Brief Announcement: A Contention-Friendly, Non-blocking Skip List*

Tyler Crain[1], Vincent Gramoli[2], and Michel Raynal[1,3]

[1] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
[2] The University of Sydney, NSW 2006, Australia
[3] Institut Universitaire de France

A skip list is a probabilistic data structure to store and retrieve in-memory data in an efficient way. In short, it is a linked structure that diminishes the linear big-oh complexity of a linked list with elements having additional shortcuts pointing towards other elements located further in the list [7]. These shortcuts allow operations to complete in $O(\log n)$ steps in expectation. The drawback of employing shortcuts is however to require additional maintenance each time some data is stored or discarded.

Non-blocking skip lists are increasingly popular alternatives to B-trees in main-memory databases, like *memsql*[1], as they are latch-free and can be traversed in sorted order. By being non-blocking, a skip list ensures that the system as a whole always makes progress. However, in a highly concurrent context the additional maintenance causes contention overheads on existing skip lists [3, 4, 8] by increasing the probability of multiple *threads* (or processes) interfering on the same shared element. Such contention could translate into performance losses in multicore applications, like in-memory key-value store.

We recently observed a similar issue in concurrent trees that led us to derive a binary search tree algorithm especially suited for transactional memory [2]. Our *contention-friendly* non-blocking skip list demonstrates that these algorithmic concepts can be adapted to improve the performance of a different data structure relying exclusively on compare-and-swap, which makes it inherently non-blocking. In addition, our skip list guarantees the atomicity of insertions, deletions and lookups of key-value pairs as shown in the companion technical report [1]:

Theorem 1. *Each of the contains, insert, delete operations implemented by the contention-friendly non-blocking skip list satisfy linearizability.*

The contention-friendly non-blocking skip list aims at accommodating contention of modern multicore machines. To this end, it exploits a genuine decoupling of each updating access into an eager abstract modification and a lazy and selective structural adaptation.

Eager Abstract Modification. The eager abstract modification consists in modifying the abstraction while minimizing the impact on the skip list itself and returning as soon

* A full version is available in [1]. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 238639, ITN project TransForm.

[1] `http://developers.memsql.com/docs/1b/indexes.html`

M.K. Aguilera (Ed.): DISC 2012, LNCS 7611, pp. 423–424, 2012.

as possible for the sake of responsiveness. Existing skip lists typically maintain a precise distribution of nodes per level, hence each time the abstraction is updated, the invariant is checked and the structure is accordingly adapted as part of a single operation. While an update to the abstraction may only need to modify a single location to become visible, its associated structural adaptation is a global modification that could potentially conflict with any concurrent update. In order to avoid these additional conflicts, when a node is inserted in the contention-friendly skip list only the bottom level is modified and the additional structural modification is postponed until later. When an element is removed the operation is separated into a logical deletion marking phase followed by physical removal and garbage collection phases.

Lazy Selective Adaptation. The lazy selective adaptation, which can be deferred until later, aims at adapting the skip list structure to the abstract changes by re-arranging elements or garbage collecting deleted ones. To guarantee the logarithmic complexity of accesses when there is no contention in the system, the structure is adapted by updating the upper levels of the skip list when contention stops.

The structural adaptation is *lazy* because it is decoupled from the abstract modifications and executed by one or multiple independent thread(s). Hence many concurrent abstract modifications may have accessed the skip list while no adaptations have completed yet. We say that the decoupling is *postponed* from the system point of view.

This postponement has several advantages whose prominent one is to enable merging of multiple adaptations in one simplified step: only one traversal is sufficient to adapt the structure after a bursts of abstract modifications. Another interesting aspect is that it gives a chance to insertions to execute faster: if the element to be inserted is marked as logically deleted, then the insertion simply needs to logically insert by unmarking it. This avoids the insertion to allocate a new node and to write its value in memory.

Performance. Our preliminary evaluations on a 24-core machine show that a Java implementation of the contention-friendly non-blocking skip list can improve the performance of one of the mostly used non-blocking skip lists, the JDK adaptation by Lea of the Harris and Michael's lists [5, 6], by a multiplying factor of 2.5.

References

1. Crain, T., Gramoli, V., Raynal, M.: A contention-friendly, non-blocking skip list. Technical Report RR-7969, IRISA (May 2012)
2. Crain, T., Gramoli, V., Raynal, M.: A speculation-friendly binary search tree. In: PPoPP (2012)
3. Fomitchev, M., Ruppert, E.: Lock-free linked lists and skip lists. In: PODC (2004)
4. Fraser, K.: Practical lock freedom. PhD thesis. Cambridge University (September 2003)
5. Harris, T.L.: A Pragmatic Implementation of Non-blocking Linked-Lists. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, p. 300. Springer, Heidelberg (2001)
6. Michael, M.M.: High performance dynamic lock-free hash tables and list-based sets. In: SPAA, pp. 73–82 (2002)
7. Pugh, W.: Skip lists: a probabilistic alternative to balanced trees. Commun. ACM 33 (June 1990)
8. Sundell, H., Tsigas, P.: Scalable and lock-free concurrent dictionaries. In: SAC (2004)