

RED HAT :: NASHVILLE :: 2006

SUMMIT



Cluster Infrastructure

David Teigland

# Introduction

- openais provides a solid cluster membership and messaging foundation
- how do we use that to manage GFS, DLM and fencing?
- how we configure and control the cluster from the command line?
- what does the DLM do?
- what does fencing do?



# RHEL5 changes

- merge kernel systems upstream: GFS, DLM
- improve performance: GFS, DLM
- move cluster infrastructure to userland: CMAN
- move bits of GFS, DLM to userland
- adopt good bits from community: openais

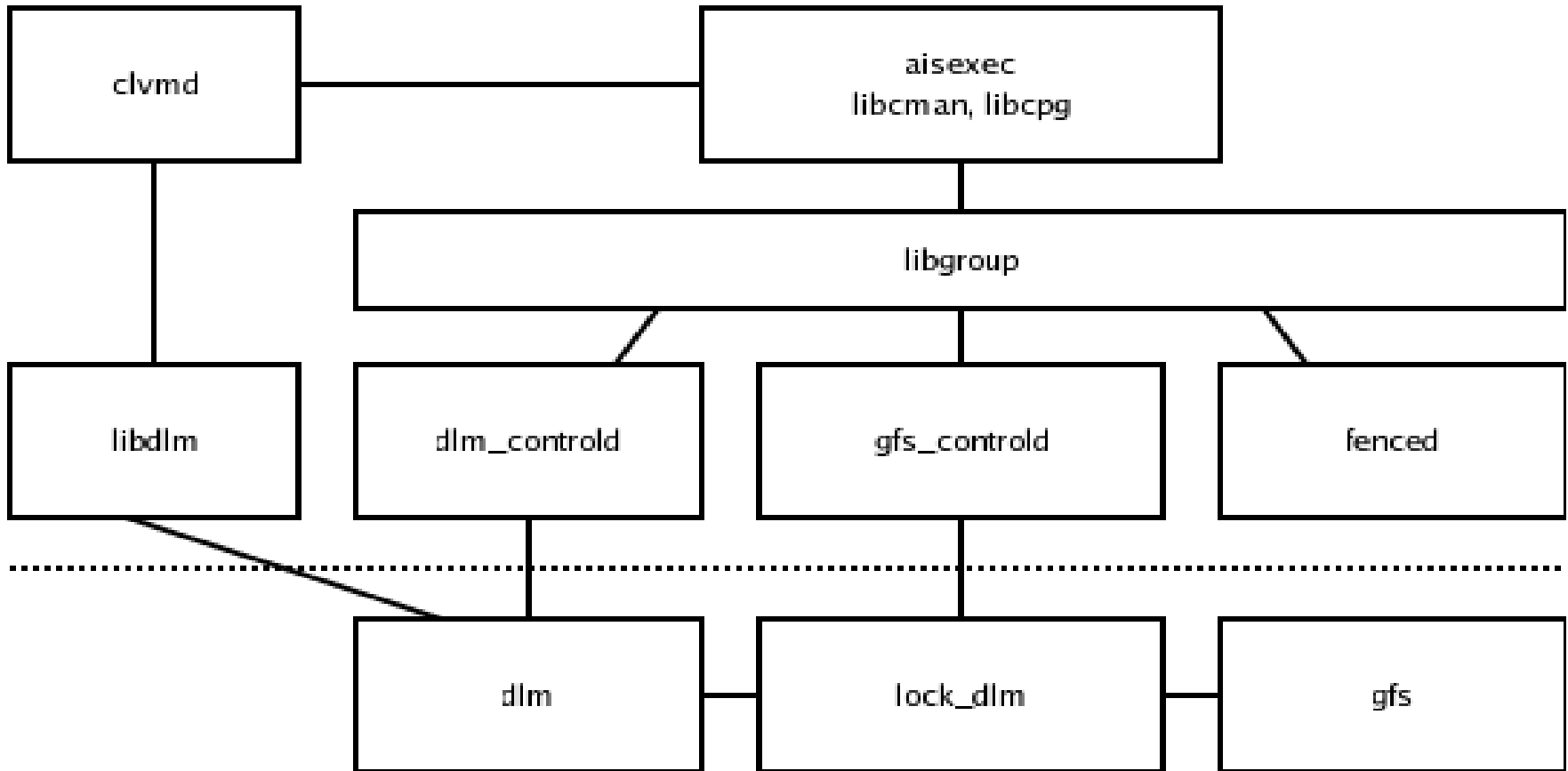


# Components

- openais: core membership & group communication
- CPG: openais service provides “closed process group” support
- CMAN: openais service for cluster configuration, management, quorum
- fenced: creates/joins group where members are fenced if they fail
- dlm\_controld: configures dlm, applies member events to dlm (recovery)
- gfs\_controld: configures gfs, applies member events to gfs (recovery)
- groupd: libgroup is common code factored from previous three for dealing with cpg events asynchronously



# Component Diagram



# openais advantages

- failures don't cause kernel crashes and are easier to debug
- faster node failure detection
- other openais services now available
- larger development community
- advanced, well researched membership/messaging protocols
- better scalability with multi-ring system
- encrypted communication possible



# openais - closed process groups

- cpg is an openais service / API (libcpg)
- standard openais callbacks are relative to entire cluster
  - message delivery
  - configuration (membership) changes
- process groups: a process joins a named group
  - members identified as nodeid/pid pairs
  - callbacks relative to group
- same VS (virtual synchrony) properties on order of delivery and configuration change callbacks



# openais - using cpg

- cpg's are used by gfs\_controld, dlm\_controld, fenced to represent:
  - the group of nodes that currently have a specific gfs fs mounted
  - the group of nodes that is currently using a dlm lock-space
  - a group of nodes that will fence each other
- mounting gfs file system foo corresponds to joining the cpg for foo, unmounting foo corresponds to leaving the cpg for foo
- gfs/dlm instances on nodes must agree about exactly who is in the gfs mount-group or dlm lock-space
- cpg's are the correct way to do this: agreed order of changes



# CMAN - Cluster Manager

- how we configure and control cluster manager (openais)
- an openais service
- configure cluster manager like we do in RHEL4: cluster.conf
- control/observe cluster manager like we do in RHEL4: cman\_tool
- query cluster manager like we do in RHEL4: libcman
- calculates quorum like we do in RHEL4 (like VMS clusters)



# CMAN - cluster.conf

- read file through libccs so we can change back-end (directory server)

```
<cluster name="alpha">
  <clusternodes>
    <clusternode name="red" nodeid="1">
      <fence>
        <method name="single">
          <device name="labapc" nodename="red"/>
        </method>
      </fence>
    </clusternode>
    <clusternode name="blue" nodeid="2">
      <fence>
        <method name="single">
          <device name="labapc" nodename="blue"/>
        </method>
      </fence>
    </clusternode>
  </clusternodes>
  <fencedevices>
    <fencedevice name="labapc" agent="fence_apc" ipaddr="labapc" login="admin"
      passwd="pw" />
  </fencedevices>
</cluster>
```



# CMAN - cman\_tool

- cman\_tool join
  - join the cluster
- cman\_tool leave
  - leave the cluster
  - fails if systems are still using the cluster
- cman\_tool status
  - local view of cluster status
- cman\_tool nodes
  - local view of cluster membership



# CMAN - libcman

- compatible with RHEL4 libcman
- not a standardized API
- nice, convenient cluster membership API
  - `cman_get_node_count()`, `cman_get_nodes()`, `cman_get_node()`,  
`cman_is_quorate()`, `cman_get_cluster()`, `cman_send_data()`
  - struct `cman_node` fields: `nodeid`, `address`, `name`, `member`, `jointime`



# CMAN - quorum

- majority voting scheme to deal with split-brain situations
- each node has configurable number of votes, default 1
  - `<clusternode name="foo" nodeid="1" votes="2">`
- sum of votes of current members is over half of possible votes =quorum
- cluster applications (and parts of infrastructure) will often only operate if the cluster has quorum
- fencing/dlm/gfs recovery only happens if the cluster has quorum
  - otherwise nodes in a partitioned, split-brain cluster would fence each other



# CMAN - two node cluster

- there is a cluster.conf “two\_node” setting that can be used when there are only two nodes in the cluster and power fencing is used
- quorum is disabled: one node has quorum, split-brain possible
- in gfs/dlm/fencing cluster, this is safe because in split-brain situation, both nodes race to fence each other before enabling gfs/dlm
- winner goes ahead to enable gfs/dlm, loser reboots
- this is a poor solution when there's a persistent network partition and both nodes can still fence each other: reboot/fence cycle



# CMAN - quorum disk

- aims to solve the problems with fencing race in two node cluster
- also gives us some new ways to affect quorum; reduce the number of nodes that must be running for cluster to have quorum
- qdisk: a daemon running on each node that does its own heartbeating to shared disk
- node status info on shared quorum disk lets the nodes defer to each other in split-brain cases instead of racing to take over
- quorum disk is configured in cluster.conf with a number of votes
- to cman, quorum disk operates like a voting node



# DLM changes

- largely rewritten for RHEL5 to address design problems (v1 was learning experience), simplify code, fix ugly corner cases in v1, improve performance, get in shape for upstream merging
- communication from kernel, point to point using SCTP
- node address/nodeid pairs passed from user space (dml\_control)
- new options: node weights, no resource directory
- API modeled after VMS
  - kernel API not widely used
  - libdml for userland cluster apps same as v1



# DLM internal concepts

- lockspace aka namespace for resources
  - when nodes join/leave lockspace recovery is done
- a resource is a named object that can be locked
- one node in the lockspace is “master” of the resource
  - other nodes need to contact this node to lock resource
  - first node to take lock on resource becomes its master (when using a resource directory)
- resource directory says which node is the master of a resource
  - divided across all nodes, rebuilt during recovery



# DLM library setup

- `handle = dlm_create_lockspace("myprog", mode);`
  - join lockspace
- `fd = dlm_ls_get_fd(handle);`
  - callbacks with poll/select
- `dlm_dispatch(fd);`
  - run your callback function
- `dlm_release_lockspace("myprog", handle, 1);`
  - leave lockspace (1 means force leave)



# DLM library locking

- `dlm_ls_lock()` parameters, [`dlm_ls_lock_wait()` has no `astaddr/astarg`]
  - `dlm_lshandle_t handle` : lockspace
  - `uint32_t mode`: EX, SH, etc
  - `struct dlm_lksb *lksb` : passes result, lockid, lvb
  - `uint32_t flags` : CONVERT, NOQUEUE, etc
  - `const void *name` : resource name to be locked
  - `unsigned int namelen` : length of name
  - `uint32_t parent` : parent lock (not yet in v2)
  - `void (*astaddr) (void *astarg)` : completion callback function
  - `void *astarg` : caller's arg for callback functions
  - `void (*bastaddr) (void *astarg)` : blocking callback function



# DLM lock modes

- Null (NL), Concurrent Read (CR), Concurrent Write (CW), Protected Read (PR), Protected Write (PW), Exclusive (EX)
- Compatibility table

	<b>NL</b>	<b>CR</b>	<b>CW</b>	<b>PR</b>	<b>PW</b>	<b>EX</b>
<b>NL</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
<b>CR</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>
<b>CW</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>PR</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>
<b>PW</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>EX</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

# DLM to do

- orphan locks and purging
- parent/child hierarchical locks
- lock groups - locks owned by group instead of process
- lock request timeouts
- LOCAL and FINDLOCAL requests (use with caution)
- atomic counter API
- user space distributed deadlock detection?



# Fencing

- fencing is primarily based on the requirements of gfs, but it's still general enough to satisfy requirements of other systems
  - gfs: we need to fence a node when we're not sure of its kernel state and we want to recover its journal
- if we conclude incorrectly that a node is in a clean state (not hung), recover its journal, the node later revives and writes metadata to the fs, then the fs is corrupted, so...
  - need to fence a node after the membership manager tells us the node has failed and before we recover its journal
  - also need to fence a node when cluster is first formed and we can't find out its current state



# Fencing agents

- scripts that interact with devices to fence a node (power off, SAN off)
- fencing daemon is told a node has failed, it looks up what agent can be used to fence it in cluster.conf, and runs that agent
- usually not difficult to write an agent to interact with a new device
- parameters for using the device entered in cluster.conf
- current agents: apc, baytech, bladecenter, brocade, bullpap, cpint, drac egenera, ibmblade, ilo, impilan, manual, mcdata, rackswitch, rps10, rsa, rsb, sanbox2, vixel, vmware, wti, xcat, xen, zvm
- to do: agent to use persistent group reservations (SCSI-3)

