All Sliders to the Right

HARDWARE OVERKILL

Dear KV,

I work for a company that has what I can only describe as a voracious appetite for expensive servers, but it seems that simply buying systems with "all sliders to the right"—as one of our devs calls it—is not getting us the performance we're expecting. We are buying the biggest boxes with the most cores and the highest CPU frequencies, but the difference between this year's model and last year's seems not to be that great. We've optimized all our code for multithreading, and yet, something seems to be not quite right. Our team has been digging into this with various tools, some of which seem to give conflicting answers, but maybe it's just that machines really aren't getting that much faster each year, and it's time to step off this expensive merry-go-round for a year or two.

Is Bigger Really Better?

Dear Bigger,

There are many reasons why this year's model isn't any better than last year's, and many reasons why performance fails to scale, some of which KV has covered in these pages. It is true that the days of upgrading every year and getting a free—wait, not free, but expensive—performance boost are long gone, as we're not really getting single cores that are faster than about 4GHz. One thing that many software developers fail to understand is

the hardware on which their software runs at a sufficiently deep level.

Those who are working in the cloud have little choice but to trust their cloud provider's assertions about the performance of any particular instance, but for those of us who still work with real metal—and it seems that you're one of us—there are a plethora of parameters about the underlying hardware that matter to the overall performance of the system. In no particular order, these parameters include the sizes of the various caches, the number of cores, the number of CPUs, and, crucially, the oft-overlooked bus structure of the system.

The bus structure of a modern server is no longer flat, which means that access to memory and, in particular, I/O devices may go over a much slower path than you assume.

The diagrams we show computer science students that purport to illustrate how the CPU, memory, and I/O devices are connected in a computer are woefully out of date and often represent an idealized computer from about 1970 or 1975. These diagrams are, of course, complete fictions when you're looking at a modern server, but people still believe them. The most common big server used in a data center is a two-CPU system with a lot of memory and, hopefully, some fast I/O devices inside such as a 10-100G network card and fast flash storage. All the components are fast in and of themselves, but how they are connected may surprise you. The bus structure of a modern server is no longer flat, which means that access to memory and, in particular, I/O devices may go over a much slower path than you assume.

For example, each I/O device, such as a NIC (network interface card) or flash memory device, is close to only one of the two CPUs you've put into that big server, meaning that if your workload is I/O intensive, the second

CPU is actually at a disadvantage and, in many cases, is completely useless from a performance standpoint. Each I/O transaction to the second CPU may even have to traverse the first CPU to get to the I/O devices, and the bus between the CPUs may be far more constrained than the connection between either CPU and its memory or cache. It is for all these reasons that sometimes, depending on what your system does, it may be far more cost-effective, and efficient, to buy systems with only a single CPU, because on a single-CPU system all cores are as close as possible to cache, memory, and I/O devices.

Of course, it's not like the system manufacturers explain any of this, and in fact, there was a time when one had to scream at their sales reps to get them to admit that any of this was the case. KV tries not to scream at sales reps too often, not because he likes sales reps, but because he's afraid of losing his voice. Mostly, you wind up reaching various data sheets about the bus structure that exists in whatever system you're buying and then hope that the documents you found online, either at the manufacturer's website or in a more tech-focused news provider, like the ones that talk about high-end gaming systems, aren't lying to you. And when I say *lying*, I mean that I am sure the companies are totally honest and upstanding but that they forgot to put the errata online. Companies would never lie about the performance problems inherent in their highest-end systems. And if you believe that, remember, KV lives in Brooklyn, and we have bridges.

All of this gets a bit complicated because it depends on your workload and what you consider to be important. Are you CPU-bound? If so, then you care more about raw

CPU power than anything else. Are you I/O-bound? Then you care more about I/O transactions per second. And then there are the moments when management says, "Just make it go faster." At which point, KV goes out and gets blind drunk and throws up on the CEO's fancy car after first checking to make sure the garage cameras are covered with masking tape.

Another thing that makes all this difficult, as you point out, is that the tooling for looking at performance is all over the place. There are books written about this stuff (and I happen to favor things written by Brendan Gregg. See "*The Flame Graph*," published in *queue* in 2016), but they really only scratch the surface. It pays to spend a good deal of time reading over those data sheets, thinking about where your code runs and what it needs, and then designing experiments to see if this year's model really is better than last year's for your particular application. In servers, as in much else, it's not the size that matters so much as how you use what you've got.

KV

George V. Neville-Neil *works on networking and operating-system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are computer security, operating systems, networking, time protocols, and the care and feeding of large code bases. He is the author of* The Kollected Kode Vicious *and co-author with Marshall Kirk McKusick and Robert N. M. Watson of* The Design and Implementation of the FreeBSD Operating System. *For nearly 20 years, he has been the columnist better*

known as Kode Vicious. Since 2014, he has been an industrial visitor at the University of Cambridge, where he is involved in several projects relating to computer security. He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. His software not only runs on Earth, but also has been deployed as part of VxWorks in NASA's missions to Mars. He is an avid bicyclist and traveler who currently lives in New York City.

CONTENTS