DAVE JOSEPHSEN

iVoyeur: Ganglia on the brain

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

IT'S IRONIC, I THINK, THAT I'VE BEEN dissatisfied with my RRDTool data collection and display options for as long as I have. Ironic in the sense that RRDTool is such a category killer for what it does (storing and graphing time series data). It used to be novel, to see that distinctive-looking RRDTool graph popping up somewhere unexpected, but these days it's so ubiquitous that I'm surprised to see anything but. So if RRDTool is the ultimate solution for everything that happens between data coming in and graphs going out, the open source community has surely provided category killers for polling data and displaying those graphs, yes?

No. When it comes to polling and displaying that data, we're far from having a category-killer, in my opinion. It's not for lack of candidates, mind you. Freshmeat can provide you with myriad options written in all manner of languages (but mostly PHP) that do essentially the same thing—put an RRDTool graph on a Web page. It's a bit of a cop-out to say they all do "essentially" the same thing, in fact, since they all seem to make the same mistakes over and over again. Am I missing something? Hold on a second, let me make a couple of Google searches . . .

No. I don't think I am, but please (oh please) correct me if I'm wrong. Dear people who write RRDTool polling and graphing engines, here's what I really need from you:

1. Separate your data polling from your graphing engine. Better yet, just write one or the other. You'll have to account for what other people may or may not do that way. If you're writing one tool that does them both, I should be able to use it for polling while completely ignoring your graphing component and vice versa.

2. If you're writing a polling engine, it should speak more than just SNMP, it should provide sane defaults, and it should make it easy for me to discover what they are and change them. I should be able to tell it about my SNMP data sources as well as my monitoring system agents and metric-laden log files (ideally, these should be plug-ins to a larger, more generic polling engine). I should be able to *easily* set whatever custom RRDTool attributes I want per device per metric.

3. If you're writing a graphing engine, it should be able to read round robin databases (RRDs) from any location on the hard drive, including multiple locations at once. It should be able to combine data from any number of RRDs in any number of locations. It should have the capability to automatically detect metrics within the RRDs and provide predictable URLs that allow me to quickly see a graph of any single metric in any RRD in a directory it knows about. It should also allow me to save static templates for complicated stuff. I should be able to change graphing parameters of any existing or dynamically generated graph by modifying the URL.

I'll stop there; I hadn't planned on ranting. It's just that there are so many tools out there now that come so close. It's frustrating. A few years ago (well, more like 10 now, heh) my pockets were laden with gadgetry. I had a camera, an MP3 player, a cell phone, and a RIM pager/email device. This feels a lot like that did. Can't we make one pocket-sized gadget that does all of this stuff? Please?

Cacti [1] is beautiful, but just try to use something else for polling. Drraw [2] is awesome (and I use it today), but I have to statically define every graph before I'm able to see it. General-purpose monitoring systems like Nagios can be shoe-horned into polling-engine duty by bolting on any of 1000 different Perl scripts, but a dedicated task-specific polling engine could poll more often and with a lot less overhead. Enter Ganglia [3].

Ganglia

A few years ago I was doing some load-testing work and needed a tool that could poll and display the Big Four server metrics (CPU, memory, disk, network) as close to real-time as possible and without putting any load on the systems themselves. If that last sentence didn't give you pause, then you're in a different line of work than I am. I may as well have added, "and needed a never-ending supply of $20 bills." "Well, Dave," I can hear you say, "don't we all?" Anyway, I don't remember how, but I came across Ganglia, which at the time was a somewhat nascent project. The PHP-based front end didn't work, but the polling engine looked like it did, so I looked at the PHP and made a couple of simple fixes that, if I recall correctly, had something to do with how it was generating links (I probably caught their CVS server on a bad day). Once I got it working, the results were amazing.

It put zero overhead on the servers, collected every available statistic related to the Big Four plus a bunch of platform-specific statistics, and had a polling interval short enough that you could literally see data scrolling by in the RRDTool graphs when you set the page reload to 1 second. Had it been a more mature project at the time (it was in that stage where all it needed to guarantee its demise was my unconditional devotion), I would have immediately abandoned all of my metric collection efforts in its favor. A few months ago, I had the same sort of requirement, and went back to check on its progress. To my delight I found a healthy project with a goodly sized user base, forums, mailing lists, regular releases, the works. I'm currently in the process of abandoning all of my metric collection efforts in its favor. Well, at least the ones on general-purpose operating systems where Ganglia can be installed, sigh.

To be clear, it's not the category-killer I'm looking for. In fact, the frustration permeating my muse in the paragraphs above has a lot to do with what Ganglia isn't. It has, however, allowed me to consolidate a vast amount of my metric-gathering hodge-podge into a neat, superbly scalable, and easily configured set of RRDs, and for that I am grateful. Further, Ganglia never set out to be a general-purpose RRDTool front end. It was written with a very specific purpose in mind, so some of what it isn't is by design. For that it obviously cannot be blamed—it's a darn good design. Other things Ganglia isn't are probably pretty easy fixes that I'm sure will be patched away before long. For now, I'm going to focus on what it *is*, as well as the usual details about what my particular implementation looks like.

In the next issue, I plan to do a follow-up article to talk about Ganglia's plug-in architecture. Newer versions of Ganglia (like so many of my favorite tools) can be expanded with user-contributed modules, written in C or Python. If you read this column with any regularity, you know how I love my C-based shared-object plug-in architectures. It should come as no surprise, then, that I've written a couple of plug-ins for Ganglia in C, and can't wait to walk you through them in the next article (I know, lucky you!).

Ganglia is, as its sourceforge page says, a distributed monitoring system for high-performance computing systems such as clusters and grids, but that doesn't preclude its use for "normal" server systems. It consists primarily of two daemons and a Web front end written in PHP. The two daemons are called gmond and gmetad. Gmond may be thought of as the monitoring agent. It's installed on the systems you want to collect statistics from. Each gmond node will multicast its metrics to its peers at user-configurable intervals. Every node in a Ganglia cluster knows the status of the entire cluster.

Gmetad nodes collect statistics data from the machines running gmond. Since every gmond node knows the state of all its peers, gmetad only needs to talk to a single node in the cluster. Both daemons are written in C and make heavy use of the Apache Portable Runtime Library for portability. In my experience, they're lightweight in the extreme, and their use of multicast and node status consolidation does a great job of minimizing network chatter. Here are a few other things Ganglia gets right:

- Once a cluster is configured, new nodes are automatically detected. No configuration required.
- Automatic fail-over: A gmetad daemon can be pointed at every node in the cluster, and it will only attempt to contact the first configured node in each cluster. If that node dies, gmetad will try the next one.
- Automatic graphing: Ganglia's front end includes a graph.php program that will generate a graph of any metric for an entire cluster or an individual box. I can change its parameters by modifying the URL, so I can easily link to these graphs from anything that can take a URL. I can't change every RRDGraph parameter I'd like, but it's a great start.
- You can use as many gmetads as you'd like, and you can even have gmetads report up to other gmetads, so Ganglia Grids can scale beyond subnet boundaries and into the thousands of hosts.
- The Ganglia front end dynamically reorders all of its output based on the most utilized clusters and cluster nodes. It also does really nice things such as recoloring the backgrounds of the graphs based on user-provided thresholds. It's an interface I think Tufte [4] would generally approve of.

The Ganglia "Grid" is the sum total of every host you're monitoring with Ganglia. The Grid is composed of a number of "clusters," each of which is further composed of a number of individual servers, or nodes. Servers that make up a cluster are simply hosts that you've configured to use the same multicast address. The gmond.conf file allows you to name the cluster and provide other details such as latlog, and a description, but the configuration parameter that actually segments cluster nodes is the multicast address you assign them.

I make that point because it confused me initially. I was configuring nodes with different cluster names in the gmond.conf file, but some were appearing in the correct clusters and others weren't. It turned out that the Cisco switches in my environment weren't properly configured for multicast, so every node in the same subnet using the same multicast address was being grouped into a cluster by Ganglia's front end, and since gmetad only talks to a single node in each cluster for the state of the entire cluster, whichever node gmetad chooses to talk to gets to actually name the cluster.

At the moment, I'm content to run a separate Ganglia Grid per data center. Since we already have a central monitoring host per data center anyway, this works out well. The Ganglia Grid in my architecture encompasses several network subnets. You don't have to deal with multicast forwarding for this to work as long as nodes in the same cluster don't bypass a subnet boundary. Even if you do have two nodes in the same cluster that are in different network subnets, you can bypass the multicast networking headaches by configuring those hosts to update each other via unicast instead. This is easily done: sample syntax resides in the gmond.conf sample configuration file, which can be generated by running gmond with a -t switch. We use several openBSD systems in our environment, which don't support multicast by default. I've configured these hosts to use unicast Ganglia inter-cluster updates.

Gmetad uses unicast to speak to a single node in each cluster, so if you have a fairly straightforward network setup, the only firewall requirement is that the host running gmetad be able to reach port 8649 on any host running gmond. It's a symptom of an elegant design on the part of the Ganglia developers, I think, that there is so much automatic detection and so little overhead or impact inherent in this system. I can't remember the last time I picked up a new monitoring tool that installed and configured this easily.

Creating new graphs and adding new metrics are non-trivial undertakings, the price perhaps for getting so much functionality out of the box for so little work. To create new graphs from existing data you'll need to write a template in PHP, but the process is well documented [5] and should be easy enough for a decent sysadmin to follow. Adding new metrics can be accomplished in one of three ways. The first, and probably the easiest, is to use the included gmetric program, which will take input from any shell command and multicast it to all listing gmond nodes in the cluster in the same way gmond itself does. Any number of metrics can be added this way, via shell scripts running from cron, or hooks from the monitoring system of your choice.

The second and third options are to write a Ganglia plug-in in either Python or C. The process is pretty much the same for either type of plug-in, although the Python plug-ins require that gmond be compiled against the Python libs, and that requires you to have Python installed everywhere. For those of us who want to add new metrics while keeping the overhead as small as possible, there's the C option. But that's fodder for another article. Stay tuned.

Take it easy.

REFERENCES

[1] Cacti, "The Complete RRDTool-based Graphing Solution": http://www.cacti.net.

[2] Drraw: http://Web.taranis.org/drraw/.

[3] Ganglia: http://ganglia.sourceforge.net/.

[4] Edward Tufte, a dude who knows a thing or two about presenting information: http://www.edwardtufte.com/tufte/.

[5] http://sourceforge.net/apps/trac/ganglia/wiki/Custom_graphs.