Increasing HPC Resiliency Leads to Greater Productivity

Roger Moye
University of Texas – MD Anderson Cancer Center
7007 Bertner Ave.
Houston, Texas 77030
1-713-792-2134
rvmoye@mdanderson.org

ABSTRACT

Maintaining a high-performance computing (HPC) infrastructure in an academic research environment is a daunting task. Coupled with lean budgets and limited staff, the need for a self-healing cluster becomes all the more important. It is possible to achieve nearly 100% uptime on HPC compute nodes by utilizing job scheduling features that will pre-emptively terminate jobs before they cause problems on HPC systems, or prevent new jobs from running should a potential problem already exist, thereby freeing up time for the systems administrators to work on tasks other than cluster recovery.

CCS Concepts

• Computer systems organization~Dependable and fault-tolerant systems and networks • Computer systems organization~Availability

Keywords

HPC; Job Schedulers; Compute Node Availability; System Uptime

1. INTRODUCTION

Maintaining a fairly large HPC infrastructure with a small team, in an academic research environment, can be a challenging task. Experimental in nature, the compute jobs on these types of systems produce a wide variety of outcomes. Unfortunately one outcome can be failed jobs, which at times can have an adverse impact on other, "bystander" jobs, or even the entire HPC infrastructure. It is desirable to minimize these negative impacts, if not eliminate them entirely. We will demonstrate how, through pre-existing tools and technology, the University of Texas – MD Anderson Cancer Center (UTMDACC) was able to achieve an extremely reliable and robust HPC infrastructure that not only produced a high level of service to our user community but also reduced the demands on the systems administration staff.

2. COMPUTE CLUSTER RESOURCE DEMANDS

The HPC environment at the University of Texas – MD Anderson Cancer Center (UTMDACC) is a demanding one that primarily serves medical researchers. The majority of jobs can be divided into two categories: (1) next-generation sequencing (NGS) analysis, and (2) other scientific applications in areas of biostatistics, radiation physics, etc. The latter will be referred to as *basic science compute jobs* for the purpose of this discussion. NGS jobs typically involve large data sets with input and output files that are many gigabytes (GB) in size. Therefore, it follows that NGS compute jobs also require large amounts of memory on each compute node. Often NGS projects will contain hundreds of gigabytes up to a few terabytes (TB) per project. Basic science jobs typically involve small data, comparatively speaking.

The UTMDACC environment consists primarily of two HPC compute clusters, *Nautilus* and *Shark*. The Nautilus cluster contains 336 compute nodes, each with 24 processor cores per node (for a total of 8064 processor cores), and memory per node ranging from 64GB up to 192GB. Nautilus uses the Moab 8.1.1 job scheduler from Adaptive Computing [1]. The cluster is primarily used for basic science jobs (including MPI jobs) although there are some NGS jobs that run on this cluster as well. The primary storage for Nautilus is an 800 TB GPFS storage system. All of the compute nodes and storage systems use a QDR Infiniband network fabric.

The second cluster, Shark, contains 80 compute nodes, each with 24 processor cores (for a total of 1920 cores), and memory per node of 384 GB. There are also six special purpose nodes with least 2 TB of memory per node. This cluster uses the Platform LSF 9.1.3 scheduler from IBM [2]. This cluster is primarily intended for NGS compute jobs due to higher memory availability. The primary storage for Shark is a 1.6 petabyte GPFS storage system. All of the compute nodes and storage systems use a QDR Infiniband network fabric.

The two clusters combined account for over 10,000 cores, with approximately 125 active users between the two. Over the past year the clusters have operated at approximately 70% utilization with extended periods (often many weeks) above 90%. The clusters have approximately 150 applications installed. There are primarily 3 systems administrators that are responsible for all aspects of the operations of this environment, from user tickets and software installs to design and installation of new systems. The responsibility for the GPFS storage systems resides within a different team and is not discussed further in this paper.

3. NODE RELIABILITY

With NGS jobs that can easily consume most of the memory on a compute node, a frequent problem with Nautilus was node crashes due to the node running out of memory. When this happens, not only would the NGS compute jobs die, but any other compute jobs running on the same nodes would be lost as well. This would require the user and innocent bystanders to restart their jobs, delaying their research results and wasting CPU cycles. On average, node downtime following a crash could range from 1-2 hours during business hours, to many hours at night and on weekends, which had a negative impact on cluster capacity. In addition, large numbers of failed nodes in a short period of time would cause performance issues for the GPFS storage system, which would in turn cause problems across the entire cluster, due to the clustered nature of GPFS.

3.1 Demands on Staff Time

From a systems administration perspective, these problems required that the UTMDACC team continually monitor Nautilus for failed nodes during and after business hours, including weekends. The team also developed scripts that would determine which jobs were running on the failed nodes, allowing staff to inform the affected users so they could restart the appropriate jobs. The team would also attempt to ascertain which specific compute jobs caused the nodes to fail and advise the user on how to resubmit the job (such as running on a specific node that had a larger amount of memory). It was not unusual for the team to reboot 10-20 nodes per day. The team typically devoted about 2 hours per day to this clean-up effort. Obviously, for a small team with many concurrent demands this is not sustainable. From the users' perspective, failed nodes and job restarts represented lost productivity as well, especially for users who had the misfortune of running good jobs on nodes that had crashed due to the memory demands of other jobs that were sharing the same node.

3.2 Single Threaded versus Multithreaded

An additional complication was users that were unaware that the software they were using was running in a multithreaded fashion. This would often result in users requesting a single core in their job submissions only to have their software utilize all 24 cores. This is especially a problem when the remaining cores on a compute node were already in use by other jobs, negatively impacting their speed. This over-subscription of compute nodes created the risk that the jobs of every other user on the affected node could exceed the amount of run time each had requested. This results in failed jobs and often additional help desk tickets sent to the systems administration team.

4. JOB SCHEDULER TO THE RESCUE

Memory overload on compute nodes was determined to be the number one problem on the Nautilus cluster. To avoid this problem on the Shark cluster, it was decided that the job scheduler would be configured to require compute jobs to request/reserve a specific amount of memory, just as the jobs would request processor cores and run time. Further, it was decided that the job scheduler would be configured to terminate a compute job if its memory utilization exceeded the amount of requested memory. Lastly, it was determined that if a memory value is omitted from a job submission script, the job would be assigned a default value of 1 megabyte, which would most likely cause the job to exit immediately. This step was taken to force the users to request memory rather than omit it.

With the help of the vendor, IBM, the memory limit enforcement was accomplished by adding several lines to the LSF configuration as shown below (see Table 1). Note that the location of the LSF configuration files will be specific to each installation but will generally be found at the path shown in Table 2.

Table 1. LSF Configuration

Config File	Parameter
lsb.conf	`LSB_MEMLIMIT_ENFORCE=y`

lsb.queues	`MEMLIMIT = 1 377856`
	`RES_REQ = "rusage[mem=8192] span[hosts=1]"`

Table 2. LSF Configuration File Path

```
/<install-root>/lsf/conf/lsbatch/<cluster-
name>/configdir
```

The `LSB_MEMLIMIT_ENFORCE=y` parameter is a single parameter that must be set in the `lsb.conf` configuration file that will cause the job scheduler to terminate a job that exceeds the amount of memory that it has requested. With this enabled, any job that does not request an amount of memory will be automatically assigned the soft limit. The soft and hard limits are defined with the `MEMLIMIT` parameter in the `lsb.queues` configuration file and should be set for every queue. In this example, the soft limit is 1 megabyte and the hard limit is 369 gigabytes (369 * 1024 MB). While the compute nodes have 384 gigabytes of memory, 369 gigabytes was chosen as the hard limit since the Linux operating system and GPFS pagepool occupy over 10 gigabytes of memory. So it was important to prevent jobs from requesting more memory than would be available on the compute nodes since these jobs would never be dispatched. Lastly, the `RES_REQ` line must be added which defines the minimum amount of memory that will be reserved (8 gigabytes). This guarantees that a job will not be dispatched to a node unless at least 8 gigabytes of memory is available on that node. The 8 gigabyte minimum limit was chosen to guarantee that a job would never be dispatched to a node that was running out of free memory.

Job submission scripts on Shark are required to request memory by including two lines as shown below (see Table 3).

Table 3. LSF Job Submission

Parameter	Description
`#BSUB -M 8192`	Memory limit
`#BSUB -R rusage[mem=8192]`	Memory reservation

The `–M` option is the job's memory limit. Omitting this limit results in the job receiving a default limit which is the soft limit of the queue. This would be 1 megabyte on Shark.

The `–R` option is necessary in order to reserve memory. In this example the job reserves 8 gigabytes of memory (8192 MB). A job will not be dispatched to a compute node unless the node has at least 8 gigabytes of memory available. Once the job is running, 8 gigabytes will be reserved for that job whether it uses it all or not. Any attempt to reserve less than 8 gigabytes will cause the job to be rejected because the amount requested is less than that defined by the `RES_REQ` line in the scheduler configuration.

5. EXPANDING THE SCOPE

Subsequently, during a major refresh of the Nautilus cluster it was determined that the job scheduler would be configured to use similar features as Shark. Given that Nautilus was approximately four times larger and intended for a wider audience, it was decided that the job scheduler would also be configured to terminate jobs that attempted to use more cores than the job had requested. The goal was to eliminate the problem of multithreaded software overloading compute nodes.

With the help of a third-party vendor, X-ISS, the memory and processor core limits enforcement was accomplished by adding several lines into the Moab configuration file `moab.cfg` as shown below (see Table 4).

Table 4. Moab Configuration

Parameter
SERVERSUBMITFILTER /opt/moab/etc/jobFilter.pl
RESOURCELIMITPOLICY JOBMEM:ALWAYS,ALWAYS:NOTIFY,CANCEL
RESOURCELIMITPOLICY PROC:ALWAYS,ALWAYS:NOTIFY,CANCEL
RESOURCELIMITPOLICY WALLTIME:ALWAYS,ALWAYS:NOTIFY,CANCEL

Note that the location of the Moab configuration files will be specific to each installation but will generally be found at the path shown in Table 5.

Table 5. Moab Configuration File Path

/<install-root>/moab/etc

The `RESOURCELIMITPOLICY` parameters indicate that the job scheduler will always notify the user when the job's memory (`JOBMEM`), number of processors used (`PROC`), or run time (`WALLTIME`) reaches the soft limit. The jobs will be cancelled when these values reach the hard limit. If soft and hard limits are not defined in the scheduler then the limits provided by the job submission become the hard limit. The run time enforcement was never a problem on Nautilus, but its configuration is shown here for completeness.

Users on Nautilus are required in their job submission scripts to request memory along with processors, nodes, and run time (see Table 6).

Table 6. Moab Job Submission

Parameter	Description
#PBS –l nodes=1:ppn=1	Nodes and processors
#PBS –l walltime=1:00:00	Run time
#PBS –l mem=1gb	Total memory

It should be noted that the `JOBMEM` feature only works with the `MAXMEM` Moab job submission parameter.. Therefore, a job filter script (`jobFilter.pl`) was added to the configuration as specified by the `SERVERSUBMITFILTER` parameter (see Table 4). This script processes every job submitted and will rewrite the job submission to include the `MAXMEM` (see Table 7) option. If the job submission omitted the `mem` option, then a `MAXMEM` option of 1 megabyte is added which means the memory hard limit for the job is only 1 megabyte.

Table 7. Moab MAXMEM Example

Example
#PBS –W x=MAXMEM:1mb

It should be noted that the job filter script requires the use of the Moab `msub` command rather than the Torque `qsub` command, though the syntax of the job scripts is the same. For this and other reasons not relevant to this paper, Nautilus requires `msub` for job submissions.

6. IMPROVED UPTIME OF COMPUTE NODES AND STAFF

The results of these efforts were nothing short of dramatic. UTMDACC achieved 525 days of uptime on the Shark cluster, and 90% of the compute nodes were never rebooted during this time. On Nautilus, UTMDACC achieved 360 days of uptime, and 70% of the compute nodes were never rebooted. The 30% of the nodes that were rebooted can be attributed to hardware failures and some errors on the Infiniband fabric that caused the nodes to get into a bad state requiring a reboot. The only reason that uptime stopped at 525 and 360 days, for Shark and Nautilus, respectively was due to a planned maintenance schedule for the research storage systems that required GPFS client software to be upgraded on every node. This provided a convenient opportunity to reboot all nodes.

More importantly, the memory and CPU enforcement allowed the limited staff to focus on tasks other than recovery of compute nodes. The focus of the staff and leadership almost immediately changed from maintaining cluster availability to improving cluster efficiency. It also provided users with increased productivity because they were no longer focused on restarting of large numbers of compute jobs.

In the first month of operation under the new configuration, Nautilus logged 964 jobs that exceeded either the memory or processor hard limits and were terminated. This represents 964 opportunities for crashed nodes that have been avoided entirely. In fact, there were zero crashed nodes during this time. During the first year of operations, Nautilus logged 12,157 jobs that exceeded their memory hard limit, and 2,187 jobs that exceeded their processor hard limit. All of these jobs were terminated. Seldom did the staff need to reboot compute nodes in spite of over 850,000 jobs submitted during the period. In what seemed like an overnight change, Nautilus went from a very hands-on cluster to a very hands-off cluster from a systems administrator perspective. With the recovery of more than 10 hours per week of staff time, this was the equivalent of adding more than one-quarter FTE to the systems administration team.

7. COMMUNICATION MADE EASIER

Prior to the implementation of hard memory limits on Nautilus, the users were instructed to request a number of processors commensurate to their estimated memory requirements in order to reserve that memory. That is, if a job was expected to use 40% of the memory on a compute node then the job should request 40% of the processors, whether it actually needed these processors or not. In theory this would prevent the compute node memory from becoming overloaded since the node could not be oversubscribed with compute jobs. In practice, however, this proved to be only marginally successful. First, it was a difficult concept to communicate to a user community that had a very wide range of experience levels. Second, users had trouble estimating the amount of memory their compute jobs needed. Even good attempts to select the correct number of processors did not solve the problem.

Following the implementation of hard memory limits, communication of this topic to the users became much easier. The user community was warned, in advance, of the hard memory limits. Information was provided on a web site, in example job scripts, and in training classes to inform them of the new job submission requirements and to illustrate what the errors might look like. One such error from Moab is presented below (see Table 8). While the team still occasionally receives help desk tickets on this topic, communication of this issue was essentially handled with a single slide in a slide deck. The users now rely on the scheduler to communicate with them when their jobs have exceeded their hard memory limits.

Table 8. Moab Job Error Example

Example
`job 3546 exceeded MEM usage hard limit (6135 > 5120).`

8. UNINTENDED CONSEQUENCES

An unintended but beneficial consequence was that users became much more aware of how their software was operating. Previously, users often had little to no indication of why their jobs died. Now, users can self-diagnose that their jobs were terminated because their software was using more processors than they had requested. This let to the discovery that their software was multithreaded, not single-threaded. Overloading of compute nodes stopped immediately.

However, with stricter rules enforcement by the job scheduler comes some drawbacks as well. Once users found a "safe" number of processor cores and memory to run their jobs, they were inclined to simply copy job scripts from one set of jobs to another, without regard for whether the CPU and memory limits were appropriate. The clusters began running compute jobs that requested many times more memory than was actually being used, which caused the clusters to be under-utilized. That is, the job scheduler was reserving space on the compute nodes that was not being used by the jobs. This required systems administration staff to identify jobs that were behaving in this manner and talk to the users about making better decisions when specifying memory and processor requirements. However, now that significantly less time is required to deal with crashed nodes, more time is available to help individual users maximize efficiency of their jobs, which can be argued is a more fruitful effort. This topic was also included in training sessions, which helped alleviate the problem, though it does occasionally reappear.

9. HARDENING THE RESOURCES

Hardening the job scheduler against compute jobs that consume too many resources is only part of the picture, however. There are other scenarios from which the cluster needs to protect itself.

9.1 Signs of Trouble

There are often signs of trouble coming, that if detected early, can be acted on to prevent jobs or nodes from crashing. The most common cases on Nautilus and Shark are summarized in the table below (see Table 9).

Table 9. Major Items to Monitor

Item	Threshold
Storage block and inode usage	> 90%
Remote storage unmounted	Y/N
Memory available	< 2GB
Swap available	< 15GB
sshd is running	Y/N

For example, if local storage on a compute node (such as `/tmp`) is nearly full then this represents a condition that could cause jobs to crash. Thus the scheduler should be instructed to stop dispatching jobs to that node.

This is accomplished on Shark with an LSF External Load Information Manager (ELIM) script and on Nautilus with the LBNL Node Health Check (NHC) from Lawrence Berkeley National Laboratory [3]. If any of the thresholds are exceeded the job scheduler will mark the node as being "closed" or "offline". If compute jobs are successfully running on those nodes they will continue to do so. However, no new jobs will be dispatched to those nodes until the item is resolved.

While the merits of this seem obvious, one potentially disastrous scenario that was prevented by these tools is worth examining. Nautilus had a repeated occurrence of a particular software package generating thousands of small files. In this case a user had run several hundred compute jobs one evening with each job generating tens of thousands of small files. Within just a few hours over 17,000,000 files had been created on the primary storage system, which took the system above 90% inode usage. Had this user or any other repeated the same activity, the support staff could have arrived at work the next morning to find that the primary storage had run out of inodes which would have crashed compute jobs and corrupted data that was already in flight.

NHC detected this condition and marked all nodes on Nautilus offline. Jobs that were running continued to run unimpeded, but no new jobs were allowed to run. New jobs that were submitted during this time went into the queue and were pending. NHC had logged the event and our monitoring system had alerted the administrators to this fact. Upon removal of the small files, the compute nodes were automatically reopened by NHC without any intervention from the systems administrators.

9.2 Storage in Memory

Another scenario that was encountered on Nautilus was compute nodes that simply stopped responding for no apparent reason. One of the cluster power users had started using software that simply was not designed to use remote storage. It needed local storage, yet the compute nodes lacked enough local disk space for these tasks. Any time these jobs would start running the compute nodes would seemingly disappear from the cluster. The compute jobs were actually running but the systems administrators could not ssh into the nodes nor could the job scheduler communicate with the nodes. This gave the appearance that the nodes had crashed, often 50 to 100 at a time.

With network storage not an option and insufficient local storage, it was decided that a solution to this problem might be the use of a RAMDisk on the compute nodes, which is essentially a file system stored in memory. This was done with a command (see Table 10) executed at boot time from `/etc/rc.local`.

Table 10. RAMDisk Implementation

```
mount -t tmpfs -o size=15g tmpfs /mnt/tmpfs
```

The user was then instructed to direct the job I/O to this location. This immediately resolved the problem, increased the performance of the compute jobs substantially, and eliminated all of the alert messages and the worries caused by compute nodes that were not responding. Both Nautilus and Shark now have this feature enabled should the problem ever arise again. It should be noted that it is incumbent upon the user, or more to the point their compute jobs, to remove any files stored inside the RAMDisk area. Without doing this the files will remain in the RAMDisk, and therefore in memory, until the next system reboot. This creates the potential for the RAMDisk to be full preventing new jobs from using it. To avoid this possibility, tmpwatch was configured (in `/etc/cron.daily/tmpwatch`) to clean files from this area as shown here (see Table 11).

Table 11. Tmpwatch configuration

Example
`flags=-umc`
`/usr/sbin/tmpwatch "$flags" 11d /mnt/tmpfs`

This configuration was merely appended to the default tmpwatch configuration provided by the Linux operating system. Tmpwatch will purge files from the RAMDisk area (`/mnt/tmpfs`) if the files have not been accessed in 11 days. Eleven days was chosen as a reasonable file age due to the known workflow of the users who were using the RAMDisk. The users' compute jobs would usually run for several days. Therefore, it was important to ensure that the files were not purged until the jobs were completed. It was also assumed that these users, being power users, would follow our instructions and remove their data from the RAMDisk as part of their compute job just prior to the jobs exiting. The tmpwatch configuration was added as a precaution to guard against users accidentally leaving data in the RAMDisk area.

10. CONCLUSION AND FUTURE WORK

We have demonstrated through several real-world examples that it is indeed possible to achieve very high levels of uptime on HPC compute nodes and lower the amount of time required by system staff to monitor and operate a cluster. Through the use of existing job scheduler features (such as the LSF MEMLIMIT and Moab MAXMEM features), third party tools (such as NHC), and an investment of time, efficiency can be achieved that will have a considerable positive impact on the HPC staff and their customers.

In the future, it is the intention of the team to implement failover on the head node of Nautilus to further increase the fault tolerance of that cluster. Failover already exists on the Shark head node though there has not been an occasion to use it other than through routine testing. Redundancy is already built into the GPFS storage systems and most maintenance activities can be done without shutting down the storage system. With failover enabled on the Nautilus head node, the team should be able to conduct most maintenance activities without interrupting service and the team will have reasonable confidence that the systems will survive routine failures.

11. ACKNOWLEDGMENTS

We are grateful for the assistance from Nathan Baca and the team at IBM's Platform Professional Services organization for their integration and deployment services on Shark, and Larry Adams with IBM Platform Computing for his assistance with LSF. We would like to thank Deepak Khosla, Bryan Bales, Waco Millican and the team at X-ISS for their integration services on Nautilus. We would like to thank the team at Adaptive Computing for their assistance with Nautilus as well. A special thanks to Dr. Bradley Broom (UTMDACC Professor of Bioinformatics and Computational Biology) for lending his advice and technical expertise. Finally, we sincerely appreciate the entire team at UT-MD Anderson Cancer Center (Dan Jackson, Sally Boyd, Jenny Chen, Eric Sisson, Rong Yao, and Andrew Adams) for their ongoing contribution to the success of these HPC resources. We appreciate the UTMDACC IT Operations Storage Team for their contributions as the GPFS storage administrators. We also thank William Joe Allen from the Texas Advanced Computing Center (TACC) for the training services he provides to our user community.

12. REFERENCES

[1] Adaptive Computing. 2016. Moab Workload Manager. http://www.adaptivecomputing.com/.

[2] IBM. 2016. Platform LSF. http://www-03.ibm.com/systems/spectrum-computing/products/lsf/.

[3] Lawrence Berkeley National Laboratory. 2016. LBNL Node Health Check. https://github.com/mej/nhc/.