Online Analysis Processes

Shared memory servers

Each shared memory server mirrors the data targeted for one of the *dss* recording nodes. Shared memory servers can be run on multiple nodes to receive the data distributed to the multiple recording nodes. For example, if 3 *dss* nodes are used for recording data, 3 *mon* nodes can each run a shared memory server to collectively receive all the data.

Analysing the data may require several parallel processes running on a given *mon* node. The shared memory server can distribute events to each of those clients in a round-robin fashion. Assuming there are 4 client analysis processes on each node, then the monshmserver command lines would look like:

```
daq-amo-mon01:  monshmserver -i 0x1 -c 4 -d -p 0 -P AMO
daq-amo-mon02:  monshmserver -i 0x2 -c 4 -d -p 0 -P AMO
daq-amo-mon03:  monshmserver -i 0x4 -c 4 -d -p 0 -P AMO
  -i <mask>  : bit mask of event nodes to mirror
  -c <ncl>   : number of shared memory clients to serve
  -d         : distribute events round-robin
  -p <plat>  : DAQ platform number (event data registration)
  -P <name>  : base name for shared memory and message queues
               Shared memory name becomes <plat>_<mask>_<name>
```

Monitoring servers

The *ami* process analyses event data from the shared memory server and produces display data (plots) for the user display. Processing the data may require several parallel threads of execution to analyse a useful fraction of the data, so multiple instances of the *ami* process may be launched for each shared memory server. For example, the command lines to launch 4 *ami* processes against the shared memory servers in this example would look like:

```
daq-amo-mon01:  ami -n 0 -P 0_1_AMO -i lo -s <group1>
daq-amo-mon01:  ami -n 1 -P 0_1_AMO -i lo -s <group1>
daq-amo-mon01:  ami -n 2 -P 0_1_AMO -i lo -s <group1>
daq-amo-mon01:  ami -n 3 -P 0_1_AMO -i lo -s <group1>

daq-amo-mon02:  ami -n 0 -P 0_2_AMO -i lo -s <group2>
daq-amo-mon02:  ami -n 1 -P 0_2_AMO -i lo -s <group2>
daq-amo-mon02:  ami -n 2 -P 0_2_AMO -i lo -s <group2>
daq-amo-mon02:  ami -n 3 -P 0_2_AMO -i lo -s <group2>

daq-amo-mon03:  ami -n 0 -P 0_4_AMO -i lo -s <group3>
daq-amo-mon03:  ami -n 1 -P 0_2_AMO -i lo -s <group3>
daq-amo-mon03:  ami -n 2 -P 0_2_AMO -i lo -s <group3>
daq-amo-mon03:  ami -n 3 -P 0_2_AMO -i lo -s <group3>

  -n <client> : client index to shared memory server
  -P <tag>    : shared memory and message queue name
  -i <if>     : network interface for ami communication
  -s <group>  : multicast group for ami server/client discovery
```

Collection

The results of each *ami* process need to be assembled before display to the user. To distribute the load for that work, a process called *ami_collection* serves as an intermediary between the display process and a subset of the servers. As such, the *ami_collection* process forwards analysis requests from the user GUI to the *ami* servers, assembles the individual results from its subset of servers, and forwards the aggregate result to the user display. The corresponding command lines for our example would look like:

```
daq-amo-mon01:  ami_collection -i lo -s <group1> -I eth0 -S <group_top>
daq-amo-mon02:  ami_collection -i lo -s <group2> -I eth0 -S <group_top>
daq-amo-mon03:  ami_collection -i lo -s <group3> -I eth0 -S <group_top>
  -i <if>    : network interface for aggregated subset (towards _ami_)
  -s <group> : ami server/client discovery multicast for aggregated subset
  -I <if>    : network interface for aggregating superset (towards GUI)
  -S <group> : ami server/client discovery multicast for aggregating superset
```

Proxy

Discovery of the set of available *ami* servers occurs within a multicast group. If the display GUI process runs on a host that does not have an interface on the subnet which carries the multicast group, the *ami_proxy* process can run on such a host and maintain a list of available servers. Thus, it facilitates all the TCP connections necessary to communicate between the display GUI and all the servers. Note that the proxy does not see any of the event data, so it has a minimal load. An example might be:

```
daq-amo-mon01:  ami_proxy -I eth0 -i eth2 -s <group_top>
  -I <if>    : public interface
  -i <if>    : FEZ subnet interface
  -s <group> : top level server/client discovery multicast group
```

GUI

Finally, the display GUI process *online_ami* can connect to the server network by calling upon the proxy. The *online_ami* process reads the user analysis requests from the GUI, forwards those requests to the servers, and does the final result collection at the top level of this hierarchy. In this case:

```
amo-daq:  online_ami -I eth0 -i eth0 -s daq-amo-mon01
  -I <if>    : TCP interface
  -i <if>    : discovery interface (optional)
  -s <group> : ami discovery group or proxy address
```

Other *online_ami* processes may be simultaneously launched to perform independent analysis of the data using the same set of server processes.