from the original paper [1] and from the source code of the standard library itself [5].

A. Basic Operations

Some explanations about the general commands...

B. Memory Reclamation

Hazard pointers are a mechanism which aims to solve the ABA and safe memory reclamation problems [3] [4]. Each thread maintains a list of *hazard pointers* to resources it currently uses. This list usually has a fixed size and is kept small. A used resource may not be freed or modified.

In our implementation, each thread has 6 *hazard pointers* in total - 4 "trie" *hazard pointers* and 2 "list" *hazard pointers*. 6 *hazard pointers* are needed because in the worst case we reference 4 nodes (a parent *INode*, it's *CNode* child, a child *Inode* it's *MainNode*). Then, we must be able to iterate over a *LNode* linked list, which requires 2 *hazard pointers*.

Before accessing anything, a thread must first place a *hazard pointer* to ensure no one will free the desired memory. However, *hazard pointers* by themselves don't guarantee success. For example, if a thread reaches an *INode* and wishes to continue down to it's *MainNode*, it reads the pointer to the *MainNode*, places a *hazard pointer* and then accesses the node. But what if between the read and the placement of the *hazard pointer*, another thread changes the *CTrie* and frees the *MainNode*? The original thread would place a *hazard pointer* on an invalid address and wouldn't know it's already freed. To solve this problem, we added a *marked* field to *INode*, *LNode* and *CNode*. The marked field indicates that this node has been removed from the *CTrie* and is scheduled for freeing. In places where a race might occur, the accessing thread must validate it's state after placing a *hazard pointer*. In the example above, the thread must validate that:

1) The *INode* isn't marked.
2) The *Inode* still points to the expected *MainNode*.

C. Snapshot

:/...

III. Benchmarking

Fun, fun, fun, fun, fun...

References

[1] Original article representing **CTrie** - https://axel22.github.io/resources/docs/ctries-snapshot.pdf.
[2] **CTrie** wikipedia reference - https://en.wikipedia.org/wiki/Ctrie.
[3] Original article of hazard pointers - https://www.research.ibm.com/people/m/michael/ieeetpds-2004.pdf
[4] Hazard pointers wikipedia reference - https://en.wikipedia.org/wiki/Hazard_pointer.
[5] **TrieMap Scala source code -** https://www.scala-lang.org/api/current/scala/collection/concurrent/TrieMap.html.