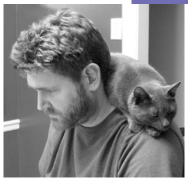
DAVE JOSEPHSEN

iVoyeur: a question of scale



Dave Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is senior systems engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.

dave-usenix@skeptech.org

"MUST I REALLY BE LECTURED EVERY time I want to use the Internet?!" demands my wife, looking at me over the lid of the laptop. This strikes me as the sort of question that warrants a politically savvy answer but I, as usual, am at a loss.

"Sorry?" I reply.

"It's always going on about how the local sysadmin should be lecturing me and what great power I'm being given. There's a word for this, what is it? Oh yes, *annoying*."

Ah-hah. The rest I can guess; she picked up my laptop wanting to get online and, finding it configured for the office network, has opened a term to run the script to reconfigure it for the home network. Upon typing "sudo home eth0", she was presented with sudo's "Are you sure you know what you're doing?" prompt. You've seen it, I'm sure. It looks something like this:

```
We trust you have received the usual lecture
from the local System Administrator. It usually
boils down to these three things:
#1) Respect the privacy of others.
#2) Think before you type.
#3) With great power comes great responsibility.
```

Great responsibility indeed. On the right box and in the wrong hands all manner of mischief might ensue. Millions of stolen credit cards, mixed up MRIs, disabled battleships, who knows? Sudo doesn't. Linux is a scalable beast if nothing else; we might be running on a wristwatch, or we might be running on . . . well, something really big, but, either way, sudo cautions us with the same message.

Although no doubt annoying to the wives of the world's techno-curmudgeons, it is the mark of great software that it scales beyond the architecture it was intended to run on. There's probably a natural law inherent here; I can only imagine what the spouses of typical VoIP systems engineers have to endure. By this yardstick, however, some classes of software that we sysadmins rely upon daily fall surprisingly short. This being a monitoring column, it shouldn't be hard for the reader to guess the general area on which my gaze is currently falling.

Every monitoring system I've ever worked with, from lofty Open View to humble Big Brother, has scalability problems. So, at least in the context of systems monitoring, I don't think this is necessarily endemic of bad design. Rather, my suspicion, simply stated, is that monitoring 1,000 services is in fact an entirely different problem from monitoring 100,000 services. Like the difference between crossing the Pacific by air and by orbit, the problems are closely related, separated mostly by scale, and require drastically different design considerations. We shouldn't be surprised to find the vehicle designed to make both trips a bit unwieldy.

Even my beloved Nagios has well publicized [1] scalability issues on the high end, but it has one thing going for it that other monitoring systems do not: the Event Broker. Over the years, sysadmins have come up with some pretty kludgy solutions to work around Nagios's scalability problems, so in this month's article I'd like to share with you two (in my opinion) very elegant Event Broker–based solutions for scaling Nagios to very large environments.

DNX

If you attended LISA '09's Nagios Guru session, you met Kyle Martin and Adam Augustine from The Church of Jesus Christ of Latter-day Saints, who spoke in depth about Nagios and Unnoc. What I wish they had been there to talk about, however, was their excellent Event Broker module "DNX" (Distributed Nagios eXecutor) [2].

As you probably already know, a normal Nagios server executes host and service checks by scheduling the execution of small, single-purpose, locally stored programs called plug-ins. These programs may be written in any language and can do any sort of checking, as long as they return standardized output back to the Nagios daemon to interpret. A DNX-enabled Nagios server does pretty much the same thing, with the small exception that just before Nagios executes the plug-in, the DNX event broker module wakes up, and checks to see if any subordinate worker nodes have asked for jobs to execute. If a worker node has asked for a job, the DNX module hands the service check to the worker node instead.

Worker nodes are remote machines in the network that are not running Nagios but have a copy of the Nagios plug-ins and are running the DNX client. The client software is run from init and requests jobs from the central Nagios host on a regular basis. They request and receive a job from the central Nagios server using a network socket and then perform the task and provide the input back to the Nagios server on a different socket.

If a worker node goes down, it, ipso facto, stops requesting jobs from the Nagios server. If a worker node goes down after it's been given a job and before it returns the status of that job, the check will time out and Nagios will reschedule it. If all the worker nodes go down, the DNX module will not have any available worker nodes to hand jobs to, and Nagios will operate as it normally does.

The Church reports running 2000 checks per minute with their DNX setup (10,000 checks in a five-minute interval) [3], and they feel it could go much higher. Steven Morrey reported to the Nagios-devel list that the Church's Nagios daemon spends two-thirds of its time reaping check results from the results ring-buffer [4], which is promising news. A DNX-enabled Nagios server is limited mainly by the speed of the reaper process.

DNX requires no modification to your existing configuration files beyond the addition of a single line to your Nagios.cfg to load the module. No replication of configuration to the client nodes is required, as there would be to configure passive checks. The clients need to know where the server is, the server needs to know what clients are allowed to connect, and that's about it; configuration is as easy as it gets for a Nagios box. Simple, efficient, and elegant, DNX is such great design that it makes me want to buy those folks beer . . . or whatever it is that's analogous to beer in their universe.

OP5 Merlin

Merlin (Module for Endless Redundancy and Loadbalancing In Nagios) is, by comparison, a fair bit more difficult to describe. This is because its goals are—despite the name—appreciably loftier than redundancy and load balancing [5].

If you've read any of my previous articles about the Event Broker, then you know the Broker allows you to hook into pretty much any aspect of a running Nagios daemon by allowing your module to register for callbacks, which trigger when an event happens. The expectation is that a given module will register for the event types it is interested in, and do something useful with the event callbacks it's given. DNX, for example, registers for the NEBCALLBACK_SERVICE_CHECK_DATA callback and uses the NEBTYPE_SERVICECHECK_INITIATE event to preempt service check execution and insert its own load-balancing framework.

Rather than registering for a particular callback and writing event handler functions inside the module, Merlin registers for *all* callbacks and exports them all to an external daemon to handle. The Merlin daemon gets events from the Merlin module inside the Nagios daemon and either sends them to other Merlin daemons on other systems or to a database of your choosing. Events that come from other Merlin daemons can be injected back into a running Nagios daemon via the Merlin Event Broker module.

So there are two very powerful things that Merlin makes possible. The first is database synchronization (and a far better, more usable DB synchronization than NDOUtils, in my opinion), which in turn enables all manner of third-party UIs, add-ons, data export, and backup scenarios. The second, and more topical for our current purposes, is load balancing, clustering, and failover. With Merlin, it's possible to update the state of one Nagios daemon with events generated by another, remote Nagios daemon. In fact, the Merlin developers describe Merlin as a "cross-host event transportation layer," and this is an accurate description. Indeed, given the extent to which Nagios stores state data in memory, I find myself thinking of Merlin as a cluster shared memory system reminiscent of the SGI Onyx.

The possibilities here are pretty interesting. DNX-like arrangements may be created where "master" Nagios daemons send their checks to subservient Nagios daemons for processing, peering clusters may be set up where two or more Nagios daemons cooperate and update each other, or various permutations of the two may be achieved. A single Nagios daemon, for example, may be a peer to its peers, a master to its nodes, and a node to its masters, all at the same time. More complex relationships are also possible whereby, for example, a rollup server in Chicago might collect and display the state of remotely administered daemons in India, Brazil, and Australia.

I'm not able to find any solid information on the practical upper limits of a "Merlinized" Nagios cluster. The op5 folks seem to believe protocol overhead to be the primary limiting factor. An interesting question (at least to me) is whether DNX and Merlin could help each other scale. For example, one could imagine a DNX cluster with multiple master nodes sharing the reaper load via the Merlin protocol. Such an arrangement would minimize the overhead wrought by the Merlin protocol as well as share the reaper load, while at the same time helping to minimize the amount of configuration necessary, since DNX doesn't require full Nagios installations for load balancing the way Merlin does.

I've been really excited about both of these projects for a while now. They're both just the kind of great tools I envisioned when the Nagios Event Broker interface was first implemented, and I look forward to the day that add-ons like this are the norm.

I should get going—my wife just got her first Nagios pages about the Pepsi supply falling below the warning threshold.

Take it easy.

REFERENCES

[1] Carson Gaspar's invited talk at LISA '07: "Deploying Nagios in a Large Enterprise Environment": http://www.usenix.org/media/events/lisa07/tech/videos/gaspar.mp4.

[2] DNX Home: http://dnx.sourceforge.net/.

[3] About DNX: http://dnx.sourceforge.net/about.html.

[4] Nagios dev archives: http://archive.netbsd.se/?ml=nagios-devel&a=2009-09&t=11599144.

[5] Merlin: http://www.op5.org/community/projects/merlin.