2007-12-18 OSC NEWS: Stocking the Toolbox: Tools No Cluster Admin Should Be Without (Linux Magazine)

Stocking the Toolbox: Tools No Cluster Admin Should Be Without

Tuesday, December 18th, 2007
By Troy Baer

Congratulations, you have your shiny brand new cluster installed! You've got your nodes and interconnect up, your file systems mounted, your resource manager running jobs, and your third-party applications ready. Users are chomping at the bit to get on the machine. Now comes the hard part: keeping the whole thing running smoothly.

Setting up a cluster can be trying enough, and maintaining it can be even more difficult. The sheer number of nodes involved in a large cluster can be daunting, as can users' expectations for quality of service. However, cluster admins have a wealth of tools available to make life easier.

Remote Access

Unless you enjoy hanging out in your machine room with a keyboard and monitor on a crash cart, being able to control your cluster nodes remotely is critical to keeping your sanity as a cluster admin.

Ideally, you should be able to run commands on your nodes, connect to their consoles, and even power-cycle them, all from the comfort of your own office. A little time spent configuring these facilities early on in your cluster's life cycle can save you considerable grief down the road.

Distributed Shell

Obviously, one of the most important things in administering a cluster is the ability to run commands on many or all of the nodes in the cluster at once, which is usually called a *distributed shell*. You'll find a huge number of distributed shell projects out there. They tend to overlap somewhat, but each one has its own distinctive set of features. Here's a small sample of the projects that are available:

- all: from the Ohio Supercomputer Center.
- cexec: from Oak Ridge National Lab's Project C3.
- dsh: from IBM's Cluster Systems Management (CSM) product.
- Numerous dsh clones, including:
    - dsh: from the ClusterIt project.
    - dsh from Junichi Uekawa.
- gexec: from the Ganglia project.
- pbsdsh: from OpenPBS, PBS Pro, and TORQUE.
- psh: from the xCAT project.

Even though most distributed shells share the bulk of their features, many have distinguishing features that may be of interest -- depending on the size and requirements of your system.

Parallelism

A good distributed shell should have options to execute a command either on one node at a time (so you can watch the output for errors) **or** on all nodes in parallel (to save time).

Scalability

When you are running a command in parallel on hundreds of nodes, fast start-up is a very good thing. A distributed shell that uses a binary tree structure to contact the nodes will outperform one that contacts the nodes using a simple list; the binary tree should have an $O(\log_2(n))$ start-up time, while the simple list will have an O(n) start up time.

Another scalability consideration is the maximum number of nodes on which you can run a command concurrently. This is particularly an issue with rsh based distributed shells, as rsh tries to use ports below 1024; this can limit it to operating on groups of 150-200 nodes at a time in parallel.

The flip side of scalability is that in many situations, you don't need to run a command on *all* the nodes in your cluster, just some of them. A good distributed shell will give you a way to do that with a simple, concise syntax.

Resource Manager Integration

It is often useful to have a distributed shell that has a certain level of integration with your resource manager or batch system of choice. For instance, having the ability to run a command on all the nodes assigned to a particular job can be extremely useful for diagnostic or monitoring purposes.

However, resource manager integration can also go too far; you don't want to have to rely on the batch system to spawn administrative or diagnostic processes on nodes if that batch system gets into a bad state.

Simplicity

The advanced functionality available in a distributed shell package should be weighed against the complexity of its build and configuration process. If a distributed shell requires elaborate configuration or multiple daemons running on all the nodes, you may want to consider using a different package.

Serial Console Access

Sometimes you need to connect to a node out of band, without using its network interface -- for instance, if a switch in the cluster's network infrastructure takes a dive, or if a node's network interface develops a hardware problem.

The most common way of handling this is connect to the node's console over a serial line, using a device called a *serial console server*. In the past, these were simply commodity computers with special high-density serial port devices, but increasingly they are purpose-built embedded systems built into rack-mount packaging.

In large clusters, you can end up with many serial console servers and a complex node-to-console-server mapping. A common way to simplify this is to use a package called Conserver, which allows many serial

concentrators to be accessed transparently from a single host. Conserver can manage who has read-only or read-write access to the individual consoles, and it can also be used to log all the traffic going over a serial line to a file.

Power Control

In some instances, all you can do for a node is reboot it or turn it off. To do that, you need some form of remote power control, which generally comes in one of two flavors. One flavor of remote power control is the *service processor*, which is a small embedded system built into the node that is accessible over the serial line and can control power for the rest of the node as well as do diagnostics on it. Service processors tend to be found in higher end server systems, and increasingly they support standardized access methods like OpenIPMI.

The second flavor of remote power control is a *network power controller*, basically a power strip with a small embedded system attached to it that can turn individual receptacles on the strip on or off. These are usually accessible over the network using an SNMP interface.

However, not all hope is lost if the nodes in a cluster have neither service processors nor network power controllers. So long as a node's serial console is available and the Linux kernel's "Alt-SysRq" functionality is enabled, the node can be rebooted over the serial line.

Logging and Monitoring

Being able to control nodes remotely is half the battle, but the other half is knowing what is happening on the nodes, both from a historical perspective and in real time. You can obtain this information via analysis of log files and active monitoring of systems.

While most Linux distributions include some basic logging and monitoring tools like syslog, sysstat, and psacct, these tools do not scale well to clusters with hundreds of nodes. However, other open source tools are available that exhibit better scalability on large systems.

Logging and Log Analysis

The standard UNIX syslog facility can be configured to send log messages to a remote host. A common practice is to designate one host as a syslog server for the cluster; in fact, a few vendors even have purpose-built syslog server appliances that advertise much higher log message rates than commodity solutions.

The larger problem is often analyzing the resulting logs to look for patterns and correlations. A number of open source log analysis tools exist, such as Epylog, Loghound, Logwatch, and Sisyphus.

Of these, Sisyphus is probably the most interesting for large clusters, as it was designed specifically with HPC environments in mind.

System Monitoring

An important part of system administration is *system monitoring*; that is, the ability to detect unusual behavior on the system by regularly querying the state of the nodes. Much of this is repetitive and can be automated, but eventually the information must be examined by an actual human being. The main difficulty in system monitoring for large systems is in finding a balance between too little information and too much.

Two of the leading open source system monitoring projects are Ganglia and Nagios. While these two

projects have significant amounts of feature overlap, they have very different goals and approaches, and as a result they complement each other well. Smaller clusters will be able to get by with one or the other, but larger systems in particular can benefit from having both available.

Ganglia's major strength is its ability to collect and display *metrics* from large numbers of nodes over time. Ganglia metrics can have either string or numeric (integer or floating point) values; for numeric metrics, Ganglia can also produce plots of the metrics' time history over various ranges. Using this capability, an administrator can often identify host-level issues such as CPU over-commitment, excessive swap, or full file systems by simple visual inspection.

On the other hand, the major strength of Nagios is in monitoring not metrics but *services*. Like many extensible systems, Nagios is built on the plugin model, with plugins that implement service tests such as network connectivity or the existence of a process.

Nagios plugins can display one of four return statuses: OK, Warning, Critical, and Unknown. Unlike Ganglia however, Nagios can be configured to provide *notification* via email if a service has been in the Warning or Critical state for too long.

Nagios also has the concept of *service dependencies*, so that a service check can specify low-level services upon which it depends. Careful configuration of Nagios service dependencies is critical to avoid being deluged with extraneous notifications during problems with core system-wide services like network connectivity or file system servers.

Job Monitoring

The vast majority of clusters are scheduled shared resource accessed through a *resource manager* (also known as a *batch system*) such as TORQUE, Grid Engine, or LSF. These resource managers generally include some level of job monitoring capabilities, with the commercial solutions often having slicker or more sophisticated tools than the open source offerings. However, it is also not uncommon for sites to implement additional job monitoring tools that reflect the realities of their environment.

For instance, the Ohio Supercomputer Center has a set of administrative tools for PBS and TORQUE systems called pbstools. Included in these tools are two job monitoring tools: qps, which displays all the processes associated with a job; and reaver, which identifies (and optionally terminates) user processes which cannot be associated with a running job.

Application Performance Monitoring

Another important aspect of monitoring that has historically been difficult on Linux clusters is the ability to measure hardware-level performance for arbitrary applications using the hardware performance counters available on virtually all modern microprocessors.

This is something that traditional supercomputer vendors like Cray have done well for years, but Linux finally is starting to catch up thanks to a portable standard interface for performance monitoring from the University of Tennessee at Knoxville's Innovative Computing Lab called the Performance API (PAPI).

A number of portable performance measurement tools have been built on top of PAPI, perhaps most notably PerfSuite
from the National Center for Supercomputing Applications. PAPI and friends are currently supported out of the box for Linux on the IA64, PPC, and SPARC architectures.

On x86 and x86_64 systems however, using PAPI and friends is complicated by the fact that there is no standard interface for hardware performance counter access in Linux for these architectures. In fact, there

are two competing interfaces, perfctr and perfmon2, and neither is included with vanilla kernel.org kernels or the kernel builds distributed by the various distributions. If you wish to use PAPI-based applications on these systems, you will have to patch and recompile the kernel for your nodes.

However, there is still hope for hardware-level performance monitoring in situations where patching the kernel to support PAPI is impossible or impractical. Most modern distributions include a system-wide profiling tool called oprofile, which also has the ability to access hardware performance counters on a system-wide basis.

Normally oprofile requires root access to start up and shut down profiling; however, since the job prologue and epilogue run by most resource managers are also run as root, these processes can be used to perform the requisite operations. The following examples are designed for TORQUE on a cluster of Opterons, but they can be adapted to virtually any batch system or Linux architecture.

Example 1: TORQUE prologue/prologue.parallel fragment for configuring oprofile:

```
#!/bin/bash
# Note:  the event list will change depending on what processor family
# you're on and what performance metrics you want to measure.
eventlist="
           --event=DISPATCHED_FPU_OPS:1000000:0x3f:0:1
           --event=L2_CACHE_MISS:1000000:0x07:0:1
           --event=L2_CACHE_FILL_WRITEBACK:1000000:0x03:0:1
           --event=DATA_PREFETCHES:1000000:0x02:0:1
          "
options="--separate=all"
opcontrol --start-daemon 2>/dev/null
opcontrol --setup $eventlist $options 2>/dev/null
opcontrol --reset 2>/dev/null
opcontrol --start 2>/dev/null
```

Example 2: TORQUE epilogue.parallel/epilogue fragment for shutting down oprofile:

```
#!/bin/bash
opcontrol --dump 2>/dev/null
opcontrol --stop 2>/dev/null
# The performance counter data will be stored in an oprofile "session"
# with the same name as the full PBS/TORQUE job id.
opcontrol --save=$1
```

Example 3: TORQUE epilogue fragment for processing job accounting and oprofile results:

```
#!/bin/bash
# compute job walltime in seconds
walltime=$(echo $7 | tr , "\n" | grep walltime | \
           sed 's/^walltime=//' | awk -F : '{print 3600*$1+60*$2+$3}')
echo "----------"
echo $1 performance results:
echo $7 | tr , "\n"
/usr/local/sbin/perfreport -a $1 $walltime
```

Here /usr/local/sbin/perfreport is a site-specific program that analyzes the performance data generated by oprofile on all the nodes assigned to the job; it is necessarily dependent on which hardware performance events are being counted, their overflow bucket sizes, and the desired metrics.

It should also be noted that oprofile will measure **all** activity on a node, so it is best to do this only in situations where the nodes allocated to a job are exclusively for that job's use.

Conclusion

Obviously, a universe of choices are available for many of the types of tools discussed here, up to and including rolling your own. Selecting the right set of tools for your system depends as much on your (and your users') requirements as it does on the various packages' feature sets. In many cases, the best approach is to narrow the list of candidates down to two or three packages and then evaluate those on your system. Before you know it, you will have a Toolbox of that makes your job easier and your users happy!

Troy Baer has been a systems engineer at the Ohio Supercomputer Center since 1998.

http://www.linux-mag.com/microsites.php?site=business-class-hpc&sid=build&p=4658