iVoyeur

Changing the Game, Part 2

DAVE JOSEPHSEN

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

Near the end of his poem "The Talking Oak," Tennyson alludes to the oldest of the pagan oracles: Jupiter at Dodona. It was quite different from the oracles that followed it in that no temple, altar, or human contrivance was ever constructed there. It was merely an oak grove on an island in the Aegean Sea. The Selli tribal priests who lived there could decipher the word of Jupiter himself from the sound of the wind rustling the leaves of those sacred oak trees (some stories say wind-chimes were also employed).

I'd read Tennyson's poem in high school but, that being pre-Google, I never understood his reference to "that Thessalian growth" until I recently happened to read about the oracle at Dodona. The resolution of that long-forgotten enigma must have made an impression on my subconscious, because I subsequently dreamt that I visited that ancient oracular forest and heard the whisper of its long-dead deity. His message to me? "Your Web server is down."

I often tell people, when the subject of my occupation arises, that I'm a plumber. This saves me from having to hear about their brother-in-law "the computer guy" with whom I must have so much in common, but really I say this because I often feel like plumbing is what I do for a living. My chimerical jaunt to Dodona, however, has me wondering whether what I do for a living, especially in the context of systems monitoring, has more in common with divination than craftsmanship.

How often, after all, do the systems just tell us what's wrong with them? My troubleshooting technique is invariably a murky blend of experience, data analysis, intuition, and luck. Things will strike me as "wrong," sometimes without my being able to articulate why, and I'll eventually arrive at root cause by following up on the "wrongness." Maybe this works because the systems and their components are both more dependent on each other than we realize and related to each other in ways we don't expect. We can stumble across some wrongness that eventually leads to an answer because everything is ultimately tied together. A deep understanding of the relationship between our systems is what allows us to hear the trouble in the wind, and we're fascinated by the interplay between systems as a result.

Event correlation work like [1] and [2] are related endeavors, but what I'm really thinking about is work like that presented by Adam J. Oliner [3] from Stanford at LISA '10 wherein they attempt to identify and quantify the extent of "influence" between components of complex systems. Once identified and quantified, cause and effect can be inferred. If you've seen the presentation, you know what I'm talking about; that isn't plumbing, it's divination.

A less mathematically rigorous, more organic version of the same thing goes on between a sysadmin and his graphs. Influence is being explored, cause and effect inferred, answers divined. This, I think, is why we get so excited about new ways to visualize our data, especially if we're provided some means to include data that we weren't visualizing before. Properly understood, every new data point provides an opportunity to understand more deeply the interoperability of a complex system.

In my last article I introduced Graphite, a data storage and visualization system that does an amazing job of giving us the means to visualize data we wouldn't have considered previously, either because it was outside the domain of Ops (e.g., sales data), or because it didn't fit RRDTool's storage paradigm (e.g., temporally asynchronous or inconsistent data). In this article I'd like to explore some of the ways we might integrate Graphite with our existing tools.

Nagios Integration

You'll recall that Graphite's Carbon daemon listens on port 2003 for a string with the form "name value date" and automatically creates a whisker database within which to store the data. It's trivial to push data from any sort of classical monitoring system into Graphite using netcat as a client. Nagios, for example, will export "performance data" for host or service checks. Performance data is officially defined as anything that follows a pipe character ("|") in the output of a check result, but the performance data command can easily be modified to include the entire text of the check result.

For example, my Nagios "process_performance_data" command looks like this:

```
command_line    /usr/bin/printf "%b\n" \
      "$LASTSERVICECHECK$::$HOSTNAME$::$HOSTNOTES$::$SERVICEDESC$::\
      $SERVICEOUTPUT$::$SERVICEPERFDATA$" \
      >> /var/log/nagios/service-perfdata.out
```

This captures the entire output of the check result (including any performance data) and logs it to /var/log/nagios/service-perfdata.out. I then use logsurfer [4] to parse data out of the log and ship it to a Graphite sender. Here's the logsurfer definition to parse the output of the Nagios check_ping command:

```
'^([^:]+)::([^:]+)::([^:]+)::([^:]+)::PING [A-Z]+ - Packet loss = ([0-9]+)%, RTA
= ([^ ]+) ms::.*$' - - - 0 continue
      exec "/usr/local/bin/gclient.sh $4.$3.$5-loss $6 $2 graphitebox 2003"
```

gclient.sh is a simple shell script that uses netcat to push data to Graphite like so:

```
#!/bin/sh
#send data to carbon
SERVER=$4
PORT=$5
[ "${SERVER}" ] || SERVER=graphitebox
[ "${PORT}" ] || PORT=2003
echo "$1 $2 $3" | /usr/bin/nc -c $SERVER $PORT
```

Ganglia Integration

When I went to look for Graphite/Ganglia [5] integration I came up a bit bare. Current versions of Ganglia support using Graphite instead of RRDTool as a rendering and storage engine for GWeb, but this isn't really what I was looking for. In my mind, Ganglia fits the bill for Ops guys looking for an easy way to get metrics across all of their hosts, and Graphite is a more corporate-wide, general-purpose data visualization solution. I wanted an easy way to export my Ganglia metrics on the fly to Graphite while still keeping the former in its native format. The options were not what I'd consider viable [6].

It seemed to me that the shortest path would be to modify gmetad with an option to export the metrics directly to Graphite as it writes them to RRDTool locally, so I took a look at the gmetad source and submitted some patches to the mailing list a few hours later [7]. These still need work, but they're functional, and have been merged into Ganglia monitor-core [8], so you can get them by checking monitor-core out from git, or taking the patches from the mailing list in the usual way. I've added the following four configuration options to gmetad.conf, two of them being optional:

1. carbon_server should be set to the remote hostname or IP address of the graphite server.
2. graphite_prefix should be whatever you want to prefix your graphite path with (ganglia.<gridname> or something like that).
3. You can optionally specify a "carbon_port." This defaults to 2003 if you don't specify it.
4. You can optionally specify a "carbon_timeout" to timeout connection attempts if/when the Graphite server is down. This defaults to 500 ms if you don't specify it.

Statsd

Graphite makes it feasible for developers to instrument their code to send metrics to Graphite relating to the inner workings of their applications. If the foo() function gets called every time someone makes a $5 purchase on a Web site, it might be interesting to maintain a counter of the number of times foo() gets called. If bar() might cause performance problems, it might be interesting to keep a gauge for how long bar() takes to execute. A problem with these scenarios is what happens when the application gets distributed to hundreds of servers. Suddenly the foo() counters need to somehow be aggregated and the bar() gauges need to be averaged. The people at Etsy [9] wrote a very popular NodeJS-based metrics aggregation daemon called StatsD [10] to deal with this problem, and Jeff Buchbinder ported it to C [11].

StatsD libraries now exist for most popular programming languages (Perl, PHP, Python, Java, Ruby, Lua, etc.). These make it easy for developers to create and maintain counters and gauges (called "timers" in StatsD) in their application. The metrics are sent to the StatsD server, where they are aggregated on normal intervals and sent on to Graphite. StatsD has the additional advantage of using UDP, so the application servers can "fire and forget."

StatsD and developer interaction makes it possible to collect some interesting business metrics. Questions like "How many users did we register?" or "How many SKU4242s did we sell?" can now be easily visualized on the same graph with system metrics (e.g., network utilization) or other dev metrics (e.g., release cycles) imported from integration systems like Hudson.

Logster

The guys at Etsy also created a dedicated log-parser for Graphite called logster [12]. Logster is a forked and simplified version of ganglia-logtailer which uses logcheck [13] along with external definitions for parsing metrics out of log files and sending them to Carbon. It comes with parsers for the Apache Web server and is intended to be run every minute from cron.

Collectd Integration

Joe Miller wrote a Graphite plugin [14] for collectd [15] that bears mentioning. I haven't personally used collectd, so I can't really provide any details, but it's there.

Reverse Integration

I'm referring here to our ability to take graphs from Graphite and re-purpose them back into your monitoring tools. Once your metrics are in Graphite, the easiest way to get graphs back out is the excellent URL interface. Every feature and function available in the Graphite CLI or Web interface is exposed as a CGI attribute in the URL interface, making it possible to graph any combination of metrics, apply functions like "average" or "derive," and control the look and feel aspects of the graph such as image size, fonts, colors, etc., all with URL parameters.

These graphs can be referenced by any external dashboard or monitoring system that will take a URL. For example, I use the "action_url" attribute in the Nagios service description for this purpose. Taking our ping example from above, we can reverse-integrate our ping data back into the Nagios UI by adding an action_url attribute in the ping service description that looks like this:

```
http://graphitebox/render?from=-6h&target=dc2.linux.$HOSTNAME$.PING
-rta&width=1024&height=768&hideGrid=true
```

The $HOSTNAME$ parameter is a Nagios macro that will be replaced at runtime with the hostname of the host to which it refers. The rest should be self-explanatory.

I doubt I'll begin introducing myself as an oracle anytime soon. Not because it'll make me sound like a crazy person (I'm sure I sound like a crazy person anyway) but, rather, because if I think too hard about the metaphor I find it a bit depressing. I realize the Selli priests had it way better than we do, because their oracles would sometimes deliver them a propitious message. Ours, on the other hand, rarely have anything but bad news.

Take it easy.

References

[1] Paul Krizak, "Log Analysis and Event Correlation Using Variable Temporal Event Correlator (VTEC)": http://www.usenix.org/events/lisa10/tech/full_papers/Krizak.pdf.

[2] Ariel Rabkin and Randy Katz, "Chukwa: A System for Reliable Large-Scale Log Collection": http://www.usenix.org/events/lisa10/tech/full_papers/Rabkin.pdf.

[3] Adam Oliner and Alex Aiken, “Using Influence to Understand Complex Systems”: http://www.usenix.org/multimedia/lisa10oliner/.

[4] Logsurfer: http://www.crypt.gen.nz/logsurfer/.

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

[6] Vladimir Vuksan, “Integrating Ganglia with Graphite” (blog post): http://blog.vuksan.com/2010/09/29/integrating-graphite-with-ganglia/.

[7] Dave Josephsen, Graphite support for gmetad (mailing list thread): http://www.mail-archive.com/ganglia-general@lists.sourceforge.net/msg06964.html.

[8] Github Ganglia Monitor Core: https://github.com/ganglia/monitor-core/pull/1.

[9] Etsy: http://www.etsy.com/.

[10] StatsD: https://github.com/etsy/statsd.

[11] StatsD-c: https://github.com/jbuchbinder/statsd-c.

[12] Logster: https://github.com/etsy/logster#readme.

[13] Logcheck: http://logcheck.org/.

[14] Collectd-graphite: http://joemiller.me/2011/04/14/collectd-graphite-plugin/.

[15] Collectd: http://collectd.org/.