Ganglia and Nagios, Part 1: Monitor enterprise clusters with Ganglia

Install, configure, and extend open source Ganglia to effectively monitor a data center

Vallard Benincosa

March 04, 2009

This is the first article in a two-part series that looks at a hands-on approach to monitoring a data center using the open source tools Ganglia and Nagios. In Part 1, see how to install and configure Ganglia, the scalable, distributed monitoring system for high-performance clusters based on a hierarchical design. Also learn how to add more monitoring capability by writing Ganglia plug-ins and by enabling external-source spoofing.

View more content in this series

As data centers grow and administrative staffs shrink, the need for efficient monitoring tools for compute resources is more important than ever. The term monitor when applied to the data center can be confusing since it means different things depending on who is saying it and who is hearing it. For example:

- The person running applications on the cluster thinks: "When will my job run? When will it be done? And how is it performing compared to last time?"
- The operator in the network operations center (NOC) thinks: "When will we see a red light that means something needs to be fixed and a service call placed?"
- The person in the systems engineering group thinks: "How are our machines performing? Are all the services functioning correctly? What trends do we see and how can we better utilize our compute resources?"

Somewhere in this frenzy of definitions you are bound to find terabytes of code to monitor exactly what you want to monitor. And it doesn't stop there; there are also myriads of products and services. Fortunately though, many of the monitoring tools are open source -- in fact, some of the open source tools do a better job than some of the commercial applications that try to accomplish the same thing.

The most difficult part of using open source monitoring tools is implementing an install and configuration that works for your environment. The two major problems with using open source monitoring tools are
• There is no tool that will monitor everything you want the way you want it. Why? Because different users will define monitoring in different ways (as I mentioned earlier).
• Because of the first problem, there could be a great amount of customization required to get the tool working in your data center exactly how you want it. Why? Because every environment, no matter how standard, is unique.

By the way, these same two problems exist for the commercial monitoring tools also.

So, I’m going to talk about Ganglia and Nagios, two tools that monitor data centers. Both of them are used heavily in high performance computing (HPC) environments, but they have qualities that make them attractive to other environments as well (such as clouds, render farms, and hosting centers). Additionally, both have taken on different positions in the definition of monitoring. Ganglia is more concerned with gathering metrics and tracking them over time while Nagios has focused on being an alerting mechanism.

As the separate projects evolved, overlap developed. For example:

• Ganglia used to require an agent to run on every host to gather information from it, but now metrics can be obtained from just about anything through Ganglia’s spoofing mechanism.
• Nagios also used to only poll information from its target hosts, but now has plug-ins that run agents on target hosts.

While the tools have converged in some functional areas, there is still enough different about them so you gain from running both of them. Running them together can fill the gaps in each product:

• Ganglia doesn’t have a built-in notification system while Nagios excels at this.
• Nagios doesn’t seem to have scalable built-in agents on target hosts (people may argue on that point) while this was part of the intentional, original design of Ganglia.

There are also other open source projects that do things these two do and some are better in certain areas than others. Popular open source monitoring solutions include Cacti, Zenoss, Zabbix, Performance Copilot (PCP), and Clumon (plus I’m sure you’ve got a favorite I didn't mention). Many of these (including Ganglia and some Nagios plug-ins) make use of RRDTool or Tobi Oetiker’s MRTG (Multi Router Traffic Grapher) underneath to generate pretty graphs and store data.

With so many open source solutions for monitoring a data center, I’m often surprised to see how many scale-out computing centers develop their own solutions and ignore the work that has already been done by others.

In this two-part article, I will discuss Ganglia and Nagios since there is some anecdotal evidence that these are the most popular. And I think there is too little written on how to integrate them together even though it is a very prevalent practice. Especially in the large HPC labs and universities.

By the end of this series, you should be able to install Ganglia and make tie-ins with Nagios, as well as answer the monitoring questions that the different user groups will ask you. It will only be a start, but it should help you get your basics down and develop a total vision of your cluster.
In this article I will walk you through:

- Installing and configuring the basic Ganglia setup.
- How to use the Python modules to extend functionality with IPMI (the Intelligent Platform Management Interface).
- How to use Ganglia host spoofing to monitor IPMI.

Our goal -- to set up a baseline monitoring system of an HPC Linux® cluster in which these three different monitoring views above can be addressed at some level:

- The application person can see how full the queues are and see available nodes for running jobs.
- The NOC can be alerted of system failures or see a shiny red error light on the Nagios Web interface. They also get notified via email if nodes go down or temperatures get too high.
- The system engineer can graph data, report on cluster utilization and make decisions on future hardware acquisitions.

Introducing Ganglia

Ganglia is an open source monitoring project, designed to scale to thousands of nodes, that started at UC Berkeley. Each machine runs a daemon called gmond which collects and sends the metrics (like processor speed, memory usage, etc.) it gleans from the operating system to a specified host. The host which receives all the metrics can display them and can pass on a condensed form of them up a hierarchy. This hierarchical schema is what allows Ganglia to scale so well. gmond has very little overhead which makes it a great piece of code to run on every machine in the cluster without impacting user performance.

There are times when all of this data collection can impact node performance. "Jitter" in the network (as this is called) is when lots of little messages keep coming at the same time. We have found that by lockstepping the nodes' clocks, this can be avoided.

Installing Ganglia

There are many articles and resources on the Internet that will show you how to install Ganglia. We will revisit the one I wrote on the xCAT wiki. I will assume for the purposes of this article that the operating system is some flavor of Red Hat 5 Update 2 (although the steps won't be that much different for other enterprise Linux operating systems).

Prerequisites

Provided you have your yum repository set up, installing prereqs should be easy for the most part. Something like this:

```
yum -y install apr-devel apr-util check-devel cairo-devel pango-devel libxml2-devel
rpmbuild glib2-devel dbus-devel freetype-devel fontconfig-devel gcc-c++ expat-devel
python-devel libXrender-devel
```

(Note: Yum is really supposed to handle most of these dependencies, but in one of my tests I saw failures to compile that were fixed by adding all these packages.)
After getting these, you need another prerequisite that is not in the Red Hat repository. You can get it and build it like this as long as your machine is connected to the Internet:

```bash
rpmbuild --rebuild libconfuse-2.6-1.fc9.src.rpm
cd /usr/src/redhat/RPMS/x86_64/
rpm -ivh libconfuse-devel-2.6-1.x86_64.rpm libconfuse-2.6-1.x86_64.rpm
```

Remember, mirrors often change. If this doesn't work, then use a search engine to find the libconfuse-2.6.-1.fc9 source RPM.

**RRDTool**

RRDTool means: Round Robin Database Tool. It was created by Tobias Oetiker and provides an engine for many high performance monitoring tools. Ganglia is one of them, but Cacti and Zenoss are others.

To install Ganglia, we first need to have RRDTool running on our monitoring server. RRDTool provides two very cool functions that are leveraged by other programs:

- It stores data in a Round Robin Database. As the data captured gets older, the resolution becomes less refined. This keeps the footprint small and still useful in most cases.
- It can create graphs by using command-line arguments to generate them from the data it has captured.

To install RRDTool, run the following (tested on versions 1.3.4 and 1.3.6):

```bash
cd /tmp/
wget http://oss.oetiker.ch/rrdtool/pub/rrdtool.tar.gz
tar zxvf rrdtool.tar.gz
cd rrdtool-
./configure --prefix=/usr
make -j8
make install
which rrdtool
ldconfig  # make sure you have the new rrdtool libraries linked.
```

There are many ways you can use RRDTool as a standalone utility in your environment, but I won't go into them here.

**The main Ganglia install**

Now that you have all prerequisites, you can install Ganglia. First you need to get it. In this article we are using Ganglia 3.1.1. Download the ganglia-3.1.1.tar.gz file and place it in the /tmp directory of your monitoring server. Then do the following:

```bash
cd /tmp/
tar zxvf ganglia.gz
cd ganglia-3.1.1/
./configure --with-gmetad
make -j8
make install
```
You should exit without errors. If you see errors, then you may want to check for missing libraries.

**Configuring Ganglia**

Now that the basic installation is done, there are several configuration items you need to take care of to get it running. Do the following steps:

1. Command line file manipulations.
2. Modify `/etc/ganglia/gmond.conf`.
3. Take care of multi-homed machines.
4. Start it up on a management server.

**Step 1: Command line file manipulations**

As shown in the following:

```
cd /tmp/ganglia-3.1.1/  # you should already be in this directory
mkdir -p /var/www/html/ganglia/ # make sure you have apache installed
cp  -a web/* /var/www/html/ganglia/  # this is the web interface
cp  gmetad/gmetad.init /etc/rc.d/init.d/gmetad # startup script
cp  gmond/gmond.init /etc/rc.d/init.d/gmond
mkdir /etc/ganglia  # where config files go
 gmond -t | tee /etc/ganglia/gmond.conf  # generate initial gmond config
cp  gmetad/gmetad.conf /etc/ganglia/  # initial gmetad configuration
mkdir -p /var/lib/ganglia/rrds  # place where RRDTool graphs will be stored
chown nobody:nobody /var/lib/ganglia/rrds  # make sure RRDTool can write here.
chkconfig --add gmetad  # make sure gmetad starts up at boot time
chkconfig --add gmond # make sure gmond starts up at boot time
```

**Step 2: Modify `/etc/ganglia/gmond.conf`**

Now you can modify `/etc/ganglia/gmond.conf` to name your cluster. Suppose your cluster name is "matlock"; then you would change `name = "unspecified"` to `name = "matlock"`.

**Step 3: Take care of multi-homed machines**

In my cluster, eth0 is the public IP address of my system. However, the monitoring server talks to the nodes on the private cluster network through eth1. I need to make sure that the multicasting that Ganglia uses ties to eth1. This can be done by creating the file `/etc/sysconfig/network-scripts/route-eth1`. Add the contents `239.2.11.71 dev eth1`.

You can then restart the network and make sure routes shows this IP going through eth1 using `service network restart`. Note: You should put in 239.2.11.71 because that is the ganglia default multicast channel. Change it if you make the channel different or add more.

**Step 4: Start it up on a management server**

Now you can start it all up on the monitoring server:

```
service gmond start
service gmetad start
service httpd restart
```
Pull up a Web browser and point it to the management server at http://localhost/ganglia. You'll see that your management server is now being monitored. You'll also see several metrics being monitored and graphed. One of the most useful is that you can monitor the load on this machine. Here is what mine looks like:

![Figure 1. Monitoring load](image)

Nothing much happening here, the machine is just idling.

**Get Ganglia on the nodes**

Up to now, we've accomplished running Ganglia on the management server; now we have to care more about what the compute nodes all look like. It turns out that you can put Ganglia on the compute nodes by just copying a few files. This is something you can add to a post install script if you use Kickstart or something you can add to your other update tools.

The quick and dirty way to do it is like this: Create a file with all your host names. Suppose you have nodes `deathstar001-deathstar100`. Then you would have a file called `/tmp/mynodes` that looks like this:

```
deathstar001
deadstar002
...skip a few...
deadstar009
deadstar100
```

Now just run this:

```
# for i in `cat /tmp/mynodes`; do
scp /usr/sbin/gmond $i:/usr/sbin/gmond
ssh $i mkdir -p /etc/ganglia/
scp /etc/ganglia/gmond.conf $i:/etc/ganglia/
scp /etc/init.d/gmond $i:/etc/init.d/
scp /usr/lib64/libganglia-3.1.1.so.0 $i:/usr/lib64/
scp /lib64/libexpat.so.0 $i:/lib64/
scp /usr/lib64/libconfuse.so.0 $i:/usr/lib64/
scp /usr/lib64/libapr-1.so.0 $i:/usr/lib64/
scp -r /usr/lib64/ganglia $i:/usr/lib64/
ssh $i service gmond start
done
```

You can restart `gmetad`, refresh your Web browser, and you should see your nodes now showing up in the list.

Some possible issues you might encounter:
• You may need to explicitly set the static route as in the earlier step 3 on the nodes as well.
• You may have firewalls blocking the ports. gmond runs on port 8649. If gmond is running on a machine you should be able to run the command `telnet localhost 8649`. And see a bunch of XML output scroll down your screen.

**Observing Ganglia**

Many system engineers have a hard time understanding their own workload or job behavior. They may have custom code or haven't done research to see what their commercial products run. Ganglia can help profile applications.

We'll use Ganglia to examine the attributes of running the Linpack benchmark. Figure 2 shows a time span where I launched three different Linpack jobs.

**Figure 2. Watching over Linpack**

As you can see from this graph, when the job starts there is some activity on the network when the job launches. What is interesting, however, is that towards the end of the job, the network traffic increases quite a bit. If you knew nothing about Linpack, you could at least say this: Network traffic increases at the end of the job.

Figure 3 and Figure 4 show CPU and memory utilization respectively. From here you can see that we are pushing the limits of the processor and that our memory utilization is pretty high too.
These graphs give us great insight to the application we're running: We're using lots of CPU and memory and creating more network traffic towards the end of the running job. There are still a lot of other attributes about this job that we don't know, but this gives us a great start.
Knowing these things can help make better purchasing decisions in the future when it comes to buying more hardware. Of course, no one buys hardware just to run Linpack ... right?

**Extending capability**

The basic Ganglia install has given us a lot of cool information. Using Ganglia's plug-ins gives us two ways to add more capability:

- Through the addition of in-band plug-ins.
- Through the addition of out-of-band spoofing from some other source.

The first method has been the common practice in Ganglia for a while. The second method is a more recent development and overlaps with Nagios in terms of functionality. Let's explore the two methods briefly with a practical example.

**In-band plug-ins**

In-band plug-ins can happen in two ways.

- Use a cron-job method and call Ganglia's `gmetric` command to input data.
- Use the new Python module plug-ins and script it.

The first method was the common way we did it in the past and I'll more about this in the next section on out-of-band plug-ins. The problem with it is that it wasn't as clean to do. Ganglia 3.1.x added Python and C module plug-ins to make it seem more natural to extend Ganglia. Right now, I'm going to show you the second method.

First, enable Python plug-ins with Ganglia. Do the following:

1. Edit the `/etc/ganglia/gmond.conf` file.

If you open it up, then you'll notice about a quarter of the way down there is a section called `modules` that looks something like this:

```plaintext
modules {
  module {
    name = "core_metrics"
  }
  ...
}
```

We're going to add another module to the `modules` section. The one you should stick in is this:

```plaintext
module {
  name = "python_module"
  path = "modpython.so"
  params = "/usr/lib64/ganglia/python_modules/"
}
```

On my `gmond.conf` I added the previous code stanza at line 90. This allows Ganglia to use the Python modules. Also, a few lines below that after the statement `include('/etc/ganglia/conf.d/`
2. Make some directories.

Like so:

```bash
mkdir /etc/ganglia/conf.d
mkdir /usr/lib64/ganglia/python_modules
```

3. Repeat 1 and 2 on all your nodes.

To do that,

- Copy the new gmond.conf to each node to be monitored.
- Create the two directories as in step 2 on each node to be monitored so that they too can use the Python extensions.

Now that the nodes are set up to run Python modules, let's create a new one. In this example we're going to add a plug-in that uses the Linux IPMI drivers. If you are not familiar with IPMI and you work with modern Intel and AMD machines then please learn about it (in Related topics).

We are going to use the open source IPMItool to communicate with the IPMI device on the local machine. There are several other choices like OpenIPMI or freeipmi. This is just an example, so if you prefer to use another one, go right on ahead.

Before starting work on Ganglia, make sure that IPMItool works on your machine. Run the command `ipmitool -c sdr type temperature | sed 's/ /_/g'`; if that command doesn't work, try loading the IPMI device drivers and run it again:

```bash
modprobe ipmi_msghandler
modprobe ipmi_si
modprobe ipmi_devintf
```

After running the `ipmitool` command my output shows

```
Ambient_Temp,20,degrees_C,ok
CPU_1_Temp,20,degrees_C,ok
CPU_2_Temp,21,degrees_C,ok
```

So in my Ganglia plug-in, I'm just going to monitor ambient temperature. I've created a very poorly written plug-in called ambientTemp.py that uses IPMI based on a plug-in found on the Ganglia wiki that does this:

**Listing 1. The poorly written Python plug-in ambientTemp.py**

```python
import os
def temp_handler(name):
    # our commands we're going to execute
    sdrfile = "~/tmp/sdr.dump"
    ipmitool = "/usr/bin/ipmitool"
    # Before you run this Load the IPMI drivers:
    # modprobe ipmi_msghandler
```

Ganglia and Nagios, Part 1: Monitor enterprise clusters with Ganglia
# modprobe ipmi_si
# modprobe ipmi_devintf
# you'll also need to change permissions of /dev/ipmi0 for nobody
# chown nobody:nobody /dev/ipmi0
# put the above in /etc/rc.d/rc.local

foo = os.path.exists(sdrfile)
if os.path.exists(sdrfile) != True:
    os.system(ipmitool + ' sdr dump ' + sdrfile)

if os.path.exists(sdrfile):
    ipmicmd = ipmitool + ' -S ' + sdrfile + ' -c sdr'
else:
    print "file does not exist... oops!"
    ipmicmd = ipmitool + ' -c sdr'

if os.path.exists(sdrfile):
    cmd = ipmicmd + ' type temperature | sed 's/ /_/g' '
    cmd = cmd + ' | awk -F, '/Ambient/ {print $2}' "
    #print cmd
    entries = os.popen(cmd)
    for l in entries:
        line = l.split()
        #print line
        return int(line[0])

def metric_init(params):
    global descriptors

    temp = {'name': 'Ambient Temp',
            'call_back': temp_handler,
            'time_max': 90,
            'value_type': 'uint',
            'units': 'C',
            'slope': 'both',
            'format': '%u',
            'description': 'Ambient Temperature of host through IPMI',
            'groups': 'IPMI In Band'}

    descriptors = [temp]

    return descriptors

def metric_cleanup():
    '''Clean up the metric module.'''
    pass

#This code is for debugging and unit testing
if __name__ == '__main__':
    metric_init(None)
    for d in descriptors:
        v = d['call_back'](d['name'])
        print 'value for %s is %u' % (d['name'], v)

Copy Listing 1 and place it into /usr/lib64/ganglia/python_modules/ambientTemp.py. Do this for all nodes in the cluster.

Now that we've added the script to all the nodes in the cluster, tell Ganglia how to execute the script. Create a new file called /etc/ganglia/conf.d/ambientTemp.pyconf The contents are as follows:
Listing 2. Ambient.Temp.pyconf

```python
modules {
    module {
        name = "Ambient Temp"
        language = "python"
    }
}

collection_group {
    collect_every = 10
    time_threshold = 50
    metric {
        name = "Ambient Temp"
        title = "Ambient Temperature"
        value_threshold = 70
    }
}
```

Save Listing 2 on all nodes.

The last thing that needs to be done before restarting `gmond` is to change the permissions of the IPMI device so that `nobody` can perform operations to it. This will make your IPMI interface extremely vulnerable to malicious people!

This is only an example: `chown nobody:nobody /dev/ipmi0`.

Now restart `gmond` everywhere. If you get this running then you should be able to refresh your Web browser and see something like the following:

**Figure 5. IPMI in-band metrics**

The nice thing about in-band metrics is they allow you to run programs on the hosts and feed information up the chain through the same collecting mechanism other metrics use. The drawback to this approach, especially for IPMI, is that there is considerable configuration required on the hosts to make it work.

Notice that we had to make sure the script was written in Python, the configuration file was there, and that the `gmond.conf` was set correctly. We only did one metric! Just think of all you need to do to write other metrics! Doing this on every host for every metric can get tiresome. IPMI is an out-of-band tool so there's got to be a better way right? Yes there is.
Out-of-band plug-ins (host spoofing)

Host spoofing is just the tool we need. Here we use the powerful gmetric and tell it which hosts we’re running on -- gmetric is a command-line tool to insert information into Ganglia. In this way you can monitor anything you want.

The best part about gmetric? There are tons of scripts already written.

As a learning experience, I'm going to show you how to reinvent how to run ipmitool to remotely access machines:

1. Make sure ipmitool works on its own out of band.

I have set the BMC (the chip on the target machine) so that I can run IPMI commands on it. For example: My monitoring hosts name is redhouse. From redhouse I want to monitor all other nodes in the cluster. Redhouse is where gmetad runs and where I point my Web browser to access all of the Ganglia information.

One of the nodes in my cluster has the host name x01. I set the BMC of x01 to have an IP address that resolves to the host x01-bmc. Here I try to access that host remotely:

```
# ipmitool -I lanplus -H x01-bmc -U USERID -P PASSW0RD sdr dump \ /tmp/x01.sdr
Dumping Sensor Data Repository to '/tmp/x01.sdr'
# ipmitool -I lanplus -H x01-bmc -U USERID -P PASSW0RD -S /tmp/x01.sdr \ sdr type
Temperature
Ambient Temp | 32h | ok | 12.1 | 20 degrees C
CPU 1 Temp    | 98h | ok |  3.1 | 20 degrees C
CPU 2 Temp    | 99h | ok |  3.2 | 21 degrees C
```

That looks good. Now let's put it in a script to feed to gmetric.

2. Make a script that uses ipmitool to feed into gmetric.

We created the following script /usr/local/bin/ipmi-ganglia.pl and put it on the monitoring server:

```
#!/usr/bin/perl
# vallard@us.ibm.com
use strict;  # to keep things clean... er cleaner
use Socket;  # to resolve host names into IP addresses

# code to clean up after forks
use POSIX "sys_wait_h";
# nodeFile: is just a plain text file with a list of nodes:
# e.g:
# node01
# node02
# ...
# nodeXX
my $nodeFile = "/usr/local/bin/nodes";
# gmetric binary
my $gmetric = "/usr/bin/gmetric";
#ipmitool binary
my $ipmi = "/usr/bin/ipmitool";
# userid for BMCS
my $u = "xcat";
# password for BMCS
my $p = "f00bar";
```
Aside from all the parsing, this script just runs the `ipmitool` command and grabs temperatures. It then puts those values into Ganglia with the `gmetric` command for each of the metrics.

3. Run the script as a cron job.
Run `crontab -e`. I added the following entry to run every 30 minutes:

```bash
30 * * * * /usr/local/bin/ipmi-ganglia.sh
```

You may want to make it happen more often or less.

4. Open Ganglia and look at the results.

Opening up the Ganglia Web browser and looking at the graphs of one of the nodes, you can see that nodes were spoofed and were updated in each node's entry:

**Figure 6. The no_group metrics**

One of the drawbacks to spoofing is that the category goes in the no_group metrics group. `gmetric` doesn't appear to have a way to change the groupings in a nice way like in the in-band version.

**What's next**

This article gives a broad overview of what you can get done using Ganglia and Nagios as open source monitoring software, both individually and in tandem. You took an installation/configuration tour of Ganglia, then saw how Ganglia can be useful in understanding application characteristics. Finally, you saw how to extend Ganglia using an in-band script and how to use out-of-band scripts with host spoofing.

This is a good start. But this article has only answered the monitoring question the system engineer posed. We now can now view systemwide performance and see how the machines are being utilized. We can tell if machines are idle all the time, or if they're running at 60 percent capacity. We can now even tell which machines run the hottest and coldest, and see if their rack placement could be better.

**Part 2** in this two-part series explores setting up Nagios and integrating it with Ganglia, including:

- Installing and configuring a basic Nagios setup for alerts
- Monitoring switches and other infrastructure
- Tying Nagios into Ganglia for alerts

As a bonus, the second part shows how to extend the entire monitoring system to monitor running jobs and other infrastructure. By doing these additional items, we'll be able to answer the other monitoring questions that different groups ask.
Related topics

- Grab Ganglia 3.1.1.
- RDDTool (Round Robin Database Tool) was created by Tobias Oetiker (bio); it provides an engine for many high performance monitoring tools.
- Gmetric is a command-line tool to insert information into Ganglia that sports lots of scripts to help with host spoofing.
- Some other monitoring tools:
  - Cacti
  - Zenoss
  - Zabbix
  - Clumon
- IPMItool is a utility for managing and configuring devices that support the Intelligent Platform Management Interface (IPMI) version 1.5 and version 2.0 specifications.
- The Multi Router Traffic Grapher monitors SNMP network devices and draws pretty pictures showing how much traffic has passed through each interface.
- In the developerWorks Linux hub, find more resources for Linux developers (including developers who are new to Linux).
- See all Linux articles and tutorials on developerWorks.

© Copyright IBM Corporation 2009
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)