Efficient Concurrent Memoization System

Author **Viacheslav Kroilov** | Supervisor **Daniel Langr**

FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Motivation

A memoization system (or a software cache) stores a limited number of recent items in local memory to speed up consequent access to them. It is often used in servers, databases, scientific applications, and more.

LRU is a simple and popular approach to evict stale items from the cache. It tracks item access order by keeping a list of all items and move an item to the list head on each access. Therefore, the least recently accessed item is located in the end.

This limits cache scalability with multiple threads as each access incurs writing to the list head that becomes a bottleneck.

Approach

The thesis presents a novel software cache, called DeferredLRU, that achieves better parallel speedup than the existing state-of-art concurrent LRU caches while delivering comparable hit-rate.

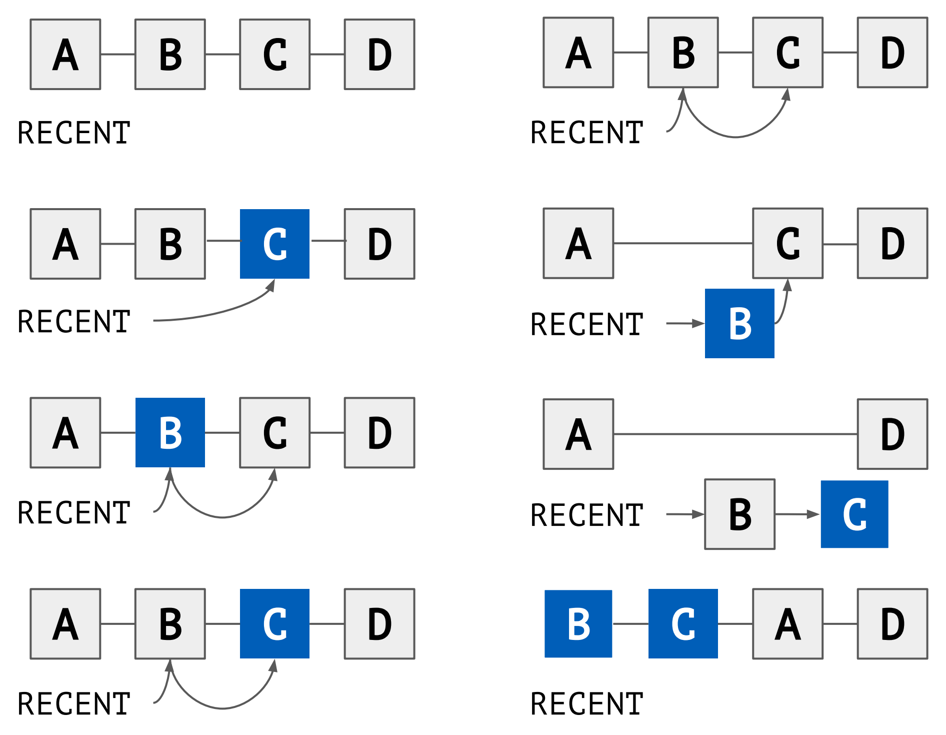
The improvement is achieved by changing the way items are reordered in the list. A second list, called the Recent list, is introduced along with the main one. When an item is accessed, it is appended to the Recent list instead of moving in the main one. Due to a simpler structure, appending to the Recent list is much faster than to the main list.

When the Recent list grows past a defined threshold, a single thread iterates over all items in it, takes them out from the main list, and then reinserts them into the main list head. This operation is called PullRecent. Inserting multiple items at once allows us to reduce the number of writes to the main list head (which is the main point of contention) by orders of magnitude.

Write-freedom for recent items

When an item is in the Recent list already, it is not inserted again. Therefore, accessing recent items requires no writes to shared memory (that are more expensive than reads) for managing the lists.

In the reference implementation, writes are still necessary for synchronizing another part of the cache. The possibility to avoid these writes is described in the thesis. To the best of the author's knowledge, it would be the first general-purpose concurrent software cache that allows for write-free lookups.



Item lookup

1) Initial state
2) **C** is accessed and added to RECENT
3) **B** is accessed and added to RECENT
4) **C** is accessed, but it is in RECENT already

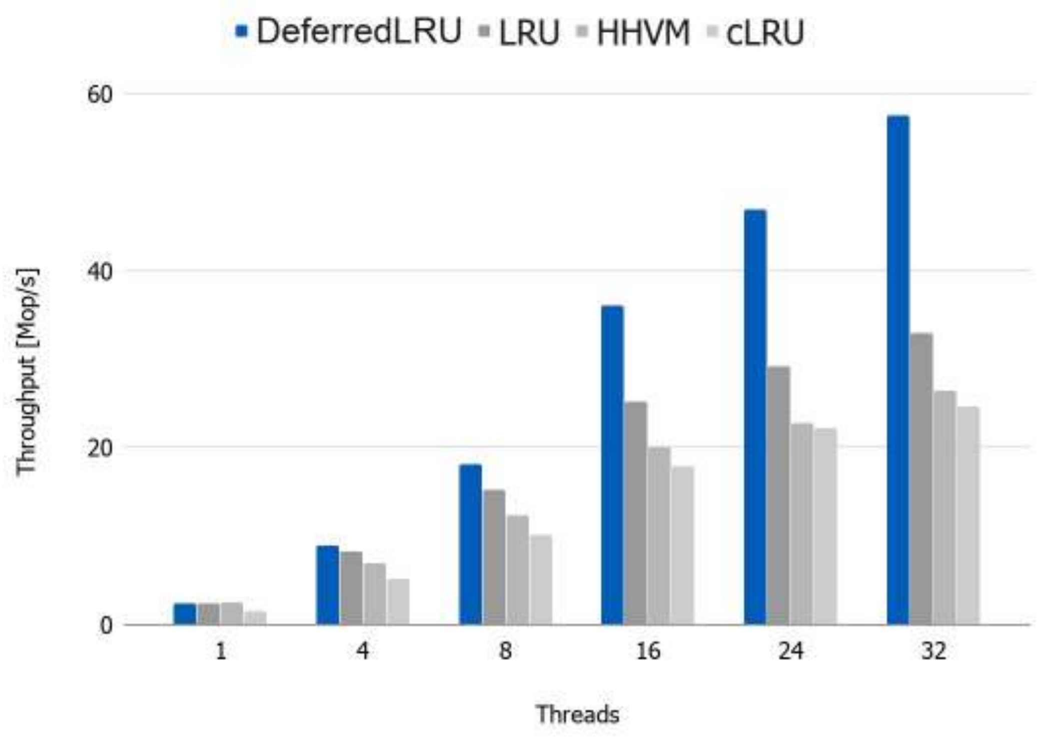
PullRecent

1) Initial state
2) **B** is removed (it is still searchable)
3) **C** is removed
4) **B** and **C** are reinserted in the main list in a single step

Evaluation

Both DeferredLRU and existing concurrent caches were evaluated on the real-world data traces (e.g., disk read accesses recorded on a search engine) with up to 32 threads.

Due to better scalability, DeferredLRU achieves higher throughput with 32 threads than the other caches in 11 of 16 tests. One of the measurements is shown in the plot below. The throughput (higher is better) is examined with varying thread count on the Wiki 1/10 trace (refer to the thesis for information on the traces).



Contribution

- A novel approach to LRU caching that achieves higher parallel speedup and performance than existing alternatives
- Extensive performance evaluation of the approach
- Recommendations and ideas for further research
- A reference C++ implementation of DeferredLRU, available at github.com/metopa/deferred_lru