Concurrent Priority Queues

Seminar in Algorithms, 2013W

Jakob Gruber, 0203440

September 23, 2015

Introduction I

Priority Queues (PQs):

- Standard abstract data structure
- Used widely in algorithmics, operating systems, task scheduling, etc
- Interface consists of two $O(\log n)$ operations:

```
void Insert(pq_t *pq, key_t k, value_t v)
bool DeleteMin(pq_t *pq, value_t *v)
```

- Typical backing data structures: heaps & search trees

Introduction II

- In the past decade, processor clock speeds have remained the same, trend towards multiple cores
- New data structures required to take advantage of concurrent execution
- The topic of this presentation: efficient concurrent PQs
- Fine-grained locking $\rightarrow$ Lock-free $\rightarrow$ Relaxed data structures

Concepts and Definitions

Safety conditions: nothing bad has happened yet

- *Linearizability*: operations appear to take effect at a single point in time, the linearization point
- *Quiescent consistency*: in a period of quiescence, semantics equivalent to some sequential ordering

Concepts and Definitions

Liveness conditions: something good eventually happens

- *Lock-freedom*: at least a single process makes progress at all times
- *Wait-freedom*: every process finishes in a bounded number of steps

Concepts and Definitions

Miscellaneous

- *Disjoint-access parallelism*: how well a data structure handles concurrent use by multiple threads within disjoint areas
- Synchronization primitives:
  - Compare-And-Swap (CAS), Fetch-And-Add (FAA), Fetch-And-Or (FAO), Test-And-Set (TAS)
  - Double-Compare-And-Swap (DCAS), Double-Compare-Single-Swap (DCSS)

```
bool CAS(T *ptr, T *expected, T value) {
  if (*ptr == *expected) {
    *ptr = value;
    return true;
  } else {
    *expected = *ptr;
    return false;
  }
}
```

Related Work

- Non-standard synchronization primitives
  - Liu and Spear: Array-based PQ with `ExtractMany`
  - Israeli and Rappoport: Wait-free PQ
- Bounded range priorities
  - Shavit and Zemach: Combining funnels & bins
- Strict PQs
  - Hunt et al.: Fine-grained locking heap
  - Shavit and Lotan: First SkipList-based PQ
  - Sundell and Tsigas: First lock-free PQ
  - Lindén and Jonsson: Minimizes contention
- Relaxed data structures
  - Kirsch, Lippautz, and Payer: k-FIFO queues
  - Wimmer et al.: k-PQ
  - Alistarh et al.: SprayList

Highlighted PQs are presented in the following

Fine-grained Locking Heaps

Hunt et al.

- Naive PQ parallelization: single global lock $\rightarrow$ sequential bottleneck
- A first improvement: fine-grained locking using a lock per node
- Galen C Hunt et al. "An efficient algorithm for concurrent priority queue heaps". In: *Information Processing Letters* 60.3 (1996), pp. 151–157

Fine-grained Locking Heaps

Hunt et al.: Innovations

- One lock per node, *but* additionally a global lock protecting the heap's `size` variable
- Insertions bottom-up, deletions top-down to reduce contention
- Successive insertions take disjoint paths towards the root

Fine-grained Locking Heaps

Hunt et al.: Limitations

- A global lock remains
- Heap is statically allocated
- Frequent complex heap reorganization
- Disjoint-access breaks down at high traffic levels
- Inherent PQ bottleneck at the minimal node
- Benchmarks show only limited scalability up to a low thread count

Lock-free Priority Queues

SkipLists

- Modern concurrent PQs are mostly based on Pugh's SkipList (SL)
- Probabilistic ordered search structure, insertions and deletions in expected $O(\log n)$ time
- No reorganizations
- Simple implementation
- Excellent disjoint-access properties

Lock-free Priority Queues

SkipLists

- Collection of linked lists with corresponding levels
- Lowest list contains all items, higher lists are shortcuts
- `Insert` chooses a `level` according to geometric distribution

```
struct slist_t {
  size_t max_level;
  node_t head[max_level];
};
struct node_t {
  key_t key;
  value_t value;
  size_t level;
  node_t *next[level];
};
```

Lock-free Priority Queues

Shavit and Lotan

- First SkipList-based PQ
- Nir Shavit and Itay Lotan. "Skiplist-based concurrent priority queues". In: *Proceedings of the 14th International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE. 2000, pp. 263–268
- Initially lock-based (linearizable), lock-free (quiescently consistent) variant published in 2008
- Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. Elsevier, 2012

Lock-free Priority Queues

Shavit and Lotan

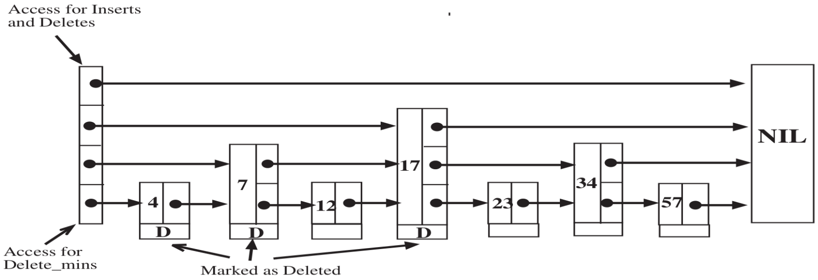

Figure: The Shavit and Lotan PQ (Image source: [9])

Lock-free Priority Queues

Shavit and Lotan

- Items are considered in the list once inserted on bottom level
- Again, insertions bottom-up and deletions top-down
- Deletions are split
  - Logical deletion sets a `deleted` flag
  - Physical deletion performs actual pointer manipulations
- `DeleteMin` attempts to logically delete the head node. On success: delete physically & return node. Otherwise, continue with next node.
- `Insert` is equivalent to the SL insertion

Lock-free Priority Queues

Shavit and Lotan

- Quiescently consistent, but not linearizable
  - Slow thread A suspended at deleted key k while in `DeleteMin`
  - Fast thread B first inserts $k-1$, then $k+1$
  - A wakes up and returns $k+1$
  - Linearizability would require returning $k-1$
- Timestamp mechanism: stamp nodes on successful insertion, `DeleteMin` ignores all stamps earlier than its own starting point
- Improved scalability, but heavy contention at list head

Lock-free Priority Queues

Sundell and Tsigas

- First lock-free PQ, linearizable, SL-based, distinct priorities
- Deletion flag packed into least significant bits of `next` pointers prevent insertion *after* deleted nodes
- Helping mechanism ensures only a single logically deleted node exists at any time
- Performs significantly better than the Hunt et al. heap, slightly better than a SkipList protected by a global lock
- Håkan Sundell and Philippas Tsigas. "Fast and lock-free concurrent priority queues for multi-thread systems". In: *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE. 2003, 11–pp

Lock-free Priority Queues

Lindén and Jonsson

- Most efficient strict PQ, SL-based & linearizable
- Concurrent strict PQ performance limited by contention and CAS failures in `DeleteMin` $\rightarrow$ Minimize CAS calls
- Deleted nodes form prefix of SL, deletion flag packed into `next` pointer of *previous* node prevents insertion *before* deleted node
- Most `DeleteMin` perform logical deletion (1 CAS) only, physical deletion only once a bound is reached
- Improves upon best previous PQs by up to factor 2
- Jonatan Lindén and Bengt Jonsson. "A Skiplist-Based Concurrent Priority Queue with Minimal Memory Contention". In: *Principles of Distributed Systems*. Springer, 2013, pp. 206–220

Lock-free Priority Queues

Lindén and Jonsson

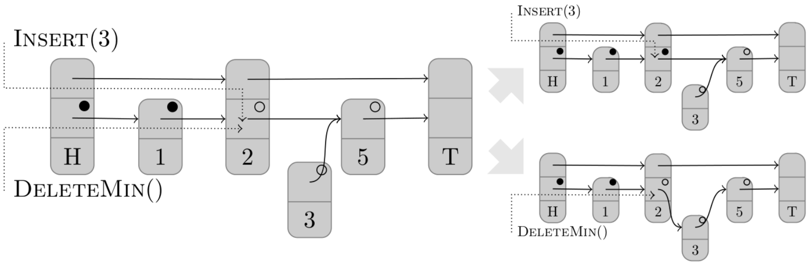

Figure: The Lindén and Jonsson PQ. Concurrent `Insert(3)` and `DeleteMin` operations. Top right: `DeleteMin` succeeds first, `Insert(3)` CAS fails. Bottom right: `Insert(3)` succeeds first, `DeleteMin` returns 3. (Image source: [6])

Relaxed Priority Queues

- Strict PQs have inherent bottleneck at minimal element
- To improve further: < 1 CAS per `DeleteMin`
- Another approach is to relax semantics, i.e. instead of returning *the* minimal element, return one of the k minimal elements

Relaxed Priority Queues

Alistarh et al.

- Relaxed SL-based PQ, safety properties unclear
- `DeleteMin` returns one of the $O(P \log^3 P)$ elements
- Degrades to random-remove if the PQ is small compared to thread count P (for $P = 80$, $P \log^3 P \approx 7000$)
- `DeleteMin` performs random walk: starting at the list head on some level l, repeatedly follow a randomized number of `next[l]` pointers, descend a randomized number of levels
- Parameters are chosen s.t. each element within the walk has approximately equal probability of being returned
- Benchmarks show scalability comparable to a random-remove `Delete` up to at least 80 threads
- Dan Alistarh et al. *The SprayList: A Scalable Relaxed Priority Queue*. Tech. rep. 2014

Relaxed Priority Queues

Wimmer et al.

- First relaxed linearizable PQs: we discuss the hybrid k-PQ which, provides a bound of kP missed elements in `DeleteMin`
- Consists of a list of globally visible elements, and per thread: a local item list, and a local PQ
- `Insert` accesses only the local structures as long as guarantees are not violated, otherwise the global list is updated the the local view is synchronized
- `DeleteMin` pops the local queue if it is non-empty; otherwise spy, i.e. copy elements from another thread's local list
- Very good scalability up to 10 threads, further limited gains with rising thread counts
- Martin Wimmer et al. "Data Structures for Task-based Priority Scheduling". In: *arXiv preprint arXiv:1312.2501* (2013)

Lock-free Priority Queues

Wimmer et al.

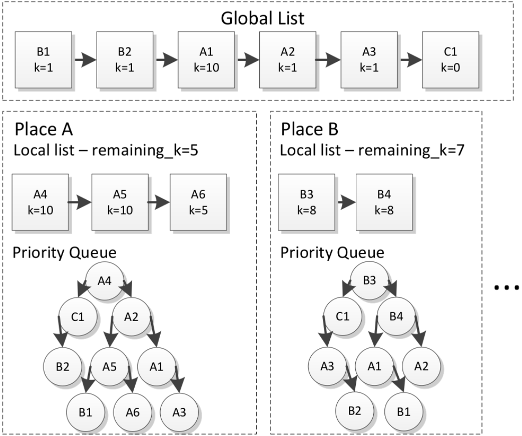

Figure: The Wimmer et al. hybrid k-PQ. (Image source: [12])

Results

- Benchmarking results from selected papers & own results
- We present throughput, i.e. the number of operations performed per second
- Each thread repeatedly chooses uniformly at random between `Insert` and `DeleteMin`

Results

Lindén and Jonsson

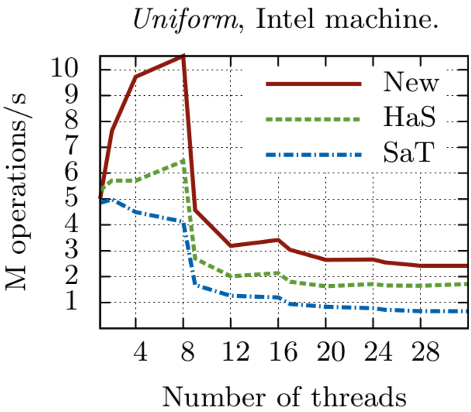

Figure: GCC 4.7.2, 32-core Intel Xeon E5-4650 @ 2.7 GHz. Initial PQ size unknown. *New*: Lindén and Jonsson, *HaS*: Shavit and Lotan, *SaT*: Sundell and Tsigas. (Image source: [6])

Results

Gruber

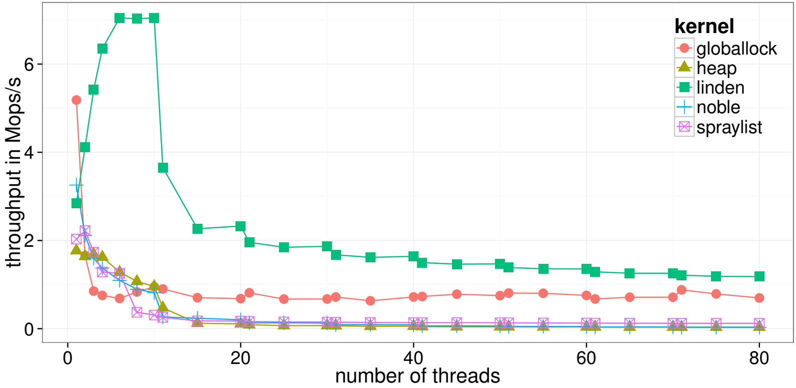

Figure: GCC 4.8.2, 80-core Intel Xeon E7-8850 @ 2.0 GHz. PQ initialized with 2^{15} elements.

Results

Alistarh et al.

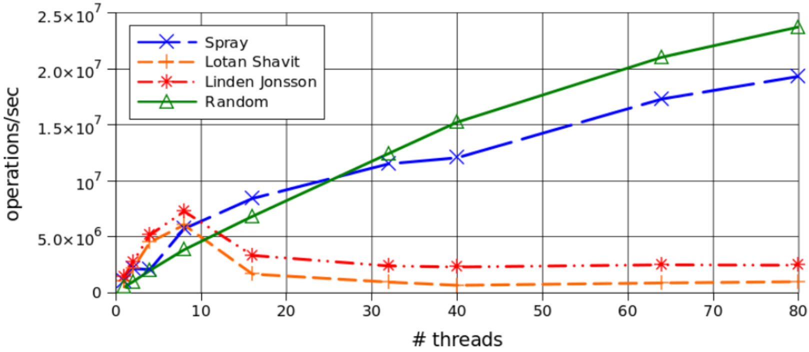

Figure: GCC version unknown, 80-core Intel Xeon E7-4870 @ 2.4 GHz. PQ initialized with 10^6 elements. (Image source: [1])

Conclusion

- Parallelizing PQs is hard
- SkipLists currently dominate practical implementations
- Main limiting factor are list head accesses in `DeleteMin`
- Lindén and Jonsson PQ is state of the art in strict semantics
- Much potential remains in relaxed PQs and data structures in general $\rightarrow$ future research

Conclusion

Questions?

References I

Dan Alistarh et al. *The SprayList: A Scalable Relaxed Priority Queue*. Tech. rep. 2014.

Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. Elsevier, 2012.

Galen C Hunt et al. "An efficient algorithm for concurrent priority queue heaps". In: *Information Processing Letters* 60.3 (1996), pp. 151–157.

Amos Israeli and Lihu Rappoport. "Efficient wait-free implementation of a concurrent priority queue". In: *Distributed Algorithms*. Springer, 1993, pp. 1–17.

Christoph M Kirsch, Michael Lippautz, and Hannes Payer. *Fast and scalable k-fifo queues*. Tech. rep. 2012-04, Department of Computer Sciences, University of Salzburg, 2012.

References II

Jonatan Lindén and Bengt Jonsson. "A Skiplist-Based Concurrent Priority Queue with Minimal Memory Contention". In: *Principles of Distributed Systems*. Springer, 2013, pp. 206–220.

Yujie Liu and Michael Spear. "A lock-free, array-based priority queue". In: *ACM SIGPLAN Notices* 47.8 (2012), pp. 323–324.

William Pugh. "Skip lists: a probabilistic alternative to balanced trees". In: *Communications of the ACM* 33.6 (1990), pp. 668–676.

Nir Shavit and Itay Lotan. "Skiplist-based concurrent priority queues". In: *Proceedings of the 14th International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE. 2000, pp. 263–268.

References III

Nir Shavit and Asaph Zemach. "Scalable concurrent priority queue algorithms". In: *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC)*. ACM. 1999, pp. 113–122.

Håkan Sundell and Philippas Tsigas. "Fast and lock-free concurrent priority queues for multi-thread systems". In: *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE. 2003, 11–pp.

Martin Wimmer et al. "Data Structures for Task-based Priority Scheduling". In: *arXiv preprint arXiv:1312.2501* (2013).