DOI:10.1145/1785414.1785434

Article development led by acmqueue
queue.acm.org

Think you've mastered the art of server performance? Think again.

BY POUL-HENNING KAMP

You're Doing It Wrong

WOULD YOU BELIEVE me if I claimed that an algorithm that has been on the books as "optimal" for 46 years, which has been analyzed in excruciating detail by geniuses like Knuth and taught in all computer science courses in the world, can be optimized to run 10 times faster?

A couple of years ago, I fell into some interesting company and became the author of an open source HTTP accelerator called Varnish, basically an HTTP cache to put in front of slow Web servers. Today Varnish is used by Web sites of all sorts, from Facebook, Wikia, and Slashdot to obscure sites you have surely never heard of.

Having spent 15 years as a lead developer of the FreeBSD kernel, I arrived in user land with a detailed knowledge of what happens under the system calls. One of the main reasons I accepted the Varnish proposal was to show how to write a high-performance server program.

Because, not to mince words, the majority of you are doing that wrong.

Not just wrong as in not perfect, but wrong as in *wasting half, or more, of your performance*.

The first user of Varnish, the large Norwegian newspaper VG, replaced 12 machines running Squid with three machines running Varnish. The Squid machines were flat-out 100% busy, while the Varnish machines had 90%

Figure 1. Comparison of runtime speeds of binary heap and B-heap.

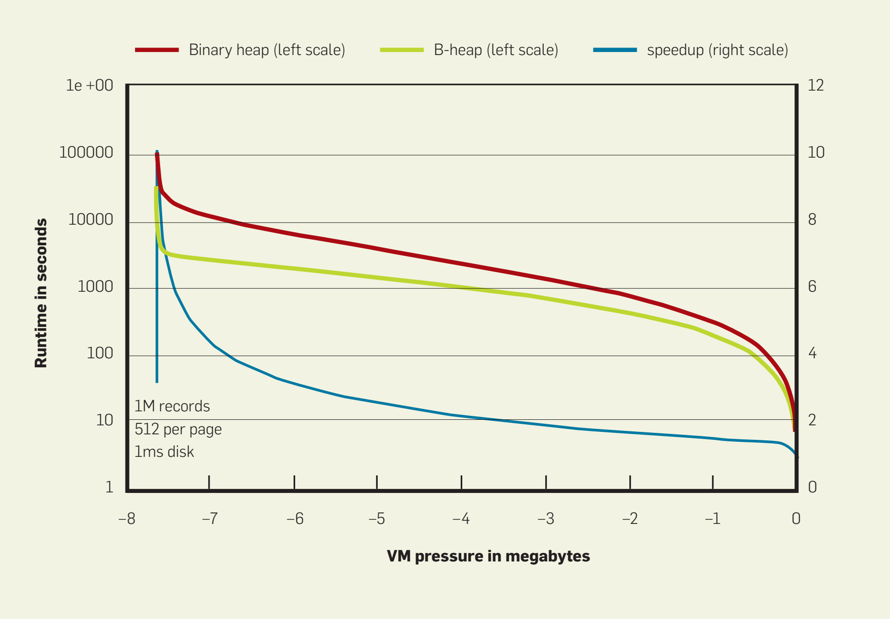
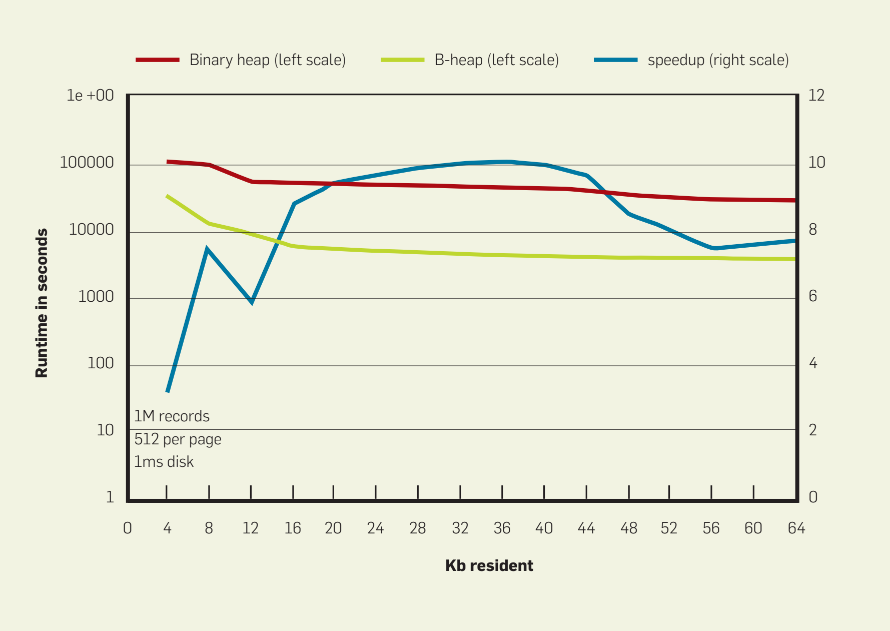


Figure 2. Close-up comparison of binary-heap and B-heap runtime speeds.



of their CPU available for twiddling their digital thumbs.[a]

The short version of the story is that Varnish knows it is not running on the bare metal but under an operating system that provides a virtual-memory-based abstract machine. For example, Varnish does not ignore the fact that memory is virtual; it actively exploits it. A 300GB backing store, memory mapped on a machine with no more than 16GB of RAM, is quite typical. The user paid for 64 bits of address space, and I am not afraid to use it.

One particular task inside Varnish is expiring objects from the cache when their virtual life-timers run out of sand. This calls for a data structure that can efficiently deliver the smallest keyed object from the total set.

A quick browse of the mental catalog flipped up the binary-heap card, which not only sports a `O(log2(n))` transaction performance, but also has a meta-data overhead of only a pointer to each object—which is important if you have over 10 million objects.

Careful rereading of Knuth confirmed that this was the sensible choice, and the implementation was trivial: "Ponto facto, Cæsar transit," and so on.

On a recent trip by night train to Amsterdam, my mind wandered, and it struck me that Knuth might be terribly misleading on the performance of the binary heap, possibly even by an order of magnitude. On the way home, also on the train, I wrote a simulation that proved my hunch right.

Before any fundamentalist CS theoreticians choke on their coffees: don't panic! The *P vs. NP* situation is unchanged, and I have not found a systematic flaw in the quality of Knuth et al.'s reasoning. The findings of CS, as we know it, are still correct. They are just a lot less relevant and useful than you think—at least with respect to performance.

The oldest reference to the binary heap I have located, in a computer context, is J.W.J. Williams' article published in the June 1964 issue of *Communications of the ACM*, entitled "Algorithm Number 232—Heapsort."[2,b] The trouble is, Williams was already out of touch, and his algorithmic analysis was outdated even before it was published.

In an article in the April 1961 issue of *Communications*, J. Fotheringham documented how the Atlas Computer at Manchester University separated the concept of an address from a memory location, which for all practical purposes marks the invention of virtual memory (VM).[1] It took quite some time before VM took hold, but today all general-purpose, most embedded, and many specialist operating systems use VM to present a standardized virtual machine model (such as POSIX) to the processes they herd.

Of course, it would be unjust and unreasonable to blame Williams for not realizing that Atlas had invalidated one of the tacit assumptions of his algorithm: only hindsight makes that observation possible. The fact is, however, 46 years later most CS-educated professionals still ignore VM as a matter of routine. This is an embar-

[a] This pun is included specifically to inspire Stan Kelly-Bootle.

[b] How wonderful must it have been to live and program back then, when all algorithms in the world could be enumerated in an 8-bit byte.

rassment for CS as a discipline and profession, not to mention wasting enormous amounts of hardware and electricity.

Performance Simulation

Enough talk. Let me put some simulated facts on the table. The plot in Figure 1 shows the runtime of the binary heap and of my new B-heap version for one million items on a 64-bit machine.[c] (My esteemed FreeBSD colleague Colin Percival helpfully pointed out the change I have made to the binary heap is very much parallel to the change from binary tree to B-tree, so I have adopted his suggestion and named my new variant a B-heap.[d])

The *x*-axis is VM pressure, measured in the amount of address space not resident in primary memory, because the kernel paged it out to secondary storage. The left *y*-axis is runtime in seconds (log-scale), and the right *Y*-axis shows the ratio of the two runtimes: (binary heap/B-heap).

Let's get my "order of magnitude" claim out of the way. When we zoom in on the left side in Figure 2, we see there is indeed a factor 10 difference in the time the two algorithms take when running under almost total VM pressure: only 8 to 10 pages of the 1,954 pages allocated are in primary memory at the same time.

Did you just decide that my order of magnitude claim was bogus because it is based on only an extreme corner case? If so, you are doing it wrong, because this is pretty much the real-world behavior seen.

Creating and expiring objects in Varnish are relatively infrequent actions. Once created, objects are often cached for weeks if not months, and therefore the binary heap may not be updated even once per minute; on some sites not even once per hour.

In the meantime, we deliver gigabytes of objects to clients' browsers, and since all these objects compete for space in the primary memory, the VM pages containing the binheap that are not accessed get paged out. In the worst case of only nine pages resident, the binary heap averages 11.5 page transfers per operation, while the B-heap needs only 1.14 page transfers. If your server has solid state drives (SSD), that is the difference between each operation taking 11 or 1.1 milliseconds. If you still have rotating platters, it is the difference between 110 and 11 milliseconds.

Figure 3. Close-up of the effect of VM pressure on binary-heap and B-heap runtime speeds.

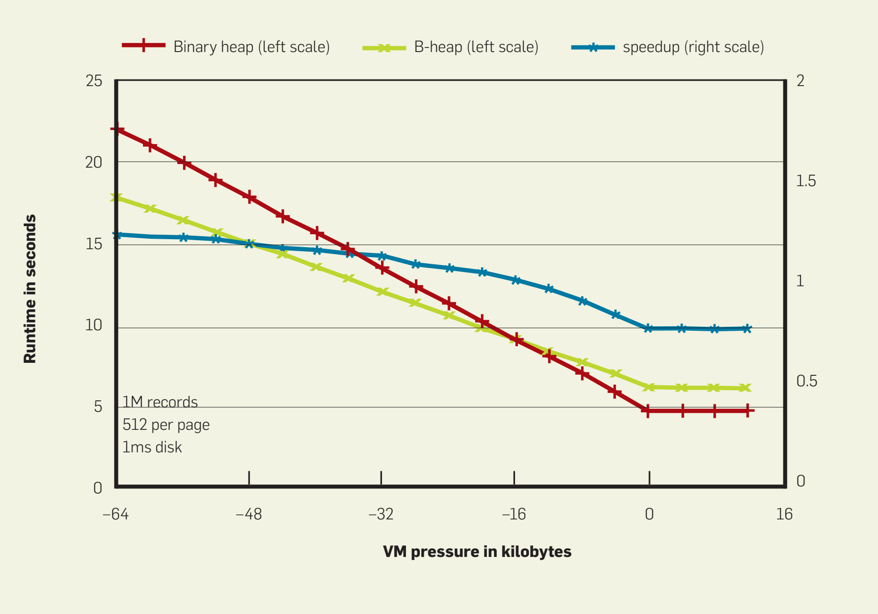
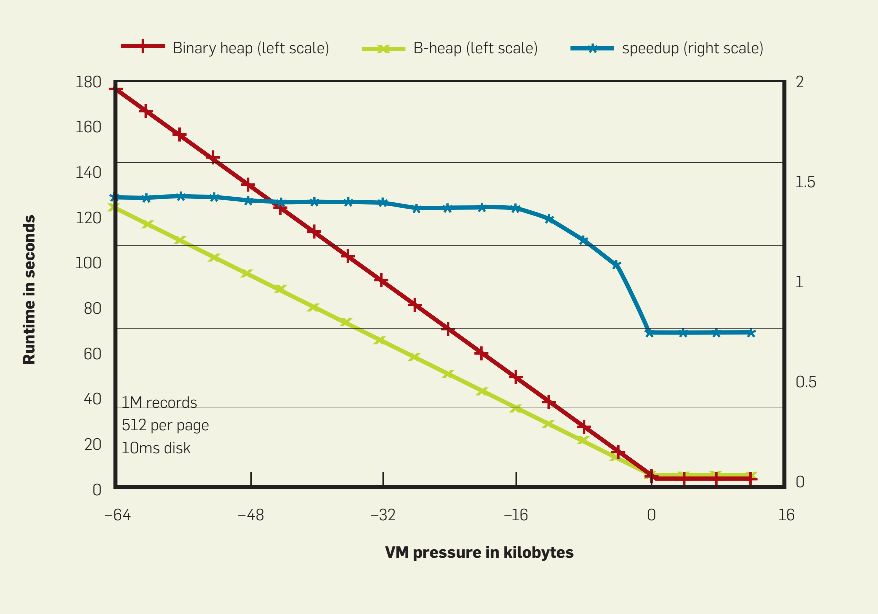


Figure 4. Comparisons of runtime speeds of binary heap and B-heap on a mechanical disk.



At this point, is it wrong to think, "If it runs only once per minute, who cares, even if it takes a full second?"

We care because the 10 extra pages needed once per minute loiter in RAM for a while, doing nothing—until the kernel pages them back out again, at which point they get to pile on top of the already frantic disk activity, typically seen on a system under this heavy VM pressure.[e]

[c] Page size is 4KB, each holding 512 pointers of 64 bits. The VM system is simulated with dirty tracking and perfect LRU page replacement. Paging operations set to 1 millisecond. Object key values are produced by random(3). The test inserts one million objects, then alternately removes and inserts objects one million times, and finally removes the remaining one million objects from the heap. Source code is at http://phk.freebsd.dk/B-Heap.

[d] Does *Communications* still enumerate algorithms, and is eight bits still enough?

[e] Please don't take my word for it: applying queuing theory to this situation is a very educational experience.

Next, let us zoom in on the other end of the plot (Figure 3). If there is no VM pressure, the B-heap algorithm needs more comparisons than the binary sort, and the simple parent-to-child / child-to-parent index calculation is a tad more involved: so, instead of a runtime of 4.55 seconds, it takes 5.92 seconds—a whopping 30% slower; almost 350 nanoseconds slower per operation.

So, yes, Knuth and all the other CS dudes had their math figured out right.

If, however, we move left on the curve, then we find, at a VM pressure of four missing pages (= 0.2%) the B-heap catches up, because of fewer VM page faults; and it gradually gets better and better, until as we saw earlier, it peaks at 10 times faster.

That was assuming you were using an SSD, which can do a page operation in 1 millisecond—pretty optimistic, in particular for the writes. If we simulate a mechanical disk by setting the I/O time to a still-optimistic 10 milliseconds instead (Figure 4), then B-heap is 10% faster as soon as the kernel steals just a single page from our 1,954-page working set and 37% faster when four pages are missing.

Figure 5. Binary-heap tree structure.

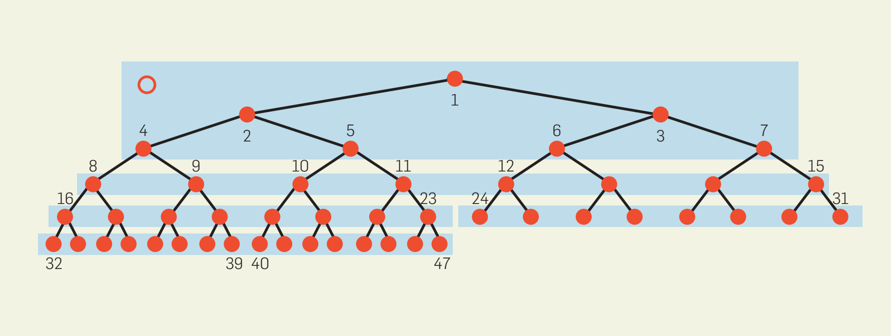


Figure 6. B-heap tree structure.

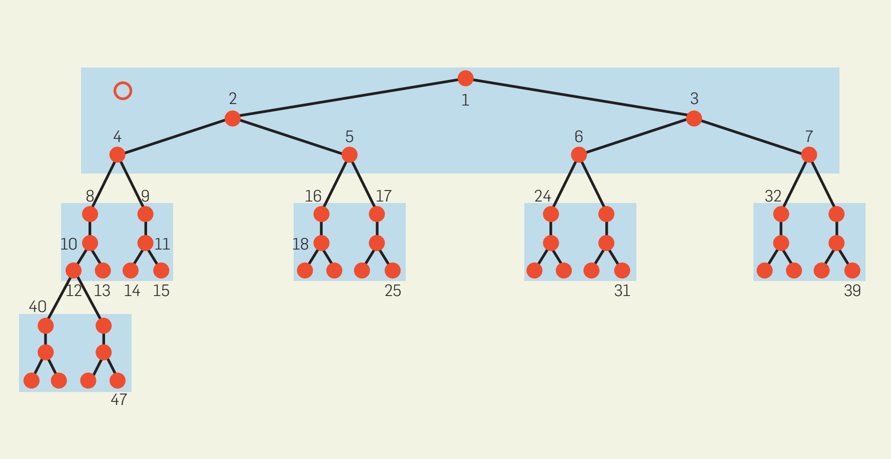
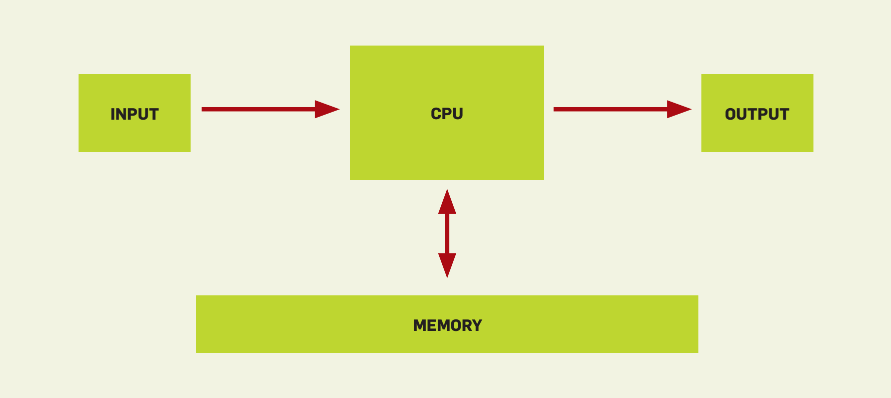


Figure 7. Outdated computer model.



So What is a B-Heap, Anyway?

The only difference between a binary heap and a B-heap is the formula for finding the parent from the child, or vice versa.

The traditional `n -> {2n, 2n+1}` formula leaves us with a heap built of virtual pages stacked one over the next, which causes (almost) all vertical traversals to hit a different VM page for each step up or down in the tree, as shown in Figure 5, with eight items per page. (The numbers show the order in which objects are allocated, not the key values.)

The B-heap builds the tree by filling pages vertically, to match the direction we traverse the heap (Figure 6). This rearrangement increases the average number of comparison/swap operations required to keep the tree invariant true, but ensures that most of those operations happen inside a single VM page and thus reduces the VM footprint and, consequently, VM page faults.

Two details are worth noting:

- Once we leave a VM page through the bottom, it is important for performance that both child nodes live in the same VM page, because we are going to compare them both with their parent.
- Because of this, the tree fails to expand for one generation every time it enters a new VM page in order to use the first two elements in the page productively.

In our simulated example, failure to do so would require five pages more.

If that seems unimportant to you, then you are doing it wrong: try shifting the B-heap line 20KB to the right in figures 2 and 3, and think about the implications.

The parameters of my simulation are chosen to represent what happens in real life in Varnish, and I have not attempted to comprehensively characterize or analyze the performance of the B-heap for all possible parameters. Likewise, I will not rule out that there are smarter ways to add VM-clue to a binary heap, but I am not inclined to buy a ticket on the Trans-Siberian Railway in order to find time to work it out.

The order of magnitude of difference obviously originates with the number of levels of heap inside each VM page, so the ultimate speedup will be on machines with small pointer sizes and big page sizes. This is a pertinent observation, as operating system kernels start to use superpages to keep up with increased I/O throughput.

So Why are You, and I, Still Doing it Wrong?

An (in)famous debate, "Quicksort vs. Heapsort," centered on the fact that the worst-case behavior of the former is terrible, whereas the latter has worse average performance but no such "bad spots." Depending on your application, that can be a very important difference.

We lack a similar inquiry into algorithm selection in the face of the anisotropic memory access delay caused by virtual memory, CPU caches, write buffers, and other facts of modern hardware.

Whatever book you learned programming from, it probably had a figure within the first five pages diagramming a computer much like the one shown in Figure 7. That is where it all went pear shaped: that model is totally bogus today.

Amazingly, it is the only conceptual model used in computer education, despite the fact that it has next to nothing to do with the execution environment on a modern computer. And just for the record: by modern, I mean VAX 11/780 or later.

The past 30 or 40 years of hardware and operating-systems development seems to have only marginally impinged on the agenda in CS departments' algorithmic analysis sections, and as far as my anecdotal evidence, it has totally failed to register in the education they provide.

The speed disparity between primary and secondary storage on the Atlas Computer was on the order of 1:1,000. The Atlas drum took two milliseconds to deliver a sector; instructions took approximately two microseconds to execute. You lost around 1,000 instructions for each VM page fault.

On a modern multi-issue CPU, running at some gigahertz clock frequency, the worst-case loss is almost 10 million instructions per VM page fault. If you are running with a rotating disk, the number is more like 100 million instructions.[f]

What good is an `O(log2(n))` algorithm if those operations cause page faults and slow disk operations? For most relevant datasets an `O(n)` or even an $O(n^2)$ algorithm, which avoids page faults, will run circles around it.

Most CS-educated professionals still ignore VM as a matter of routine. This is an embarrassment for CS as a discipline and profession, not to mention wasting enormous amounts of hardware and electricity.

Performance analysis of algorithms will always be a cornerstone achievement of computer science, and like all of you, I really cherish the foldout chart with the tape sorts in Volume 3 of *The Art of Computer Programming*. But the results coming out of the CS department would be so much more interesting and useful if they applied to real computers and not just toys like ZX81, C64, and TRS-80.

[f] And below the waterline there are the flushing of pipelines, now useless and in the way, cache content, page-table updates, lookaside buffer invalidations, page-table loads, etc. It is not atypical to find instructions in the "for operating system programmers" section of the CPU data book, which take hundreds or even thousands of clock cycles, before everything is said and done.

Related articles on queue.acm.org

Thread Scheduling in FreeBSD 5.2
Marshall Kirk McKusick and George V. Neville-Neil
http://queue.acm.org/detail.cfm?id=1035622

Flash Storage Today
Adam Leventhal
http://queue.acm.org/detail.cfm?id=1413262

High Performance Web Sites
Steve Souders
http://queue.acm.org/detail.cfm?id=1466450f

References

1. Fotheringham, J. Dynamic storage allocation in the Atlas Computer, including an automatic use of a backing store. *Commun. ACM 4*, 19 (Apr. 1961), 435–436.
2. Williams, J. W. J. Algorithm 232—Heapsort. *Commun. ACM 7*, 6 (June 1964), 347–348.

Poul-Henning Kamp (phk@FreeBSD.org) has programmed computers for 26 years and is the inspiration behind bikeshed.org. His software has been widely adopted as "under the hood" building blocks in both open source and commercial products. His most recent project is the Varnish HTTP accelerator, which is used to speed up large Web sites such as Facebook.