LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO-T1500227-v1 *LIGO* May 12, 2015

aLIGO DAQ hardware, software setup

K. Thorne, R. Bork

Distribution of this document:

LIGO Scientific Collaboration

This is an internal working note of the LIGO Laboratory.

California Institute of Technology
LIGO Project – MS 18-34
1200 E. California Blvd.
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project – NW22-295
185 Albany St
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
P.O. Box 1970
Mail Stop S9-02
Richland WA 99352
Phone 509-372-8106
Fax 509-372-8137

LIGO Livingston Observatory
P.O. Box 940
Livingston, LA 70754
Phone 225-686-3100
Fax 225-686-7189

http://www.ligo.caltech.edu/

Contents

Tables

1 Description

The document attempts to detail how to setup aLIGO DAQ computers for their different roles (stand-alone frame-builder, data concentrator, frame-writer, NDS server, etc.). This includes unique hardware, drivers and software. At present, it covers the existing IFO machines running an older Gentoo (kernel 2.6.35) and proposed new machines running Ubuntu 12 (kernel 3.2).

2 References

LIGO-T980024 Data Acquisition Daemon (DAQD) Client Server Communication Protocol

LIGO-T0900638 CDS Real-time Data Acquisition Software

LIGO-T0900636 Frame Builder and Network Data Server

LIGO-D1400003 aLIGO DAQ Configuration and Networks

LIGO-T1300038 EPICS Modifications for CDS

LIGO-T1000248 aLIGO CDS File System Directories

FramewriterRaid Setting up raw trend RAID arrays on IFO framewriters (wiki)

LIGO-T1000379 CDS Environment Configuration Scripts

3 Overview

The aLIGO Data Acquisition Daemon (DAQD) software takes data from the real-time control computers (front-ends) and writes it to files for storage. It also provides an interface (NDS1) for uses to retrieve both live and archived data.

There is also a separate software package (NDS). It supports requests for temporary broadcasts for test point data from the front-ends.

Being a multi-threaded process with a built-in client/server protocol between threads. It can be packaged in several configurations.

It can be installed in several ways, including (with the build label in parentheses)

- Standalone, installed directly on the front-end computer itself (SA)
- dedicated frame-builder (FB) handling all 'daqd' functions
- separate machines: data-concentrator (DC), frame-writer (FW) and NDS1 server (NDS). May also include a frame re-broadcaster (BCST)

We are currently using a specific Gentoo release (2.6.35) on the IFO systems (H1, L1). These were set up and maintained by Alex Ivanov, with some input from others. The MIT LASTI framebuilder was upgraded by Alex to Ubuntu 12, due to kernel 2.6.35 incompatibility with new server hardware.

We now need to move on to upgrading these systems and setting up new ones. This document attempts to pull together all the hardware, software configuration information for these systems.

4 Abbreviations

- SA – stand-alone DAQ on front-end computer
- DC – data concentrator computer
- FB – frame-builder computer
- FW – frame-writer computer
- NDS – NDS1 server computer
- BCST – frame re-broadcaster computer
- DAQD – Data Acquisition Daemon
- FE-DAQ – network from front-ends to data concentrator/frame builder
- DAQ-OUT – broadcast network from data concentrator to frame-writer, NDS

5 DAQ system configuration

5.1 Chassis Requirements

The computer chassis requirements differ depending on the type of DAQ server. The stand-alone systems require the same PCIe slots as a front-end computer – at least two (PCIe extender for the IO Chassis, IRIG-B receiver). They also should have PS/2 ports for keyboard, mouse so the USB support can be disabled for the front-end real-time models.

The single-machine frame-builder (FB) requires at least two PCIe slots (10G adapters, IRIG-B receiver). Additional slots may be needed for adapters to external disk arrays.

The current data concentrators (DC) on the IFO systems require 4 PCIe slots (2 10G adapters for FE-DAQ network, 1 10G adapter for DAQ-OUT network, IRIG-B receiver). To support this may require upgrading the PCIe riser cards on a 2U SuperMicro chassis for this.

The current frame writers (FW) on the IFO systems require 2 PCIe slots (10G adapter for DAQ-OUT network, adapter for external trend RAID chassis). They require at least two network ports, one for the local LAN, and one for dedicated access to the QFS server hosts the frames disk array.

The NDS servers (NDS) and GDS re-broadcasters (BCST) on the IFO systems only require 1 PCIe slot (10G adapter for DAQ-OUT network). The re-broadcasters require two network ports, with the second dedicated to the GDS broadcast network.

5.2 Memory Requirements

On the standalone systems (integrated with front-end), memory requirements are modest and accommodated by typical configurations (6Gb and above). On larger DAQ systems with multiple front-ends, the requirements are larger. In particular, the frame-writers require RAM to hold all the data before writing it to frames, and even more memory to build the frames at the same time. The NDS servers hold the same data to support queries. I surveyed existing usage on some systems, the MIT test stands (M1), the LLO DAQ test stand (X2) and the L1 IFO DAQ (L1).

DAQ	Server	Memory (GB)	Input Rate (KB/s)	# Channels	Frame Duration (s)	Rate to Disk (MB/s)
M1	FB	5	10	25 K	16	29
X2	DC	3	15	26 K		
X2	FW	11	15	26 K	64	5
X2	NDS	9	15	26 K		
L1	DC	11	53	175 K		
L1	FW	66	53	175 K	64	19
L1	NDS	11	53	175 K	64	
L1	BCST	6	--	--	1	

Table 1: DAQ Memory Requirements

There is a rough scaling between the input rate and memory requirements on the frame-writers, assuming fixed frame duration. Note that the frame-writer memory will increase if the frame duration increases (even though the rate to disk does not).

5.3 BIOS settings

5.3.1 Standalone systems

When the DAQ is run on the same machine as the front-end code, it clearly requires the BIOS settings for the real-time models. This is detailed elsewhere, but includes disabling power management, USB ports, hyperthreading, CPU frequency scaling and idling.

5.3.2 Dedicated DAQ servers

On systems with dedicated DAQ computer, the following BIOS settings are recommended

- Hyperthreading – Enabled (DAQ code is multi-threaded)
- CPU Frequency scaling, idling – Disabled (these slow the CPU to save power and screw up the DAQ real-time scheduler)
- Power Management –Maximum Performance, power saving disabled (again, DAQ code
- USB ports – can be enabled

5.4 Network settings

The Ethernet ports used for the local LAN can use the default MTU (1500 bytes). However, a frame-writer port (typically eth1) used for writing frames to a QFS server over an NFS mount should have the MTU set to 9000.

On the Gentoo 2.6.35 systems, we configure this network port ‘eth1’ by creating a link in /etc/init.d and editing /etc/conf.d/net. To create the eth1 link,

```
cd /etc/init.d
ln –s net.lo net.eth1
```

Then we edit the network start file ‘/etc/conf.d/net’ to add the following.

```
#  ETH1 – QFS server NFS mount (10.110.146)
config_eth2=( "10.110.146.102/24 brd 10.110.146.255" )
mtu_eth2="9000"
```

On the Ubuntu systems, Ethernet port specs are set in ‘/etc/network/interfaces’.

```
auto eth1
iface eth1 inet static
      address 10.110.146.102
      netmask 255.255.255.0
      broadcast 10.110.146.255
      mtu 9000
```

To simplify the NFS mount, add unique names for these ports in ‘/etc/hosts’

```
10.110.146.102    l1daqfw0-frames
10.110.146.104    l1daqgw0-frames
```

6 Driver installation and configuration

The DAQ computers use some specialized hardware and drivers. The following sections detail how drivers are installed and configured. Note that different DAQ computers may use different features, optimizations of the same driver (i.e. Myricom).

6.1 EPICS CA Repeater

The DAQ system uses the EPICS package for asynchronous exchange of status data. It is built against the CDS installation of EPICS (see LIGO-T1300038). To support that use, the DAQ computers need to run the EPICS CA repeater (caRepeater) to avoid redundant sockets, etc. This has been handled in different ways on the Gentoo and Ubuntu systems

6.1.1 Setup with Gentoo 2.6.35

On the Gentoo system, EPICS is installed at /opt/rtapps/epics. The existing method on these systems is to start 'caRepeater' at boot time by the following line in '/etc/conf.d/local.start'

```
/opt/rtapps/epics/base/bin/linux-x86_64/caRepeater&
```

6.1.2 Setup with Ubuntu 12

On the Ubuntu 12 systems, EPICS is installed at /opt/rtapps/ubuntu12/epics. Here we use the init script provided in the EPICS base build, doing the following (we could do similar things on other recent kernels

1) sudo cp /opt/rtapps/ubuntu12/epics/base/src/util/rc2.caRepeater /etc/init.d/caRepeater

2) edit /etc/init.d/caRepeater to define INSTALL_BIN as

```
INSTALL_BIN="/opt/rtapps/ubuntu12/epics/base/bin/linux-x86_64"
```

3) make the script executable

```
cd /etc/init.d
sudo chmod +x ./caRepeater
```

4) Add to system startup

```
sudo update-rc.d caRepeater defaults
```

6.1.3 Check for working driver

You can simply check that caRepeater is running

```
controls@x2daqdc0 ~ $ ps -ef | grep caRep
root 5449 1 0 May05 ? 00:00:07 /opt/rtapps/epics/base/bin/linux-
x86_64/caRepeater
```

6.2 IRIG-B driver (SA, FB, DC)

The LIGO DAQ systems use an IRIG-B receiver to get the GPS time. This is compared against the GPS time in the packets received from the front-ends to check for timing errors/data dropouts. Two different cards (Symmetricom, SpectraCom) are supported. The driver code is released with the RCG packages. That is where it is built and installed from.

Only the machine receiving data directly from the front-ends needs this card and driver. So it is only found on the front-ends, data concentrators, and frame-builders

6.2.1 Driver build and install

You go to the RCG checkout to build the driver. Here we assume the default checkout.

```
cd /opt/rtcds/rtscore/release
cd src/drv/symmetricom
make clean
make
sudo make install
```

6.2.2 Driver startup

On the Gentoo systems, the driver is started with the following in '/etc/conf.d/local.start'

```
# Symmetricom IRIG-B start
/sbin/modprobe symmetricom
mknod /dev/symmetricom c `grep symmetricom /proc/devices|awk '{print $1}'` 0
chown controls /dev/symmetricom
```

On the MIT framebuilder (Ubuntu) there is a similar call in '/etc/rc.local'.

The driver puts its output in a proc file '/proc/gps'

6.2.3 Check for working driver

If the driver is running, you should have GPS time data in the proc file. For example

```
controls@x2daqdc0 ~ $ cat /proc/gps
1115136784.91
```

6.3 MX driver (FB, DC)

On DAQ machines that receive data from the front-ends over a network (FE-DAQ), a Myricom 10G card may be used together with the 'mx' driver. The driver supports the mx_stream driver that is run on each front end using an 'open-mx' driver on its vanilla 1G port.

6.3.1 OS compatibility

The vendor (Myricom) is no longer providing updates to the 'mx' for newer Linux kernel. As of now, we know that it will work up through Linux kernel 3.2 . Thus, DAQ operating systems should not be progressed beyond this kernel.

6.3.2 Driver build and install

The 'mx' package is provided by Myricom (Myricom web site.) The release in use is mx 1.2.16. This had been previously downloaded and is stored for retrieval from the LIGO DAQ software site (mx_1.2.6.tar.gz). Copy it to /opt/rtcds/drivers so you can access it for builds. We built it following the MX Installation Instructions

Log onto the DAQ computer as 'root'

```
cp /opt/rtcds/drivers/mx_1.2.16.tar.gz /root
```

```
cd /root
gunzip -c mx_1.2.16.tar.gz | tar xvf - (creates /root/mx-1.2.16)
cd mx-1.2.16
./configure --enable-ether-mode
make
make install (this installs the driver in /opt/mx, creates /etc/init.d/mx)
```

6.3.3 Driver startup

On the Gentoo systems, the following is added to '/etc/conf.d/local.start'.

```
/etc/init.d/mx start
```

On the Ubuntu systems, we likely need to add it to the startup with the following. You can check that it is installed by checking for links like /etc/rc2.d/S**mx

```
sudo update-rc.d mx default
```

6.3.4 Configuring for multiple end-points, Myricom cards

For frame-builders on the smaller systems (i.e. MIT), the default driver settings are fine. However, on the large IFO systems, we have resorted to providing a unique end-point for each front-end computer by installing 2 10G cards for this and enabling 16 end-points on each card. On the data concentrator, this is done by editing the init script '/etc/init.d/mx'

```
# add/modify/uncomment MX_MODULES_PARAMS lines to change defaults
# 2 10G cards - L1, H1
MX_MODULE_PARAMS="mx_max_instance=2 mx_max_endpoints=16 $MX_MODULE_PARAMS"
```

To use this, the open-mx driver and mx_stream used on the front-ends needs to be updated, but that is covered elsewhere.

On the IFO data concentrators, we actually have three such 10G adapters installed, but we only use two of them for the MX input. Thus we need to figure out which cards these are. The PCIe riser cards often have a non-obvious bus ordering. The best way is to

1) note MAC address of each10G adapter when installed. See label on each adapter.
2) Get the mx driver running
3) Check the system log 'dmesg' for records 'mx', 'MAC'. For example, from the test stand:

```
  controls@x2daqdc0 ~ $ dmesg | grep mx | grep MAC
[   21.057748] mx INFO: Board 0: MAC address = 00:60:dd:43:c3:4e
[   27.041735] mx INFO: Board 1: MAC address = 00:60:dd:44:b2:16
```

The board that is not in the list is available for broadcasting on the DAQ-OUT network (see below)

6.3.5 Check for working driver

If the driver is working, the mx mapper should see things. Run '/opt/mx/bin/mx_info' to check. An example from the test stand:

```
controls@x2daqdc0 ~ $ /opt/mx/bin/mx_info
MX Version: 1.2.16
MX Build: root@x2daqdc:/root/mx-1.2.16 Mon Nov 10 09:51:52 CST 2014
2 Myrinet boards installed.
The MX driver is configured to support a maximum of:
      16 endpoints per NIC, 1024 NICs on the network, 2 NICs per host
===================================================================
Instance #0:  364.4 MHz LANai, PCI-E x8, 2 MB SRAM
      Status:           Running, P0: Link Up
      Network:    Ethernet 10G

      MAC Address:      00:60:dd:43:c3:4e
      Product code:     10G-PCIE-8B-S
      Part number:      09-04228
      Serial number:    472794
      Mapper:           00:60:dd:44:b2:16, version = 0x00000000, configured
      Mapped hosts:     6

                                                        ROUTE COUNT
INDEX    MAC ADDRESS     HOST NAME                        P0
-----    -----------     ---------                        ---
   0) 00:60:dd:43:c3:4e x2daqdc0:0                        1,0
   1) 00:60:dd:44:b2:16 x2daqdc0:1                        1,0
   2) 00:25:90:0d:6a:ff x2oaf0:0                          1,0
   3) 00:25:90:37:f8:9d x2seiex:0                         1,0
   4) 00:25:90:0d:78:77 x2lsc0:0                          1,0
   5) 00:25:90:37:fa:45 x2susex:0                         1,0
===================================================================
Instance #1:  364.4 MHz LANai, PCI-E x8, 2 MB SRAM
      Status:           Running, P0: Link Up
      Network:    Ethernet 10G

      MAC Address:      00:60:dd:44:b2:16
      Product code:     10G-PCIE-8B-S
      Part number:      09-04228
      Serial number:    443349
      Mapper:           00:60:dd:44:b2:16, version = 0x00000000, configured
      Mapped hosts:     6

                                                        ROUTE COUNT
INDEX    MAC ADDRESS     HOST NAME                        P0
-----    -----------     ---------                        ---
   0) 00:60:dd:43:c3:4e x2daqdc0:0                        1,0
   1) 00:60:dd:44:b2:16 x2daqdc0:1                        1,0
   2) 00:25:90:0d:6a:ff x2oaf0:0                          1,0
   3) 00:25:90:37:f8:9d x2seiex:0                         1,0
   4) 00:25:90:0d:78:77 x2lsc0:0                          1,0
   5) 00:25:90:37:fa:45 x2susex:0                         1,0
```

6.4 Myricom driver (DC)

On the IFO DAQ systems, we have installed dedicated 10G network switches that pass packets of merged front-end data broadcast from the data concentrator (DC) to the DAQ machines (FW, NDS, BCST) that received these packets. We use the same Myricom 10G cards for this purpose as above, but here we use the conventional 10G Ethernet driver (myri10ge). Currently different revisions and configurations of this driver are used on the broadcaster than on the receivers.

6.4.1 Driver build and install

Current data concentrators are only running Gentoo (2.6.35), so I'll only cover that case here. We are currently using myri10ge 1.5.1. You can get the source tar-ball file myri10ge-linux.1.5.1.tgz from Myricom web site or the LIGO DAQ software site (myri10ge-linux.1.5.1.tgz). We need to do the build such that we include the firmware in the kernel module. We also set things for jumbo frames (which can be done when configuring the port as well)

1) login to data-concentrator as 'controls' and copy the file to the 'home' directory

```
2) tar -xzf myri10ge-linux.1.5.1.tgz
3) cd myri10ge-linux.1.5.1/linux
4) make clean
5) make MYRI10GE_BUILTIN_FW=1 MYRI10GE_JUMBO=1
6) su
```

For some reason, instead of running 'make install' to install the driver, Alex simply copied the kernel object file 'myri10ge.ko' from this directory to /etc/conf.d.

```
cp myri10ge.ko /etc/conf.d
```

I think this was to simplify testing and a need to reboot. We have not had time to check that a normal install will work.

6.4.2 Driver startup

On the Gentoo system, we configure the network port eth2 (skipping the two built-in Ethernet ports) by creating a link in /etc/init.d and editing /etc/conf.d/net. To create the eth2 link,

```
cd /etc/init.d
ln –s net.lo net.eth2
```

Then we edit the network start file '/etc/conf.d/net' to add the following.

```
#  ETH2 - DAQ-OUT connection 10.148
config_eth2=( "10.148.0.100/24 brd 10.148.0.255" )
mtu_eth2="9000"
```

Note that specifying the MTU to be 9000 does the same thing as building the driver with the JUMBO option.

We then add the following to '/etc/conf.d/local.start' to increase the max read-queue memory.

```
/sbin/sysctl -w net.core.rmem_max=8388608
```

We also add the following later in '/etc/conf.d/local.start' to load the driver from the location is what put at (/etc/conf.d). It is unclear why it is loaded twice. It may have to do with getting the firmware loaded.

```
insmod /etc/conf.d/myri10ge.ko
rmmod myri10ge
insmod /etc/conf.d/myri10ge.ko
```

Of course, to make these active we need to reboot the computer.

6.4.3 Check for working driver

We should check the eth2 port configuration and driver modules loaded. Check that MTU:9000 set for the port and the myri10ge module is loaded

```
controls@x2daqdc0 /etc/init.d $ /sbin/ifconfig eth2
eth2      Link encap:Ethernet  HWaddr 00:60:dd:45:f7:2b
          inet addr:10.148.0.100  Bcast:10.148.0.255  Mask:255.255.255.0
          inet6 addr: fe80::260:ddff:fe45:f72b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
          RX packets:884658 errors:0 dropped:0 overruns:0 frame:0
          TX packets:546383906 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:85214204 (81.2 MiB)  TX bytes:4355076297430 (3.9 TiB)
          Interrupt:101

controls@x2daqdc0 /etc/init.d $ lsmod
Module                  Size  Used by
myri10ge               48972  0
mx_driver             126410  34
mx_mcp                499107  1 mx_driver
symmetricom             5698  1
```

6.5 Myricom driver (FW, NDS, BCST)

The DAQ computers that received broadcasts are the frame-writer (FW), NDS server and GDS broadcaster. On the large IFO DAQ systems, we have had to customize the Myricom driver and system settings to handle the broadcast rate. This is done using a newer version of the driver.

6.5.1 Driver build and install

On the large IFO DAQ systems, we have had to customize the Myricom driver. This requires either a new driver build (Gentoo) and firmware installation (Ubuntu).

6.5.1.1 Gentoo kernel 2.6.35

We need to get a recent myri10ge release from the vendor that supports the 'myri10ge_big_rxring' load-time option. See https://www.myricom.com/software/myri10ge/856-would-you-explain-the-myri10ge-load-time-option-myri10ge-big-rxring.html. This load-time option configures the driver to load firmware that uses larger receive rings (4096 buffers vs 1024 buffers). This feature is available in Linux Myri10GE 1.5.3.p2 and later. We can get this from the vendor website or the LIGO DAQ software site (myri10ge-linux.1.5.3.p3.tgz). Once you have the file, then

1) login to frame-writer as ‘controls’ and copy the file to the 'home' directory

```
2) tar -xzf myri10ge-linux.1.5.3.p3.tgz
3) cd myri10ge-linux.1.5.3.p3/linux
4) make clean
5) make MYRI10GE_BUILTIN_FW=1 MYRI10GE_JUMBO=1
```

Again, instead of running ‘make install’ to install the driver, Alex simply copied the kernel object file ‘myri10ge.ko’ from this directory to /etc/conf.d.

```
cp myri10ge.ko /etc/conf.d
```

This was likely done for speed, to copy it to multiple machines. We have not had time to check that a normal install will work.

6.5.1.2 Ubuntu 12

For Ubuntu 12 (linux 3.2), there is an in-kernel ‘myri10ge’ driver. This in-kernel driver does not support the ‘myri10ge_big_rxring’ load-time option used on the Gentoo machines. However, since the myri10ge_big_rxring parameter simply tells the driver to load a firmware variant that supports larger receive rings, it is possible to achieve the myri10ge_big_rxring functionality by doing the following workaround:

Download and install the latest firmware TAR.GZ file (myri-fw-1.4.58.tar.gz) from https://www.myricom.com/myri10ge-firmware.html. Once downloaded, do the following:

```
tar xvzf myri-fw-1.4.58.tar.gz
cd myri-fw-1.4.58
sudo bash install.sh
```

This procedure copies the firmware files to /lib/firmware.
Load the "big" firmware via myri10ge_fw_name.

```
sudo rmmon myri10ge
sudo modprobe myri10ge myri10ge_fw_name=myri10ge_eth_big_z8e.dat
```

HOWEVER: Testing of frame-writer performance on a full-up IFO DAQ shows that this option gives worse performance under Ubuntu 12. This it should not be used.

6.5.2 Driver Startup

To use the driver, we also must configure the Ethernet port. To support the high network throughput rates on the frame-writer, network parameters are tuned.

6.5.2.1 Gentoo kernel 2.6.35

As on the data-concentrator, we configure the network port eth2 by creating a link in /etc/init.d and editing /etc/conf.d/net. To create the eth2 link,

```
cd /etc/init.d
ln –s net.lo net.eth2
```

Then we edit the network start file '/etc/conf.d/net' to add the following.

```
#  ETH2 - DAQ-OUT connection 10.148
config_eth2=( "10.148.0.102/24 brd 10.148.0.255" )
mtu_eth2="9000"
```

We then add the following to '/etc/conf.d/local.start' to increase the max read-queue memory to an even higher level than on the concentrators

```
/sbin/sysctl -w net.core.rmem_max=33554432
```

We add the following later in '/etc/conf.d/local.start' to load the driver with the special firmware. It is unclear why it is loaded twice. It may have to do with getting the firmware loaded

```
insmod /etc/conf.d/myri10ge.ko myri10ge_big_rxring=1
rmmod myri10ge
insmod /etc/conf.d/myri10ge.ko myri10ge_big_rxring=1
```

Of course, to make these active we need to reboot the computer.

6.5.3 Ubuntu 12

On the Ubuntu system, Ethernet port specs are set in '/etc/network/interfaces'.

```
auto eth2
iface eth2 inet static
      address 10.148.0.102
      netmask 255.255.255.0
      broadcast 10.148.0.255
      mtu 9000
```

We looked more deeply at how to optimize the performance, using recommendations from https://www.myricom.com/software/myri10ge/392-how-do-i-troubleshoot-slow-myri10ge-or-mx-10g-performance.html.

We added the following to '/etc/sysctl/conf'

```
net.core.rmem_max = 33554432
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.core.netdev_max_backlog = 250000

sudo sysctl –p /etc/sysctl.conf
```

Because the driver is in-kernel, we did not have to do anything special to start it. However, a system reboot was done to ensure things were done in a permanent fashion.

6.5.4 Check for working driver

As with the data concentrator, you can check on the network port with '/sbin/ifconfig eth2' and on the kernel modules with 'lsmod'.

On the Gentoo 2.6.35 systems, check that the big_rxring firmware is loaded with 'ethtool –g:

```
cdsadmin@x2daqfw0:~$ ethtool -g eth2
Ring parameters for eth2:
Pre-set maximums:
RX:          8192
RX Mini:     8192
RX Jumbo:    0
TX:          16384
Current hardware settings:
RX:          8192
RX Mini:     8192
RX Jumbo:    0
TX:          16384
```

7 DAQ server setup

7.1 'controls' accounts

Create a 'controls' account and group with the same userid and groupid used on the front-end systems (typically '1001') for both.

In '/etc/passwd'

```
controls:x:1001:1001::/home/controls:/bin/bash
```

In '/etc/group'

```
controls:x:1001:
```

7.2 NFS mounts

We need to mount the common file system shared with the front-end computers. On existing systems, these are NFS mounted from the boot server (i.e. l1boot) as a simple mount point (/opt). Here is an example of that from '/etc/fstab' on L1.

```
# <fs>                  <mountpoint>      <type>      <opts>      <dump/pass>
l1boot:/opt             /opt              nfs         defaults     0  0
```

We are migrating the larger systems to a dedicated NFS server. In that case, we have moved to separate mount points for the required directories (/opt/cdscfg, /opt/mx, /opt/rtapps and /opt/rtcds). This better supports the use of mixed operating systems. Here is an example piece from '/etc/fstab' on X2 DAQ.

```
# <fs>                  <mountpoint>      <type>      <opts>      <dump/pass>
x2file:/rtopt/cdscfg    /opt/cdscfg       nfs         defaults     0  0
x2file:/rtopt/rtapps    /opt/rtapps       nfs         defaults     0  0
x2file:/rtopt/rtcds     /opt/rtcds        nfs         defaults     0  0
x2file:/rtopt/mx        /opt/mx           nfs         defaults     0  0
```

The dedicated frame writers need to mount the /frames disk on the QFS server, but using the dedicated Ethernet port (which has write access). Here is an example Frame writer '/etc/fstab/

```
# <fs>                  <mountpoint>      <type>      <opts>      <dump/pass>
l1boot:/opt             /opt              nfs         defaults     0  0
```

```
l1gw1-frames:/daq-l1-frames   /frames     nfs
      vers=3,async,rw,rsize=524288,wsize=524288       0 0
LABEL=rawtrend          /rawtrends        ext4        defaults,noatime  0 0
LABEL=rawarchive        /rawarchive       ext4        defaults,noatime  0 0
```

The last two lines are mounting the raw trend RAID arrays. This setup is currently covered in the LLO CDS wiki, see FramewriterRaid. This was required to accommodate the incredible number (~175,000) raw trend files by using an SSD raid for initial creation.

The dedicated NFS servers need to mount the /frames disk on the regular LAN port (as they only need read access). They also need to mount the raw trends, as see in this '/etc/fstab':

```
# <fs>                  <mountpoint>      <type>      <opts>      <dump/pass>
l1boot:/opt             /opt              nfs         defaults    0 0
l1daqgw0:/dcs-l1-frames /frames           nfs         defaults    0 0
l1daqfw0:/rawtrends     /rawtrends        nfs         defaults    0 0
l1daqfw0:/rawarchive    /rawarchive       nfs         defaults    0 0
```

7.3 Frame disk directories

The following directories are the conventional ones used for frames

- /frames/full – For full data frame files (1st acquire bit set)
- /frames/science/ - For science-mode frame files (2nd acquire bit set)
- /frames/trend/second – For second trend frames
- /frames/trend/minute – For minute trend frames
- /frames/trend/minute_raw – For raw trend files (one per channel)

On systems with dedicated frame-writers, these should all be set to ownership by the 'controls' account. This is particularly important when using the QFS-served frame disks, as this is the only way to get both Linux and Solaris systems to have a common userid, groupid for these files (as super-user account IDs do not match).

7.4 System files

Each operating system has a slightly different way it is customized for running the daqd package. Templates for them are starting to be distributed with the RCG code under the 'setup' folder.

7.4.1 Gentoo 2.6.34-cs (standalone with front-end models)

On the front-end computers, the DAQ processes are started by the 'inittab'. If you disable/enable the restart of the DAQ (because it keeps crashing or is now OK), you can comment the start-up calls in 'inittab', then do 'init –q' to activate the updated file.

- */etc/inittab* – starts daqd, nds process
- */opt/rtcds/<site>/<ifo>/target/fb/start_daqd* – daqd start script
- */opt/rtcds/<site>/<ifo>/target/fb/start_nds* – nds start script
- */etc/ld.so.conf.d/epics-x86_64.conf* – put EPICS libraries on path
- */etc/ld.so.conf.d/framecpp-x86_64.conf* – put framecpp libraries on path

7.4.2 Gentoo 2.6.35 (dedicated DAQ hardware)

On the dedicated machines, the start/stop got moved to monit process through its '/etc/monitrc' file. The initialization of drivers, etc. is done in the local.start, local.stop files. The files in ld.so.conf.d list the library directories that the daqd, nds executables need to run.

- */etc/conf.d/local.start* – startup script (caRepeater, symmetricom, mx, myri10ge, monit)
- */etc/conf.d/local.stop* – shutdown script
- */etc/conf.d/net* – network configuration (eth0, eth1, eth2)
- */etc/init.d/daqd_** - daqd start/stop script
- */etc/init.d/nds_** - nds start/stop script [SA, FB, NDS only]
- */etc/init.d/mx* – MX driver start/stop script [FB, DC only]
- */etc/ld.so.conf.d/epics-x86_64.conf* – put EPICS libraries on path
- */etc/ld.so.conf.d/framecpp-x86_64.conf* – put framecpp libraries on path
- */etc/ld.so.conf.d/mx_x86_64.conf* – put MX libraries on path [FB, DC only]
- */etc/monitrc* – monitors daqd, nds processes, restarting them as needed

To stop the daqd process manually (and prevent restarts), you login to DAQ computer and type

```
sudo /etc/init.d/monit stop
sudo /etc/init.d/daqd_* stop
```

To restart the daqd process, you can restart either init script.

7.4.3 Ubuntu 12

Under Ubuntu 12, any driver initialization has moved to '/etc/init.d/rc.local'. The monit process still starts/stops the daqd, nds processes, with init scripts still in '/etc/init.d'. However, variables for invoking these scripts are in '/etc/default'. The files in ld.so.conf.d list the library directories that the daqd, nds executables need to run.

- */etc/init.d/rc.local* – startup script (symmetricom, mx)
- */etc/network/interfaces* – network configuration (eth0, eth1, eth2)
- */etc/init.d/daqd* - daqd start/stop script
- */etc/init.d/nds* - nds start/stop script [SA, FB, NDS only]
- */etc/init.d/mx* – MX driver start/stop script [FB, DC only]
- */etc/ld.so.conf.d/epics-x86_64.conf* – put EPICS libraries on path
- */etc/ld.so.conf.d/framecpp-x86_64.conf* – put framecpp libraries on path
- */etc/ld.so.conf.d/mx_x86_64.conf* – put MX libraries on path [FB, DC only]
- */etc/monitrc* – monitors daqd, nds processes, restarting them as needed
- */etc/defaults/daqd* – define variables for init script
- */etc/defaults/nds* - define variables for init script
- */etc/security/limits.conf* – Allow controls users access to real-time priority queues

To stop the daqd process manually (and prevent restarts), you login to DAQ computer and type

```
sudo /etc/init.d/monit stop
sudo /etc/init.d/daqd stop
```

To restart the daqd process, you can restart either init script.

7.5 daqd init scripts

The daqd init scripts on the dedicated DAQ server are each slightly different

7.5.1 Data Concentrator

On the data concentrator, we want to move all the channel lists in the master file to a 'current' directory that is only updated when the data concentrator daqd is restarted. By using this for the downstream DAQ machines (FW, NDS, BCST), we ensure that daqd restarts on those machines do not pick up channel list files modified since the data concentrator's daqd was restarted.

Note that the daqd process runs as 'root' on the data concentrator. This ensures that its threads have access to the real-time priority scheduler, to keep up with the data sent from the real-time front-ends.

#!/sbin/runscript

```
opts="${opts}"
hostname=`hostname`
daqdir="/opt/rtcds/tst/x2/daq"

start() {
      ebegin "Starting Data Concentrator"

      # Copy INI, PAR files to secure location. Create a master file
      # This permits restarts to frame writers, nds machines against the
      # channel configuration at the time of the data concentrator start,
      # and prevents modified INI, PAR files from being used.
      # D.Barker 23May2012

      # remove good-to-go flag
      rm -f ${daqdir}/running/READY
      # INI, PAR files copied into a directory named by the current date-time
      # a symlink called running is pointed to the latest directory
      DATETIME=`date +%d%m%y_%H:%M:%S`
      mkdir -p ${daqdir}/${DATETIME}
      rm -rf ${daqdir}/running
      ln -s ${daqdir}/${DATETIME} ${daqdir}/running
      for f in `grep "^/opt" /opt/rtcds/tst/x2/target/x2daqdc0/master`
      do
         cp $f ${daqdir}/${DATETIME}
      done
      # create new master file referencing this running configuration
      for f in ${daqdir}/running/*
      do
         echo $f >> ${daqdir}/running/master
      done
      # End of INI-PAR copy change. DB.

      # archive old log files, creating archive if needed
      cd /opt/rtcds/tst/x2/target/x2daqdc0
      mkdir -p logs/archive
      mv -f logs/daqd.log*  logs/archive/

      # start up data concentrator with date-stamped log
      stamp=`date +%s`
      start-stop-daemon --start --quiet -b -m --pidfile /var/run/daqd.pid --
exec /opt/rtcds/tst/x2/target/x2daqdc0/daqd -- -c
/opt/rtcds/tst/x2/target/x2daqdc0
/daqdrc -l /opt/rtcds/tst/x2/target/x2daqdc0/logs/daqd.log.${stamp}
      eend $?

      # set good-to-go flag
      touch ${daqdir}/running/READY
}

stop() {
      ebegin "Stopping Data Concentrator"
      start-stop-daemon --stop --signal 9 --exec
/opt/rtcds/tst/x2/target/x2daqdc0/daqd
      eend $?
}
```

7.5.2 Frame writer

On the frame writers, the daqd process runs as the 'controls' user. As with the data concentrator, they archive the old log files.

7.5.2.1 Gentoo 2.6.35

On the older Gentoo systems, we have simply 'start' and 'stop' states. Also note that (unlike the Ubuntu scripts), there is no attempt to set the real-time priority or nice. These were not allowed for non-root users under this older kernel.

```
#!/sbin/runscript

opts="${opts}"
hostname=`hostname`

start() {
      ebegin "Starting Frame Writer"
      # archive old log files, creating archive if needed
      cd /opt/rtcds/llo/l1/target/l1daqfw0
        mkdir -p logs/archive
      mv -f logs/daqd.log*  logs/archive/
      # start up frame writer with date-stamped log
      stamp=`date +%s`
      start-stop-daemon --start --quiet -b -m --pidfile /var/run/daqd.pid --
chuid controls --exec /opt/rtcds/llo/l1/target/l1daqfw0/daqd -- -c
/opt/rtcds/llo/l1/target/l1daqfw0/daqdrc -l
/opt/rtcds/llo/l1/target/l1daqfw0/logs/daqd.log.${stamp}
      eend $?
}

stop() {
      ebegin "Stopping Frame Writer"
      start-stop-daemon --stop -s 9 --exec
/opt/rtcds/llo/l1/target/l1daqfw0/daqd
      eend $?
}
```

7.5.2.2 Ubuntu 12

Here we use the template init script. Only the customized parts are shown here.

```
#!/bin/bash
### BEGIN INIT INFO
# Provides:          daqd
# Required-Start:    $network $local_fs
# Required-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: <Enter a short description of the software>
# Description:       <Enter a long description of the software>
#                    <...>
#                    <...>
### END INIT INFO

# PATH should only include /usr/* if it runs after the mountnfs.sh script
PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC=daqd             # Introduce a short description here
NAME=daqd             # Introduce the short server's name here
DAEMON=/opt/rtcds/tst/x2/target/x2daqfw0/daqd # Introduce the server's location
here
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME
#
do_start()
{
      ulimit -r 99
      mkdir -p $LOGDIR/archive
      if stat -t $LOGDIR/$NAME* >/dev/null 2>&1 ; then
          mv -f $LOGDIR/$NAME*  $LOGDIR/archive/
      fi
      chown -R controls $TARGET
      # Return
      #   0 if daemon has been started
      #   1 if daemon was already running
      #   2 if daemon could not be started
      start-stop-daemon -N -20 -m -b -c controls --start --quiet --pidfile
$PIDFILE --exec $DAEMON --test > /dev/null \
            || return 1
      start-stop-daemon -N -20 -m -b -c controls --start --quiet --pidfile
$PIDFILE --exec $DAEMON -- \
            $DAEMON_ARGS \
            || return 2
      # Add code here, if necessary, that waits for the process to be ready
      # to handle requests from services started subsequently which depend
      # on this one.  As a last resort, sleep for some time.
}
```

Note the use of 'ulimit –r 99' to allow use of the real-time priorities and '–N -20' to set nice. These only work because the 'controls' user is authorized due to the following in '/etc/security/limits.conf':

```
controls    hard  rtprio 99
controls    hard  nice -20
```

8 Building DAQ software

8.1 Required packages

The DAQ software requires the source code from the RCG repository (advLigoRTS). That is covered in the next section. In addition, it needs two other packages, EPICS and framecpp.

You can find tar-balls of source and pre-built libraries on the CDS DAQ download site:

https://llocds.ligo-la.caltech.edu/daq/software/

8.1.1 EPICS

The CDS sys-admins maintain tar-balls of source and pre-built EPICS collaboration packages. See LIGO-T1300038. For the DAQ software build, we need the EPICS Base package. Typically that is installed under '/opt/rtapps' (Gentoo) or '/opt/rtapps/ubuntu12 (Ubuntu 12). We have recently added files to support 'pkg-config' use to simplify builds. We store them at revision-specific sub-directories, using the 'epics' soft-link to point to the current one.

8.1.2 Framecpp

The framecpp package is maintained by the LIGO Laboratory. It is currently distributed as part of the 'ldas-tools' package. If built from source, it is usually installed at the same location as EPICS. We use the 'framecpp' or 'ldas-tools' soft-link to point to the current one. Source-code tar-balls for the various releases are available at http://software.ligo.org/lscsoft/source/. This is needed for Gentoo. For Ubuntu 12, you may be able to use the Debian 'lscsoft' package 'ldas-tools-framecpp-devel'.

8.2 Target Directories

This setup follows the directory structures detailed in LIGO-T1000248. Some adjustments are made for new cases where the DAQ computers may not share an OS with the front-ends.

Set up a dedicated target area, under /opt/rtcds/<site>/<ifo>/target. For stand-alone system, this can be the 'fb' directory. On systems with dedicated servers, create a directory for each server (i.e. l1daqdc0, x2daqfw0, etc.). This typically has sub-directories for archived executables (bin_archive), log files (logs) and archived log files (logs/archive). This directory will need the following (in addition to the daqd, nds executables)

8.2.1 daqdrc – client command file

The daqd package has an established communication protocol where it listens for commands on port 8087 for client requests. The syntax is described in LIGO-T0900636. When we start the 'daqd', we start its operation using a file of these client requests, typically named 'daqdrc'. It can be difficult to set up this file, and there are (currently) several lines of commands required to satisfy no-longer-used features. We typically take an existing one and modify it.

8.2.2 master – list of channel lists

The 'daqdrc' file will point to a 'master' file. This file is itself a list of files containing channels to be included in the DAQ. The syntax of these channel list files is shared with the real-time code (See LIGO-T0900638) to define what channels go to frames (and which frames) and what are test-points that can be activated as needed. There should only be one such file for each DAQ system at a time.

8.2.3 frame_wiper.pl

The frame_wiper.pl script is used by the framewriter (FW), framebuilder (FB) or stand-alone system (SA) to keep the archive of frame files from overflowing the disk. It should be run about every hour on IFO systems, and perhaps daily on smaller test stands.

8.2.4 broadcast.ini

The GDS frame re-broadcaster needs a list of channels for its 1-second frame. These are listed in the 'broadcast.ini' file. Each channel name is on a single line, enclosed in square brackets [ ].

8.3 RCG Setup

Check out the desired version of the RCG code from the repository. If building on a stand-alone system, you can use the version checked out for the front-end code (usually at '/opt/rtcds/rtscore/release'). If you are building for dedicated servers, it is now best to use a different set of directories for this, such as '/opt/rtcds/daqcore/release'. Export to a release-specific directory. Then change the 'release' softlink to the new location if you want to make it easier to navigate to. For example

```
cd /opt/rtcds/daqcore
svn ex https://redoubt.ligo-wa.caltech.edu/svn/advLigoRTS/tags/advLigoRTS-
2.10.1
rm release
ln –s advLigoRTS-2.10.1 release
```

If you have a system with DAQ computers with different operating systems, you will want to export it to different locations for each operating system.

Log in to the DAQ computer as 'controls'. You can do all builds for a particular OS from one DAQ computer. First we need to do some work in the directory where we exported the code. From a Terminal window, change to that directory (here I assume it is the one suggested above). First build the GDS libraries to support calls to awgtpman from the DAQ code.

```
cd /opt/rtcds/daqcore/advLigoRTS-2.10.1
cd src/gds
make clean
make
cd ../..
```

If using an RCG release of 2.10 or greater, then we need to do set up autoconf tools to use info on the EPICS and framecpp packages to simplify builds. So we run './bootstrap' in the daqd and nds source directories.

```
cd src/daqd
./bootstrap
cd ../nds
./bootstrap
cd ../..
```

Now we can set up a different directory to do the builds in. Typically, this is under '/opt/rtcds/<site>/<ifo>/daqbuild. Create a directory there and run the configure script of the DAQ RCG check out area. For example

```
cd /opt/rtcds/tst/x2/daqbuild
mkdir daq-2.10.1
cd daq-2.10.1
/opt/rtcds/daqcore/advLigoRTS-2.10.1/configure
```

We can make this the default DAQ build area using the softlink

```
cd /opt/rtcds/tst/x2/daqbuild
rm current
ln –s advLigoRTS-2.10.1 current
```

Now we are ready to build ‘daqd’ and ‘nds’ executables.

8.4 Building for different DAQ configurations

In the build area (‘/opt/rtcds/tst/x2/daqbuild/release’) will be a ‘Makefile’ that has several pre-made daqd ‘make’ commands to choose different options based on a label.

8.4.1 Standalone (SA)

We need to build both a DAQ daemon (standiop) and NDS process. Here I use ‘tst’ and ‘x2’ as <site> and <ifo>. You need to customize them to your set up.

Login as ‘controls’, then

```
cd /opt/rtcds/tst/x2/daqbuild/current
make standiop
make nds
cp –p build/standiop/daqd /opt/rtcds/tst/x2/target/fb/bin_archive/daqd-2.10.1
cp –p build/nds/nds /opt/rtcds/tst/x2/target/fb/bin_archive/nds-2.10.1
```

You will notice that I copied the new daqd, nds executables to the archive area first, with a revision specific name. That is to stage them without interrupting the currently running daqd, nds executables. To start the new ones, we need to stop the auto-start of daqd, nds in the inittab, move the new ones into

Edit /etc/inittab to comment out the fb and nds lines.

```
sudo init q
cd /opt/rtcds/tst/x2/target/fb
cp –p bin_archive/daqd-2.10.1 daqd
cp –p bin_archive/nds-2.10.1 nds
```

Edit /etc/inittab again (or swap in the old one) to un-comment the lines

```
sudo init q
```

8.4.2 Framebuilder (FB)

This also needs a daqd, nds. I will assume here that ‘monit’ is used to auto-start the executables. For the frame-builder, choose the ‘mx’ option.

Login as ‘controls’, then

```
cd /opt/rtcds/mit/m1/daqbuild/current
make mx
make nds
cp –p build/mx/daqd /opt/rtcds/mit/m1/target/m1fb0/bin_archive/daqd-2.10.1
cp –p build/nds/nds /opt/rtcds/mit/m1/target/m1fb0/bin_archive/nds-2.10.1
```

Now we just stop/start monit to switch

```
sudo /etc/init.d/monit stop
sudo /etc/init.d/daqd stop
sudo /etc/init.d/nds stop
cd /opt/rtcds/mit/m1/target/m1fb0
cp –p bin_archive/daqd-2.10.1 daqd
cp –p bin_archive/nds-2.10.1 nds
sudo /etc/init.d/monit start
```

8.4.3 Data Concentrator (DC)

The data concentrator only needs ‘daqd’ built using ‘dc’ option. Again, monit is in use.

Login as ‘controls’ on the data concentrator, then

```
cd /opt/rtcds/tst/x2/daqbuild/current
make dc
cp –p build/dc/daqd /opt/rtcds/tst/x2/target/x2daqdc0/bin_archive/daqd-2.10.1
sudo /etc/init.d/monit stop
sudo /etc/init.d/daqd_dc0 stop
cd /opt/rtcds/tst/x2/target/x2daqdc0
cp –p bin_archive/daqd-2.10.1 daqd
sudo /etc/init.d/monit start
```

8.4.4 Frame writer (FW)

The frame writer only needs ‘daqd’ built using ‘fw’ option, if there is a dedicated NDS server as well. Again, monit is in use.

Login as ‘controls’ on the framewriter, then

```
cd /opt/rtcds/tst/x2/daqbuild/current
make fw
cp –p build/fw/daqd /opt/rtcds/tst/x2/target/x2daqfw0/bin_archive/daqd-2.10.1
sudo /etc/init.d/monit stop
sudo /etc/init.d/daqd_fw0 stop
cd /opt/rtcds/tst/x2/target/x2daqfw0
cp –p bin_archive/daqd-2.10.1 daqd
sudo /etc/init.d/monit start
```

Note that if multiple framewriters are used, you can build on one machine, then copy to the target of the other machine as well.

8.4.5 NDS server (NDS)

The NDS server needs ‘daqd’ built using ‘rcv’ option to include the test points. It also needs the ‘nds’ executable built.

Login as ‘controls’ on the NDS machine, then

```
cd /opt/rtcds/tst/x2/daqbuild/current
make rcv
make nds
cp –p build/rcv/daqd /opt/rtcds/tst/x2/target/x2daqnds0/bin_archive/daqd-2.10.1
cp –p build/nds/nds /opt/rtcds/tst/x2/target/x2daqnds0/bin_archive/nds-2.10.1
sudo /etc/init.d/monit stop
sudo /etc/init.d/daqd_nds0 stop
sudo /etc/init.d/nds_nds0 stop
cd /opt/rtcds/tst/x2/target/x2daqnds0
cp –p bin_archive/daqd-2.10.1 daqd
cp –p bin_archive/nds-2.10.1 daqd
sudo /etc/init.d/monit start
```

8.4.6 GDS broadcaster (BCST)

The GDS frame re-broadcaster needs 'daqd' built using 'bcst'.

Login as 'controls' on the GDS broadcaster machine, then

```
cd /opt/rtcds/llo/l1/daqbuild/current
make bcst
cp –p build/bcst/daqd /opt/rtcds/llo/l1/target/l1daqgds0/bin_archive/daqd-
2.10.1
sudo /etc/init.d/monit stop
sudo /etc/init.d/daqd_gds0 stop
cd /opt/rtcds/llo/l1/target/l1daqgds0
cp –p bin_archive/daqd-2.10.1 daqd
sudo /etc/init.d/monit start
```

9 Tuning DAQ processes

The DAQ daemon 'daqd' is a multi-threaded application. Not all instances will have all threads, but only those thread it requires (and whose creation is typically initiated by a client request line). For RCG 2.10, we have added the ability to put unique labels on every thread so they can be followed using 'top' or a similar process.

9.1.1 Thread prioritization, CPU affinity options

For RCG 2.10, we updated the daqd routines to centralize where parameter on thread prioritization and CPU affinity are set to some #defines in 'daqd.hh'. Here is a sample

```
/// Define real-time thread priorities (mx receiver highest, producer next,
frame savers lowest)
#define MX_THREAD_PRIORITY 10
#define PROD_THREAD_PRIORITY 5
#define SAVER_THREAD_PRIORITY 2
#if defined(USE_BROADCAST)
#define PROD_CPUAFFINITY 1
#define FULL_SAVER_CPUAFFINITY -1
#define SCIENCE_SAVER_CPUAFFINITY -2
#define SECOND_SAVER_CPUAFFINITY -3
#define MINUTE_SAVER_CPUAFFINITY -4
#else
#define PROD_CPUAFFINITY 0
#define FULL_SAVER_CPUAFFINITY 0
#define SCIENCE_SAVER_CPUAFFINITY 0
#define SECOND_SAVER_CPUAFFINITY 0
#define MINUTE_SAVER_CPUAFFINITY 0
#endif
```

The MX threads (DC, FB) have highest real-time scheduler priority, followed by producer thread (all), then the frame saver threads (SA, FB, FW). We have only tried to use CPU affinity for DAQ daemons that receive broadcasts (FW, NDS, BCST) over the DAQ-OUT network from a data concentrator. A positive affinity is the offset from CPU 0, negativity affinities count done from the highest indexed CPU. 0 means no affinity.

These can be modified as needed for your case. They are still in a bit of flux. Note that currently the thread priorities will not work on the non-DC Gentoo 2.6.35 machines.

9.2 Data Concentrator daqd threads

The data concentrator is mostly mx threads (dqmx##) to get data from front-ends on the FE-DAQ network, a producer thread (dqprod) to combine these into the circular buffer, and transmitter threads (dqxmit) to put them on the DAQ-OUT network

Use 'top –u root', use 'f' , 'j' to add the CPU, then 'H' to show threads. Here is an example

```
  PID USER       PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+   P COMMAND
 6431 root        0 -20 8022m 2.2g 395m S    7 18.9 670:24.71  7 dqxmit
 6432 root        0 -20 8022m 2.2g 395m S    7 18.9 594:43.43  8 dqxmit
 6433 root        0 -20 8022m 2.2g 395m S    1 18.9  81:17.01 11 daqd
 6467 root       -6 -20 8022m 2.2g 395m S    9 18.9 816:02.48  5 dqprod
 6468 root      -11 -20 8022m 2.2g 395m S    1 18.9  35:27.37  1 dqmx000
 6469 root      -11 -20 8022m 2.2g 395m S    1 18.9  38:30.05  3 dqmx001
 6484 root      -11 -20 8022m 2.2g 395m S    1 18.9  27:37.75 10 dqmx100
 6485 root      -11 -20 8022m 2.2g 395m S    0 18.9  36:25.05  2 dqmx101
 6501 root      -11 -20 8022m 2.2g 395m S 9999 18.9  21:59.54  4 dqmx000
```

The system generates a 'dqmx' thread for every possible combination of card (0-1) and endpoint (0-E) even if they aren't active. Note that the processes also have the 'nice' priority set to -20.

9.3 Framewriter daqd threads

On the frame-writer, the producer thread (dqprod) takes in data from DAQ-OUT to fill the circular buffers. There are threads to save full frames (dqfulfr), science frames (dqscifr), second trend frames (dqstrfr), minute trend frames (dqmtrfr) and raw minute files (dqmtraw). There are helper threads for the trends, a second trend consumer (dqstrco), worker (dqstrwk) and minute trend consumer (dqmtrco). There are also profiler threads, a main one (dqmain), trends (dqptrend) and raw minute (dqpmt). There are also listener threads on ports 8087, 8088 (dql8087, dql8088). Here are the main ones in action

```
  PID USER       PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+   P COMMAND
 1220 controls    0 -20 15.0g  11g 2264 S  0.0 64.0   0:00.01  6 daqd
 1396 controls    0 -20 15.0g  11g 2264 S  0.0 64.0   0:00.68  3 dqpmain
 1397 controls   -3 -20 15.0g  11g 2264 R 99.9 64.0 166:07.24 11 dqfulfr
 1398 controls   -3 -20 15.0g  11g 2264 S  0.0 64.0   0:39.44  2 dqmtrco
 1399 controls    0 -20 15.0g  11g 2264 S  1.3 64.0   9:11.49  0 dqstrwk
 1400 controls   -3 -20 15.0g  11g 2264 S  1.3 64.0   9:43.42  4 dqstrco
 1401 controls   -3 -20 15.0g  11g 2264 S  0.3 64.0  10:04.51  9 dqstrfr
 1403 controls   -3 -20 15.0g  11g 2264 S  0.0 64.0   0:50.73  8 dqmtrfr
 1404 controls   -3 -20 15.0g  11g 2264 S  0.0 64.0   1:49.36  7 dqmtraw
 1405 controls   -3 -20 15.0g  11g 2264 R 99.9 64.0  96:13.43 10 dqscifr
 1406 controls    0 -20 15.0g  11g 2264 S  0.0 64.0   0:00.66  4 dqptrend
 1407 controls    0 -20 15.0g  11g 2264 S  0.0 64.0   0:00.68  0 dqpmt
 1409 controls    0 -20 15.0g  11g 2264 S  0.0 64.0   0:17.33  7 dqepics
 1408 controls   -6 -20 15.0g  11g 2264 S 14.3 64.0  95:08.08  1 dqprod
 1410 controls    0 -20 15.0g  11g 2264 S  0.0 64.0   0:00.00 10 dql8087
 1411 controls    0 -20 15.0g  11g 2264 S  0.0 64.0   0:00.00  9 dql8088
```