ANDREW HUME

how's your OS these days?

Andrew Hume is a senior researcher at AT&T Labs. Over the past 10 years or so, he has worked on big data problems and the cluster infrastructure needed to support such applications. He believes in end-to-end checks, and in both the compelling price/performance and utter fallibility of modern PC hardware running UNIX-like operating systems.

■ *andrew@research.att.com*

IN 2003 I GAVE A KEYNOTE ADDRESS at HotOS IX about the reliability, or lack thereof, of OSes commonly used to build computing systems and clusters. I spent much time detailing examples of aberrant behavior, some of which were entertaining (if it didn't happen to you) and some just outright perplexing. While certain people, notably including those who sell software, understood my points well, I think many were not quite sure what to make of my charges.

To be truthful, I was not sure what response I wanted either. Certainly, I wanted to challenge the (often smug) complacency of the FREENIX crowd who believe in the technical superiority of their particular OS. I put forward the notion that properties that let you combine individual systems into clusters—for example, predictable and bounded behavior—used to be fairly common but nowadays seem less so.

The obvious question is, has anything changed over the last two years? The answer is clearly yes. Every release of every OS brings its own new set of "features," or bugs, as we used to call them. For example, the egregious I/O problems we had with the 2.4 Linux kernels seem to have gone away with the 2.6 kernels. Of course, new problems appear; when we are pounding away at writing to SCSI tape (at a massive 5MB/s), the buffer cache seems to vanish and becomes very slow to replenish, especially for pages read in nonsequential order. Scanning a 100MB gdbm database, which normally takes 1 or 2 seconds, starts taking anywhere from 5 to 50 minutes. I understand full well the consequences of flushing the buffer cache, but writing to a slow tape seems an inadequate reason. We find fewer bugs with each release, but the number is still decidedly nonzero.

I also want to make clear what I mean by "bug" here. I do not just mean when the OS does something wrong (more on this below), but when it does a right thing in an untimely fashion. We saw one example above, when the time needed to scan a modest database averages 1 to 2 seconds but can take 50 minutes.

So what's the problem? When we try, as my team does, to build reliable and/or highly available computing infrastructure out of nodes that are only modestly reliable, it is necessary to detect node failure. When we execute some work on a node, there is an associated time limit (called a "lease") for that work, and if the lease expires before the work completes, we assume that the node died and assign the work to another node. This allows the general workflow to continue despite nodes failing—but we now have to parameterize the leases. If the leases are too short, there will be wasted work as we re-execute the work unnecessarily. If the leases are too long, as in the case of an actual node failure, we'll spend unnecessary time waiting for work to finish when it never will.

Another example of a time-related bug is recycling a server. Recycling a server means halting a service (which would result in the open port being closed). On most UNIX-like OSes, one can simply unmount the bind and exit; almost immediately (in a second or less), one can re-execute the server process, which will then be able to bind and proceed on. On all the Linuxen we've tried, this just fails, and we end up waiting a fairly long time before we can restart successfully (we initially wait 15 seconds, and back off exponentially to a maximum of 75 seconds). The variance of how long a wait is required (again, it seems to depend on how busy the system is) is annoying, and directly increases service unavailability.

For outright bugs, two examples come to mind. The first is the weakness of the FreeBSD SCSI system; we cannot reliably write tapes on our FreeBSD nodes (although at least we get told about the errors!). Again, the tape is slow (5MB/s) and should not be an issue, and we can reliably write them on Linux (on more or less identical hardware). Although this is annoying, it turns out reading a tape works just fine, so we're not too annoyed.

The other example is perhaps not an OS bug per se, but rather, I think, a hardware weakness, so common these days. We configure our PCs with a 3Ware controller plugged into the PCI bus, and all our disks (2–7 per node) plug into the 3Ware. Because our nodes are typically 1U systems, we need an extender board so that we can mount the 3Ware controller horizontally. It turns out that the 3Ware board is overly fussy about termination and really only operates reliably when *actively* terminated, not passively (as is the norm). We didn't really care which way it needed to be terminated; what did piss us off was the complete lack of error detection by everyone involved. No errors were logged or detected by either drivers or diagnostics. Perhaps the driver doesn't see an error, or the hardware doesn't have ECC, but this is bad.

The only diagnostic that worked was "copy 2GB to 5GB of files and checksum every copy and verify that the copies were good." In many ways, this is an admirable end-to-end test, but it also always seemed a very gross test.

Which brings me to my final thought. Many people have listened to my tales of woe, and the almost universal response is "Why are you so unlucky?" (because they not only don't see these problems, they've never even heard of them before). Certainly, I pound on my systems and work them hard. But I'm sure I'm not alone in this (although relatively few people schedule jobs in batches of 75,000–150,000, or move TBs of files around a 10 to 12-node cluster). I think the significant difference is that I *check* everything I can. All file movement is md5summed and, where plausible, we add consistency checks to verify our processing.

For example, we have a distributed logging system where the logging routine ensures that at least three systems got the log message. Each system then generates a file of the recent log messages every five minutes, and these are collected and coalesced on a central node into a single file per week. After the coalescing, we check the result by sorting all the five-minute files into an "input" pile, sorting all the weekly files we updated into an "output" pile, and then verifying that the input pile is a strict subset of the output file. You might think this a tedious, expensive check of demonstrably correct code (the shell script that does this is quite simple). But so far this check has found at least seven bugs that would have otherwise probably not been found. This includes not only bugs in the five-minute file generator, but also rude behavior by the system sort utility (returning success even though the temp file system ran out of space), and even by rcp (copying to a full file system not only returns success but also sets the file's length to the right amount, even though it failed earlier on). And thus every time I think about taking out this apparently redundant test, I think of how the system is out to get me, and I leave the test in.

So my standard answer to the question, "Why do you see so many errors?" is, "I care about the answers and check that they're right." Sometimes I wonder why more people don't have the same answer. Don't you?