IMPLEMENTING HIGH AVAILABILITY WITH COTS COMPONENTS AND OPEN-SOURCE SOFTWARE

Rainer Schwemmer, Niko Neufeld, CERN, Geneva, Switzerland

Abstract

High Availability of IT services is essential for the successful operation of large experimental facilities such as the LHC experiments. In the past, high availability was often taken for granted and/or ensured by using very expensive high-end hardware based on proprietary, single-vendor solutions. Today's IT infrastructure in HEP is usually a heterogeneous environment of cheap, off the shelf components which usually have no intrinsic failure tolerance and can thus not be considered reliable at all. Many services, in particular networked services like the Domain Name Service, shared storage and databases need to run on this unreliable hardware, while they are indispensable for the operation of today's control systems. We present our approach to this problem which is based on a combination of open-source tools, such as the Linux High Availability Project and home-made tools to ensure high-availability for the LHCb Experiment Control system, which consists of over 200 servers, several hundred switches and is controlling thousands of devices ranging from custom made devices, connected to the LAN, to the servers of the event-filter farm.

MOTIVATION FOR HA

The question, why one would want High Availability for control systems in HEP experiments is quickly answered by three main concerns. HEP experiments are typically very expensive to build, but also to operate. It is imperative for an experiment to capture as much luminosity as possible while particle beams are available. This is especially true for the big LHC experiments, where every minute of downtime wastes a lot of tax payers' money. Additionally every second of downtime translates to a loss in statistics which reduces the measurement precision of the experiment.

Even when the accelerator is not running, it is vital that control of the sensitive detector hardware is always available, to monitor/prevent unwanted behavior or damage to the devices. Experience shows, that this has always been a major concern to sub detector responsibles during any system outage.

Of course, man power is also expensive, and it is important to make sure people have a system to work with when they are on site, which is usually 24 hours a day. Also the notion of moral must not be underestimated. A system that is plagued by one major breakdown after the other subverts peoples' moods and trust quickly, resulting in a breakdown of productivity even long after the system has been recovered.

SERVICES THAT NEED TO BE HA

While it is good to strive for as much availability as possible in any part of the control system, there are certain sub systems that are more important than others. These systems are usually not the ECS itself, but services that the ECS relies on. For example: the experiment might be able to continue for minutes or even hours without control of the readout boards of the tracking sub-detector, but everything will come to a quick stop if the central file services or databases are unavailable for even a minute.[1]

For LHCb the sub systems identified as most critical, in no particular order, are:

- Databases
- Domain Controllers
- Domain Name Service
- Central File System services
- Some experiment specific services (Event Writers, Database Interfaces, etc)

The first three items on this list are easy to implement on the software side, because Databases, DC and DNS usually come with their own redundancy and HA mechanics. Of course it is still important to come up with a scheme for deploying these services and their backups in a way that does not introduce any single points of failure, like same power source or same PC chassis. In this paper we will focus mostly on the central File System and experiment specific services though.

LHCb ONLINE CORE SYSTEM ARCHITECTURE

During the previous generation of large HEP experiments, it was common to buy all hardware from one vendor, which would then tailor the system to the specific needs. This had the advantage, that the whole system was homogeneous, easy to administer and usually had HA capabilities built in. This option still exists today. The downside is, that it is extremely expensive. Nowadays raw CPU power and general purpose servers are so cheap, that it is more appealing to just invest directly into cheap servers. The drawback is, that cheap hardware can usually not be considered reliable and software solutions have to be found

[1] Although LHCb was not designed this way, experience has shown, that the experiment can, in principle, continue taking data for some time, even if the ECS is completely halted.

to detect failures and migrate services between machines when necessary.

The order of magnitude of the LHCb Online system is about 1000 PCs, distributed to the Event Filter Farm, ECS machines, readout boards and workstations. The core system has to supply all these machines with file systems, databases and authentication and authorization capabilities.

LHCb Online Core System Powering Scheme

The primary reason for major failures of the LHCb Online System so far have been unforeseen power cuts. While there are two redundant power circuits available for powering devices, the main cause for power cuts are due to mistriggers of the safety system which cuts off both circuits. A power cut on the main experiment power circuits typically comes with major recovery work afterwards. Although this usually requires a restart of all the higher level ECS services, it was still found to be important to protect the lower level services with HA techniques.

An especially noteworthy system here are the central file system services. In case of power failure, even with battery backed caches on the storage appliance, File System corruption can occur when the disks lose power and start behaving erratically. Losing the FS in an uncontrolled way means several hours of downtime at best, if the damage is little and a consistency check can recover the FS. At worst it can be days, if everything has to be restored from tape storage. The same holds true for the central database services.

All machines running the central services are thus protected from power failures by using both independent power circuits. Additionally an Uninteruptable Power Supply (UPS) has been deployed to protect from a failure of both circuits. In case of a power outage that exceeds the battery runtime, a daemon process, which monitors the UPS [1], will gracefully shut down all the critical machines, before the power goes out completely.

SAN and Fiber Channel Network

The central FS service of the experiment is based on a SAN consisting of 4 main storage nodes connected to a DDN 9900 storage appliance via fiber channel network. Each component in the FC network exists at least twice to not introduce a single point of failure. The 4 main nodes are connected via two independent FC switches to the two controllers of the storage appliance. Since the switches don't have redundant power supplies, each of the two FC switches is connected to one of the two power circuits.

The store nodes have two redundant power supplies and two redundant FC host ports. Each of them is connected to both switches and each switch is connected to both of the DDN controllers. The connection from the store nodes to the switches are FC4 while the connections to the DDN controllers are FC8. This allows for maximum utilization of the FC4 host ports on the store nodes. The current operating system is Redhat Enterprise Linux 5.3 with kernel version 2.6.18-128.

A distributed FS is running on top of this SAN. Since the FS can utilize multiple FC paths in parallel, we get the additional bonus of running all components in a fully symmetric Active-Active configuration. In case of a single failure the system will run with degraded but still adequate performance.

Because it is too expensive to buy dedicated LAN client licenses of the distributed FS for an order of 1000 clients, the FS is redistributed to all client machines via NFS and Samba.

DETECTING AND ACTING ON FAILURES

Heartbeat/Pacemaker

Heartbeat [2] and Pacemaker [3] are two projects that are part of the Linux HA project suite. While Heartbeat is a tool that offers monitoring and action services, Pacemaker implements a decision engine, that uses the information it gets from Heartbeat to calculate the current state of the cluster and to decide what actions to take on failure of a node or service.

The main entities of Heartbeat are nodes and resources. A resource is any entity that can be made highly available. This can be any kind of abstract or concrete service, like a shared disk volume, a program or an IP address. Resources run on cluster nodes and are started or stopped by Heartbeat.

Resources can have constraints that restrict the nodes that a resource can run on or that make sure that a certain resource runs on the same node as another resource. If a node on the cluster fails, all resources that have been running on that node so far will be distributed over the remaining nodes.

To make sure a node that has erroneously been declared as dead comes in conflict with another node that has taken over its resources, a service called STONITH[2] is used to forcefully restart the presumed dead node. In our case this service is implemented as IPMI calls and uses a different network than the Heartbeat network. This is essential to prevent concurrent access to resources that might become corrupted if they are not used exclusively.

In order to defuse split brain situations, Heartbeat can be configured to require a minimum number of nodes to be in the cluster, before it starts any resources. In our cluster we use 4 active nodes, which run all the essential services plus one node as a tie breaker in permanent standby. This node hosts some non critical services that are not controlled by Heartbeat. To run the cluster, a majority of 3 nodes is necessary. This way we can have one node in maintenance mode, without jeopardizing the cluster quorum by a failure of another active node on top of it.

[2] Shoot The Other Node In The Head

Turning Normal Services into HA Services

Heartbeat has the ability to turn any Linux service that is started via an LSB script, as they can usually be found in /etc/init.d, into a HA service. Of course, this works best if the service is stateless. In case it is statefull, the state needs to be saved to some shared storage device, like the SAN or a database.

Services that we have added to Heartbeat so far are:

- The Run Database interfaces, needed for creating new run numbers and file names for the runs.
- The data movers, which move the output files from temporary storage at the experiment site to CASTOR.
- The monitoring daemon for the UPS.
- The snmptrapd, which is a standard Linux daemon, that captures SNMP[3] traps and, in our case, has been programmed to forward these traps as SMS to the cellphones of the administrators.
- Several IP addresses that are associated with above services.

HA Active-Active NFS and Samba Services

This is certainly the main reason for deploying HA capabilities on our cluster. The idea is to run an NFS and Samba server on each of the 4 storage nodes and to have a virtual IP address per NFS and Samba instance. The IP addresses are added to two round robin entries on the DNS server. One for NFS and one for Samba. The clients then mount the NFS or Samba share with the address of the DNS entry and are thus load balanced over the four nodes.

When one of the store nodes fails or is brought down for maintenance, the IP addresses are failed over to another node. The network communication will then just pick up at the point where it was interrupted on the failing node, making the fail over almost transparent to the user. There are of course a few pitfalls that have to be taken care of with this scenario.

Samba: Although Samba uses TCP connections, it works almost out of the box in this mode. During fail over the TCP connection is broken, but the SMB protocol will immediately reconnect to the new server and continue with the transfer that it might have been doing during the interruption.

A problem is the combination of Samba and a Domain Controller. Since all four Samba servers pretend to be the same server, but the Kerberos Domain Controller allows only one, they will start kicking each other out of the domain, essentially losing the possibility to authenticate any user accounts. This was solved by using RPC instead of ADS when joining the servers to the domain.

NFS: Although NFS V3 uses UDP and it should be even more easy to make the fail over work, it has a few nasty misfeatures that have to be overcome.

First one has to make sure, that all NFS services like mountd, rpcd, etc. are running on the same UDP port on all machines. This can be set in */etc/sysconfig/nfs*. In case of fail over, NFS will try to resume the communication on the same port it had on the previous node.

The reason why we are using V3 right now is due to a bug in the TCP version of NFS. During fail over the TCP version would sometimes go into an ACK storm between the server and the client, bringing down the whole network in between. It is possible this has been fixed somewhere between SLC4 and RHEL 5.3, but we never tested this again so far.

Once the connection fails over smoothly, one has to make sure, that each file and directory has the same NFS handle on all four server nodes. Since NFS uses the physical location of the file on the disk for calculating the handles, this will only work on a SAN with shared storage or some special block device. Moreover, one has to use the *fsid* directive for every export statement to make sure they are the same on all four nodes for each share.

There are also drawbacks that are not so easy to overcome. One problem is, that the FS cache of a node is lost during failure. One has to make sure that either write back caching is disabled on the back end FS or set it to a very small amount.

Another problem is, that file locks can not be guaranteed anymore. NFS, at least the kernel level version, does not use the file locks of the underlying FS but keeps its own locking information to improve performance. This issue is already there if a share is exported via Samba and NFS from the same node. So far this has not been an issue for our users though.

CONCLUSION

We have shown how we achieve High Availability of the LHCb ECS by utilizing mostly cheap, off the shelf components and OSS on top of it to detect and act on failures. The reaction and fail over time of Heartbeat is so fast, that minor failures of one of the core service components are usually not even visible to the user any more. The utilization of HA software has not only increased the robustness of our system against failures, but also decreased the downtime that is usually attached to updates of software or upgrades of hardware. Nodes can just be switched into stand by mode and upgraded one by one in a rolling fashion, while the user keeps working on the rest of the system.

REFERENCES

[1] The APCUPSD project, http://www.apcupsd.com.

[2] The Linux High Availability Project, http://www.linux-ha.org.

[3] Pacemaker: A scalable High-Availability cluster resource manager, http://clusterlabs.org.

[3] Simple Network Management Protocol