Online Diagnostics at Scale

Don Maxwell
National Center for Computation Sciences
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
mii@ornl.gov

Dr. Jeff Becklehimer
Cray Inc.
Oak Ridge, Tennessee, USA
jlbeck@cray.com

***Abstract*—The Oak Ridge Leadership Computing Facility (OLCF) housed at the Oak Ridge National Laboratory recently acquired a 200-cabinet Cray XK6. The computer will primarily provide capability computing cycles to researchers through the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. The OLCF has a tradition of installing very large computer systems requiring unique methods in order to achieve production status in the most expeditious and efficient manner.**

More often than not, new computer systems of this size employ the most recent processor, memory and interconnect technology available from computer vendors. While installation-to-production is an exciting time in the life cycle of a computer, fielding systems of this size brings its own set of challenges. New computer parts have neither time nor experience on their side. Statistically speaking, a large system will see many more early-life part failures than smaller systems, and only time spent running the system will cause those parts to fail. As for experience, new technology used in these machines has traditionally not made it to the general commodity market yet, so the benefit of having assistance in finding potential issues that might be lingering in a new design cannot be realized.

This paper will explore some of the methods that have been used over the years at the OLCF to not only address many of the issues noted above at installation but also ongoing issues that occur during the life of the machine. Experience has shown that parts can fail in many different ways causing different failure modes for applications. While some applications run just fine on a particular part, others fail every time on that same part. Traditional offline diagnostics are unable to detect these issues. However, ongoing online diagnostics can be very helpful in tracking down these issues and alerting systems staff to problem parts in the machine. Early detection can prevent job failures and ongoing frustration for users as they debug issues which are ultimately hardware failures. Algorithms and applications that have been developed for use in online diagnostics will be discussed as well as tracking and reporting mechanisms. Ongoing and future work will also be discussed.

I. INTRODUCTION

Computer users have an expectation that running the same code on the same computer will generate the same results over and over. While this is generally true, computers are machines, and machines have parts that fail. When parts do fail, the goal is always to minimize the impact to the user with the ultimate goal being to hide the failure entirely from the user.

Given that hardware will ultimately fail at some point, software has to be designed to deal with these failures. There are many different levels at which software can be written to provide the user with the machine expected. Offline diagnostics are the first line of defense in screening parts for failures. These diagnostics are designed to test each component for failures and to stress parts to the point that marginal ones fail. Parts that have survived this round of testing then move on to the next phase of testing which involves a fully functioning operating system. The focus of this paper will be the online portion of testing with the operating system booted.

Some of these online tests are performed early in the life of the machine during the bring-up and acceptance testing while others are ongoing tasks that continue throughout the life of the machine. The algorithms and infrastructure needed to find and track parts that can fail in unconventional ways are not complicated but require some thought and effort. Many common tools already used at the OLCF including MySQL databases and the Simple Event Correlator (SEC) package are being employed to provide the tracking and logic needed to attack this problem.

Following this process to its logical conclusion, once a defective part is found, the next task is to replace it. The new XK6 with the Gemini interconnect provides the opportunity to replace defective parts while the machine continues to run user jobs. This requires that the machine be quiesced and the Gemini network rerouted. Generally, this should just work, but experience has proven that at times, it does not, which leads to a reboot of the machine. Methods for attempting to ensure a successful machine reroute as well as the actual warmswap procedure itself will

be discussed to complete the lifecycle of the defective part in the machine.

II. Motivation and Experience

Experience in bringing very large systems online at the OLCF has shown that achieving a minimal state of stability and reliability is a multistep process that begins with processes designed and implemented by the manufacturer. Often these state-of-the-art supercomputers are the first to receive newly designed commodity processors, so the first concern is generally centered on the design and manufacture of the part. If there was either a flaw in the design of the part that some in the business have seen in past deployments or an issue in the manufacturing process, it can lead to delays which have both scheduling implications and financial impacts to the commodity part vendor, the supercomputer vendor, the consumer and ultimately the user of the computer. Not having realized the experience of the market at this early stage in the life of the part, the challenge becomes putting together the right processes at different levels to deal with the risk.

First, once parts are manufactured, there is an intensive screening process performed by the commodity part manufacturer to eliminate defective and weak parts. This process subjects the parts to environmental conditions outside normal operating boundaries using both thermal and electrical extremes. Using extremes, the goal is obviously to provide the consumer with a part that should operate properly under normal circumstances by eliminating weak parts. Further, the hope is that this process eliminates the need for further screening down the line by the consumer, but as many have learned, that is often not the case.

The next attempt at verifying the part is usually the responsibility of the supercomputer manufacturer. At times, schedules dictate that parts bypass the supercomputer vendor and instead ship directly to the final consumer, but that is obviously not the preferred path. At a minimum, parts will undergo a second round of screening once installed in the machine of their ultimate destination using a series of tests developed by the commodity manufacturer, the supercomputer manufacturer, and the computing community at large. These offline tests are used not only to identify defective parts but other problems such as issues with seating parts in the machine, etc. There are generally a few rounds of this testing performed over several days with a system of any significant size, but it is at this point that the online diagnostic process begins.

Cray has gathered a collection of applications from customers over the years that have traditionally stressed machines of days gone by in different ways. Certainly the Linpack benchmark is one of the most stressful applications that any new supercomputer will run and has historically been important to many sites. The online diagnostic process begins by booting the new machine into an operating system to provide an environment necessary for running Linpack and the suite of other applications assembled for diagnostic purposes. Ideally, after a few rounds of this testing are performed over several days, the end result is a productive machine; but again, experience says that this final goal has not been reached just yet.

The next step in the process introduces customer involvement with acceptance testing. During this phase, customer applications are developed on new hardware providing ample opportunity for any flaws and defects to be brought to the forefront. Given the fact that this is the first exposure of the customer to the hardware, it is no surprise that issues arise at this stage. Problems can present themselves in a variety of ways. Early-life failures that are typical in most supercomputers manifest themselves in the forms of complete hardware failures taking nodes down, segmentation faults on either all applications or a select few applications, and divergent answers.

III. Diagnostic Harness

A. Diagnostic Identification

In order to mitigate risk to users' jobs, the OLCF embarked upon a project to provide an ongoing online diagnostic with the goal of finding early-life failures before the users found them. Given that only certain applications seem to stress the machine is such a way to identify these nodes, applications had to be identified that have historically met this requirement and that could be run in a reasonable timeframe. Clearly, running these diagnostics takes compute cycles from the users, so a balance had to be achieved to minimize both run time and frequency.

During the initial thought process, the hope was to find an application or small set of applications that met the requirement of finding early-life failures and also ran in such a small amount of time that it would be reasonable to run them in the Cray NodeKARE™ package which provides the node health check service for the Cray XK6 machine. This package provides a mechanism for running site applications but unfortunately does not provide a mechanism for running a parallel MPI application. The check simply runs the application specified on each individual node in the reservation. While this did not meet the need for the diagnostic application, a batch system prologue or epilogue script could have accomplished a similar task, but ultimately, the real problem proved to be finding a short-running application that could identify the early-life failures.

The next best alternative to a short-running application running at the beginning or ending of each job is one that runs in a fairly short amount of time during batch processing. The application that seemed to meet all the criteria proved to be the Linpack benchmark. This application had identified many of the early-life failure nodes seen during several Cray X-series acceptance testing periods, it could scale to the size of the machine, and it ran in approximately 30 minutes with very predictable results making verification simple. While other codes were successful in identifying individual failed parts, Linpack

proved the most reliable in consistently finding the failures. As for the frequency to employ this simulation to test for parts failure, again a balance between compute cycles away from the users versus confidence in the machine had to be achieved. A 72-hour period between runs seemed to be a reasonable compromise of the two factors. Two runs per week minimizes the time taken from users but doesn't prolong the period between verification runs. Certainly, as confidence is gained in the machine, this period will increase and will ultimately just become a verification run after a maintenance period.

B. Identifying the Failure

Now that the application has been identified, it is necessary to determine what defines a failure and which node is responsible for the failure. For our purposes there are two types of failures.

A hard error is defined as a failure which results in the application exiting. In these cases identifying the responsible node is usually done by simple examination of the error logs. Hardware errors such as machine check exceptions are a good example of a hard error. Also, for this discussion we consider segmentation faults as hard errors. This is because the behavior of the application is well known and any type of error exit is considered abnormal.

A soft error occurs when the application completes normally but the resulting output does not match the expected output. In those cases identifying the responsible node is not possible using only one run. Instead we must introduce a twist into the process.

Rather than run one large application across the entire system we split the system into blocks of $\mathbf{N}^2$ nodes. For each block we run **N** jobs which are **N** nodes wide as portrayed in Fig. 1. If that block runs error free then the nodes are good.

If there is a soft error we regroup the nodes along the columns and repeat the run as in Fig. 2. Hopefully the failure will repeat. The faulty node is then identified by the node which intersects both sets of nodes involved in the failing jobs.

C. Tracking the Diagnostic Runs

In order to ensure that each node in the machine was verified in the appropriate interval of time, a simple MySQL database was created to track the last diagnostic runtime on each node in the system. Using this database described below in Table I, determining which nodes were eligible for a new diagnostic run became a simple matter of querying the database.

A few experiments were conducted to determine what combination of diagnostic jobs might be less intrusive for the users. In order to ensure the jobs got executed in a timely fashion, Moab® advance reservations were created based on the node list query from the database.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	21	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Run one job per row **N** *nodes wide*

Figure 1. One Job Per Row

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	17	24
25	26	27	28	29	30	21	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Run one job per column **N** *nodes wide*

Figure 2. One Job Per Column

Table I. MySQL Schema

Field	Type
last_runtime	Datetime
processor_id	int(10) unsigned
job_id	varchar(80)
Hostname	varchar(80)

Initially, the thought was that smaller diagnostic runs would allow more user jobs to complete around those waiting to run on particular nodes, but ultimately, the number of Moab reservations needed to accommodate these smaller runs became unmanageable and caused performance issues with Moab, making the idea impractical. The implementation used to accomplish the runs now uses the flowchart outlined in Fig. 3. To summarize the steps to identify failed parts: one large job runs, the database is queried to pick up the residual nodes not covered by the large job, a Moab reservation is created to cover those nodes, and a new job is submitted against that reservation. This has proven to be the most effective method to date.

IV. DIAGNOSING PERSISTENT SEGMENTATION FAULTS

A. The Problem

Persistent segmentation faults are difficult to detect and report in an automated fashion. Most prevalent during acceptance testing, segmentation faults can be another manifestation of early life failures but they can also simply be a programming error by the user. Differentiating a programming error from a hardware failure can be difficult.

A few patterns have been observed in console logs where segmentation faults are reported. First, a particular node can be seen reporting faults on several different applications typically from different users over a period of time. While this would seem to indicate that the node has defective hardware, the possibility exists that this could simply be different users in debug cycles with their codes that are coincidentally scheduled on this same node over time.

Next, a particular node has been seen generating segmentation faults on the same code from the same user on a very regular basis. This could also be interpreted as either the user debugging code or a node experiencing hardware problems. While this would seem to indicate a debugging cycle, the OLCF has found through several different generations of processors that particular codes can and do generate segmentation faults on particular nodes over and over while other codes run without problems on the same nodes. Codes that have experienced this behavior include LSMS, S3D, Madness and others. These codes cover several different scientific disciplines with very different coding practices and complexities, so it is believed that this unique behavior is simply a defect in the part caused by either a particular instruction or more likely a sequence of instructions unique to the code.

Finally, a node might report only one segmentation fault during a boot session. Again, how should this be interpreted – a code problem or a defective part? As presented above, particular applications can and do generate segmentation faults when no other codes experience the behavior, so making a definitive decision either way is challenging.

B. The Compromise

Given so many different possible interpretations of the cause of segmentation faults, how can an algorithm be developed to correctly diagnose each fault and associate it with the correct root cause? There is no distinction in the data reported for a segmentation fault generated by user error versus hardware defect, so an automated diagnosis becomes very difficult. Furthermore, a user engaged in a debugging cycle often doesn't know if problems being experienced are hardware or software. Human nature is to blame conditions beyond the control of the user for problems being experienced, so the likelihood of getting a meaningful diagnosis by including input from the user is probably very low.

With these circumstances in mind, the OLCF has decided to implement a compromise which attempts to capture segmentation faults due to hardware failures. It is a compromise since it attempts to limit the exposure of these failures to the users by making assumptions that cannot necessarily be confirmed. The algorithm monitors for applications that experience failures on a particular node multiple times while also using both context and thresholds to limit false positives. Nodes will be flagged as potential hardware failures when a single application experiences repeated segmentation faults on a particular node three times with no segmentation faults from any other node. This would seem to indicate that only that node is causing the application to fail since a coding problem should experience segmentation faults from other nodes as jobs move around the system. To cover the case whereby multiple applications could be experiencing failure due to defective hardware, a node that generates segmentation faults from different applications over a three-day period will also be flagged as a potential hardware failure. The hope is that statistical probability will identify defective parts while keeping false positives to a minimum.

The implementation of this algorithm will take advantage of the SEC infrastructure already installed on the system management workstation (smw). The OLCF has relied on this existing system to monitor and diagnose problems with the Cray X-series for several years now [1]. By adding new SEC rules that have the ability to both count faults and check context, the OLCF should be able to limit user exposure to hardware failures that manifest themselves as segmentation faults.

V. REMOVING FAILED NODES FROM THE SYSTEM

With the introduction of the Cray XK6, the OLCF now has the capability of removing failed parts while the system continues to run. The new Gemini interconnect has the ability to dynamically reroute the network unlike the previous generation Cray Seastar interconnect, so replacing failed nodes no longer requires a complete system downtime. However, the OLCF has occasionally experienced failed reroutes of the Gemini network when attempting to replace failed parts, so procedures have been developed to attempt to verify that the entire node replacement process will be successful. Once that verification has been accomplished, the warmswap procedure itself can be implemented.

A. Verifying Successful System Reroute

In order to remove a module from the system for repair, the network must be rerouted to divert traffic away from that module. This is done automatically when a module fails on a running system. However, just as any other software can fail, the reroute of the system can also fail under the right conditions, and the OLCF has experienced these failures. If the reroute fails, the entire system must be rebooted resulting in the loss of multiple jobs and wasted

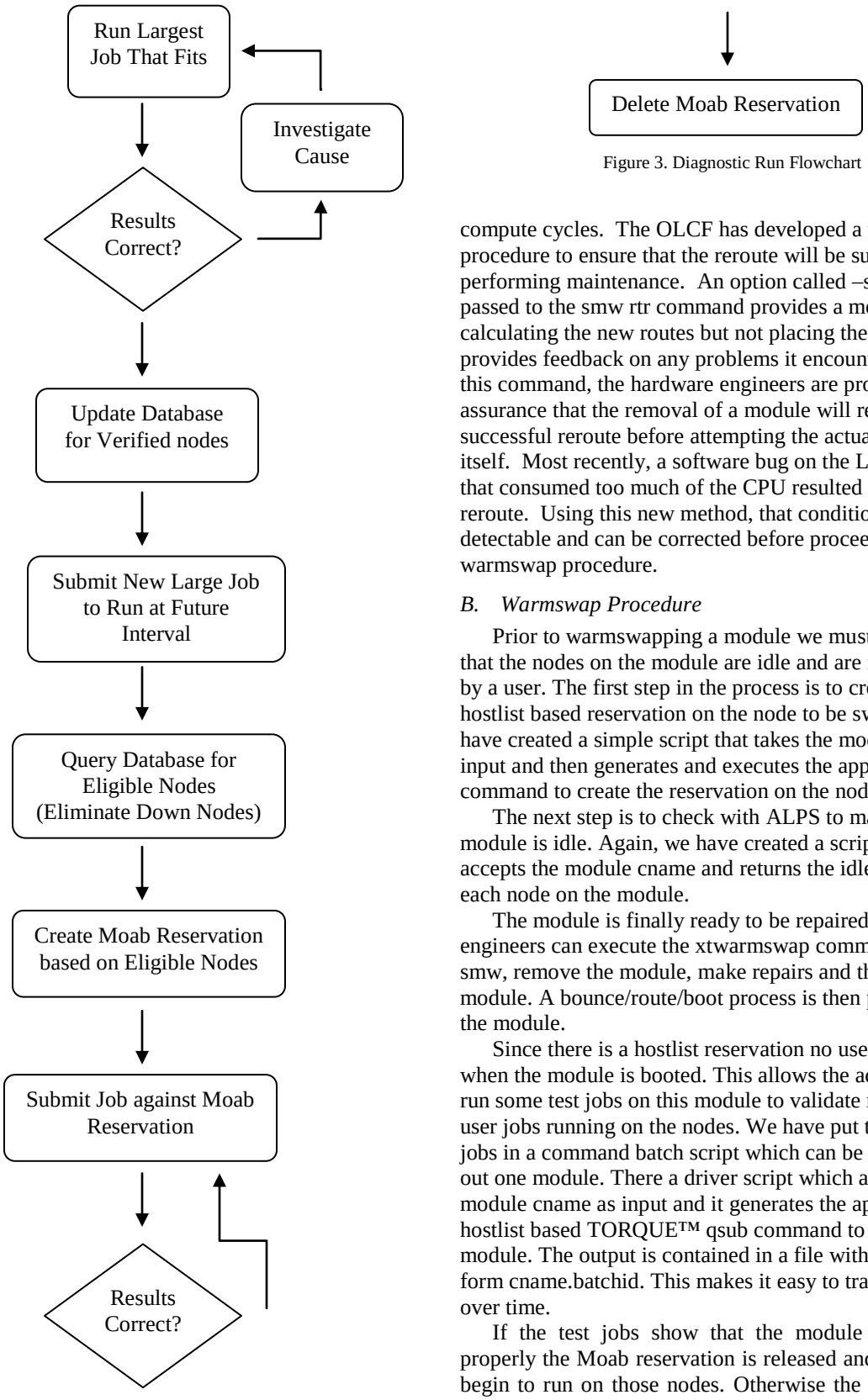

Figure 3. Diagnostic Run Flowchart

compute cycles. The OLCF has developed a very simple procedure to ensure that the reroute will be successful when performing maintenance. An option called –stage-routes passed to the smw rtr command provides a mechanism for calculating the new routes but not placing them and provides feedback on any problems it encounters. Using this command, the hardware engineers are provided some assurance that the removal of a module will result in a successful reroute before attempting the actual procedure itself. Most recently, a software bug on the L0 controllers that consumed too much of the CPU resulted in a failed reroute. Using this new method, that condition is now detectable and can be corrected before proceeding with the warmswap procedure.

B. Warmswap Procedure

Prior to warmswapping a module we must first ensure that the nodes on the module are idle and are not reserved by a user. The first step in the process is to create a Moab hostlist based reservation on the node to be swapped. We have created a simple script that takes the module cname as input and then generates and executes the appropriate Moab command to create the reservation on the nodes.

The next step is to check with ALPS to make sure the module is idle. Again, we have created a script which accepts the module cname and returns the idle/busy status of each node on the module.

The module is finally ready to be repaired. The hardware engineers can execute the xtwarmswap command on the smw, remove the module, make repairs and then replace the module. A bounce/route/boot process is then performed on the module.

Since there is a hostlist reservation no user jobs will start when the module is booted. This allows the admins time to run some test jobs on this module to validate repairs prior to user jobs running on the nodes. We have put together a few jobs in a command batch script which can be used to check-out one module. There a driver script which accepts the module cname as input and it generates the appropriate hostlist based TORQUE™ qsub command to test the module. The output is contained in a file with a name of the form cname.batchid. This makes it easy to track modules over time.

If the test jobs show that the module is functioning properly the Moab reservation is released and user jobs will begin to run on those nodes. Otherwise the module can be returned to the hardware engineers for further work.

VI. CONCLUSION

In summary, early-life failures cause loss of productivity and frustration for users. At scale, these issues are compounded not only by the sheer number of parts, but also by issues that arise with software when attempting to run diagnostics at scale. Methods can be developed to minimize the impact of these problems to users, but a balance must be struck between providing confidence in the system and taking computing cycles away from the users. Work continues at the OLCF to improve diagnostics along with their runtimes in anticipation of incorporating runs into prologues at job launch. With a proven diagnostic and a small runtime, both goals of minimal impact and system confidence could be achieved for the life of the system.

REFERENCES

[1] J. Becklehimer, C. Willis, J. Lothian, D. Maxwell, and D. Vasil, "**Real Time Health Monitoring of the Cray XT Series Using the Simple Event Correlator (SEC),"** Cray User Group Conference 2007.