Mark 6 Utility Programs and Commands

Roger Cappallo

Last update: 2015-10-09

Introduction

There are a number of utility programs which facilitate use of the mk6, making it more straightforward and robust. Also, there are some Linux commands that have particular applicability for operating a mk6 system, both at a site and a correlator. Since some of the utilities may be employed on machines other than a mk6 (e.g. *dqa*), where possible there is no longer any dependency on the pf_ring library, or other special-purpose software. This document details the usage of the utilities.

Table of Contents

da-client

da-client –h <machine> - where *<machine>* is the target Mark 6 system (defaults to localhost). It is a small, standalone program with a simple operator interface that allows commands to be sent and responses to be received from *cplane. cplane* must be running on *<machine>*.

dboss

dboss is a user interface client for dplane. It provides limited, but direct control of dplane for recording data, without the need for c-plane to be running. Executing dboss without any arguments will produce the following help menu:

```
rjc@gibbs: dboss
Usage:
dboss a <drain_buffers>                  -- abort
dboss c <start_cntl> <dur> <time>        -- capture data
  where <start_cntl> is 0|1|2|3|4 for immed.|wall-clock|vex|vdif|mk5
        <dur> is duration in secs
        <time> is start time in appropropiate format
dboss f <bit-mask>                       -- abandon files by bit #
dboss i                                  -- request module information
(NYI)
dboss n <vdif|mk5b> <1..4 input devices> -- defines new stream(s)
dboss o <sg|raid> <filename> <sg-group>  -- specifies output file(s)
dboss s <interval secs>                  -- request status
dboss t [0|1]                            -- terminate dplane
  where 0|1 is do_not|do force termination without buffer drain
rjc@gibbs:
```

dpstat

dpstat monitors *dplane* UDP messages and displays status information (only). It is typically run in its own window, where it monitors the current activity of dplane.

dqa

dqa is a data quality analyzer program. It is quite simple, yet capable, and it can be run on any vdif or mk6 data file. *dqa* provides information about the file format and its contents. It can prove very useful in determining what thread_id's are present within a file, its time range, etc. As with many of the utility programs, just running dqa without any arguments will provide a usage list:

```
rjc@gibbs: dqa
Usage: dqa <file_name> to get quality summary
 ~or~  dqa <file_name> <#packets> for detailed report
 ~or~  dqa -d <file_name> to also de-thread the file
```

```
rjc@gibbs:
```

Providing just the single argument of a file name to *dqa* will cause the whole file to be read and summarized:

```
rjc@gibbs: dqa gat1438_0.vdif

opening gat1438_0.vdif
vdif file
packet payload size 8224
read 7710000000 bytes
time span 12753503:0 --> 12753532:31249
legacy mode FALSE
vdif epoch 29
#chans 16
vdif version 1
station ID adb
#bits/sample 2
complex mode
total   data rate 2056.0 Mb/s
channel data rate  128.5 Mb/s

thread                                0
number of packets (by thread):     937500
number of packets out of order:       0
number of second jumps:               0
number of invalid/fill-frame:         0
rjc@gibbs:
```

Most fields are self-explanatory. Note that the time range given is in the vdif system – seconds after the 6-month epoch that is listed, followed by the frame # within the second. A quick look, without reading the whole file, can be had by restricting the number of frames read, as specified by a 2nd argument. In this mode additional debug printout is offered, such as the block #'s that are read if one is in scatter-gather mode. The printed information can help one to find errors in the format of data, if they exist.

One other mode of running dqa is to specify the de-threading option via the –d flag. This mode is used when a file contains data from multiple threads, and one wishes to separate it into separate files for each thread. The names of the de-threaded files are created automatically from the input file name, by appending _n for thread id n. In addition to the thread separation, the usual *dqa* summary is also created.

dspeed

dspeed is a rudimentary program to test disk performance by writing a number of 10 MB blocks from memory to a single file. It defaults to a total file size of 10 GB, which can be overwritten by the 2nd parameter. It is a useful utility for checking the raw speed of a single disk.

```
rjc@gibbs:~$ dspeed
Usage: dspeed <filename> [<file size (GB)]
rjc@gibbs:~$
```

gather

gather reassembles scan data from a scatter-gather file system, which is a group of files originally written by the mk6. It uses threading and circular buffers for each thread in order to enhance performance. It is run by specifying file names for n input files, and one output file. The –o flag before the output file name is required as a safety measure, to keep one from accidently over-writing an input file (if the output file name was omitted).

```
rjc@gibbs: gather
Usage:
gather <ifile1> ... <ifilen> -o <ofile>
rjc@gibbs:
```

For example:

```
gather /mnt/disks/*/*/data/scan_xyz.vdif –o /raid_array/scan_xyz.vdif
```

gather416

gather416 is a variant of gather. In addition to the usual re-assembly of a single vdif file from the scatter-gather files it merges together multiple threads into a single thread having more channels.

gather416 is a de-threading version of gather. It reassembles a single file from a group of files originally created by *dplane*. Additionally, it splices together on a sample-by-sample basis contemporaneous data taken on different threads. This program creates a single-thread x 64-channel vdif file from multiple 4-thread x 16-channel files. For performance reasons, a separate reader thread is created for each input (scattered) disk file. The real samples from the various threads are merged together into a single thread with more channels per sample.

By way of the optional –t flag, one can specify non-standard thread id's to be used. Normally, thread id's are 0, 1, 2, or 3. Thread 0 is put into the lowest 16 channels of the output sample word, thread 1 into the next lowest 16 channels, and so on. If, for example, the recording was made using thread id's 11, 12, 21, and 22, then the five arguments –t 11 12 21 22 should be inserted in the command line.

```
rjc@gibbs:~$ gather416
Usage:
gather416 <ifile1> ... <ifilen> -o <ofile>
~or~
```

```
gather416 -t <thr0> <thr1> <thr2> <thr3> <ifile1> ... <ifilen> -o
<ofile>
rjc@gibbs:~$
```

gather464

gather464 is another de-threading version of gather, but in this case for complex samples, whereas *gather416* dealt with real samples. It reassembles a single file from a group of files originally created by *dplane*. Additionally, it splices together on a sample-by-sample basis contemporaneous data taken on different threads. This program creates a single-thread x 64-channel vdif file from multiple 4-thread x 16-channel files, where each 16-channel file contains complex data and has 64 bit sample words. For performance reasons, a separate reader thread is created for each input (scattered) disk file. The complex samples from the various threads are merged together into a single thread with more channels per sample.

By way of the optional –t flag, one can specify non-standard thread id's to be used. Normally, thread id's are 0, 1, 2, or 3. Thread 0 is put into the lowest 16 channels of the output sample word, thread 1 into the next lowest 16 channels, and so on. If, for example, the recording was made using thread id's 11, 12, 21, and 22, then the five arguments –t 11 12 21 22 should be inserted in the command line.

```
rjc@gibbs: gather464
Usage:
gather464 [-v] <ifile1> ... <ifilen> -o <ofile>
~or~
gather464 [-v] -t <thr0> <thr1> <thr2> <thr3> <ifile1> ... <ifilen> -o
<ofile>
rjc@gibbs:
```

gator

gator is a python script/program that accesses (potentially multiple) Mark 6 scatter-gather file sets, re-assembles data as necessary, and creates output file(s) on a destination fileserver (often RAID). It makes multiple calls to gather, when necessary, in order to accomplish the work. It handles automatic mounting of groups, if necessary, and returns the mk6 system to the state in which it found it. For example, if a group is mounted by *gator*, then it is unmounted when the program concludes. c-plane must be running to allow *gator* to mount groups.

```
rjc@gibbs: gator -h
Usage:
  gator [options] <group> "<wild_carded_input_files>"
<destination_path>
```

```
  gathers multiple m6 filesets and writes them to raid
  note that double quotes are needed around the input file expression
  iff the expression contains *, [], or ? symbols
  example:   gator.py 12 "scan01[0-3].m5b" /data/2345/westford/

Options:
  -h, --help            show this help message and exit
  -f, --force           force overwrite of duplicate output files
(false)
  -i HOST, --ip=HOST    mk6 host ip (127.0.0.1)
  -p PORT, --port=PORT  mk6/cplane port (14242)
  -q, --quit            quit if an error is encountered (false)
  -v, --verbose         verbose mode (false)
rjc@gibbs:
```

modspeed

modspeed is a Python program that sets up a module and runs *scatspeed* to measure disk performance, with a minimum of user typing. It requires c-plane to be running, in order to auto-mount the module when necessary.

```
rjc@gibbs: modspeed -h
Usage: modspeed [options]
  measures the write speed of a mk6 module

Options:
  -h, --help            show this help message and exit
  -g GB, --gb=GB        gigabytes to write (60)
  -i HOST, --ip=HOST    mk6 host ip (127.0.0.1)
  -p PORT, --port=PORT  mk6/cplane port (14242)
  -s SLOT, --slot=SLOT  module slot (1)
  -v, --verbose         verbose mode (false)
rjc@gibbs:
```

scatspeed

scatspeed is a *C* program to test multiple-disk scatter write performance on a mounted module. It uses the same file-writing routines as used in *dplane*, but with dummy data just read out from memory. By explicitly naming the files to be written, including their full path, one can measure true performance on mounted mk6 modules.

```
rjc@gibbs: scatspeed
Usage: scatspeed <file size (GB)> <file1> <file2> ... <file n>
       number of files is variable (1..32)
rjc@gibbs:
```

For example, the command:

```
scatspeed 100 /mnt/disks/1/0/data/test.vdif
/mnt/disks/1/1/data/test.vdif /mnt/disks/2/4/data/test.vdif
/mnt/disks/2/5/data/test.vdif
```

would write 4 files, 2 on the module in slot 1, and 2 on the module in slot 2. More typically (albeit tediously), one would specify all 8 disks per slot for 1, 2, or 4 slots.

m6-erase

m6-erase is a standalone disk-erase and disk-conditioning program