Cray SV1 SuperCluster System Resiliency

Mike Wolf

1 Overview

Clustering of computer systems brings unique challenges for resiliency. This paper discusses those challenges, and what SGI is doing to address them. While the features discussed in this paper apply to any shared GigaRing channel environment, this paper will focus on the Cray SV1 SuperCluster system.

2 Goals

The first goal of cluster resiliency is to maintain cluster operation when one node becomes unavailable. This goal can be subdivided into two categories: one is to correct problems with nodes as soon as possible, the second is failover. With the loss of a node the cluster is degraded in compute power. Depending on the configuration, jobs on that node could be aborted. New jobs can continue to be submitted, and network access to all running nodes will not be disrupted

A second goal is to recover a node as quickly as possible. This is called "Auto Recovery".

3 Ring Resiliency

The first step in cluster resiliency for the Cray SV1 SuperCluster system is making sure that the GigaRing channel stays stable and that no other node on the cluster panics or hangs as a result of another node going down. When a node goes down on a shared ring, packet backups can occur and eventually take down every node on that shared ring. To prevent this from happening, three steps have been taken:

- UNICOS and UNICOS/mk operating system resetting the client chip on panic.
- check*xxx* resets the client chip on a hang or panic of mainframe.
- Proxy locking on ION kernels.

These steps are discussed in detail in the following sections.

3.1 UNICOS and UNICOS/mk Operating System resetting client chip on Panic

In the UNICOS/mk operating system, the client chip is set to discard GigaRing packets once a mainframe panics. This capability is being added to the UNICOS operating system as well. This feature clears the client chip after the operating system panics and is no longer able to process GigaRing packets. This keeps the node chip on the mainframe from backing up with unprocessed packets. This, in turn, keeps other nodes on the GigaRing from backing up.

3.2 check*xxx* resets the client chip on hang or panic of the mainframe

The check*xxx* commands are being modified to initialize the client chip when a hang or panic is detected. If the site is running a version of the operating system that clears the client chip, it will be initialized again. This doesn’t impact performance and makes sure that the client initialization is done on hangs as well. The checkxxx commands are necessary because the operating system cannot detect when it is hung and automatically do the clear. The client initialization performed by the check*xxx* commands allows older versions of the operating system to benefit from this feature as well.

3.3 Proxy locking on ION kernels

Proxy locking was added to the ION kernels to allow more than one dring(8) command to be running simultaneously without colliding and causing ring problems. Basically, proxy locking controls access from the SWS to the GigaRing channel. The feature allows critical dring commands to be guaranteed access to the ring to perform operations.

For example, two users can run dring on the same ring, or dring can be run by cron(1) or hwmd(8) and also by a user. The dring(8) with the lower priority action waits for the higher priority one to finish, then the lower priority dring runs to completion.

4 Node Resiliency

Another new feature is the unattended operations or auto-recovery feature. This feature uses the state server and the hardware monitor daemons to watch over the system. When a mainframe hangs or panics, the state change is detected by new recovery and notification mechanisms.

The auto-recovery feature is built upon the foundation of the state server daemon, the hardware monitor daemon, and the dring command. The auto-recovery operation depends on hwmd(8) to periodically run commands to check the state of the system. These commands include checksv1(8) to monitor the mainframe for hangs and panics, checkion(8) to make sure the IONs are operating properly, and finally dring(8) to make sure that there are no ring problems. These commands are, by default, run once every minute. If checksv1(8) or checkion(8) detect problems, they stabilize the ring and then update the state server daemon. The state server daemon launches the notification and recovery processes. Auto-recovery is a series of commands and scripts that configure the actions to be taken for recovery. The hardware monitor, recovery, and notification programs are configured individually. Recovery and notification are independent of one another but depend on the hardware monitor updating the state server to initiate their actions.

For example, assume that one of the nodes in the Cray SV1 SuperCluster system panics. When the checksv1(8) command is run by the hardware monitor daemon, it detects the panic. The checksv1(8) command runs dring to stabilize the ring. The checksv1 command will then inform the state server of the state change. This is detected simultaneously by the notification and recovery daemons. The notification program sends out email that the node is in a panic state. The recovery daemon dumps the node’s domain and then reboots it to multiuser mode. Once the mainframe is rebooted, the notification daemon sends out email that the mainframe is available again.

5 Auto-Recovery command synopsis

The following is a discussion of the auto-recovery feature. It consists of the current monitoring code and two new sets of commands: recovery and notification. A site has a choice on whether or not to use recovery or notification, however, recovery and notification depend on monitoring. The monitoring code (hwmd(8)) checks the GigaRing channel and components and updates the state server, when the state of a component changes. This state change is what initiates recovery and notification, provided they are enabled.

5.1 Recovery

Recovery consists of a monitoring daemon that watches for state changes in the state server daemon, a control program that configures how recovery works, and a set of scripts that implement the recovery action. Recovery can be configured off; a site does not have to have recovery running in order to take advantage of monitoring or notification.

5.1.1 recd

The recd(8) daemon calls recovery commands that restarts the system in the event of a hang or panic. The daemon receives component and node updates from the state server. When it notices a state or setting that requires action, it calls the appropriate recovery script to restart the system and return it to a state where users can run jobs.

5.1.2 recdcontrol

The recdcontrol(8) command controls the recovery that is initiated by the recd(8) daemon. You can use it to indicate whether recovery should take place and the mainframe boot level.

If you invoke recdcontrol(8) without options, it displays the monitoring flag and the mainframe boot level. This information is derived from the state server. The state server is initialized to contain the following information:

- Recovery monitoring is off.
- Mainframe boot level is 3.
- Retry count is 1.

The recdcontrol command accepts the following arguments:

-a on|off	Turns recovery monitoring on or off for all IONs, all mainframes, and all rings. You cannot specify this option with the -p or -u option.
-b bootlevel	Sets the boot level to use during booting of the system. This value is passed to the recover*xxx* command, which uses it to reboot the mainframe. You cannot specify this option with the -p, or -u options.
-p	Displays information in a persistent mode. As control information changes, output is generated. The default is to display the current settings and exit. You cannot specify this option with the -a, -b, or -r option.
-r retries	Sets the number of times to retry booting the mainframe domain. By default, it will be 1. You cannot specify the option with the -p or -u option.

-u	Specifies that unformatted output is displayed. By default, the output is formatted with line breaks and labels. You cannot specify this option with the -a,-b, or -r option.

5.1.3 recover*xxx*

The recover*xxx* command is the mechanism for rebooting the mainframe. By default, this command first dumps the mainframe, then it reboots the mainframe domain. If the reboot fails, it will try to boot again. The number of reboots attempted is controlled by the recdcontrol(8) command.

A script recovermf(8) is provided and does recovery suitable for any mainframe type. This can be done since the recovermf command is based on the *sys commands: (haltsys(8), dumpsys(8), bootsys(8)). Recovert90(8), recoverj90(8), recovert3e(8), and recoversv1(8) are all linked to the recovermf command. If a site wants mainframe specific actions to occur, they simply write the script and break the link for the appropriately named script. For example, if they want to recover the Cray SV1 mainframe differently than the supplied default, they would remove the default recoversv1(8) script (breaking the link) and then supply their own recoversv1(8) script. Next time there is an error the site script is invoked.

The recover*xxx* commands accept the following arguments:

info	A colon delimited list of information that the state server holds for the node/component.

5.1.4 post*xxx*

The post*xxx* command is a user exit. If the script exists, it is called by the recover*xxx* script by default, once it is done rebooting the system. This command is provided by the site and completes any actions that the site wants completed before users resume submitting jobs. Again this command is structured like the recover*xxx* commands. A postmf command exists and by default the post*xxx* commands are a link to postmf. If a site wants specific actions done based on mainframe type they would simply supply a new post*xxx* script.

The post*xxx* commands accept the following argument:

info	A colon delimited list of information that the state server holds for the node/component.

5.1.5 dring monitor

Traditionally, dring(8) is run as a two step process. The first dring(8) command is run to detect any GigaRing channel or client problems. If a problem is detected, a second iteration of dring is run to fix the problem. There are two problems with this approach:

- The dring command to fix the ring has to duplicate the analysis of the dring command that detected the problem.
- This duplicate analysis allows for a window of time that the ring can experience more errors across other nodes.

To remedy this, a new mode has been added to dring(8) called the monitor mode. The monitor mode checks the GigaRing channel and its nodes periodically. If a problem is detected, it immediately goes to isolate mode and starts fixing the problem. Since it still has the analysis information in memory, valuable time is saved with the isolating mode.

5.2 Notification

Notification consists of a notify daemon that watches for state changes in the state server daemon, a control program that configures how notification works, and a script that implement the notification action. Notification can be configured off; a site does not have to have notification running in order to take advantage of monitoring or recovery.

5.2.1 notifyd

The notifyd(8) daemon sends out email stating that the system is in an error state, or that the system recovered and is now available for user jobs. It receives state and setting changes from the state server. It looks at the state and settings to see if mail needs to be sent out regarding the component or node.

5.2.2 ndcontrol

The ndcontrol(8) command controls the notification that is initiated by the notifyd daemon. You can use it to change whether notification takes place and to change the list of people who receive the notification.

If you invoke ndcontrol(8) without options, it displays the notification monitoring and the list of people who receive notification. The state server is initialized to contain the following information.

- Notification monitoring is off.
- Email list is empty.

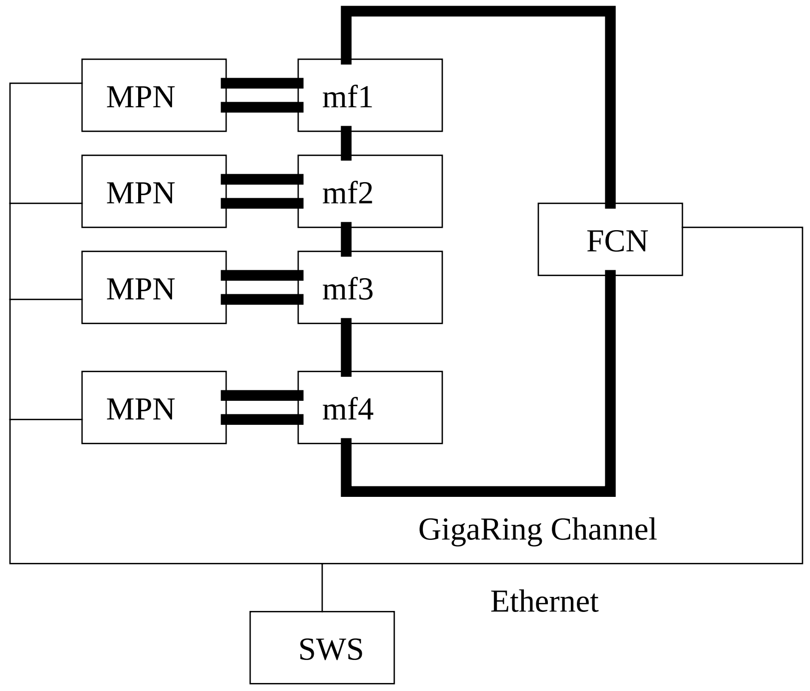
The ndcontrol command accepts the following arguments:

-a on|off	Turns recovery monitoring on or off for all IONs, all mainframes and all rings. You cannot specify this option with the -p or -u option.
-p	Displays in persistent mode. As control information changes, output is generated. The default is to display the current settings and exit. You cannot specify this option with the -a or -l option.
-l list	Lists the people who will receive email when notification occurs. It is a comma separated list.
-u	Specifies that unformatted output is displayed. By default, the output is formatted with line breaks and labels. You cannot specify the option with the -a, or -l option.

6 Ring Stabilization Examples

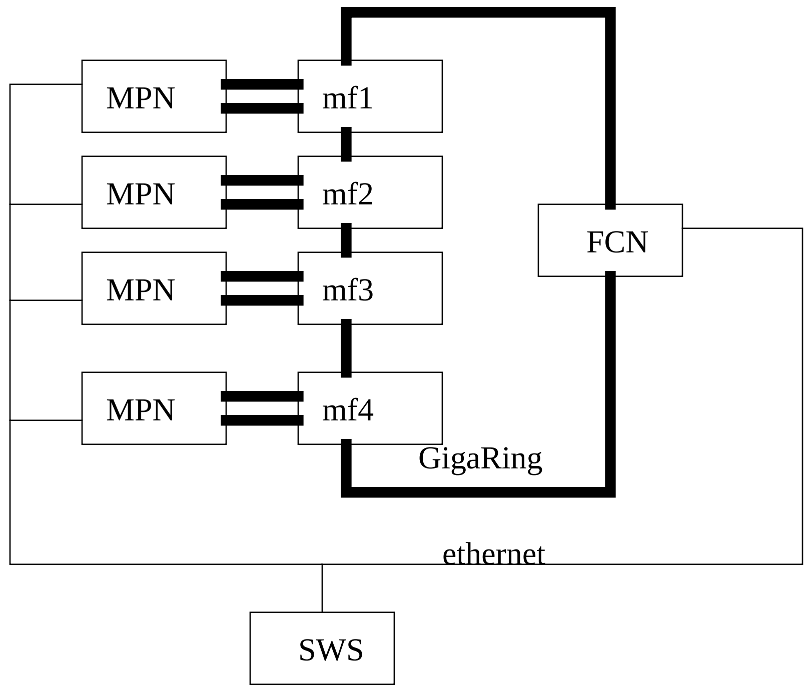
.

6.1 Without resiliency features



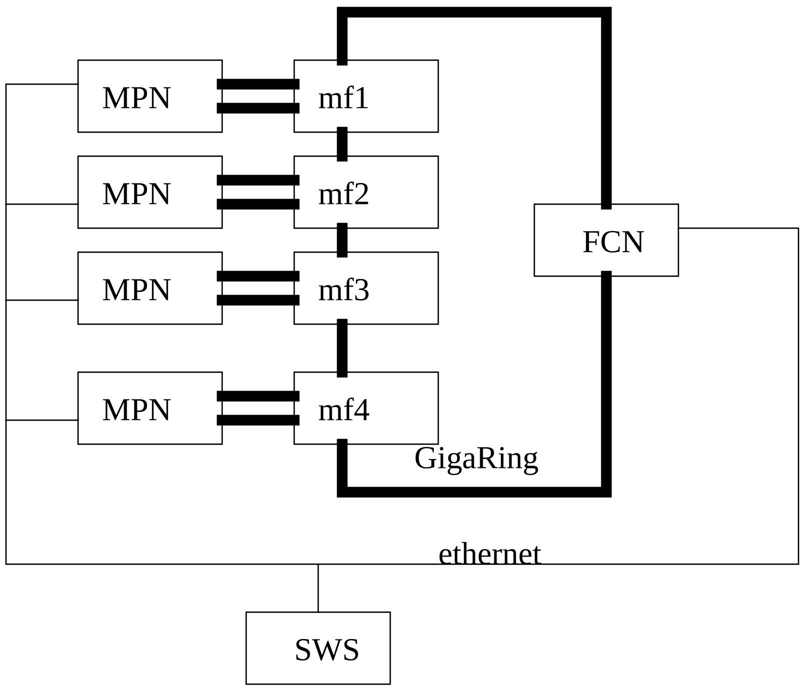
- mf1 panics
- mf2 and other nodes on the shared ring keep sending packets to mf1. These packets are either commands or communications or system heartbeats that are sent out.
- mf1 is not able to process the packets at all. The packet goes around the ring, back to the originator. The response from mf1 is *busy*.
- The other nodes get their packets back from mf1 marked as *busy*. The other nodes keep these packets and try to resend them. Eventually, the send queues on these nodes fill and the nodes are no longer able to communicate to the ring.
- As a result, one by one all the nodes on the ring backup and cannot function. The entire cluster then needs to be rebooted.

6.2 With resiliency features



- mf1 panics
- The GigaRing connection for mf1 is reset. mf1 continues to process packets it receives. Since the operating system is down, it returns the packets marked as *nack.*
- mf2 and the other nodes on the rings get their packets marked as *nack.* mf2 and the other nodes realize the mf1 is no longer responding.
- mf2 goes through its peer-down sequence and just sends system heartbeats to mf1.
- mf1 is able to process these heart-beat requests, so the ring is stable and mf2 and the other nodes do not back up on their GigaRing send buffers.
- As a result, one node is down but the rest of the ring keeps functioning.

7 Auto-Recovery Example



- mf1 panics
- The GigaRing connection for mf1 is reset. mf1 continues to process packets it receives. Since the operating system is down, it returns the packets marked as *nack*.
- mf2 and the other nodes on the rings get their packets marked as *nack*. mf2 and the other nodes, detect that the mf1 is no longer responding.
- mf2 goes through its peer-down sequence and just sends system heartbeats to mf1.
- mf1 is able to process these heart-beat requests, so the ring is stable and mf2 and the other nodes do not back up on their GigaRing send buffers.
- The recovery daemon detects that the mainframe is in a panic state and starts the recovery process. This includes dumping the domain, rebooting the domain, and then issuing an mflevel command to put the mainframe into multiuser mode.
- The bootsys(8) command informs the state server that the mainframe is now in a booted state. The notify daemon detects this state change and again sends out email. This time stating that the mainframe is now up and running.
- mf2 and the other nodes detect that the heartbeats are now being acknowledged by mf1 and the nodes go through their peer-up sequences. The cluster is now back to full operation with no manual intervention.