FROM THE EDITOR

Size Still Matters

Despite servers with terabytes of RAM, executable size still matters.

by Andrew Binstock

We now live in an age in which servers can host more than 1 TB of RAM and where so-called spindle farms are populated with hard drives weighing in at 12 TB or more of capacity. For the average consumer, smartphones with 128 GB of storage are no longer uncommon. You would think with all this space (at amazingly low price points) that we could finally let go of concerns about the size of executable programs. But we can't. While the capacity has grown, the problem of executable size remains frustratingly constant.

Take, for example, the theme of this issue: lightweight frameworks. The three frameworks we cover (Javalin, Micronaut, and Helidon) all tout their small size, which is presented as a proxy for speed and simplicity of programming. But in fact, the main benefit of the small size is to fit within the architectural metaphor of microservices. Those microservices are universally deployed in containers, which themselves were created in response to the considerable size of virtual machines.

It's important to note here that the argument for microservices is not the usual one: less complexity. Microservices bring many benefits, but it is far too early—in my opinion and that of many others—to know whether in fact they deliver an equivalent computing experience with less complexity. If you ever have tried to locate an intermittent bug in a microservices implementation, you have firsthand knowledge of the complexity problem.

But returning to the question of size. The size of executables has been an issue for programmers since the beginning of business computing. Memory arrays for IBM computers in the early 1960s cost $1 a byte (which would be $8.57 a byte in today's dollars). Because of this cost, most development was done in assembly language and programmers had to be exceedingly knowledgeable about the effect of their code on the binary size. They worked much like embedded programmers—equally skilled at coding and shoehorning.

Two decades later, the problem still existed but without such tight strictures. For example, UNIX workstations for much of the 1980s and early 1990s were considered top of the line if they had 16 MB of RAM. And software that used a lot of that memory was heavily criticized because it greatly slowed other operations. For example, emacs, the editor upon which millions of developers depended, was widely mocked for its memory consumption as being an acronym for "eight megabytes and continually spooling." The irony is that today, no editor is small enough to use only 8 MB of memory. In those days, memory was a hard limit—if you blew the limit, performance crawled, or on some systems, like the early Macs, the system would simply hang.

Today, with hundreds of gigabytes of RAM on high-end workstations and servers, the limits are soft. This is especially true in the cloud, where servers of more than 1 TB are commonly available. Executable size still matters, though, because in a microservice architecture, you could have

hundreds of instances of services running in the same memory space. In such a situation, you want those instances to be small so that the remaining RAM can be dedicated to the data itself. And if that data's needs decline, you want to be able to shrink your memory usage to lower your cloud infrastructure bill. So for these soft reasons, economy of resources and of spending, executable size remains an important aspect.

What is not entirely clear is whether microservices do in fact lower memory usage. Containers hold duplicated code—both user and system code. Run enough containers and you are likely to surpass the execution footprint of the monolithic server that those services aim to replace. However, services deliver a greater scalability that servers cannot match, and this benefit alone might offset the potential for excess memory consumption.

But regardless of whether your architecture reaches that tipping point, you'll always be well served by true economy in executable size.

Also in This Issue

Photograph by Bob Adler/Getty Images

Andrew Binstock

Andrew Binstock (javamag_us@oracle.com, @platypusguy) is the editor in chief of *Java Magazine*. Previously, he was the editor of *Dr. Dobb's Journal*. He co-founded the company behind the open-source iText PDF library, which was acquired in 2015. His book on algorithm implementation in C went through 16 printings before joining the long tail. Previously, he was the editor in chief of *UNIX Review* and, earlier, the founding editor of the *C Gazette*. He lives in Silicon Valley with his wife. When not coding or editing, he studies piano.

Share this Page