The SprayList: A Scalable Relaxed Priority Queue[1]

Seminar Advanced Algorithms and Data Structures - Student Report

Adrian Balla

2018-11-11

Motivation and Introduction

$\Omega(n \log n)$ for sorting and many more lower bounds provide a theoretical limit to computational speed. A definite bound for speed in hardware is the physical limit for processor frequency, almost hit today.
So no hope for further increase in overall performance on these two axes? There is in fact a 3rd axis, which can be resorted to for specific but common cases, called parallelism. According to Amdahl's Law: Whenever an algorithm has parallelizable part (i.e. certain steps can be carried out independently), it can be computed in a fraction of the time by distributing it over multiple processors.
Nowadays, multicore machines are ubiquitous. Such a computer distributes its given work over potentially more than 100 cores. This distribution can be achieved via work-requesting, where each processor fetches a job from a priority queue, which is in shared memory. A difficulty in this environment is that a collision of read with write operations results in interruptions (stalls [3]) responsible for slowdowns and ultimately a reduction in throughput. Hence, contention on the top element of the priority queue becomes a bottleneck.

Traditionally, binary heaps are employed for the priority queue. In this new setting however, many complications evolve. For instance, an insertion requires moving a key upwards by swapping until the heap property is restored. Blocking the entire structure for the execution of an operation would be an option that performs poorly, while allowing concurrent access of the heap would be undesirable because of high number of collisions (e.g. when multiple items are inserted) and difficulties maintaining consistency. The SkipList is superior in this setting, it requires less intricate mechanisms.

The SkipList

SkipLists are ordered lists containing n elements with varying *height* (from 0 to ∞) which are accessible from *levels* starting at 0. (See Figure 1 for an example)
Similar to a linked list, a node consists of links to its successors, one for each level that is less than its height. At level l of a node we thus can reach the next node with $height > l$ or descend to the level below.
Inserting a new element consists of finding the position corresponding to its value (doable in $O(\log n)$) and the linking process (takes constant time).
The advantages over heaps are evident: Once a node is added, there are no further modifications and collisions are less frequent. A downside is the randomness and, in practice, the use of additional pointers.
Theoretically, the first element can be removed in constant time, but competition for extracting the top element, or rather numerous simultaneous accesses, delay success and therefore throttle throughput.

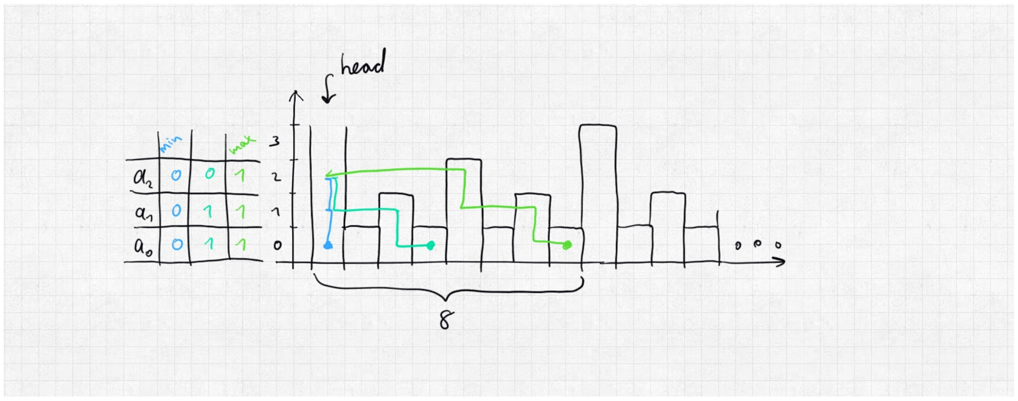
Relaxation

In place of a single point of contention, the head, an obvious way to reduce collisions is to spread the choice of extraction over multiple elements at the top of the queue. Most algorithms using priority queues won't break when receiving errors of this type, or at least can be adjusted slightly to withstand them. Wasted work is a result, e.g. when elements have to be re-added because of too low priority.Say we allowed the queue to return an element that is off by k from the optimum. We call such a data structure k-relaxed in general. Because we opted for a SkipList, we do not have to linearly traverse, but may skip elements. Before constructing an algorithm, let us review the core property of the SkipList:

The height of the elements is geometrically distributed:

$$height \sim \text{Geo}(1/2)$$

Figure 1: A `Spray` execution on an ideal SkipList



Whenever an item is to be added to this list, we will flip a coin until it shows head and set the height of our new element to the number of coin flips performed. I.e. $\Pr[height > j] = 1/2^j$.

Let us consider ideal instances for now: exactly $1/2^j$ of the items have $height > j$. So one out of 2^j elements satisfy the latter condition. In addition, a step down the list at level l means moving over 2^l elements total.

The SkipList depicted in Figure 1 is a special case. For the fixed interval of size $k = 8$ we can start at the *head* and explore this interval as with binary search or a binary tree. Starting at level 2, we choose to descend if we would like to land in the left half or else move along to the next node at this level (green line) for the right half. While the blue line marks the path to the up-most element, the green line walks furthest and the cyan one is in-between. In fact, we have 8 paths, one for each element, so we can choose uniformly among them by choosing the step length a_l at level l u.a.r. from $\{0, 1\}$. The table in Figure 1 shows the corresponding step lengths for each plotted path.

In this particular SkipList it makes sense to take at most one step per level. Over all SkipLists, more steps are preferable and the process above can be generalized to the `Spray` algorithm:

The `Spray` Algorithm

Note: In this report, $\log x$ means $\log_2 x$ and is assumed to be integer.

The procedure `Spray` takes parameters H, the starting height, L, the maximum jump length and D, the number of steps to descend after a horizontal walk.

We start at the head at height H and repeat the following:
Walk a jump length, chosen u.a.r. from $[0, L]$, horizontally and then, if the bottom level has not been reached yet, descend D steps; else return the current element.

The i-th *part* of a `Spray` is the walk at level iD. The *i-suffix* of a `Spray` is the 0-th up to the i-th part (i.e. $a_0, ..., a_i$). In other words: the $(i+1)$ lowest horizontal walks of a spray.
A tuple $(a_0, ..., a_{l_p})$ of length $(l_p + 1)$ describes a `Spray` execution, where a_i steps are taken at level iD.
The runtime of `Spray` clearly is in $\mathcal{O}(LH/D)$

In Figure 1, `Spray` was executed with $H = 2$, $L = 1$, $D = 1$.
In order to vary the relaxation, we introduce the parameter p, that should be at most the umber of clients executing `DeleteMin`/`DeleteMax` operations concurrently. When all of these access the SkipList at the same time, a selection of at least p elements needs to be provided.

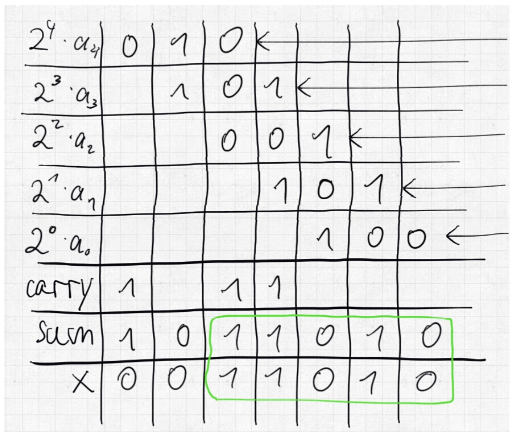
Bounding the hit probability on a perfect SkipList

By bounding the probability of landing on each element, we obtain a limit for collisions and for uneven hit distribution.

Intuitively, a SkipList that can be traversed most efficiently should have its elements separated by exponential distance with regard to its height.

Definition. A *perfect* SkipList has distance $2^{\min(h1,h2)}$ between any two consecutive elements of height h_1 and h_2.

Figure 2: The sum of walked distances in binary representation for $p = 32$, $L = \log p = 5$, $H = \log p - 1 = 4$, where the step lengths $a_i \in [1, 5]$.



$2^4 \cdot a_4$	0	1	0	←				
$2^3 \cdot a_3$		1	0	1	←			
$2^2 \cdot a_2$			0	0	1	←		
$2^1 \cdot a_1$				1	0	1	←	
$2^0 \cdot a_0$					1	0	0	←
carry	1		1	1				
sum	1	0	1	1	0	1	0	
x	0	0	1	1	0	1	0	

The distance a `Spray` walks in a perfect SkipList is

$$\sum_{i=0}^{H} a_i 2^i$$

In this section, let $L = \log p$, $H = \log p - 1$ and $D = 1$.
The furthest walk then is of distance $L(2^{\log p} - 1)$. Being roughly $p \log p$, this is an upper bound for the number of reachable elements and for low p, $\frac{1}{p \log p}$ is roughly $\frac{1}{p}$, so the following theorem shows that the distribution among those elements is close to uniform and thus probability of collision is small.

To simplify the analysis, we assume $L = \log p$ is even and we set the jump length to be at least 1. That is, $a_i \in [1, L]$.

Theorem 1

The probability that a `Spray` (with $L = \log p$, $H = \log p - 1$, $D = 1$) hits any element is $\leq \frac{1}{p}$.

Proof: Let the element which the `Spray` lands on be x. The following equation states that the walking distance d equals x. We will bound the probability of this equation being satisfied by providing the probability of the first $\log p$ bits of the distance d matching those of x. Let the corresponding event of match be $\mathcal{E}$ (In short: $\forall k \in [1, \log p] : d^{(k)} = x^{(k)}$)

$$d := \sum_{i=0}^{\log p - 1} a_i 2^i = x$$

Before proving the general case, consider the example $p = 32$. Figure 2 shows the binary representation of the summands, the sum (total distance) and carry bits as well as the position x. If the bits in the green frame do not match, the equation cannot be satisfied.
Say we generated a_0 first. The chance of a match is $1/2$ because we select from $[1, 5]$. Then, a match at bit 2 has probability $1/2$ for the same reason. This repeats until there is a mismatch at the $i+1$-th bit after generating a_i, or all green bits match. This happens with probability $(1/2)^5 = (1/2)^{\log p} = 1/p$.

The equation from above gives us the equation $a_0^{(1)} = x^{(1)}$, denoting equal parity, where $y^{(j)}$ is the j-th bit of y and a_i are again the jump lengths. Since L is even and $a_i \in [1, L]$ (chosen u.a.r.), the probability of a_i being even (i.e. $a_0^{(1)} = 0$) is $1/2$. Thus, the equation is satisfied with the same probability due to symmetry.
Similarly, the equation $a_1^{(1)} + a_0^{(2)} = x^{(2)} \mod 2$ is satisfied with probability $1/2$ again:

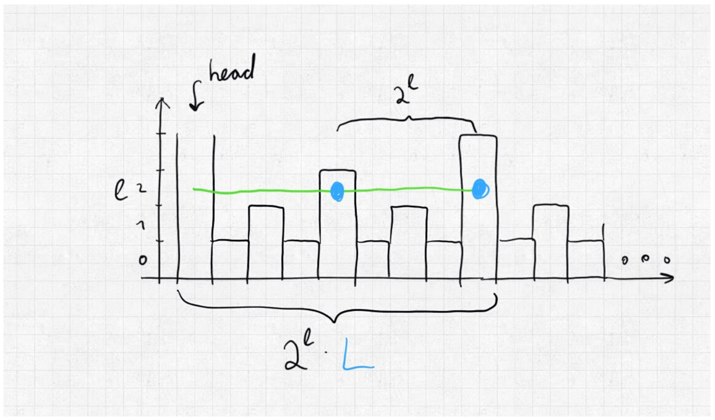
$$\begin{aligned}
\Pr[x^{(1)} = 0] &= \Pr[a_1(1) = 0] \cdot \Pr[a_0(2) = 0] + \Pr[a_1(1) = 1] \cdot \Pr[a_0(2) = 1] \\
&= \frac{1}{2} \Pr[a_0(2) = 0] + \frac{1}{2} \Pr[a_0(2) = 1] \\
&= 1/2
\end{aligned}$$

$a_{k-1}^{(1)} + a_{k-2}^{(2)} + ... + a_1^{(k-1)} + a_0^{(k)} + c = x^{(k)} \mod 2$ for a carry bit c is fulfilled with probability $1/2$ as well, for the same reason as with $k = 2$. The argument is that $a_{k-1}^{(1)}$ is independent of all other variables in the current equation and is symmetric. Thus flipping this one bit will flip the entire left-hand side of the equation, entailing symmetry for the latter too. Finally, $\Pr[\mathcal{E}] = (1/2)^{\log p} = 1/p$.

Bounding the furthest walk

In contrast to above, we now consider any SkipList (the general case) and ensure that not too many elements are skipped by the `Spray`, or else low

Figure 3: The expected walking distance for the greatest step choice



priority elements are returned. Since we have no control over the SkipList's element height distribution, we can only give probabilistic guarantees.

Basically, the goal is to bound the sum of the distances each part of the furthest `Spray` walks. (This means each part takes L steps.) We can loosen this by bounding each summand. The Markov or Chernoff bounds are powerful tools which require knowledge of our variable's expectation: In the previous section, we knew the exact distance of two consecutive elements with height $> l$. Interestingly, here this is the average value. Let E_l be the expected value of the distance of a walk of L steps on level l.

$$E_l = 2^l L$$

The argument is the very same: On average, one out of 2^l elements satisfy the condition *height* $> l$. Therefore a step at level l results in a distance of 2^l. For certain parameter choices, the probability of skipping the $(1 + \frac{1}{\log p})\frac{\log p}{\log p - 1}p$-th element is low ($p^-\Omega(1)$).

Proposition 1 will bound the distance of the individual parts of the `Spray`, while Lemma 1 will bound the entire distance, both probabilistically.

Fix $H = \log p - 1$, $L = M \log^3 p$ (M is some constant) and $D = \max(1, \log\log p)$. (These are the parameters that were chosen for the final algorithm.)

Let d_i be the distance which the i-th part of a `Spray` walks.

Recall the definition of E_l, which we will instantiate with $L = M \log^3 p$:

$$E_l = M 2^l \log^3 p$$

Proposition 1

If $k \leq \log p$ and $\alpha > 0$, then the distance traveled at level k is bounded by $(1+\alpha)M2^k \log^3 p$ with probability $\mathrm{e}^{-\Omega(M\alpha^2 \log^3 p)}$.

Proof: We apply a Chernoff bound, which yields the result:

$$\Pr[\mathit{distance} \geq (1+\alpha)M2^k \log^3 p = (1+\alpha)2^k L = (1+\alpha)\mathbb{E}[\mathit{distance}]] \leq \mathrm{e}^{-\Omega(M\alpha^2 \log^3 p)}$$

distance (at level k) can be thought of the sum of indicator variables, one for each element, that are 1 iff their corresponding elements are skipped.

Lemma 1

For a fixed α, the k-suffix of any `Spray` will go a distance of at most $M(1+\alpha)\frac{\log p}{\log p - 1} 2^{kD+1} \log^3 p$, with probability at least $1 - \mathrm{e}^{-\Omega(M\alpha^2 \log^3 p)}$ over the choice of the SkipList.

Proof:

$$\begin{aligned}
\Pr[\sum_{r=0}^{k} d_r \geq (1+\alpha)\sum_{r=0}^{k} E_{rD}] &\leq \sum_{r=0}^{k} \Pr[d_r \geq (1+\alpha)E_{rD}] && \text{(by the union bound)} \\
&\leq k \cdot \mathrm{e}^{-\Omega(M\alpha^2 \log^3 p)} && \text{(by Proposition 1)} \\
&\leq (\log p)\mathrm{e}^{-\Omega(M\alpha^2 \log^3 p)} \\
&\in \mathrm{e}^{-\Omega(M\alpha^2 \log^3 p)}
\end{aligned}$$

With the above probability following also holds:

$$\begin{aligned}
\sum_{r=0}^{k} d_r &\leq (1+\alpha)\sum_{r=0}^{k} E_{rD} \\
&= M(1+\alpha)\log^3 p \sum_{r=0}^{k} 2^{rD} \\
&\leq M(1+\alpha)\log^3 p \sum_{r=0}^{kD} 2^{r} \\
&\leq M(1+\alpha)\frac{\log p}{\log p - 1} 2^{kD+1} \log^3 p
\end{aligned}$$

Theorem 2

No Spray will return an element beyond the first $M(1+\frac{1}{\log p})\frac{\log p}{\log p-1}p\log^3 p$ with probability at least $1-p^{-\Omega(M)}$.

Proof: Instantiate Lemma 1 with $\alpha = \frac{1}{\log p}$ and $k = l_p$ (i.e. the full-height Spray).

The SprayList

So far we randomly selected a high-priority element by doing a Spray. This random walk with the parameter choice from the above section combined with the SkipList's Delete operation constitutes the DeleteMin/DeleteMax operation of our priority queue, the *SprayList*. The runtime of $\mathcal{O}(\log^3 p)$ is proven in the original paper.
Inserting is equivalent for the queue and the SkipList and can be done in $\mathcal{O}(\log n)$.

Conclusion

To conclude, the paper has presented a relaxed priority queue that attempts to land on one of the first $O(p\log^3 p)$ elements with (approximately) uniform distribution, utilizing random walks. It takes errors into account but scales better than any other proposed up to that point in terms of throughput. A user of the SprayList has to balance wasted work against dequeueing throughput by adjusting the relaxation parameter.

Additionally, low collision probabilitywas *proven* (the paper's Theorem 3 does so for the general case).

Several follow-up papers were exploring the gain of relaxation and provided comparisons with the SprayList (for up to 10^8 elements).
Exemplarily, the MultiQueue [2] performs the SprayList in both throughput and quality, according to the paper's benchmarks.
It operates on an array of priority queues, from which, on every MultiQueue-operation, one is pulled out at random. The original operation is then applied to the chosen priority queue. However, in the case of a DeleteMin, multiple priority queues are selected, their smallest values are compared and the smallest of these is returned. The taken queues are then put back.

An internal priority queueis blocked entirely during these operations. Nevertheless, this data structure seems to perform well. Though there is a minor downside: The error depends on $\sqrt{n}$, whereas the `Spray` is completely independent of the list's length.

References

[1] Dan Alistarh, Justin Kopinsky, Jerry Li, and Nir Shavit. The spraylist: A scalable relaxed priority queue. Technical report, September 2014. Best Artefact Award.

[2] Hamza Rihani, Peter Sanders, and Roman Dementiev. Multiqueues: Simpler, faster, and better relaxed concurrent priority queues. *arXiv preprint arXiv:1411.1209*, 2014.

[3] Nir Shavit. Data structures in the multicore age. *Commun. ACM*, 54(3):76–84, March 2011.