A Look Into Generalized Methods for Creating Concurrent Data Structures

Oliver Tipton

April 21, 2024

Novelty

A NUMA machine, also known as a non-uniform memory access machine, utilizes an architecture which links nodes together that contain specific cores and caches. NUMA machines have quickly become an industry standard used by most if not all tech companies. Therefore, in order for present-day concurrent data structures to utilize the full power of NUMA, data structures ought to be NUMA-aware. NUMA-awareness means that the data structure is aware of the specific NUMA artifacts. However, the main challenge with designing NUMA-aware data structures is dealing with *operation contention*, the idea that two operations may be contingent upon one another so removing one changes the other. In "How to implement any concurrent data structure for modern servers," Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K. Aguilera describe a Node Replication (NR) method for "black-box" creation of concurrent data structures. This essentially means that no understanding of the inner data structure is required to create a NUMA-aware data structure using this generalized method. Replication is not the novel part of the method however. It is the broad-based, generalizable application of replication to creating concurrent NUMA-aware data structures which is novel and new.

Main Result(s):

Through comprehensive testing against other methods for creating NUMA-aware concurrent data structures, it was seen that "NR outperforms data structures obtained from other methods by up to 14x" (25). The main drawbacks seen in the research on NR were memory overhead as a result of replication, the blocking mechanism which can halt the progress of other threads (though no issues were found in the experiments with this, it creates the opportunity for an interesting study on non-blocking NR), and how NR may be outperformed by specifically catered methods like a lock-free skip list running on low-contention workloads. That being said, additional experiments showed NR outperforming catered methods like the lock-free skip list method when run on a high-contention workload. These results show that NR is not only generalizable, but actually optimal in a lot of cases. Overall, it was found that NR is best suited for small to medium sized data structures with operation contention.

Impact

As mentioned early on in the paper, NUMA machines have become increasingly useful and important in the tech industry. With that, NUMA-aware data structures are becoming more and more important. A generalized method for creating these types of data structures gives room for further development and usage of data structures which haven't been usable due to the difficulty in creating NUMA-aware structures. The paper goes little into any major impacts in the industry as a result of the creation of NR (presumably because it is still relatively new). However, examples like Redis show that NR is fully capable of making an impact, and it seems likely that NR will become more commonplace among NUMA machine users in the tech industry.

Evidence

Within the paper, there were two main testing categories covered: a handful of different concurrency methods (spinlock, One big readers-writer loc, Flat combining, Flat combining with readers-writer lock, Lock-free algorithm NUMA-aware algorithm) tested against NR on different data structures and a real life application, Retis. The 4 data structures tested were a skip list priority queue, a pairing heap priority queue, a skip list dictionary, and a stack. In the skip list priority queue, NR had the fewest cache misses and fell the least in comparison to the other methods. While NR has a cost associated in terms of overall memory usage, it makes up for it with better performance than all of the other methods. In a pairing heap priority queue, NR was also found to outperform the other methods once past the first node. The skip list dictionary had slightly different results, with the lock-free algorithm outperforming all other methods with low contention due to the parallelism being unaffected by contention. However, the lock-free algorithm loses its effectiveness as the number of threads increases, while NR's effectiveness increases. Lastly, the NUMA-aware algorithm and NR scaled the best with the stack. Once 14 threads or more were reached though, the NUMA-aware algorithm outperformed NR by 3.6x, showing that "black-box algorithms can be limited by their generality" (29).

The Redis server was also used as a means for running tests using NR. Through a myriad of different test cases and methods, it was found that NR was the best performer overall.

Prior Work

Memprof is another work which mentions replication as a way to optimize NUMA

Systems. However Memprof involves the replication of read-only objects, as opposed to NR which replicates data structures on each node and coordinates threads within and across these nodes. Munin performs replication through message passing in the cache coherence mechanism. Barrelfish also uses a message-passing mechanism for maintaining replicas of kernel data structures. However, these works all heavily differ in terms of scope and "generality" in applying to all sequential data structures. While other methods have also used in-memory logs, it was with a different scope and goals than those of NR. NR looks to branch out from these other methods in an effort to make a generalizable method for creating NUMA-aware data structures through replication.

Reproducibility

The tests involving skip lists, dictionaries, and a myriad of other data structures being adjusted by NR and other methods like spinlock and lock-free algorithms would be relatively simple to reproduce. Looking at overall cache misses, runtime, and memory storage necessary for specific computations amongst NR and other methods would be an easy way to reproduce those sets of tests/findings. However, the Redis test would be significantly more difficult to reproduce. It involved the usage of a 512 GB ram Dell Server, as well as 56 cores on 4 separate Intel processors. This type of test requires more resources and time allocated to properly test and reproduce those types of findings.

Question

If there is a generalizable method which can beat a majority of other methods but not all, how come it seems so difficult to create a specific method for a specific data structure that beats

ALL other methods for that data structure? (e.x. “Optimal” lock-free skip list gets outperformed by NR on high-contention workloads, showing how it isn’t really optimal)

Criticism

The ordering of sections made this a slightly more difficult read than I think it needed to be. Section 3 would have felt more helpful if it replaced section 2. I found myself asking a lot of questions about what the method and algorithm even was, yet we were already comparing it to other algorithms (section 2). Instead, I wish that the basics of NR (ie. section 3) was given after the introduction, then section 4 which introduces the more complicated algorithm, and then section 2 which compares NR to other methods. That way I could have gotten a concrete understanding of what NR actually was before comparing it to other methods and learning about its real-world applications.

Ideas for further work

In looking at blockchain protocols like Ethereum and Solana, oftentimes there are high-transaction costs associated with bridging funds from one network to the other (ie. from Ethereum to another protocol like Solana or Arbitrum). As mentioned on page 24, accessing data from another node comes with a higher cost. This is because cores share local caches within a node which requires little to no cost as it is local data. While I’m unsure how feasible this would be, I see a potential application of some kind which could be possible between blockchain protocols and NUMA machines. It would be interesting to “group” Layer 1 protocols like

Etehereum and Solana into a node so that “bridging” funds would be inexpensive and like moving from local to local storage. Then, Layer 2 protocols could be grouped into another node and connected to the Layer 1 protocols via an interconnector.