Science & Technology Facilities Council

Technical Report

Computational cluster monitoring using Ganglia

Wadud Miah

January 2010

RAL-TR-2010-003

© **Science and Technology Facilities Council**

Enquires about copyright, reproduction and requests for additional copies of this report should be addressed to:

Library and Information Services
SFTC Rutherford Appleton Laboratory
Harwell Science and Innovation Campus
Didcot
OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44(0)1235 446403
Email: library@rl.ac.uk

The STFC ePublication archive (epubs), recording the scientific output of the Chilbolton, Daresbury, and Rutherford Appleton Laboratories is available online at:
http://epubs.cclrc.ac.uk/

ISSN 1358-6254

Neither the Council nor the Laboratory accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigation

RAL-TR-2010-003

Computational cluster monitoring using Ganglia

Wadud Miah[1]

This report will describe how Ganglia is utilised within the Scientific Computing Technology (SCT) group at the eScience department of STFC. Ganglia is extensively used to monitor usage of Linux computational clusters, e.g. CPU, memory usage and network traffic. In addition to the standard metrics monitored by Ganglia, custom metrics are also stored and visualised. This allows the group to monitor the status of the machine room and LSF job status. This report does not cover details of the technology, and for readers who require this, they are referred to [4, 3].

---

[1] wadud.miah@stfc.ac.uk, Rutherford Appleton Laboratory

Scientific Computing Technology Group
eScience Department, Rutherford Appleton Laboratory
Oxfordshire, OX11 0QX

January 2010

Contents

1 Ganglia structure

Ganglia is a highly scalable distributed monitoring system which provides near real time status of computational clusters or a group of machines. It runs a PHP web front end which collates and presents the data from client machines, and typically runs on either a monitoring machine or head node. Ganglia is reported to run on numerous architectures and operating systems and has scaled to clusters with 2000 nodes [1]. Ganglia is written in the C programming language and is an open source project, which is particularly advantageous as custom metrics developed in C can use the available API. An example usage of the Ganglia gmetric API can be found in Section 6.1 and instructions on how to build the program in Section 6.2 of the Appendix.

The two daemons that form part of Ganglia are:

- The monitoring daemon - gmond - runs on client machines and multicasts resource metrics. See [2] for a description the multicast protocol;

- The meta daemon - gmetad - runs on the same host as the web server and collates data from the clients. Data is stored in round robin data (RRD) format using the RRD tool [5].

The monitoring daemon, gmond, is an extremely lightweight process and presents the data in XML format, multicasting packets in XDR format to the meta daemon, gmetad. A typical structure is shown in Figure 1.

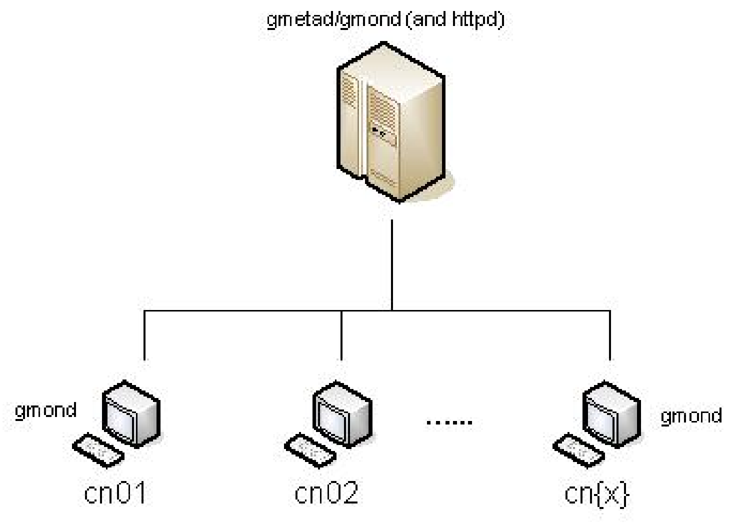

Figure 1: Basic Ganglia structure

At SCT, there are a number of clusters or host groups, namely distinct multicast groups. A multicast group in Ganglia represents a logical group, and this loosely follows Nagios host groups. A ‘master’ gmetad exists for the group which points to all Ganglia groups and this configuration is shown in Figure 2.

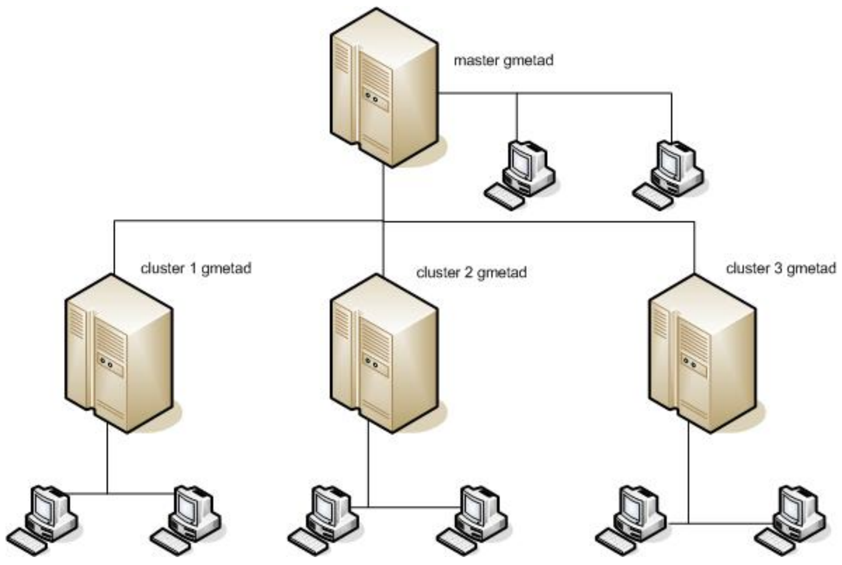

Figure 2: Basic Ganglia structure

The 'master' gmetad polls the cluster gmetad to obtain aggregate information using the following directive in `/etc/gmond.conf`:

```
data_source "NGS cluster" ngs-head-node.rl.ac.uk:8651
```

The TCP port 8651 is where the cluster gmetad listens on. Multiple data sources can be specified, as well as the multicast group the host itself belongs to, namely:

```
data_source "SCT services" localhost:8649
```

The above directive can point to multiple sources for redundancy.

2 Monitoring custom metrics

In addition to the standard metrics provided by Ganglia, numerous custom metrics are stored for the purpose of trend analysis and visualisation using the `gmetric` command. Examples include:

- Temperature sensor values from server racks in a machine room;
- Power consumption from Power Distribution Units (PDUs);
- Load Sharing Facility (LSF) statistics.

The temperature obtained from racks is also monitored in Nagios, where alerts are sent if the values exceed a specified threshold, and at the same time is fed into Ganglia. The check is executed via a Nagios plugin using the shell code:

```
VAL=`$SNMP_GET -v1 -m IT-WATCHDOGS-MIB -Cf -c public $IP_ADDR \
     $OID_NAME $OID_AVAIL $OID_VAR 2>/dev/null | awk '{ print $NF }'`

gmetric --name=$GANGLIA_NAME --type=int16 --units=$UNIT \
        --spoof=172.16.140.157:sensor.rl.ac.uk --value=$VAL > /dev/null
```

This method avoids the duplicate poll for the same data. The metric `$GANGLIA_NAME` appears in the PHP front end under the spoofed host sensor.rl.ac.uk. A spoof host in Ganglia is used to group custom metrics fed via `gmetric`; if the host is not spoofed, the metric will appear under the host that issued the `gmetric` command. Additional metrics of the same type can also be sent as this spoofed host, and is a convenient way to group such metrics. However, the spoofed host will appear as dead and reappear as alive if a 'heartbeat' is not sent to the gmetad. This is achieved using a daemon which is spawned at boot time:

```
while true
do
  /usr/bin/gmetric --heartbeat --spoof=172.16.140.157:sensor.rl.ac.uk

  sleep 30
done
```

Since it requires a delay of 30 seconds, this can not be executed by crond (in Linux systems) as it executes commands in units of integer minutes. A delay of greater than 30 seconds causes the host to momentarily disappear and then reappear.

3 System configuration

The gmetad writes metric data to an RRD file stored at a path specified by `rrd_rootdir` in `/etc/gmetad.conf`. Metrics are being received by gmetad from every host in the mutlicast group typically 30 seconds for every metric and is written to disk. This results in extensive hard disk activity which can be circumvented using the RAM file system[1]. To configure this:

1. Assign `rrd_rootdir` to, say, `/var/lib/ganglia/ramrrds`;

2. In the initialisation script `/etc/init.d/gmetad` include the shell code:

```
start)
  TMPFS=`/bin/awk '{ if ( $1 ~ /^tmpfs$/ ) print $2 }' /proc/mounts`

  [ "x${TMPFS}" != "x${RAMRRD}" ] && { # is RAMRRD mounted?
    mount -t tmpfs tmpfs ${RAMRRD}
    /usr/bin/rsync -a ${RRD}/ ${RAMRRD}/
    chown nobody:nobody ${RAMRRD}
  }
# more code
```

[1]this stores the file in RAM

```
stop)
  TMPFS=`/bin/awk '{ if ( $1 ~ /^tmpfs$/ ) print $2 }' /proc/mounts`

  [ "x${TMPFS}" = "x${RAMRRD}" ] && {
    /usr/bin/rsync -a ${RAMRRD}/ ${RRD}/
    umount ${RAMRRD}
  }
```

When the daemon is stopped, the data from RAM file system is copied to disk; conversely, when it is started, it is copied from disk to RAM file system.

3. To ensure data is saved in the event of a system crash, regularly copy the data from RAM file system to disk via crond:

```
*/15 * * * * root /usr/bin/rsync -a /var/lib/ganglia/ramrrds \
                  /var/lib/ganglia/rrd
```

To allow client hosts to receive multicast packets, the IP tables firewall rule must be configured to allow it. To achieve this ensure the following rules are included:

```
-A INPUT -d <MCAST_IP> -p igmp                      -j ACCEPT
-A INPUT -d <MCAST_IP> -p udp  -m udp --dport 8649 -j ACCEPT
```

where the <MCAST_IP> is a multicast IP address for the group as specified by RFC1112, e.g. 239.2.11.71. Even though the above rules are specified for the INPUT chain, it must contain the -d option and not -s as would be normally. This is an exception for multicast packets.

If the gmetad is further down the Ganglia hierarchy, e.g. cluster 1 in Figure 2, the gmetad host IP tables must be configured to allow the 'master' gmetad to poll the gmetad port:

```
-A INPUT -s <MASTER_IP> -p tcp -m tcp --dport 8651 -j ACCEPT
```

where <MASTER_IP> is the IP address of the master gmetad (see Figure 2). In addition to configuring local firewall rules, some managed network switches may also need configuring to allow IGMP (Internet Group Management Protocol) packets. If they are not, the gmetad will not be able to receive multicast packets from gmond clients.

A PHP configuration that requires a higher memory value, particularly for large clusters, is `memory_limit` in `/etc/php.ini`. In a cluster of approximately 280 hosts, the value of `8M` is sufficient. The Ganglia PHP configurations in `ganglia/conf.php`[2] require the following configuration:

```
$gmetad_root = "/var/lib/ganglia";
# ensure this points to the RAM file system
$rrds = "$gmetad_root/ramrrds";
# location of the rrdtool command
define("RRDTOOL", "/usr/bin/rrdtool");
```

[2]relative to DocumentRoot

4 Future plans for Ganglia

As the Ganglia monitoring system is also a service and falls under the service life cycle, the following are future plans for the Ganglia installation at SCT:

1. Monitoring statistics provided by the intelligent platform management interface (IPMI). This has been recently configured due to the instability/complexities in earlier versions of IPMI;
2. Monitor power consumption by the CRAC units;
3. Monitor power consumption by extractor fans in a closed hot aisle configuration. Currently, the SCT machine room has a cold and hot aisle set up where the hot air is dispersed into the atmosphere;
4. Upgrading Ganglia to version 3.1.2 from 3.0.5, although the main feature of monitoring each CPU core has not been very useful for SCT;
5. Using temperature sensor data in Ganglia to validate computational fluid dynamics (CFD) models.

5 Summary

The Ganglia monitoring system has allowed the SCT group to get a more detailed picture of cluster utilisation. Historical data can be viewed for the past hour, day, week, month and year which allows the group to monitor peak times. One factor that is not represented by the default installation of Ganglia is the number of slots reserved by jobs, and thus giving the impression that not all CPU resources are being fully utilised. However, this important metric is being monitored using the gmetric API using the program in Section 6.1. Ganglia data on cluster utilisation also indicates whether more hardware is required to satisfy the requirements of the scientific community, or whether the infrastructure is being efficiently utilised. Historical data has also allowed the group to tweak LSF queue parameters to ensure higher throughput of jobs.

One of the aims of the SCT group is pursuing the ‘green’ agenda of efficiency: reducing power consumption whilst increasing the work completed by computational nodes. To achieve this, the power consumption of the hosts/clusters is being monitored and is visualised to ensure the aims are being achieved. The power consumption data has also proved useful in comparing with the manufacturer’s data and experience at SCT has shown the latter to be generally conservative. This type of monitoring will be further extended to monitor power used to cool the hosts/clusters.

As the power to cool machines can be as high as the power to operate them, reducing this is another major aim of SCT. The computer room air conditioner (CRAC) units are being continuously tweaked, e.g. increasing the temperature of the chilled water system of the CRAC units, and subsequently monitoring the temperature in Ganglia to ensure this has not had any negative effects. Thus far, there has been no negative repercussions of this endeavour, thanks to the available data in Ganglia. When new racks of nodes are installed, the subsequent environmental effects need to be studied and analysed to ensure the conditions are not adversely effecting other racks, and this is achieved again using historical data in Ganglia. Thus it is evident that historical resource and environmental data is invaluable in managing clusters and the machine room.

6 Appendix

6.1 Example use of the gmetric API in C

The C program below queries LSF queues passed as a command line argument. It obtains the number of jobs slots (NJOBS) held in the queue, number of job slots used by pending jobs (PEND), number of job slots that are reserved (RSV) and number job slots used by running jobs.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#include <ganglia.h>
#include <cmdline.h>
#include <lsf/lsf.h>
#include <lsf/lsbatch.h>

Ganglia_pool global_context;
Ganglia_gmetric gmetric;
Ganglia_gmond_config gmond_config;
Ganglia_udp_send_channels send_channels;

/* The commandline options */
struct gengetopt_args_info args_info;

int main( int argc, char *argv[] ) {
  int rval, args_count = 6, i;
  struct queueInfoEnt *qInfo;
  int numQueues = argc - 1, options = 0;
  char *host = NULL, *user = NULL;
  char *args[6] = { "lsf_queue", "--name=scarf-NJOBS", "--type=uint16",
                    "--units=jobs",
                    "--spoof=172.16.140.199:lsf-queues.rl.ac.uk",
                    "--value=15" };
  char value_name[50], value[30];

  /* create the global context */
  global_context = Ganglia_pool_create( NULL );
  if ( !global_context ) {
    fprintf( stderr, "Unable to create global context. Exiting.\n" );
    exit( 1 );
  }

  /* parse the configuration file */
  gmond_config = Ganglia_gmond_config_create( "/etc/gmond.conf", 1 );
```

```
/* build the udp send channels */
send_channels = Ganglia_udp_send_channels_create( global_context,
                                                  gmond_config );
if ( ! send_channels ) {
  fprintf( stderr, "Unable to create ganglia send channels. Exiting.\n" );
  exit( 1 );
}

/* create the message */
gmetric = Ganglia_gmetric_create( global_context );
if ( ! gmetric ) {
  fprintf( stderr, "Unable to allocate gmetric structure. Exiting.\n" );
  exit( 1 );
}

/* initialise lsb */
if ( lsb_init( argv[0] ) < 0 ) {
  lsb_perror( "lsb_init() failed" );
  exit( 3 );
}

*argv++;   /* get info of queues provided at the command line */
qInfo = lsb_queueinfo( argv, &numQueues, host, user, options );
if ( NULL == qInfo ) {
  lsb_perror( "lsb_queueinfo() failed" );
  exit( 3 );
}

for ( i = 0; i < numQueues; i++ ) {
  sprintf( value_name, "--name=%s-NJOBS", qInfo[i].queue );
  sprintf( value, "--value=%d", qInfo[i].numJobs );
  args[1]=&value_name[0];
  args[5]=&value[0];

  /* process the commandline options */
  if ( cmdline_parser( args_count, args, &args_info ) != 0 ) {
    fprintf( stderr, "could not parse command line\n" );
    exit( 1 );
  }

  rval = Ganglia_gmetric_set( gmetric, args_info.name_arg,
                              args_info.value_arg,
                              args_info.type_arg, args_info.units_arg,
                              !strcmp( args_info.slope_arg, "zero" ) ? 0: 3,
                              args_info.tmax_arg, args_info.dmax_arg );
  /* send the message */
  rval = Ganglia_gmetric_send_spoof( gmetric, send_channels,
                                     args_info.spoof_arg,
                                     args_info.heartbeat_given );
```

```
sprintf( value_name, "--name=%s-PEND", qInfo[i].queue );
sprintf( value, "--value=%d", qInfo[i].numPEND );
args[1]=&value_name[0];
args[5]=&value[0];

/* process the commandline options */
if ( cmdline_parser( args_count, args, &args_info ) != 0 ) {
  fprintf( stderr, "could not parse command line\n" );
  exit( 1 );
}

rval = Ganglia_gmetric_set( gmetric, args_info.name_arg,
                            args_info.value_arg,
                            args_info.type_arg, args_info.units_arg,
                            !strcmp( args_info.slope_arg, "zero" ) ? 0: 3,
                            args_info.tmax_arg, args_info.dmax_arg );

/* send the message */
rval = Ganglia_gmetric_send_spoof( gmetric, send_channels,
                                   args_info.spoof_arg,
                                   args_info.heartbeat_given );

sprintf( value_name, "--name=%s-RUN", qInfo[i].queue );
sprintf( value, "--value=%d", qInfo[i].numRUN );
args[1]=&value_name[0];
args[5]=&value[0];

/* process the commandline options */
if ( cmdline_parser( args_count, args, &args_info ) != 0 ) {
  fprintf( stderr, "could not parse command line\n" );
  exit( 1 );
}

rval = Ganglia_gmetric_set( gmetric, args_info.name_arg,
                            args_info.value_arg,
                            args_info.type_arg, args_info.units_arg,
                            !strcmp( args_info.slope_arg, "zero" ) ? 0: 3,
                            args_info.tmax_arg, args_info.dmax_arg );

/* send the message */
rval = Ganglia_gmetric_send_spoof( gmetric, send_channels,
                                   args_info.spoof_arg,
                                   args_info.heartbeat_given );

sprintf( value_name, "--name=%s-RSV", qInfo[i].queue );
sprintf( value, "--value=%d", qInfo[i].numRESERVE );
args[1]=&value_name[0];
args[5]=&value[0];
```

```
    /* process the commandline options */
    if ( cmdline_parser( args_count, args, &args_info ) != 0 ) {
      fprintf( stderr, "could not parse command line\n" );
      exit( 1 );
    }

    rval = Ganglia_gmetric_set( gmetric, args_info.name_arg,
                                args_info.value_arg,
                                args_info.type_arg, args_info.units_arg,
                                !strcmp( args_info.slope_arg, "zero" ) ? 0: 3,
                                args_info.tmax_arg, args_info.dmax_arg );
    /* send the message */
    rval = Ganglia_gmetric_send_spoof( gmetric, send_channels,
                                       args_info.spoof_arg,
                                       args_info.heartbeat_given );

  } /* end for */

  /* cleanup */
  Ganglia_gmetric_destroy( gmetric );
  /* not really necessary but for symmetry */
  Ganglia_pool_destroy( global_context );

  exit( 0 );
}
```

6.2 Building gmetric C programs

To compile the C program, change directory to `ganglia-3.0.5/gmetric`[3]:

```
gcc -I/opt/lsf/6.2/include -I. -I..  -I../lib -I../gmond -I../libmetrics -O2 \
    -o lsf_queue.o -c lsf_queue.c
```

To build the program:

```
gcc -I../lib -I../gmond -I../libmetrics -g -O2 -o lsf_queue \
    lsf_queue.o cmdline.o ../lib/.libs/libganglia.a \
    /opt/lsf/6.2/linux2.6-glibc2.3-x86_64/lib/libbat.a \
    /opt/lsf/6.2/linux2.6-glibc2.3-x86_64/lib/liblsf.a \
    ../lib/libgetopthelper.a ../libmetrics/.libs/libmetrics.a \
    ../srclib/confuse/src/.libs/libconfuse.a \
    ../srclib/apr/.libs/libapr-0.a -lrt -lm \
    -lcrypt -ldl -lresolv -lnsl -lpthread
```

[3]relative to where Ganglia was un-tar'd

References

[1] Ganglia community. Official Ganglia web site. `http://ganglia.sourceforge.net/`.

[2] Internet Engineering Task Force (IETF). Host Extensions for IP Multicasting. RFC 1112. `http://www.ietf.org/rfc/rfc1112.txt`.

[3] M. Massie, *et al.* The Ganglia Distributed Monitoring system: design, implementation and experience. *Parallel Computing*, 30:817–840, 2004.

[4] Linux magazine. Cluster monitoring with Ganglia. `http://www.linux-mag.com/id/1433`.

[5] Tobias Oetiker. RRD tool. `http://oss.oetiker.ch/rrdtool/`.