Research Proposal - CiCTrie

Or karni & Daniel Solomon

July 15, 2017

1 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.

2 Goals and Objectives

CTrie suffers from a memory reclamation problem and, like all CAS-based data structures, from the ABA problem. Therefore, up until now most implementations of this data structure rely on the existence of a garbage collection mechanism in the targeted platforms. The first implementation was in *Scala* by its very own author *Alexander Prokopec*. Since then there were few more implementations for *Java*, *Go* and more.
CiCTrie - *C implementation of* ***CTrie*** aims for the following objects:

- Implement **CTrie** in *C* using *hazard pointers*[3][4].
- Bench mark our implementation versus the *Java* implementation.

Hazard pointers are a mechanism which aims to solve the ABA and safe memory reclamation problems. 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 order to make sure our implementation has no memory leaks we will use the *valgrind*[5] framework tool.

3 Previous Work

A quick search on the web will result a few projects that aimed to achieve our first goal, all of them are incomplete or missing memory management:

- **ctries**[6] (c implementation) - **incomplete**.
- **unmanaged-ctrie**[7] (c++ implementation) - **no attempt to manage memory allocation**.
- **concurrent-hamt**[8] (Rust implementation) - The only complete implementation using hazard pointers known.

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] Valgrind home page `http://valgrind.org/`.

[6] `https://github.com/Gustav-Simonsson/ctries`.

[7] `https://github.com/mthom/unmanaged-ctrie`.

[8] `https://github.com/ballard26/concurrent-hamt`.