THE NO HOT SPOT NON-BLOCKING SKIP LIST FOR UNMANAGED LANGUAGES

Ian Dick
Dr. Vincent Gramoli
School of Information Technologies
FACULTY OF ENGINEERING & INFORMATION TECHNOLOGIES

INTRODUCTION

In-Memory Databases

- In-memory databases have become a common method for increasing performance, removing the need for disk-based storage during normal database operation
- When the disk storage bottleneck has been removed from database operations, the performance bottleneck imposed by the lock manager becomes more apparent
- Recent research has focused on providing in-memory databases with concurrency management tools that do not require locking mechanisms, since locks tend to serialise parallel systems when there are many threads of execution

Non-blocking Data Structures

- Non-blocking data structures can provide concurrent access to resources without the need for locks and mutual exclusion
- If a data structure is non-blocking it is guaranteed that some forward progress will always be made, i.e. conditions such as system-wide deadlock cannot occur
- Implementing non-blocking parallel systems can be much more complicated than implementing parallel systems using locks, especially when the systems must work on weak-memory architectures [3]

The Skip List

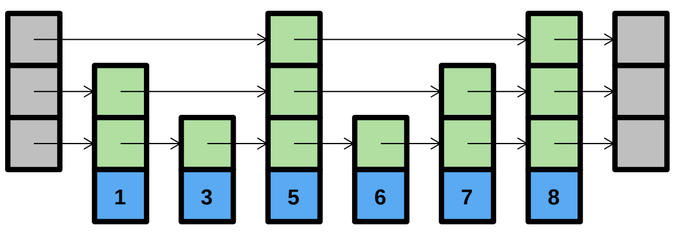

Figure 1: A Skip List

- A skip list is similar to a linked list, but with index nodes that allow searches to skip over blocks of elements, which can result in logarithmic query times
- Numerous research efforts have aimed to provide concurrent skip lists that are non-blocking, since non-blocking skip lists can be simpler to implement than other non-blocking logarithmic structures such as B-Trees

THE NO HOT SPOT SKIP LIST

- The No Hot Spot Non-Blocking skip list [1] proposed by Crain, Gramoli and Raynal is one of the latest research efforts to provide a scalable non-blocking skip list
- Contention hot spots are avoided by delegating all index-level modifications to a background thread, so that normal threads performing search/insert/delete operations on the skip list do not contend with each other over these modifications

THE CONTRIBUTION

- The contribution is to showcase the performance benefits of the No Hot Spot Non-Blocking skip list when it is implemented in C (with garbage collection!)
- Changes to the original data structures were necessary: node structures were collapsed into arrays and accessed using modulo arithmetic (see right) – these changes were made to reduce pointer indirection and increase memory locality to improve the cache friendliness of the data structures

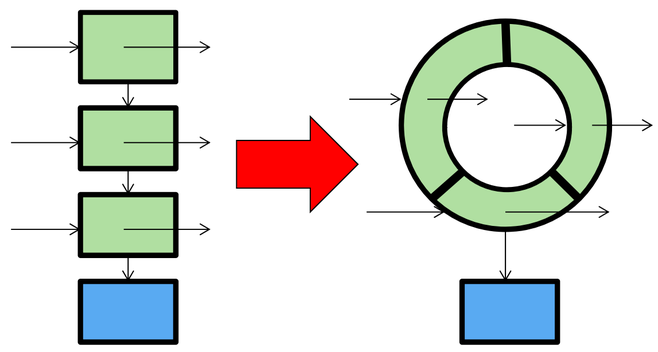

Figure 2: Evolution of index nodes: Distinct nodes (left), became nodes consolidated into an array and accessed using modulo arithmetic (right)

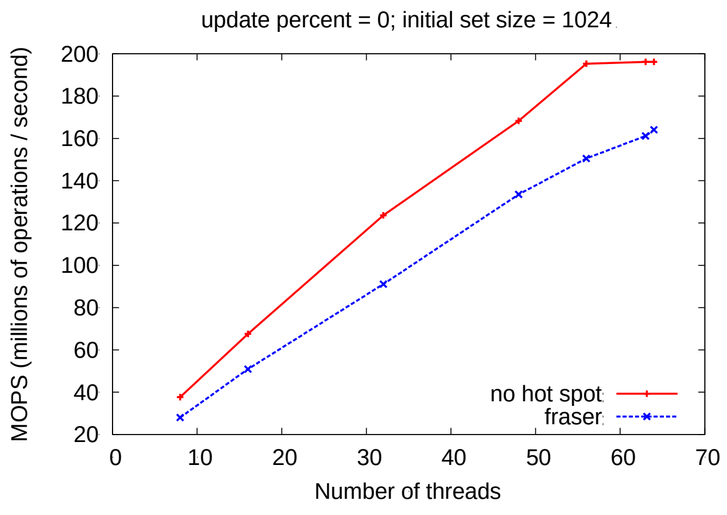

Figure 3: Performance experiments for the No Hot Spot skip list and Fraser's skip list

THE EXPERIMENT

- The No Hot Spot skip list was implemented in the C programming language and benchmarked against Fraser's non-blocking skip list [2] using a micro-benchmark suite (a variant of Fraser's list is in JDK 1.6+ and is commonly used as a benchmark [4])
- Transaction throughput was documented in various experiments, where higher throughput equals better performance
- Different parameters were manipulated to see how well the two lists cope under different scenarios

THE RESULTS (see Figure 3)

- As the percentage of updates increases to 10, we see the effects of contention on the two skip lists
- The No Hot Spot skip list scales better with the number of threads than Fraser's skip list when many threads are trying to conduct modifications such as inserts or deletes
- When the test set size increases to 65536 the impacts of contention are slightly lessened since threads are less likely to be modifying the same item simultaneously, but the No Hot Spot list is still at an advantage

FUTURE WORK

- In the experiments corresponding to the results in Figure 3, the No Hot Spot skip list background thread is run very infrequently
- Running the background thread too frequently hurts performance, but if the background thread is run too infrequently and the list becomes unbalanced then it takes a long time to become balanced again
- Future work will involve improving the background thread so that it can respond dynamically to changes in transaction workload to address the above issue

REFERENCES

1. Crain, T; Gramoli, V; Raynal, M. (2013) The No Hot Spot Non-Blocking Skip List, *ICDCS July 2013, IEEE.*
2. Fraser, K. (2003) Practical Lock-Freedom, *PhD Thesis, Cambridge University Computer Laboratory*.
3. Linux Kernel Documentation, Memory Ordering and Barriers, https://www.kernel.org/doc/Documentation/memory-barriers.txt
4. Doug Lea's ConcurrentSkipListMap, http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentSkipListMap.html