The Observer Effect

FINDING THE BALANCE BETWEEN ZERO AND MAXIMUM

who is KV?

click for video

Dear KV,

The company I work for rolled out a new monitoring system one weekend, and it didn't go as well as we would have liked. When we first brought up the monitoring system, several of our servers started to show very high CPU load. Initially, we could not figure out why. The monitoring processes on each server were very busy, so we turned off the monitoring system and the servers got less busy. Eventually, we realized it was the number of polls being issued by the monitoring system that was causing the servers to use so much CPU time. We decreased the polling frequency to every 10 minutes, and this seemed to be the sweet spot for system performance. What I would like to know is how one should go about tuning such systems, as it seems still to be done via trial and error.

Polled Too Frequently

Dear Polled,

Trial and error? The problem here is usually a failure to appreciate just what you are asking a system to do when polling it for information. Modern systems contain thousands—sometimes tens of thousands—of values that can be measured and recorded. Blindly retrieving whatever it is that might be exposed by the system is bad enough, but asking for it with a high-frequency poll is much worse for several reasons.

The first reason is the one that you bring up in your letter: the amount of overhead introduced by simply asking for the data. Whenever you ask the system for its configuration state, whether that's a routing table or the state of various sysctls (system control variables), the system has to pause other work to provide a consistent picture of what's going on. KV knows that in recent years the idea of consistency has been downplayed in favor of performance—in particular, by various database projects. In the systems world, however, we still think that consistency is *a good thing*™ and therefore the system will try either to snapshot the data you request or to pause other work while the data is read out. If you ask for a few thousand items, and a random `sysctl -a` shows 9,000+ elements on a server I am using, then that's going to take time—not forever but not nothing, either.

The second reason that polling for data frequently is a problem is that it actually hides the information you might be looking for in the noise generated by retrieving and communicating the values you asked for. Every time you ask the system for some stats, it has to do work to get those stats, and the system doesn't account for your request separately from any other work it has to do. If your monitoring system is banging away at the server asking for data every minute, then what you will see in your monitoring system is the load that the system itself is generating. Such Heisen-monitoring, where your monitoring system is overwhelmingly affecting the measurements, is completely pointless.

In a monitoring system, there is always the tension between too much and too little information. When you're

debugging a problem, you always wish you had more data, but when your system is running normally, you want it to do the work for which it was deployed. Unless you get off on just pushing monitoring systems—and, yes, there is definitely a handle for those people somewhere on social media—you need to find the Goldilocks zone for your monitoring system. To find that zone, you must first know what you're asking for. Figure out which commands the monitoring system is going to execute on your servers, and then run them individually in a test environment and measure the resources they require. You care about runtime, which can be found to a coarse level with the `time(1)` command. Here is an example from the server just mentioned.

```
time sysctl -a > /dev/null
sysctl -a > /dev/null 0.02s user 0.24s system 98%
cpu 0.256 total
```

Here, grabbing all of the system's various system-control variables takes about a quarter of a second of CPU time, most of which is system overhead—that is, time spent in the operating system getting the information you requested. The `time(1)` command can be used on any utility or program you choose.

Now that you have a rough guess as to the amount of CPU time that the request might take, you need to know how much data you're talking about. Using a program that counts characters, such as `wc(1)`, will give you an idea of how much data you're going to be gathering and moving off the system for each polling request.

```
sysctl -a | wc -c
378844
```

You would be grabbing more than a quarter of a megabyte of data here, which in today's world, isn't much, but it still averages out to 6,314 bytes per second if you poll every minute; and, in reality, the instantaneous rate is much higher, causing a 3-Mbps blip on the network every time you request those values.

Of course, no one in his or her right mind would just blindly dump all the `sysctl` values from the kernel every minute—you would be much more nuanced in asking for data.

Of course, no one in his or her right mind would just blindly dump all the `sysctl` values from the kernel every minute—you would be much more nuanced in asking for data. KV has seen a lot of unsubtle things in his time, including monitoring systems that were set up to do just this sort of ridiculous level of monitoring. "We don't want to lose any events; we need a transparent system to find bugs!" I hear the DevOps folks cry. And cry they will, because sorting through all that data to find the needle in the noise will definitely not make them happier or give them the ability to find the bug.

What is needed in any monitoring system is the ability to increase or reduce the level of polling and data collection as system needs dictate. If you're actively debugging a system, then you probably want to turn the volume of data up to 11, but if the system is running well, you can dial the volume back down to 4 or 5. The volume can be thought of as the polling frequency times the amount of data being captured. Perhaps you want more frequent polling but less data per request, or perhaps you want more data for a broader picture but polled less frequently. These are the horizontal and vertical adjustments you should be

able to make to your system at runtime. A one-size-fits-all monitoring system fits no one well. The fear, of course, is that by not having the volume at 11 you will miss something important—and that is a valid fear—but unless the whole reason for your existence is to capture all events at all times, you will have to find the right balance between 0 and maximum volume.

KV

Kode Vicious, *known to mere mortals as 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 code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). 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. Neville-Neil is the co-author with Marshall Kirk McKusick and Robert N. M. Watson of* The Design and Implementation of the FreeBSD Operating System (second edition). *He is an avid bicyclist and traveler who currently lives in New York City.*

Related articles

Kode Vicious Bugs Out
http://queue.acm.org/detail.cfm?id=1127862

A Conversation with Bruce Lindsay
Designing for failure may be the key to success.
http://queue.acm.org/detail.cfm?id=1036486

Software Needs Seatbelts and Airbags
Emery D. Berger
Finding and fixing bugs in deployed software is difficult and time-consuming. Here are some alternatives.
http://queue.acm.org/detail.cfm?id=2333133