Methodologies for Advance Warning of Compute Cluster Problems via Statistical Analysis: A Case Study[*]

Jim Brandt
Sandia National Laboratories
MS 9159, P.O. Box 969
Livermore, CA 94551 U.S.A
brandt@sandia.gov

Ann Gentile
Sandia National Laboratories
MS 9152, P.O. Box 969
Livermore, CA 94551 U.S.A
gentile@sandia.gov

Jackson Mayo
Sandia National Laboratories
MS 9159, P.O. Box 969
Livermore, CA 94551 U.S.A
jmayo@sandia.gov

Philippe Pébay
Sandia National Laboratories
MS 9159, P.O. Box 969
Livermore, CA 94551 U.S.A
pppebay@sandia.gov

Diana Roe
Sandia National Laboratories
MS 9152, P.O. Box 969
Livermore, CA 94551 U.S.A
dcroe@sandia.gov

David Thompson
Sandia National Laboratories
MS 9159, P.O. Box 969
Livermore, CA 94551 U.S.A
dcthomp@sandia.gov

Matthew Wong
Sandia National Laboratories
MS 9152, P.O. Box 969
Livermore, CA 94551 U.S.A
mhwong@sandia.gov

ABSTRACT

The ability to predict impending failures (hardware or software) on large scale high performance compute (HPC) platforms, augmented by checkpoint mechanisms could drastically increase the scalability of applications and efficiency of platforms. In this paper we present our findings and methodologies employed to date in our search for reliable, advance indicators of failures on a 288 node, 4608 core, Opteron based cluster in production use at Sandia National Laboratories. In support of this effort we have deployed OVIS, a Sandia-developed scalable HPC monitoring, analysis, and visualization tool designed for this purpose. We demonstrate that for a particular error case, statistical analysis using OVIS would enable advanced warning of cluster problems on timescales that would enable application and system administrator response in advance of errors, subsequent system error log reporting, and job failures. This is significant as the utility of detecting such indicators depends on how far in advance of failure they can be recognized and how reliable they are.

Catagories and Subject Descriptors: C.4 [Computer Systems Organization]: Performance of Systems – *fault tolerance, reliability, availability, and serviceability*; D.2.8 [Software Engineering]: Metrics – *complexity measures, performance measures, process metrics*; G.3 [Probability and Statistics]: Statistical computing

General Terms: Reliability, Performance

Keywords: reliability, RAS, fault tolerance, failure prediction

[*]This work was supported by the United States Department of Energy, Office of Defense Programs. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

1. INTRODUCTION

Resource failures on high performance compute (HPC) platforms can be catastrophic for large MPI based scientific simulations as MPI is currently not robust to loss of compute resources and such loss typically results in the application hanging until external intervention, typically in the form of killing the application and re-queuing it, takes place. Because, to date, such failure events have been asymptomatic the only way for an application to make progress is to checkpoint based on the statistical probability of resource failure given the number of resources being employed by the application and the historic resource failure rate of the platform being used. Additionally, since it is generally unknown (see Section 2) which resource(s) are more likely to fail, checkpoints must be saved for all compute resources which may, for large applications, require substantial network bandwidth, storage bandwidth, and time.

Our tool OVIS [4], in development at Sandia National Laboratories, for scalable data collection, analysis, and visualization of high performance compute platforms has been deployed on Sandia's Glory cluster, an Opteron quad core based cluster with an infiniband interconnect. This is a proof of concept deployment on a production HPC platform to explore our premise that anomalous behavior, in particular, statistical outliers with respect to system characteristics, could provide early indication of problems that would eventually manifest themselves in resource failure. Defining statistical outliers is a difficult problem as many system characteristics are convolved with others that can't easily be extricated and used for normalization. One example is the complex interaction between CPU temperature, CPU utilization, fan speed, and cooling air temperature where CPU temperature is driven by CPU utilization, intake air temperature, and fan speed as well as other confounding factors such as voltage and clock speed. Another example, and one that we address in this particular case study, is memory utilization by an application, whose memory requirements are a priori unknown to the system, convolved with operating system memory needs and perhaps current or pending I/O memory needs. In order to deal with these different scenarios OVIS employs a variety of analysis mechanisms such as descriptive statistics, multi-variate correlative analysis, and Bayesian inference which are discussed further in Section 3.

This paper is divided as follows: Section 1 gives an introduction and the motivation for this work, Section 2 discusses related work, Section 3 discusses our approaches and methodologies, Section 4 covers the Glory case study to date, and Sections 5 and 6 contain summary and conclusions.

2. RELATED WORK

Although HPC systems have grown tremendously in scale and complexity, the underlying component robustness has not improved significantly [8]. Therefore as systems grow and application runs span larger numbers of components, the mean time to failure of some component involved is decreased. There is work (e.g., [8, 12]) that projects such failures to be so frequent on exa-scale systems that large applications utilizing substantial fractions of the platforms will not be able to make forward progress as they will be unable to perform enough work and checkpoint it before a failure occurs. By enabling preemptive action targeting only affected resources, failure prediction, if accurate, could significantly improve overall system and application performance for large HPC systems.

The majority of predictive work done in this area has focused on classification of types of failures, their symptoms, and their temporal distributions. Leangsuksun, Scott et al [10] have analyzed failures on the ASC White machine at LLNL in order to come up with an algorithm for calculating a predicted time to failure for a system of nodes each with a known failure history. Oliner, Sahoo et al [13] have done a study on LLNL's BlueGene/L similar in nature. Fu and Xu [9] have analyzed nine years worth of failure and log data collected at LANL [1] and written a framework for utilizing this data to classify failure types and their temporal distributions. Using this framework allows classification of recent history behaviors and hence prediction of related failure distributions over near future time windows. Schroeder et al [14] have also done an in depth study and modeling work on the same data sets in order to classify and provide models for failure types and modes on HPC systems. Schroeder et al [15] also have done work in not only failure rate projection but also strategies for coping with such failures in peta-scale sized platforms. This body of work uses historic failure data collected from log files to make mean time to failure (MTTF) predictions which can increase system and application performance by allocating resources based on an application's size and expected run time and allowing the application to optimize its checkpointing strategy based on the projected MTTF for that pool of resources.

There is another body of work that attempts to discover impending failure of resources on a per resource basis based on discovering signatures of pre-failure conditions and detecting them in real time with sufficient time margin for the system and/or application to react before the failure occurs. Stearley and Oliner [16] take this approach in their use of frequency analysis and sorting of words and word groupings from system log files. Sahoo et al [11] explore using occurrences of non-fatal events in log data to predict fatal ones. Our OVIS project (see for example, [4] and references therein), rather than using log files to look for signatures associated with failure, uses raw data such as voltages, temperatures, fan speeds, memory utilization etc. in order to discover behaviors of raw events that may either singly or as an aggregate give indication of failures earlier than log data, the latter of which by nature means that a reportable event has already occurred. We only use log files as a guide to what kinds of failures occur and hence the combinations of metrics in which to look for such signatures. The premise of our work is that anomalous behavior in these metrics could be a precursor to resource failure [7] and used within an information-sharing infrastructure to provide warnings and resource management decisions [5].

Beyond the work to utilize data, whatever the source, to predict failures or failure trends, is the work supporting this in the form of data sources and access to these. These range from tools such as Ganglia [3] to failure repositories being set up to give researchers access to historic data to help them solve these difficult problems. In addition to the LANL repository cited above, Schroeder et al [15] have put up such a repository called the Computer Failure Data Repository [2].

3. APPROACHES AND METHODOLOGIES

The OVIS user interface, shown in Figure 1 (last page) consists of a components list (on the left), each of which has a sub-list of metrics associated with the component. In the case of Glory the only component with associated metrics currently is the node. A rendering of the system components in a 3D physical view allows for coloring the components based on associated metrics either raw or derived. This physical view (in the center) can be zoomed, translated, and rotated. This visualization allows the user to easily identify patterns (e.g. thermal gradients) as well as extreme outliers in the raw metrics (e.g. all but one component at extreme of color scale if auto-scaling) or relative to a calculated model (e.g., outliers outside of a probabilistic threshold are colored red, while those within are colored green; alternatively, outliers may be colored blue or red to indicate if they are outside of the probabilistic range due to the high vs low values relative to the model). It also can enhance understanding of how events unfold temporally as it allows the user to scroll through time and play back historic data. An analysis pane is shown on the right.

The current analytic capabilities of OVIS are described below though few of them were utilized in this case study due to the nature of the problem (only required analysis of one metric). It is expected, however that other major problems such as power supply failure and CPU failure could benefit from several of these capabilities used in conjunction. Besides the particulars of the statistical analyses described below, the overarching features of OVIS that make the rest truly useful for real time monitoring and analysis are its scalability and robustness to failures. OVIS uses a distributed database for both collection and parallel analysis. The underlying communication mechanisms for analysis are not MPI based and are robust to individual failures in the distributed system. Loss of one database entity (if the distributed feature is being used) results only in loss in fidelity of the result(s) and not loss of a result. (For details of the OVIS architecture see [6].)

There are currently three types of analyses supported by OVIS, each having the option to be run using either the learn or monitor modes of operation[1]. In learn a *model* is calculated or inferred from unmodified data. Such a model can take several forms, such as statistical moment estimators, PDFs, *etc.*. In monitor the roles are here interchanged with those of the learn mode: the data is now assessed with respect to a given model. The output of the monitor mode is a collection of *outliers*, described in a way that allows for unambiguous and efficient retrieval of the particular components and times to which these correspond; the output may also be presented as an ordered list so as to reflect a gradation in severity or abnormality of behavior. The output may also be seen in the physical view, where components values at the displayed time can be compared to the calculated model and colored accordingly. Note that reportable cases may occur either when a particular event diverges from the model by more than what has been set as acceptable or because no (or fewer than specified) events of a particular type occurred. For instance, outliers – which may be defined in several ways depending on the type of model being used – can be identified as elements of the data set that deviate from what the model predicts within pre-defined acceptability bounds.

Within this framework, the currently available engines are the following:

Descriptive statistics: In learn mode, descriptive statistics of the data set of interest are calculated (estimators of the mean, standard deviation, skewness, kurtosis, as well as bounds). These statistics can be interpreted directly by the user, or be used as input parameters to the monitor mode of the descriptive engine itself or even of another engine, e.g., to complement expert knowledge prior to Bayesian parameter estimation. In monitor mode, relative distance in terms of mean and standard deviation (which, as indicated, *may* be the result of a prior learn stage) is the criterion according to which outliers are detected, based on user-specified probabilistic thresholds.

Multivariate Correlative statistics: The goal of this engine is to seek anomalous behaviors by calculating (in learn mode) or devising (with "expert knowledge") multivariate correlation statistics, via mean vectors and covariance matrices – and thus, implicitly, a multiple linear regression model – for a set of tuples of variables of interest, and examining (in monitor mode) how individual observations of these tuples of variables of interest deviate from the aforementioned model. Such deviations are characterized in terms of the the multivariate Mahalanobis distance computed with the mean vector and covariance matrix. This is especially useful to prevent the user from conducting more advanced and costly analysis such as running a Bayesian engine when linear correlation between metrics can be evinced.

Bivariate Bayesian: In learn mode, the parameters of a probabilistic model that describes the dependency of a metric on another are inferred from the input data viewed as *training data*. In this mode, a parametric model as well as a prior must be provided to the analysis engine before the calculation can proceed; the descriptive and correlative analysis engines are useful here since they allow the user to come up with a "first cut" that is not completely uninformed – thus ensuring faster parameter identification and/or better accuracy. The monitor mode calculates the likelihood of the data as it is sifted through the model with its parameter values provided as an input, for instance after they have been calculated in learn mode with "trustable" training data. Details of the Bayesian engine methodology are outside the scope of this paper, but suffices to say that it is here in the context of this paper to allow the user to construct models of spatial gradients due to environmental effects but is applicable to modeling of other bi-variate dependencies.

[1]Note that a third mode, validate also exists but is currently implemented only for the Bayesian engine and is thus not discussed here.

4. GLORY CLUSTER CASE STUDY ON MEMORY RELATED FAILURES

Sandia's Glory cluster is a 288 node, 4608 core Opteron cluster with an Infiniband interconnect. Discussions with the system administrators and examination of log files indicated that the majority of failures in this system stem from three causes — power supply failure, running out of memory, and a core ceasing to function. Of these failure modes the most prevalent appears to be running out of memory hence this paper describes our memory related analysis and results for this platform.

As mentioned in Section 3, OVIS collects numeric data from many sources. In the case of Glory, due to the nature of the instrumentation, we collect all numeric information in band from the following sources: LM sensors — voltages, temperatures, fan speeds, /proc — memory related metrics, and EDAC — memory error metrics on a per-Dimm basis. Because we collect data in band we are limited by the system administrator to a collection period of once a minute so as to not perturb running applications. Additionally we collect text data from Glory's SLURM database as well as console logs to discover application/node mappings and failed nodes/application runs. The latter information is needed in order to correlate anomalous behaviors/characteristics in the numeric data with failures in the system.

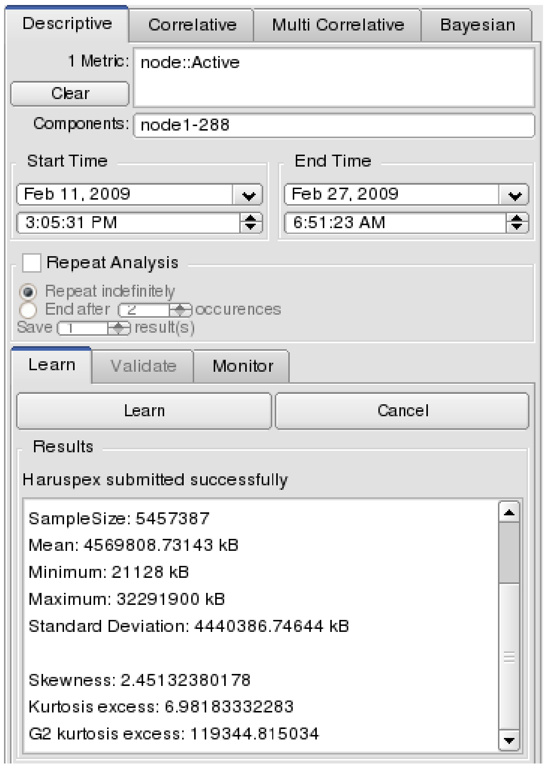
The end goal is to discover anomalous behavior in some metric(s) that precede failure of a resource far enough in advance and with enough reliability that knowledge of the behavior could allow the system and/or application to take pro-active action to save appropriate state and initiate a lower impact recovery than is traditionally possible by discovery at or after the time of failure.

In this section we discuss the process by which we established that there is such an anomalous behavior, that the behavior is indeed an indicator of the failure that is occurring, and that there are statistical analyses that can be used for automated discovery of indicators that precede the failure on reactable timescales.

4.1 Establishment of the Behavioral Precursor of Failure

4.1.1 General Statistical Consideration

Since we were looking for metric behavior indicative of an impending out of memory condition we chose to first investigate the Active Memory metric taken from /proc/meminfo.



1 Metric: node::Active
Components: node1-288
Start Time: Feb 11, 2009 3:05:31 PM
End Time: Feb 27, 2009 6:51:23 AM
Repeat Analysis
Repeat indefinitely
End after 2 occurences
Save 1 result(s)

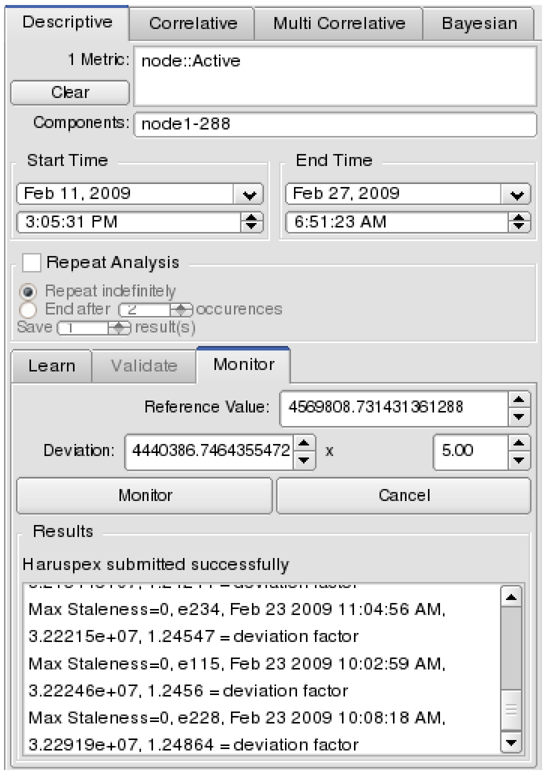
Results

Haruspex submitted successfully

```
SampleSize: 5457387
Mean: 4569808.73143 kB
Minimum: 21128 kB
Maximum: 32291900 kB
Standard Deviation: 4440386.74644 kB

Skewness: 2.45132380178
Kurtosis excess: 6.98183332283
G2 kurtosis excess: 119344.815034
```

Figure 2: Learn tab for descriptive statistics, component type "node", and metric Active Memory. The Components box and the Start and End times respectively specify which components (all 288 nodes in this case) and time interval over which data will be used as an input to the analysis. Results of the analysis are displayed in the Result box.



1 Metric: node::Active
Components: node1-288
Start Time: Feb 11, 2009 3:05:31 PM
End Time: Feb 27, 2009 6:51:23 AM
Repeat Analysis
Repeat indefinitely
End after 2 occurences
Save 1 result(s)
Reference Value: 4569808.731431361288
Deviation: 4440386.7464355472 x 5.00

Results

Haruspex submitted successfully

```
Max Staleness=0, e234, Feb 23 2009 11:04:56 AM,
3.22215e+07, 1.24547 = deviation factor
Max Staleness=0, e115, Feb 23 2009 10:02:59 AM,
3.22246e+07, 1.2456 = deviation factor
Max Staleness=0, e228, Feb 23 2009 10:08:18 AM,
3.22919e+07, 1.24864 = deviation factor
```

Figure 3: Monitor tab for descriptive statistics. Outliers are displayed in the Result boxes. The outlier threshold (number of standard deviations) is specified in the box to the right of Deviation. A value outside of $(ReferenceValue) \pm (ThresholdxDeviation)$ will be flagged as an outlier

ActiveMemory was used in preference to MemFree, the latter of which includes cache and Inactive memory in addition to the application's Active Memory.

Using data collected over a sixteen day period we first performed a descriptive statistics "Learn" analysis (described in Section 3) on the Active Memory metric over this whole period and over all nodes (Figure 2) irrespective of what was running on the cluster. The reason for using this simple single variable analysis first (as opposed to using the multivariate correlation engine and all memory related metrics) is that it is computationally lighter weight and involves fewer database accesses. A subsequent "Monitor" analysis (Figure 3) established a list of nodes that were exhibiting outlier behavior in the Active Memory metric and at what times that behavior occurred to direct us in further analysis. Note that error log file information would only give us times of the actual out of memory (OOM) event, and not of previous outlier behavior. Note also that outlier behavior in this metric considered alone is not sufficient to establish it as a meaningful abnormality, because the memory utilization is strongly dependent upon the job and machine state.

4.1.2 Time-Dependent Behavior

We next investigated the temporal behaviors of Active Memory on nodes seen to be outliers in the aforementioned analysis (Figure 3) where we defined an outlier as anything greater than or equal to five standard deviations from the group mean. The following shading conventions are used in all Active Memory plots referenced in this section: yellow shaded regions represent time periods in which the nodes being represented have no job running on them, pink shaded regions represent a period over which there was a job scheduled that resulted in a failed job event in the scheduler logs, and gray shaded regions represent a time period over which a job either ran to completion or was canceled by the user. The Y axis on all plots is the fraction of available system memory that is designated as Active. The X axis is the time (Note that this is Pacific time, the time zone of the authors, rather than the time zone of the Glory cluster which is in Mountain time).

Figure 4 is a plot of Active Memory on Glory node 228, which was flagged as an outlier in the previous analysis, over a time period when it was both flagged as an outlier in this metric and had an OOM event. During the time preceding the first idle region a particular user, designated here as user **A**, has an application running on this node (unshaded). Subsequent to the first idle region, **A** continues to run applications on this node (alternates with another user)(unshaded) until the rightmost idle region.

There are two items of interest in this case. One is that during the rightmost idle period an OOM event, represented by the red x, occurs on this node resulting in the death of a process belonging to **A** even though all state instantiated

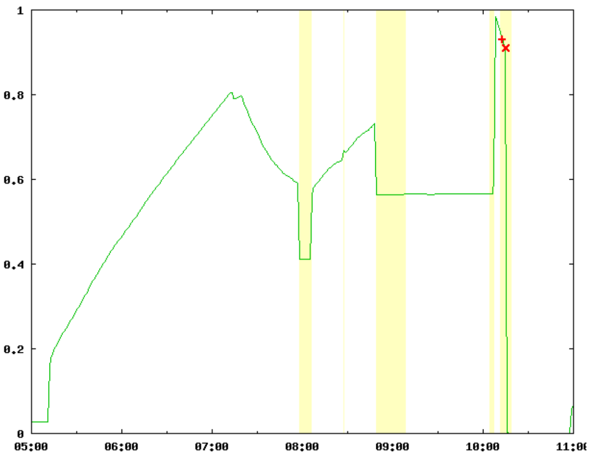


Figure 4: Active Memory for Glory node 228 (shown in green) normalized to the total system memory (32GB) for the time interval of 0500 to 1100 on February 23, 2009. Idle periods on the nodes are shaded in yellow. Node not responding log events are indicated by a red +; out-of-memory events resulting in killed processes are indicated by the red x. Note that Active Memory remains high, even during idle periods, and that user processes are killed during idle periods.

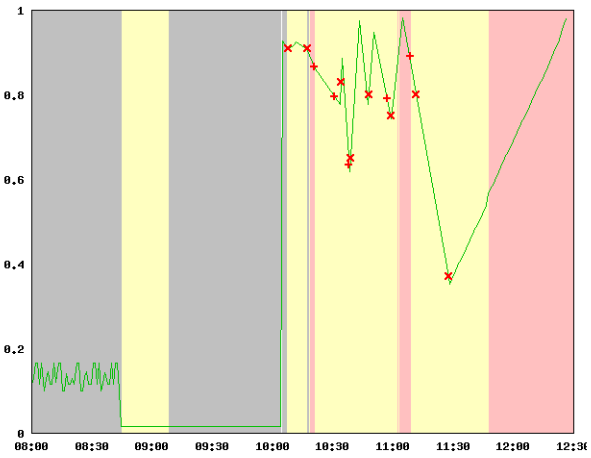


Figure 5: Active Memory for Glory node 234 (shown in green) normalized to the total system memory (32GB) for the time interval of 08:00 to 12:30 on February 23, 2009. Idle periods are shaded in yellow, canceled or complete jobs are shaded in gray, failed jobs are shaded in pink. Note that Active Memory remains high, even during idle periods, and that user processes are killed due to OOM events (red x) during idle periods.

by a user should have been removed upon exit. The second is that during all idle periods, until the OOM event, this node's Active Memory value remains unexpectedly high.

In Figure 5[2], a similar plot for Glory node 234 which was also flagged as an outlier during this period, it can be seen that, as on node 228, there is a high Active Memory utilization during some of the idle periods. Additionally there are many OOM events occurring during the idle regions on the right side of the figure, many of which involve the death of processes belonging to an earlier user, also **A**, but which include system processes as well. Most jobs landed on this node in this region of time fail (pink regions) and are not those of user **A**.

Finally, Figure 6 shows four nodes, one (node 279, green) of which was used by yet another user **C** (during the time shaded in dark gray) and left in an idle state with a high amount of Active Memory. Upon a subsequent job of user **D** shared by this and three other nodes, the Active Memory on node 279 is abnormally high compared with that of the 3 other nodes. Note that the increase in memory is the same for all four nodes involved in **D**'s job as one would expect for a well-balanced job.

In this case the high amount of Active Memory while in an idle state does not result in an OOM related death for the follow-on job because the memory requirements for that job did not cause the Active Memory value to reach the OOM threshold, and hence **D**'s job was able to run to completion.

These examples indicate that some users' processes and/or allocated memory are not being properly cleaned up by the system upon job completion/exit resulting in unintentionally high Active Memory utilization during subsequent jobs.

[2]A error in the shading of this figure that does not affect the results has been corrected in this version after publication.

While these initial jobs may complete normally, follow-on jobs may not. The combination of the memory utilization of the previous processes and the current job's processes can result in sufficient memory utilization to invoke the OOM condition.

We do not yet know if this is a system problem (memory leaks) triggered by some user memory access patterns or is simply a problem of cleanup after job completion (everything not belonging to root should be terminated but is not). However, our goal in this work is to identify precursors of this failure scenario that can be discovered automatically, and with enough advance warning to allow for mitigating intervention.

4.1.3 Advanced Automated Discovery of Potentially Failure Conditions

In order to achieve automated advanced discovery of this type of problem, we cannot use as an indicator merely a memory threshold for flagging problems because we do not know a priori the expected memory requirements of any particular job.

Thus there are two possible methods that we can use. The easiest would be to look for Active Memory outliers during idle time and address these by appropriate system cleanup (reboot if necessary) before allocation to job requests. In Figure 7 it is clear that four of the fourteen nodes that share an idle time (shaded in yellow) are abnormally high in their Active Memory metric. Descriptive statistical analysis of the Active Memory of all fourteen nodes during this time returns a mean value of $3.6GB$ with minimum utilization of 0.2 G and maximum utilization of $15GB$ and a standard deviation of $5GB$. In contrast, a similar analysis over the only the ten well behaved nodes during the same period results in a mean value of $0.7GB$ with minimum utilization of $0.2GB$ and maximum utilization of $1.2GB$, with a stan-

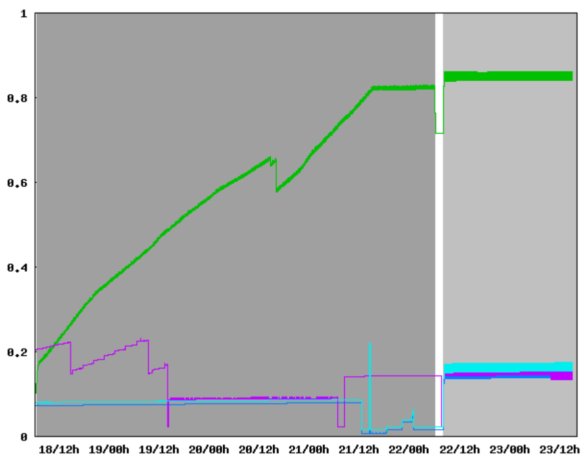


Figure 6: Active Memory normalized to the total system memory for the time interval of noon February 18, 2009 through noon February 23 for node 279 (shown in green). The dark gray time range corresponds to a previous job on node 279 only. The time range unshaded (white) corresponds to idle time on node 279. Note that Active Memory is abnormally high on node 279 during the shared job.

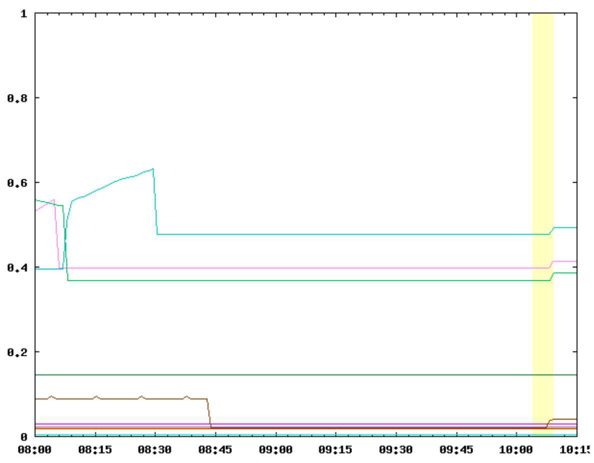
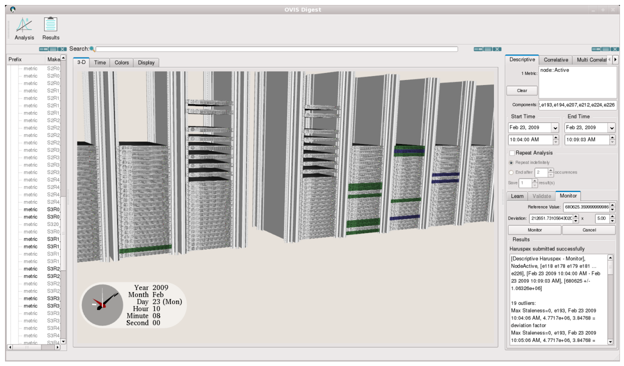


Figure 7: Active Memory during a shared idle time (shaded in yellow) on a subset of nodes normalized to the total system memory (32GB) on the day of February 23, 2009. Note that active memory is high during a subset of nodes during the idle time.



1 Metric: node::Active

Components: ,e193,e194,e207,e212,e224,e226

Start Time: Feb 23, 2009 10:04:00 AM

End Time: Feb 23, 2009 10:09:03 AM

Reference Value: 680625.359999999986

Deviation: 212651.7310564302 x 5.00

Haruspex submitted successfully

[Descriptive Haruspex - Monitor], NodeActive, [e118 e178 e179 e181 ... e226], [Feb 23 2009 10:04:00 AM - Feb 23 2009 10:09:03 AM], [680625 +/- 1.06326e+06]

19 outliers:
Max Staleness=0, e193, Feb 23 2009 10:04:06 AM, 4.7717e+06, 3.84768 = deviation factor
Max Staleness=0, e193, Feb 23 2009 10:05:06 AM, 4.7717e+06, 3.84768 =

Figure 8: OVIS Screenshot of monitor analysis to determine outlier behavior in the nodes in Figure 7. Nodes shown in green fit the learned model; nodes shown in blue have Active Memory values outside the allowable range for the model. Analysis results correspond to those obtained in Figure 7.

dard deviation of $0.2GB$. In the first case, mostly due to the fact that the distribution is highly skewed towards the right, in particular because of the relatively large number of true outliers on that side, using standard deviations as a means to evince outliers is not conclusive: for instance, only a single true outlier lies farther than two standard deviations off the sample mean. In the second case, however, all four true outliers are flagged as outliers even with a threshold of five standard deviations from the mean. Using the second case as a reference distribution we use OVIS's monitor mode and display to show the nodes flagged as outliers in this sample group (see Fig. 8). In general application of this method we expect that with consideration of all nodes the majority would be well behaved during idle and so we would expect the reference distribution to resemble the second case (i.e., reasonably low mean and small standard deviation).

High values of Active Memory during idle time, either in comparison with other idle nodes, or learned models of Active Memory usage on idle systems, can indicate nodes with unintentionally excessive Active Memory before a subsequent job is placed upon that node, allowing appropriate action to be taken to clear the condition and thus avoid future failure. This type of analysis could have discovered the problem in node 228 shown in Figure 4 at the time of the first idle period in that figure, more than 2 hours before the first indication of a problem in the log file.

The other method would be to assume that most distributed applications are reasonably balanced (possibly excluding MPI rank(0)) in Active Memory across all compute resources in which case doing the same type of analysis would flag problematic nodes early on in the application run, as was the case in the Figure 6. Although this method may detect meaningful outliers, practical application of this method has two drawbacks: 1) the statistical methods employed rely on a statistically significant sample size and hence would only work well for jobs employing a reasonably large number of resources and 2) pro-active response would require a mechanism for feeding this information back to the running application (vs. the idle time analysis that, through interaction with the scheduler, would prevent a job from landing on a problematic node).

In either case, both methods will require the monitoring system to directly incorporate resource state information in order to perform such ongoing analysis (an OVIS feature in development).

Such monitoring and analysis when used to evaluate potential compute resources for allocation in terms of their Active Memory metrics relative to a reference distribution learned over all idle nodes can provide a probabilistic basis for "healthy" (in terms of this metric) node allocation. In the cases studied over this sixteen day period such health based allocation would have eliminated the majority of OOM related job deaths (there were a few that were due to applications that were attempting to use more memory than was available) and thus increased system efficiency.

5. SUMMARY

We have investigated out of memory (OOM) related failures in Sandia's Glory cluster, a 288 node quad core Opteron cluster with an Infiniband interconnect, using hardware related metrics collected over a sixteen day period (using our monitoring, analysis, and visualization tool OVIS), system logs, and resource manager logs in order to determine whether or not there are mechanisms that could be used to give advanced warning of impending OOM related failures in time for the system to react. We discovered that the majority of OOM related failures occurring on the cluster over this period could have been prevented by not allocating resources exhibiting anomalous behavior in the Active Memory metric during the idle time preceding the allocation. Further we discovered that after various users' codes exit large amounts of Active Memory are still retained by the compute resources. Resources left in such a state correlate well with OOM related job failure on subsequent jobs. Finally, we have shown that statistical consideration of the Active Memory utilization over the cluster, over groups of nodes sharing jobs, and over idle time, is a methodology that can be used for automated outlier detection that would give indication of this problem in advance of failure on affected nodes.

6. CONCLUSIONS

We have discovered advance indicators of OOM related problems that can be utilized in various ways by the system, application, or a combination to largely mitigate these problems; however, we have not discovered the root cause of these OOM related problems. Some work should be done to try to discover the root cause of the problem and if possible fix it. In the absence of such an investigation and fix the next best thing would be to detect it during idle time on resources so that the condition can be cleared before the resource is allocated. In order to detect the condition, one could set an absolute threshold for Active Memory based on average memory use over a large number of nodes known to be idle (such as during system time after a system reboot). This method could run into difficulty as memory use may change during operation or due to to system configuration changes. A more robust method would be to sample this metric on each node upon job completion when it enters the idle state and build a learned model (which can evolve with system changes) from these samples over the whole cluster. Setting a probabilistic threshold relative to this learned model would allow flagging of outliers that would not be released into the pool of allocatable resources until it had been returned to a normal operational state.

7. ACKNOWLEDGMENTS

The authors would like to thank Jerry Smith of Sandia National Laboratories for his assistance in deploying OVIS on the Glory cluster and providing access to and direction in the use of system tools, databases, and log files.

8. REFERENCES

[1] Data used in the referenced study was obtained from. `http://www.lanl.gov/projects/computerscience/data/`, 2006.

[2] COMPUTER FAILURE DATA REPOSITORY. `http://cfdr.usenix.org`.

[3] GANGLIA. `http://ganglia.info`.

[4] OVIS. `http://ovis.ca.sandia.gov`.

[5] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pébay, D. Thompson, and M. Wong. Using probabilistic characterization to reduce runtime faults on hpc systems. In *Proc. of the 8th IEEE Symposium on Cluster Computing and the Grid (Workshop on Resiliency in High-Performance Computing)*, Lyon, France, May 2008.

[6] J. Brandt, A. Gentile, B. Debusschere, J. Mayo, P. Pebay, D. Thompson, and M. Wong. OVIS 2: A robust distributed architecture for scalable RAS. In *Proc. of the 22nd IEEE International Parallel & Distributed Processing Symposium (4th Workshop on System Management Techniques, Processes, and Services)*, Miami, FL, Apr. 2008.

[7] J. M. Brandt, A. C. Gentile, Y. M. Marzouk, and P. P. Pébay. Meaningful automated statistical analysis of large computational clusters. In *IEEE Cluster 2005*, Boston, MA, Sept. 2005. Extended Abstract.

[8] J. T. Daly. Performance challenges for extreme scale computing. `http://www.pdsi-scidac.org/publications/`, Oct. 2007.

[9] S. Fu and C.-Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Proceedings of SC'07*, Reno, NV, Nov. 2007.

[10] N. R. Gottumukkala, Y. Liu, C. B. Leangsuksun, R. Nassar, and S. Scott. Reliability analysis in hpc clusters. In *Proc. of High Availability and Performance Computing Workshop 2006*, Sante Fe, NM, Oct. 2006.

[11] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo. Bluegene/l failure analysis and prediction models. In *Proc. of the 2006 International Conference on Dependable Systems and Networks*, Philadelphia, PA, June 2006.

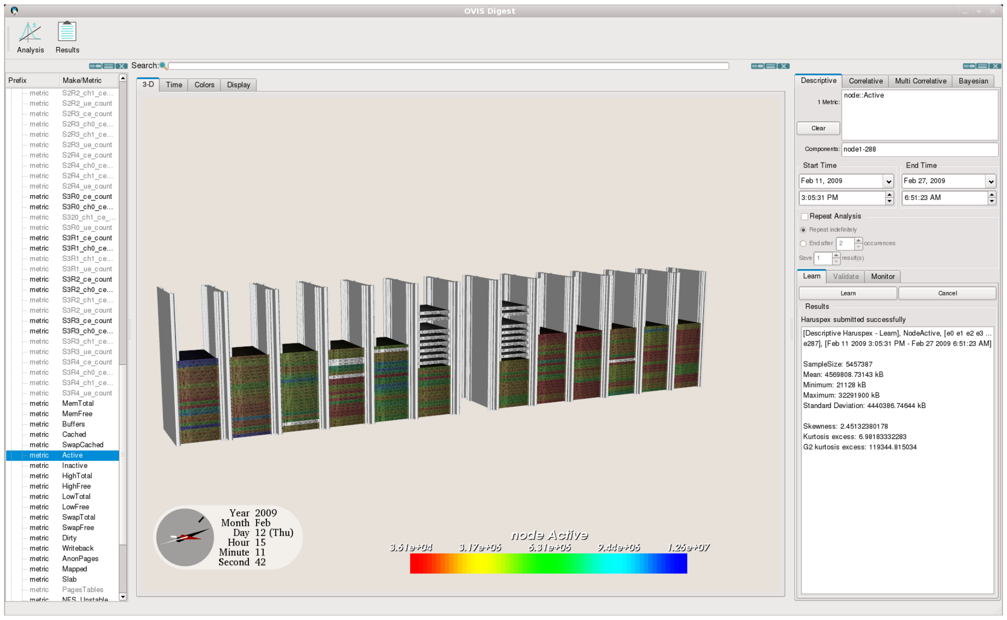
[12] R. A. Oldfield. Investigating lightweight storage and overlay networks for fault tolerance. In *High Availability and Performance Computing Workshop*, Santa Fe, New Mexico, Oct. 2006.

[13] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for blue gene/l systems. In *Proc. of the International Parallel and Distributed Processing Symposium*, Santa Fe, NM, Apr. 2004.

[14] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proc. of the 2006 International Conference on Dependable Systems and Networks*, Philadelphia, PA, June 2006.

[15] B. Schroeder and G. A. Gibson. Understanding failures in petascale computers. In *Journal of Physics: Conf. Ser. 78*, June 2007.

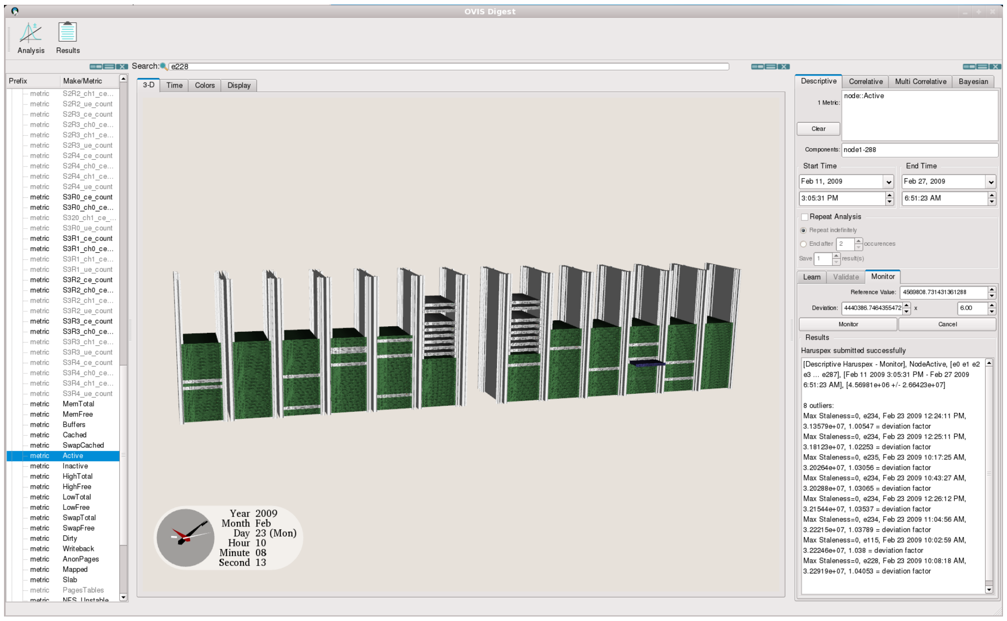
[16] J. Stearley and A. Oliner. Bad words: Finding faults in spirit's syslogs. In *Proc. of the 23rd International Parallel and Distributed Processing Symposium (Workshop on Resiliency in High-Performance Computing)*, Lyon, France, May 2008.



```
Descriptive | Correlative | Multi Correlative | Bayesian
1 Metric: node::Active
Components: node1-288
Start Time: Feb 11, 2009  3:05:31 PM
End Time: Feb 27, 2009  6:51:23 AM
Repeat Analysis
Repeat indefinitely
End after 2 occurences
Save 1 result(s)
Learn | Validate | Monitor
Results
Haruspex submitted successfully
[Descriptive Haruspex - Learn], NodeActive, [e0 e1 e2 e3 ...
e287], [Feb 11 2009 3:05:31 PM - Feb 27 2009 6:51:23 AM]

SampleSize: 5457387
Mean: 4569808.73143 kB
Minimum: 21128 kB
Maximum: 32291900 kB
Standard Deviation: 4440386.74644 kB

Skewness: 2.45132380178
Kurtosis excess: 6.98183332283
G2 kurtosis excess: 119344.815034
```



```
Descriptive | Correlative | Multi Correlative | Bayesian
1 Metric: node::Active
Components: node1-288
Start Time: Feb 11, 2009  3:05:31 PM
End Time: Feb 27, 2009  6:51:23 AM
Repeat Analysis
Repeat indefinitely
End after 2 occurences
Save 1 result(s)
Learn | Validate | Monitor
Reference Value: 4569808.731431361288
Deviation: 4440386.7464355472 x 6.00
Results
Haruspex submitted successfully
[Descriptive Haruspex - Monitor], NodeActive, [e0 e1 e2
e3 ... e287], [Feb 11 2009 3:05:31 PM - Feb 27 2009
6:51:23 AM], [4.56981e+06 +/- 2.66423e+07]

8 outliers:
Max Staleness=0, e234, Feb 23 2009 12:24:11 PM,
3.13579e+07, 1.00547 = deviation factor
Max Staleness=0, e234, Feb 23 2009 12:25:11 PM,
3.18123e+07, 1.02253 = deviation factor
Max Staleness=0, e235, Feb 23 2009 10:17:25 AM,
3.20264e+07, 1.03056 = deviation factor
Max Staleness=0, e234, Feb 23 2009 10:43:27 AM,
3.20288e+07, 1.03065 = deviation factor
Max Staleness=0, e234, Feb 23 2009 12:26:12 PM,
3.21544e+07, 1.03537 = deviation factor
Max Staleness=0, e234, Feb 23 2009 11:04:56 AM,
3.22215e+07, 1.03789 = deviation factor
Max Staleness=0, e115, Feb 23 2009 10:02:59 AM,
3.22246e+07, 1.038 = deviation factor
Max Staleness=0, e228, Feb 23 2009 10:08:18 AM,
3.22919e+07, 1.04053 = deviation factor
```

Figure 1: Top: OVIS GUI provides the analyst with components and associated metrics (L), geometrically correct physical 3D representation of the cluster (C), and access to analysis engines (R). Metrics are dragged onto either the physical view or the analysis box for display or processing. Bottom: The Glory cluster with the results of Monitor coloring the cluster. Outliers on the high side are colored blue whereas outliers on the low side (non-existent in this case) are colored red and everything else is colored green. Since an outlier at one time may not be an outlier for all time, all nodes flagged as outliers in Figure 3 do not show up at the same time on the physical view.