CESR CONTROL SYSTEM UPGRADE TO LINUX HIGH AVAILABILITY CLUSTER*

M. J. Forster, S. Ball, L. Bartnik, D. Bougie, R. Helmke, M. Palmer, S. Peck, D. Riley, R. Sholtys, C. Strohman, CLASSE, Ithaca, New York, USA

Abstract

The Cornell Electron Storage Ring (CESR) accelerator complex is used to support the Cornell High Energy Synchrotron Source (CHESS) x-ray user facility and the CESR Test Accelerator (CESRTA) ILC development program. Several hundred electro-magnetic elements as well as several thousand sensors are controlled and monitored in real-time via a Multi-Port Memory device (MPM). MPM access and control programs have used Hewlett Packard (originally DEC) Alpha and VAX computers running OpenVMS since 1988.

Due to the demanding throughput, computational and storage requirements of the CESRTA experimental program, as well as a desire to upgrade to more supportable hardware, we have implemented a new Linux control cluster based on an Infortrend 10 GbE Internet Small Computer System Interface (iSCSI) storage device and the Red Hat Cluster Suite. This paper will describe the hardware and software changes required to upgrade our VMS cluster to a high availability, high performance, Linux control cluster.

INTRODUCTION

The CESR control system interfaces to the CESR field bus, the XBus, via a VME-base MPM device [1]. In preparation for the CESRTA research program, it was clear that a new generation of high bandwidth and data volume devices would exceed the capabilities that could be provided by the existing OpenVMS architecture based on Alpha and VAX nodes. In response, an Ethernet-based extension of the accelerator XBus was developed to support a new generation of beam instrumentation. In conjunction with these modifications, processing capability for the new data sources was provided using standard Linux data servers and a preliminary interface to the CESR MPM was developed. Based on the experience developed as a result of this CESRTA endeavor, the CESR operations group was able to develop a migration plan to implement the full functionality of the control system in a high availability Linux-based cluster. That migration is presently underway.

OPENVMS-BASED CONFIGURATION

The current OpenVMS-based control system configuration consists of a cluster of OpenVMS Hewlett Packard (originally DEC) Alpha nodes networked by DECnet and OpenVMS Cluster Communications protocols over 100Mbit Ethernet connections. Six of these nodes have direct access to the MPM, which provides access to control and monitoring of all accelerator devices. Historically, all of our background and interactive control system programs have used these six OpenVMS nodes [1].

LINUX-BASED CLUSTER SETUP

See Fig. 1 for an overview of the control cluster setup.

Data Storage

The new Linux-based control system is centered around an Infortrend EonStor iSCSI storage system. This system offers high-throughput storage, power-outage protection, and fault-tolerent data integrity. To ensure high availability, the EonStor uses a modular design including redundant RAID (redundant array of independent disks) controllers, power supplies, and cooling systems. The RAID system provides for multiple drive failure while allowing uninterrupted data access, and failed drives can be replaced without taking down the system. An Infortrend iSCSI solution was targeted to take advantage of our in-house expertise while meeting our projected throughput, performance and reliability needs.

The data on the EonStor is made accessible through an Ethernet-based storage area network (SAN). This provides block level access to the data. The SAN is used for the cluster protocols and iSCSI traffic.

The shared SAN data is made available to the cluster nodes through the Global File System 2 (GFS2). The nodes function as peers and are allowed equal access to the shared data. This shared access must be protected with a distributed lock mechanism, or DLM. GFS2 was selected as the cluster file system due to the file system consistency it offers and its potential for better bandwidth utilization. The DLM modules as supplied contained a bug, which caused lock recovery time to be proportional to the square of the number of locks to recover. When the cluster was being used for operations the number of locks was above 30,000, resulting in recovery time over 10 minutes when a cluster node was fenced. A patched local build of the DLM module reduces the recovery time to less than a minute, and is expected to be incorporated into the next version of Red Hat Enterprise Linux (RHEL 6.4).

Linux Cluster Nodes

Initially, there were three types of computer processors installed as a cluster largely determined by what was readily available on-hand. A painful lesson was learned that the cluster was not very stable with nodes of differing configurations. When all of the cluster nodes were updated to the machines which were behaving the most reliably, namely IBM System x3550 M3, 12 core, 6 NIC machines with redundant power supplies and disks, the number of node failures decreased dramatically.

Network

Two Blade Networks G8124 10 Gb Ethernet switches connect the cluster members. The EonStor iSCSI device

*Work supported by U.S. National Science Foundation, Award PHY-0734867 and Award PHY-1002467, as well as, U.S. Department of Energy, Award DE-FC02-08ER41538.

ISBN 978-3-95450-115-1

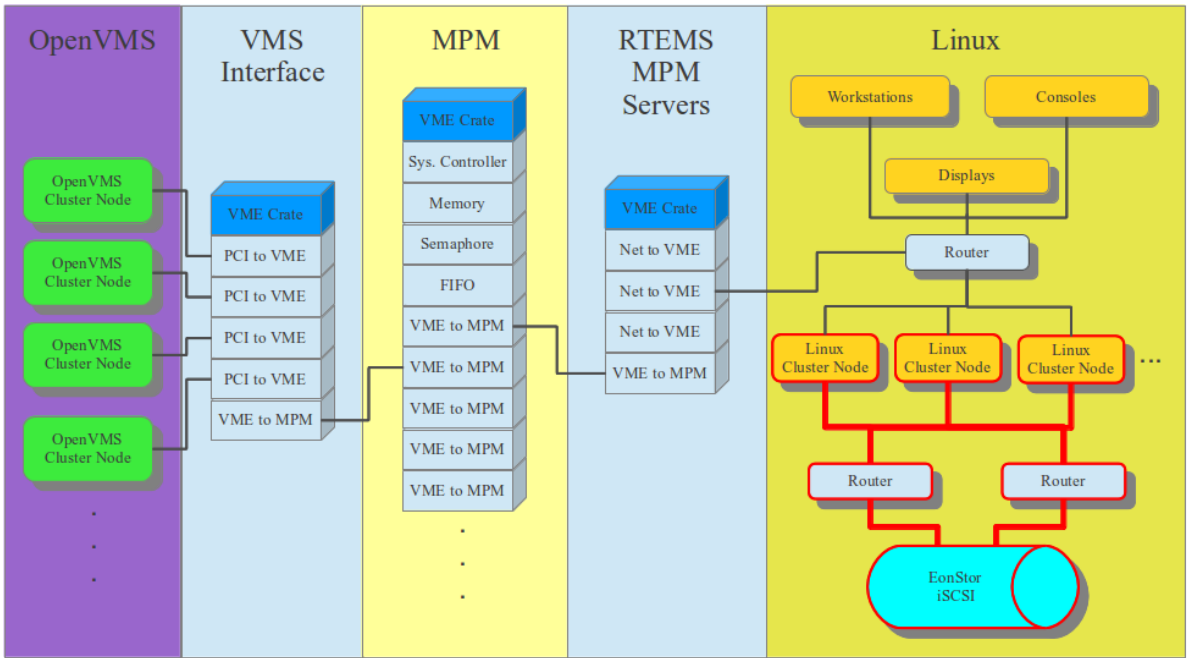

Figure 1: The CESR control system layout. 10 Gb network connections are shown in red.

has two controllers with two 10 Gb NICs (network interface cards) each, connected to both switches. Each cluster member connects to the SAN through an active/passive channel bonded network interface created using 10 Gb connections to each switch. In addition, Linux Device Mapper Multipath (DM-Multipath) is used to provide redundant paths to the iSCSI device. This setup provides full redundancy and enables automatic failover and continued functionality in the event of a NIC or cable failure, switch failure, or iSCSI controller failure.

Moving forward, we plan to test cluster stability using two 10 Gb interfaces instead of a single bonded interface, and the effect of moving the cluster traffic to a separate network from the iSCSI traffic.

Cluster Management

The cluster configuration has been evolving with our understanding of the underlying cluster protocols and their interactions with the network, dm-multipath, and iscsid configuration. The current setup uses Scientific Linux 6.2 (SL6) with Red Hat's High Availability Add-On, which uses the Corosync Cluster Engine (version 1.4.1) for core cluster functionality. The initial setup used Scientific Linux 5.4 (SL5) with the Red Hat Cluster Suite. However, SL5's cluster2 protocol and cluster messaging would not easily support our end goal of programmatic cluster service control.

Our initial plan for upgrading to SL6 involved a mixed cluster of SL5 and SL6. This should have been possible according the specifications, but an attempt at this produced corrupted entries in the GFS2 file system. We quickly identified this incompatibility, and proceeded by splitting the cluster in two. Half of the cluster remained at SL5 and continued serving the GFS2 file systems and running control system services. The other half of the members were upgraded and used to create a parallel SL6 cluster. When testing was complete, we simply moved the GFS2 file systems and all cluster services from the SL5 to the SL6 cluster. When we were satisfied with the transition and reliability of the SL6 cluster, the SL5 cluster was dissolved and the machines were then upgraded and added to the SL6 cluster as they became available. This process went very smoothly and demonstrated the advantages of being able to easily swap cluster services from node to node.

In the event of a cluster node failure, the cluster management will: fence and power-cycle the failed node, move its services to a live cluster member chosen according to a preset failover scheme, and then return the services to the recovered node if desired. A failed node is generally identified within 10 seconds and currently takes about five minutes to recover automatically. The transfer time of services to a new node depends on the resources the service is using, but in general is under 30 seconds.

LINUX TO ACCELERATOR CONTROL INTERFACE

The Multi-Port Memory (MPM)

The main aggregation point for monitoring and controlling the CESR accelerator system has been and will continue to be a centralized multi-port memory or MPM. The MPM consists of VMEbus-based, fast access memory boards, a system controller board, a FIFO (first in, first out) buffer board, and a semaphore array board. The FIFO board provides messaging between processors connected to the MPM. The semaphore board implements mutual exclusion protection for critical areas of memory. The MPM has ports to allow up to 16 processors to connect and have shared access to the data [2].

RTEMS MPM Servers

Server programs were created to run in an RTEMS (real-time executive for multiprocessor systems) environment on an MVME5500 board with a direct input/output connection to an MPM port. These servers allow clients to connect using Ethernet-based socket communication. The RTEMS servers have been programmed to provide the MPM access functions as they were available from OpenVMS. A corresponding set of functions has been made available as a client side library for control system programs requiring MPM access to link against. Three RTEMS servers are currently in use: an operational server, a developmental server, and a debugging server. Each server allows up to 100 client connections simultaneously. The servers have been successfully providing MPM connections to all control system programs ported to Linux for over three months. These client programs include those written in Fortran, C, Java, Python and Matlab.

There were several challenges to the implementation. The real-time nature of RTEMS brings with it little memory protection across tasks, and a memory error in the server program can crash the whole kernel. Another struggle was the small scope of debugging tools available for RTEMS. Currently, the best option is to run a GDB server on one of the boards, which we dedicate to debugging only. Linux GDB (Gnu Debugger) clients can then connect to this board and issue debugging commands. However, a crash often takes down the GDB server as well, preventing any further debugging. In general, these challenges have made RTEMS debugging slower than anticipated.

Control System Program Porting

The majority of control system code in use on our OpenVMS cluster was written in Fortran77. When porting the code to Linux, the decision was made to update the source code to the Fortran90 standard. This decision has slowed down the porting process, but will leave a more maintainable result on the Linux side.

There was some time lost in replacing or working around the use of some OpenVMS specific Fortran conventions or functions. Another hurdle has been with the X11 windowing system, which is used ubiquitously for control system displays. Some of the X11 standards were interpreted rather loosely on OpenVMS and, as a result, several long-standing bugs needed to be fixed when a display program was moved to Linux.

Roughly 20 percent of the control system programs have been ported to Linux so far.

Control System Services

Control system programs such as monitoring displays and knob console control have been ported from running as detached programs on OpenVMS to running as Linux service daemons on the cluster. The services are set up with standard init.d style scripts which provide functions for inquiring about the running status of the service, starting the service, and stopping the service.

Currently, a set of text files defines which control system service runs on a particular node. A background job scheduled to run every minute checks each node and service, making the init.d status inquiry. If a job is found to be not running, it is restarted. This setup is similar to the automatic restart procedure which was in place in the OpenVMS-based control system and proved sufficient for over two decades of CESR use.

The plan is to further improve the recovery for a service failure by converting these standard Linux daemon services to cluster services. This will allow the services to switch nodes automatically on a node failure and restart within seconds. Furthermore, it is planned to add in the capability for programmatic control of the services so that they may be paused, stopped or restarted in response to control system events. This implementation is pending cluster messaging subroutines being made available to our code libraries.

CONCLUSIONS

Only after several Linux node hardware and cluster configuration iterations is the implementation of clustering provided by the Linux environment approaching the level of robustness that has been provided by OpenVMS for many years. It appears to be cost-effective and provides a very effective platform for future control system development.

The software porting process has gone more slowly than expected due to some unexpected challenges. Fortunately, the migration path to Linux has allowed for the OpenVMS cluster to remain fully operational so programs can continue to run on the OpenVMS side. However, our accelerator code libraries have added some changes which are not easily backwards compatible to OpenVMS, and the OpenVMS library is becoming dated. This cannot be maintained indefinitely and provides further impetus to finish the porting of the control programs quickly.

System level cluster services have demonstrated automatic failover with good response times. This now needs to be implemented for accelerator control services.

ACKNOWLEDGMENT

We would like to thank Tim Wilksen, now at DESY, for his groundwork and insight with the RTEMS MPM server and client framework.

REFERENCES

[1] R. G. Helmke, D. H. Rice, and C. Strohman, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 247 (1986) 93, ISSN 01689002.

[2] C. R. Strohman and S. B. Peck, "Architecture and performance of the new CESR control system," Proc. of the 1989 Particle Accelerator Conference, vol. 3, pp. 1687-1689 (1989).

ISBN 978-3-95450-115-1