cman 2.00 and OpenAIS

Version history

0.1 24th April 2006	First draft
0.2 26th April 2006	Fix some typos
0.3 11th October 2006	Add totem config example
0.4 10th November 2006	Add information on “Disallowed” nodes
0.5 4th November 2007	Add recovery information for “Disallowed” nodes, and a few general updates

Introduction

This document describes how cman and its components in Red Hat Cluster Suite use and interact with OpenAIS and highlights some of the important differences between the kernel-based cman in cluster suite 4 and the new one which runs in userspace. It also tries to explain why these changes have been made and the advantages that should accrue.

It will touch on things like the DLM which are affected by the changes but it is not a whitepaper on the new DLM. I will not mention GFS (except I just did, damn) or rgmanager as others are far better placed to do so than I. (oh, and I apologise in advance for overusing brackets).

The old stuff

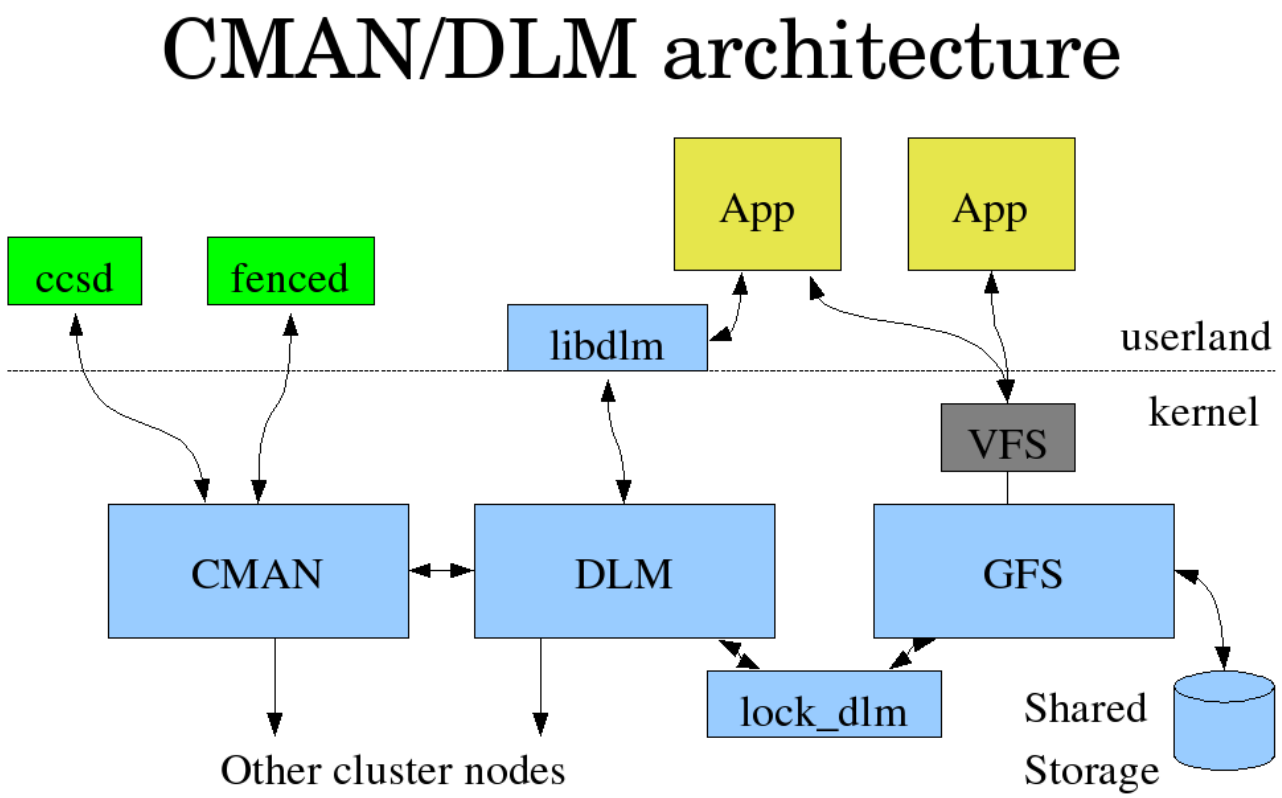
The cman code in RHEL4 (hereinafter known as “old cman” [can you tell I studied law?, no? probably just as well.]) is a kernel module that is split into two distinct parts. There is CMAN itself which provides a reliable communications layer using UDP multicast/broadcast and membership services for the cluster as a whole, and there is SM (Service Manager) which manages service groups. These can be thought of as sub-clusters which are created for particular services. So, for this section only, “cman” refers to the cman.ko kernel module (that incorporates CMAN & SM) and “CMAN” refers to the cluster manager portion (without SM), clear?

The CMAN component has its own protocol (based on UDP) for communications between nodes – using broadcast/multicast where the destination is the whole cluster and unicast (but still UDP) for sending to a single node. This protocol is exposed to userland as a protocol family (PF_CLUSTER number 30) and other kernel services via an internal API. The membership and SM components both use this for their communications. The DLM does not – it uses TCP/IP. Although this protocol is

exported to userland it is not suitable for general sustained or bulk use as it uses a single channel for all "ports" including its own; as far as I know, only clvmd uses this protocol for intra-cluster communication. Anyone attempting to use this for anything more than moderate and intermittent data transfer is likely to suffer nodes leaving the cluster at irregular intervals – yes, it really is that bad.

CMAN exports its API via ioctls on a socket on the above-mentioned protocol. These are encapsulated in the libcman API. Any clients of CMAN should always use the library and not the ioctl interface. This interface includes such things as passing cman its configuration information at join time. Because it's in the kernel, cman does not keep a copy of the config information with it at all times and can't poll for changes in it so it needs to be told when things change – this is the purpose of the *cman_tool config* utility. SM also exports some functionality via ioctls on the same socket but these are not in the libcman API as they are not forward compatible.

Here's a picture showing how the elements of this "old" architecture fit together and which bits sit in user or kernel space. You really don't want to know (particularly not if you're my boss) how much effort it took to get this picture into OpenOffice (and, yes, it was originally drawn in OpenOffice).



The new stuff (and why)

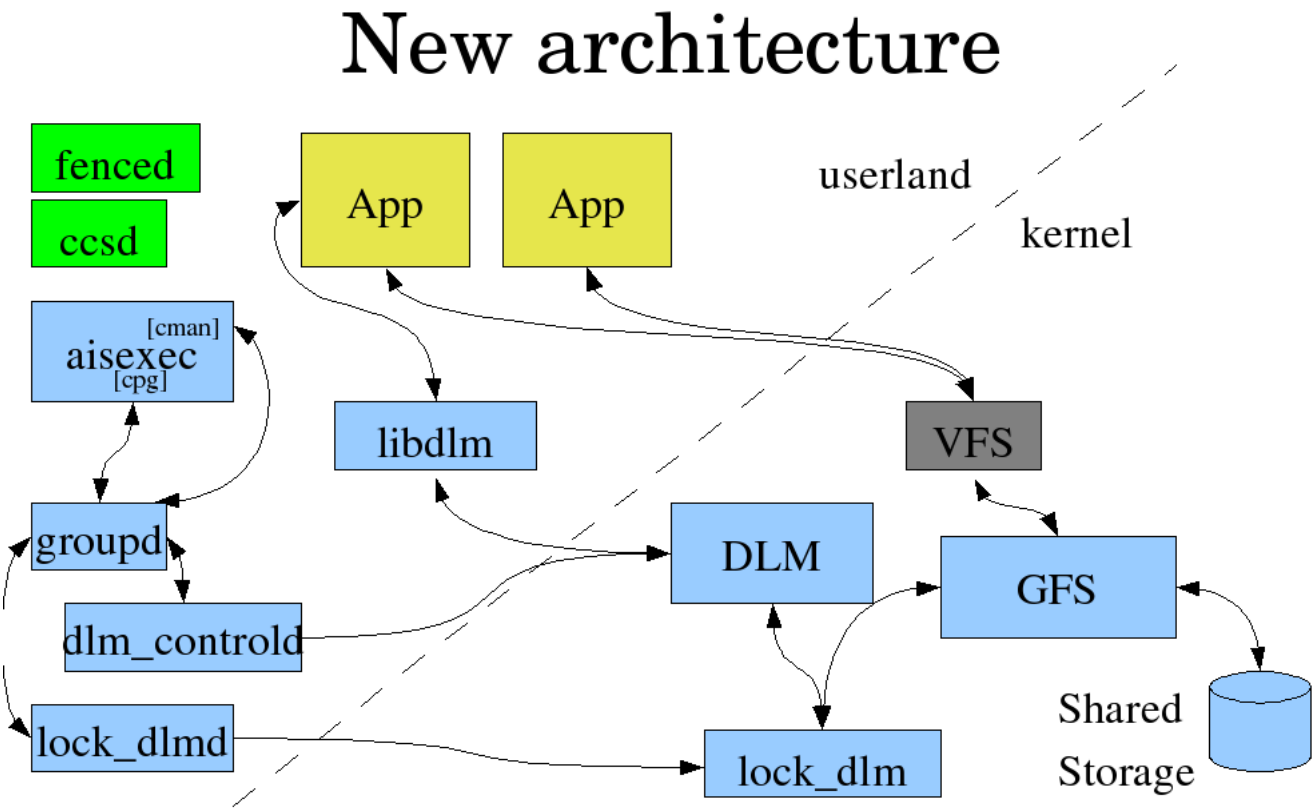
It became clear that there was no compelling reason for the cluster manager to live in the kernel. Kernel code is difficult to write/debug/maintain and update. Clients of the cluster manager only refer to it infrequently (eg for membership updates) so there is no real performance bottleneck for a service like the DLM (which *is* performance-critical, particularly when servicing a filesystem like GFS [which I said I wasn't going to mention]) referring back to userspace for information.

The first process I went through was simply to “port” old cman to userland. This worked fine as far as it went but provided nothing new to users, nor resolved any of the problems associated with the cman protocols, though it did highlight a few bugs in the kernel version that I hadn't spotted before!

I was quite struck with the virtual synchrony features of OpenAIS and wondered how hard it would be to make cman work in a similar manner. The answer was “very hard” as it turned out. So I decided that the obvious solution was actually to use OpenAIS as the communications layer for cman. This has the advantages that there are more people working on that code than just me, it works to a documented protocol and pulls in all the other features that OpenAIS has that are now available for customers.

Other kernel components have also been moved out into userspace. The SM component of cman has become a daemon called *groupd*, and there are daemons for controlling the (still in-kernel) DLM and GFS [aw, not again] mounts. This looks like a whole lot of new daemons but in fact all that's happened is that kernel threads have become userland daemons.

Another picture should (it had better, given the amount of effort it took) make this clear:



A quick introduction to OpenAIS

OpenAIS is a cluster manager in its own right. It has several subsystems that already provide membership/locking/events/communications services and other useful things. The external APIs for many of these are written to conform to Service Availability Forum standards. The core messaging system used is called “totem” and it provides reliable messaging with predictable delivery ordering.

The core of OpenAIS is the *aisexec* daemon. This is a modular executive into which the various services load. Shared libraries provide the APIs for client programs to talk to the services via named pipes. Configuration of the executive is usually done using configuration files /etc/ais/openais.conf but other configurator modules can be loaded that get their config information from other places.

For more information about OpenAIS visit the web site at http://openais.org/

The new components

The new cman is a service module that loads into aisexec. That means the other OpenAIS services are also available to users and cman itself can take advantage of the totem messaging system. So if you're looking for the cman process, remember it's now called *aisexec*.

As already mentioned, *groupd* replaces the service manager, and much of it's functionality is now provided by the CPG (Closed Process Groups) service inside OpenAIS.

dlm_controld is a daemon that connects to cman and groupd to manage the DLM groups. Those are the things you were shown by the command “cman_tool services”, and indeed you still will be. dlm_controld listens for node up/down events from cman and requests to create/destroy lockspaces and passes those into the DLM (which is still in the kernel) via configfs.

lock_dlmd is a daemon that manages the interaction between the DLM, groupd and GFS [eek, I said it again!]

clvmd is the same old clvmd we know and love (or hate, depending on your politics). Because it uses libcman to talk to the cluster manager, at least in later versions, you probably won't even need to recompile it. Simply replace the old libcman.so.x with the new one.

The two important libraries remain unchanged as far as most users are concerned. If you've been good girls and boys and linked your applications dynamically with them then things may even just work.

libcman provides the same API as before with some extras for good measure. You can conditionally compile these in by checking for LIBCMAN_VERSION == 2. But remember, the old libcman.h doesn't define this at all.

libdlm is also unchanged as far as most users are concerned provided your application is dynamically linked. There are some old DLM features not yet implemented in the new version but I hope they will turn up sooner or later.

So what does cman actually do then?

Good question. To some extent it is largely a compatibility layer for existing cluster-suite applications but it's also a configuration interface into CCS.

OK, calling it a compatibility layer is rather simplistic, the cman API is a nice simple API to use and provides, in one place, most of the things many applications need from a cluster manager. All the “get_info” calls are in one place and the messaging API is a model of simplicity, though I say so myself. Also useful is the “islistening” feature which keeps a track of which nodes have a cman client listening on a particular “port”. clvmd uses this to ensure that the daemon is running on all cluster nodes before changing any metadata.

cman also provides additional features such as quorum disk APIs, conditional shutdown, barriers and functions for managing quorum, which it also maintains.

Compatibility

At an API level, the new code is compatible with the old. In most cases simply replacing the libraries will make things work.

You cannot mix nodes running the old cman and the new. They speak very different protocols and will not see each other. In fact trying to run old and new code on the same UDP port on the same network could cause serious problems to one or the other so please don't try it. The default port for the new cman has been changed from 6809 to 5405 (the default OpenAIS port) to try and mitigate this.

cluster.conf files will be compatible except for one small change, see below.

User-visible changes

I've tried to keep the user and programmer visible changes to a minimum, but some annoying ones have been necessary and some new ones are nice.

The first thing you'll notice is that dead-node detection is *much* faster than before. This is a consequence of the OpenAIS totem token-passing protocol.

The output from cman_tool is formatted slightly differently, so if you were parsing it in a script or program then some small changes may be necessary. If you were naughty enough to look directly at files in /proc/cluster (not quite as dangerous as looking directly at the sun, but nearly as inadvisable) then you'll need to change your code to call cman_tool commands. The cman_tool -X switch is no longer supported. You *must* have a CCS config file for cman to work now.

All of which brings us neatly to:

Configuring the openAIS-based cman

The new userland openAIS-based CMAN can be configured in the same way as the old (kernel) one. It recognises the same main CCS tags as before with a few small differences.

I'll deal with it in sections. Remember all these sections are already subsections of `<cluster></cluster>`

<clusternodes>

This is fundamentally the same as before. The most important difference is that `nodeid=` is now a mandatory field. There is a new subcommand in ccs_tool (cunningly named *addnodeids*) to add these to an existing config file.

The `<altname>` tag is supported and the DLM can use it too, so it should now be useful. Now, altname specifies extra rings in the openAIS multi-ring protocol which is possibly subtly different to what you might expect. I'll leave Steven to document that...
The `<altname>` tag also allows you to specify the multicast name and UDP port to use for communications on each ring too. eg:

```
<altname name="mynode2" mcast="239.1.1.2" port="6810"/>
```

mcast and port are optional, the defaults for the first ring will be used if they are omitted.

cman now loads the node list into memory when it starts up (you'll notice that "`cman_tool nodes`" shows the full node list regardless of which nodes have or have not joined the cluster. If a new node joins the cluster with an updated config file then all nodes will automatically re-read the <clusternodes> section from CCS. This can be forced by running "`cman_tool version -r`"

Fencing options are the same as before.

<cman>

This section takes the following keys:

`quorum_dev_poll`	millisecond poll time for quorum devices
`shutdown_timeout`	millisecond timeout period to allow a service to respond during a shutdown
`debug_mask`	debuglog mask. (only useful if you know the sources!)

`debug_mask` can be overridden on the cman_tool command line using '`-d<n>`' This will also force the debugging output to go to stderr. If you want to customise the logging output more use the `<logging>` section as detailed in openais.conf(5) man page. It's worth mentioning here that in later versions of cman you can alter the debugging level at run-time, so that might help you debug any problems you might be having, though I hope you won't need it. Look for the `cman_tool debug` command and whoop for joy if you find it.

Others

In addition to these cman-specific sections, cman also loads in configuration for openAIS from

CCS from the following keys:

```
<totem>
<logging>
<event>
<aisexec>
<group>
```

See the openais.conf(5) man page for more information on keys that are valid for these sections. Note that the `<clusternodes>` section will overwrite things in these sections and options on the `cman_tool` command line will override both. In particular setting things like `bindnetaddr, mcastaddr, mcastport & nodeid` in this section will *always* be replaced by the values in `<clusternodes>`.

The cman/CCS component will try to read down the configuration hierarchy, but I'm not yet convinced it works well, as the CCS interface isn't really up to it.

openais.conf

cman does not use the openais.conf file at all. What it does do is load the values from CCS into internal OpenAIS data structures that map onto the entries to openais.conf. So that's why you can add an entry in a <totem> section in CCS and it will be used by totem. Refer to the openais.conf(5) man page for the openais-specific configuration keys.

So, those people who like to tweak heartbeat settings and suchlike, have some new concepts and new CCS keys to learn!

Well, maybe just a couple of examples then

Here's how to increase the token timeout to five seconds:

```
<totem token="5000"/>
```

And this is how to add extra AIS logging options to cman:

```
<logging to_stderr="yes">
  <logger ident="CPG" debug="on" to_stderr="yes">
  </logger>
  <logger ident="CMAN" debug="on" to_stderr="yes">
  </logger>
</logging>
```

If you then start cman using *cman_tool join -d* then you will get debugging output from the CMAN & CPG services on the terminal you are using.

Please don't just copy and paste those cluster.conf entries into your file. For a start, they have changed if you're using the openais from subversion trunk (at the time of writing, November 2007).

The new libcman

Externally, the new libcman looks very much like the old. There are a small number of extra API calls for new features. The first thing to notice is that there are two cman_init() calls. There is the old (hopefully) familiar cman_init() and a new cman_admin_init(). Any user can open a connection to cman using cman_init() but only privileged users can call cman_admin_init(). The protection here is achieved by having two named pipes with different permissions on them

Some calls require an admin socket to work. They are the ones that have “set” in the name mainly, cman_killnode()/cman_shutdown() and the quorum device calls. Admin sockets cannot send messages to other nodes using the messaging API, nor can they receive event notifications. This shouldn't be a problem as all the admin functions are one-shot operations. And it is possible to have two open sockets.

Most libcman calls are simple request/response calls. There are a few exceptions though. If a process registers for notification or data reception then it will get unsolicited data, this is delivered to libcman down the normal pipe and then passed onto the client using callbacks. Because of the request/response system you *cannot* call other libcman functions whilst in one of these callbacks as it will deadlock the process. The only call you can make whilst in a callback is cman_replyto_shutdown() which does not receive a response. (see below for more information on shutdown).

Another thing to be careful of with libcman is the file descriptor passed back from cman_get_fd(). It is vitally important *always* to call cman_get_fd() (and to use the result!) before calling select() or poll() on your file descriptors and you must never read that file descriptor yourself – always call cman_dispatch(). libcman does message caching and filtering so it might sometimes be the case that the actual pipe is not active, but the library has data for you squirrelled away. In this case it will pass you a file descriptor opened to /dev/zero so it will always be active.

See libcman.h for lots of information on the libcman API.

Getting it all going

So, here's a quick overview of what happens when you've got everything installed and configured and you finally, fingers all a-quiver, get to type in *cman_tool join.*

Not much; you forgot to press enter. Ah, here we go.

The first thing cman_tool does is to check that CCS is running. That's just a nicety really, cman_tool never reads anything from CCS any more. But it provides quicker feedback in the annoying case where you forgot to start ccsd beforehand.

After that cman_tool takes any command-line overrides and defines them as environment variables for cman to pick up when it starts. It then defines OPENAIS_DEFAULT_CONFIG_IFACE to "cmanconfig" then forks and execs the aisexec binary. That oddly named environment variable tells aisexec to use cman as its configurator, so that the config information gets read from CCS rather than openais.conf.

Now that aisexec has control, it tells cman to read CCS and populate its internal configuration database with the values found there. cman reads the openais-specific keys first (eg <totem>) and then the information about the local node from <clusternodes>. It then sets some internal values and adds the cman service handler to the list of services to load and passes control back to aisexec.

aisexec then loads up the service components it needs (including cman) and cman loads the full nodes list at this point. When it gets its first confchg message from totem it sends out a join message to the rest of the cluster. The rest of the nodes don't actually do much with this, but they do check the version number of cluster.conf that the new node started with and if it's higher than the one they are using then they will re-read it from CCS.

And that's about it really. The totem protocol notifies cman about node ups & downs and it does what it needs to do (including informing people listening for those events using libcman). libcman users who send messages to each other send them to cman, which adds on a nodeid and passes the messages around the cluster using totem.

All you need to do now is to start up your daemon farm and mount GFS volumes. [sigh, again. Ah well it was the last time.]

Shutdown

cman provides for a staged shutdown so that daemons can manage their state and control whether it is OK for cman to close down.

Shutdown is initiated by the command "cman_tool leave" which calls cman_shutdown(). (Note there is still a cman_leave() call which is deprecated; this call initiates an unconditional and immediate shutdown). On receipt of the shutdown command, cman sends all of its local clients (that are registered for notifications) a CMAN_REASON_TRY_SHUTDOWN notification and waits for responses. On receipt of this notification a daemon should then call cman_replyto_shutdown() with either a 1 (yes I will shutdown) or a 0 (no I don't want to shutdown). If a daemon does not respond in 5 seconds (default), cman assumes that the daemon does not want to shut down.

If all daemons agree that it is OK to shut down then cman will send a LEAVE message to all other cluster nodes with the reason for its shutdown (and whether they should reduce quorum). When this

message arrives back at its starting point, cman then shuts down, closing all of the client pipes.

Clients notice that cman has shutdown by either poll() returning POLLHUP on the FD or cman_dispatch returning -1 (with errno set to EHOSTDOWN).

If the clients refuse to let cman shutdown, it returns EBUSY to the caller of cman_shutdown() which can then go and sulk as it sees fit.

Disallowed Nodes

Occasionally (but very infrequently I hope) you may see nodes marked as “Disallowed” in cman_tool status or “d” in cman_tool nodes. This is a bit of a nasty hack to get around mismatch between what the upper layers expect of the cluster manager and OpenAIS.

If a node experiences a momentary lack of connectivity, but one that is long enough to trigger the token timeouts, then it will be removed from the cluster. When connectivity is restored OpenAIS will happily let it rejoin the cluster with no fuss. Sadly the upper layers don't like this very much. They may (indeed probably will have) have changed their internal state while the other node was away and there is no straightforward way to bring the rejoined node up-to-date with that state. When this happens the node is marked “Disallowed” and is not permitted to take part in cman operations.

If the remainder of the cluster is quorate the the node will be sent a kill message and it will be forced to leave the cluster that way. Note that fencing should kick in to remove the node permanently anyway, but it may take longer than the network outage for this to complete.

If the remainder of the cluster is inquorate then we have a problem. The likelihood is that we will have two (or more) partitioned clusters and we cannot decide which is the “right” one. In this case we need to defer to the system administrator to kill an appropriate selection of nodes to restore the cluster to sensible operation.

The latter scenario should be very rare and may indicate a bug somewhere in the code. If the local network is very flaky or busy it may be necessary to increase some of the protocol timeouts for OpenAIS. We are trying to think of better solutions to this problem.

It's worth noting here that a similar problem can occur with “old” cman. If the network splits a cluster precisely into two halves and then recovers there will be two inquorate clusters in a stand-off; neither daring to fence the other. This problem is very uncommon and easily avoided by having an odd number of nodes in the cluster so there will always be a quorate “half” to do the dirty deed of fencing the others.

Recovering from this state can, unfortunately, be complicated. Fortunately, in the majority of cases,

fencing will do the job for you, and the disallowed state will only be temporary. If it persists, the recommended approach it is to do a *cman tool nodes* on all systems in the cluster and determine the largest common subset of nodes that are valid members to each other. Then reboot the others and let them rejoin correctly. In the case of a single-node disconnection this should be straightforward, with a large cluster that has experienced a network partition it could get very complicated!

Example:

In this example we have a five node cluster that has experienced a network partition. Here is the output of *cman_tool nodes* from all systems:

```
Node  Sts   Inc   Joined               Name
   1   M   2372   2007-11-05 02:58:55  node-01.example.com
   2   d   2376   2007-11-05 02:58:56  node-02.example.com
   3   d   2376   2007-11-05 02:58:56  node-03.example.com
   4   M   2376   2007-11-05 02:58:56  node-04.example.com
   5   M   2376   2007-11-05 02:58:56  node-05.example.com

Node  Sts   Inc   Joined               Name
   1   d   2372   2007-11-05 02:58:55  node-01.example.com
   2   M   2376   2007-11-05 02:58:56  node-02.example.com
   3   M   2376   2007-11-05 02:58:56  node-03.example.com
   4   d   2376   2007-11-05 02:58:56  node-04.example.com
   5   d   2376   2007-11-05 02:58:56  node-05.example.com

Node  Sts   Inc   Joined               Name
   1   d   2372   2007-11-05 02:58:55  node-01.example.com
   2   M   2376   2007-11-05 02:58:56  node-02.example.com
   3   M   2376   2007-11-05 02:58:56  node-03.example.com
   4   d   2376   2007-11-05 02:58:56  node-04.example.com
   5   d   2376   2007-11-05 02:58:56  node-05.example.com

Node  Sts   Inc   Joined               Name
   1   M   2372   2007-11-05 02:58:55  node-01.example.com
   2   d   2376   2007-11-05 02:58:56  node-02.example.com
   3   d   2376   2007-11-05 02:58:56  node-03.example.com
   4   M   2376   2007-11-05 02:58:56  node-04.example.com
   5   M   2376   2007-11-05 02:58:56  node-05.example.com

Node  Sts   Inc   Joined               Name
   1   M   2372   2007-11-05 02:58:55  node-01.example.com
   2   d   2376   2007-11-05 02:58:56  node-02.example.com
   3   d   2376   2007-11-05 02:58:56  node-03.example.com
   4   M   2376   2007-11-05 02:58:56  node-04.example.com
```

```
5   M   2376   2007-11-05 02:58:56  node-05.example.com
```

In this scenario we should kill the node node-02 and node-03. Of course, the 3 node cluster of node-01, node-04 & node-05 should remain quorate and be able to fenced the two rejoined nodes anyway, but it is possible that the cluster has a qdisk setup that precludes this.

Summary

So, what have we gained from all this work? Quite a a lot I hope, most of which has been touched on already.

- Moving to userspace makes the system easier to maintain and less likely to cause kernel crashes if it fails.
- Faster node failure detection
- Availability of other OpenAIS services to cluster suite users
- Larger development community
- Better, published cluster protocol
- Better scalability with multi-ring system
- Encrypted communication channels
- Much better inter-node communications for cman clients