SAND2007-8007C

Using Probabilistic Characterization to Reduce Runtime Faults in HPC Systems

Ann Gentile*, Jim Brandt*, Philippe Pébay°,
David Thompson*, Matthew Wong°, Bert Debusschere†, Jackson Mayo°
Sandia National Laboratories
MS *9152 / °9159 / †9051
P.O. Box 969, Livermore, CA 94551 U.S.A.
{gentile,brandt,pppebay,dcthomp,mhwong,bjdebus,jmayo}@sandia.gov

***Abstract*—The current trend in high performance computing is to aggregate ever larger numbers of processing and interconnection elements in order to achieve desired levels of computational power, This, however, also comes with a decrease in the Mean Time To Interrupt because the elements comprising these systems are not becoming substantially more robust. There is substantial evidence that the Mean Time To Interrupt *vs.* number of processor elements involved is quite similar over a large number of platforms. In this paper we present a system that uses hardware level monitoring coupled with statistical analysis and modeling to select processing system elements based on where they lie in the statistical distribution of similar elements. This approach will enable the scheduler/resource manager to deliver a close to optimal set of processing elements given the available pool and the reliability requirements of the application.**

I. Introduction

Since it appears that High Performance Computing (HPC) system elements will continue to have unexpected failures for the foreseeable future, the HPC community has been putting significant effort into building fault tolerant systems. Fault tolerance work is ongoing in system software, scheduling and resource management, and application software. Perhaps the most important fault tolerance mechanism utilized in current HPC systems is the ability to checkpoint and restart. Since use of this mechanism requires both time and system resources a lot of effort has been put into optimizing job size and checkpoint and restart intervals and strategies [1].

There has been substantial work [1], [2] done to characterize HPC platforms in terms of identifying gross system failure categories and average frequency of failures within these. These failures are grouped into hardware, software, application, I/O, human related, etc. categories, each with its own distribution of mean time to occurrence *vs.* system size. These characteristics are important because they allow system level failure models to be built. These models can be used for application node allocation and checkpoint frequency optimizations as well as for making projections for the needs of future systems.

A major impediment to scaling HPC systems is component mean time to failure. While the number of system components is undergoing exponential growth, the robustness of the components themselves seems to be relatively constant. The result is that as application runs become larger in terms of number of components involved, the time between checkpoints has to become smaller. To some extent this can be offset [1] by the speedup in checkpointing due to the larger number of participating elements in conjunction with faster file systems. There is, however, a practical limit to I/O bandwidth based on hard drive failure rates [2].

While we believe that fault tolerance in HPC can be enhanced by application awareness of underlying hardware characteristics, burdening the application programmer with low level hardware related data gathering and analysis is not practical. To date even system manufacturers have not tackled this problem and are still shipping monitoring packages with their systems that address only threshold-based fault detection which typically does not leave time between detection and failure for applications to respond.

The overall goal of this research is to increase application performance on HPC platforms through historic and runtime characterization of the underlying hardware in such a way that it can give meaningful guidance to the OS, the scheduler and/or resource manager (RM), and the application thus increasing the mean time to interrupt (MTTI) on a given number of nodes. In this paper we present a monitoring and analysis system that uses statistical and probabilistic characterizations of system elements in the context of their environments to infer the relative health of these elements. An API allows the system to be queried for a list of elements and associated characteristics from which the scheduler/RM can assign resources and an application can calculate checkpoint frequency and identify I/O resources to use.

This paper is organized as follows: first we introduce related work and its relevance. We then describe our technical approach and show how it can be used to choose job appropriate system elements in order to tailor the expected MTTI to the application, number of process elements, etc. as well as provide run time feedback to the application. We then present some preliminary findings based on our proof of concept deployment. Finally, we present planned extensions of our approach to additional aspects of the HPC reliability problem.

*°†Please note that the list of authors was randomly ordered and henceforth is not informative as to the relative importance of each author's contribution to this work.

II. RELATED WORK

There has been substantial work in the area of fault tolerance though most has focused on quantifying how MTTI scales as a platform increases in aggregate compute power and mechanisms to deal with the fact that such failures will occur.

Daly [1] has shown how run time efficiency can be optimized using historic Mean Time To Interrupt curves in conjunction with other system parameters such as I/O write rate, and Parallel Scaling. Schroeder and Gibson [2] have done extensive work in exploring failure data sets on existing large computational platforms. Their research shows that hardware failures account for more than 50% of node outages experienced over a 9 year period on 22 HPC systems at Los Alamos National Laboratory (LANL). They also show that failures increase as the number of CPU sockets in a system increases. They further make the case that it is likely that aggregate computational power for the foreseeable future will come from just such increases. Their exploration of current and alternative checkpointing techniques and associated pros and cons parallels those of others such as Oldfield [3]. All such work seems to draw the conclusion that current fault tolerance techniques, namely checkpoint and restart, in their current form coupled with the seemingly fixed relationship between number of CPU sockets and MTTI imply that run time efficiency will continue to fall as computational platforms become larger. While they all propose schemes that could increase the checkpoint and restart efficiency given some constraints on memory footprint on nodes and I/O bandwidth growth, there seems to be agreement that the per socket fault rate will hold steady.

There has also been work by Gottumukkala et al. [4] on using historical failure data from a particular platform to quantify reliability of nodes on that platform. This reliability data would in turn be used to drive application decomposition and node allocation in order to optimize job run time. The Coordinated Infrastructure for Fault Tolerant Systems (CIFTS) [5] initiative seeks to construct an extensive fault aware framework in order to address these through more open communications between all system components.

In contrast to these efforts, our research seeks to increase the MTTI for an application by hand picking, as it were, components whose relative probability of failure is minimized given the number of nodes requested, expected runtime, and available resource pool. We seek to quantify failure distribution characteristics as they relate to the real-time statistical distributions of parameter values of compute elements. Such characterizations would allow us to identify run-time system degradation issues without requiring extensive historical or platfrom specifc data. We believe that using this methodology can drive down the hardware fault related application interrupts on applications.

III. TECHNICAL APPROACH

We take a statistical approach to detection of elements having anomalous behavior. Since today's HPC systems have aggregations of thousands of identical components our expectation is that these components should behave similarly under within a similar environment. Our approach then is to statistically characterize how these elements behave and identify statistical outliers as elements being more likely to fail. In previous work [6] we have shown that for some problems this type of identification can indicate problems before they reach the failure stage.

Our approach then to this problem is to provide the low level element characterization tool and an interface for communication between it, the application, and the scheduler/resource manager. In this section we describe OVIS, our tool for element characterization, and our characterization methodology; we present motivations for our statistical characterizations to be used as the basis for resource usage decisions; and proposed interactions between Application, Scheduler/RM, an Allocation Evaluator, and OVIS.

This methodolgy allows us to target nodes based upon their component paramaters distribution characteristics, whether or not the parameter is known to be directly indicative of future failure or not. This is significant for dealing with cases where the causes of failure of a component are not well understood, or where the component is degrading in a way that is not reflected in historical failure records, such as due to aging or changing environmental conditions.

A. OVIS

This section first briefly describes our data collection and analysis engine. Next we cover its associated data collection mechanisms and the type of data currently being collected. Finally we cover its analysis engines' functionalities and how they can be used to characterize system element attributes.

OVIS [7], [8] is a distributed tool for scalable data collection and analysis. It is able to use both historic and run time data to characterize *reportable cases* in statistical and/or probabilistic terms, such as descriptive statistics, correlations, and parametric Bayesian models. In particular, on the basis of such characterizations, historic and run time data, each element's degree of "abnormality" can be quantified based on where it lies relative to an applicable model. The set of currently available analysis tools, a.k.a. *haruspices*[1] is by no means limitative; in fact, other haruspices are currently under development but are not discussed here. Currently a person setting up the analyses on a system specifies in terms of variance what they want defined as an "outlier". OVIS returns a running list of "outliers" which is updated whenever new data arrives. The distributions and models can also be dynamically modified to represent how new data may have modified them.

1) Data Collection: Getting data with which to characterize system elements is critical to this scheme. Since it is unknown, even by the system vendors, what characteristics of what

[1]Partially quoting the Wikipedia *Haruspex* entry: *"In Roman practice inherited from the Etruscans, a haruspex (plural haruspices) was a man trained to practice a form of divination called haruspicy, hepatoscopy or hepatomancy. Haruspicy is the inspection of the entrails of sacrificed animals, especially the livers of sacrificed sheep."*

elements are critical in diagnosing system health, we attempt to collect as much data as possible with as high a frequency as possible. Our hope is that we will be able to discover critical health metrics and an optimal collection frequency that will be a small subset of what is possible. Currently, depending on platform, we collect voltages, temperatures, fan speeds, NIC, and SMART disk controller data. These data are collected via SNMP, IPMI, in band, and proprietary samplers with a periodicity ranging from once a second to once an hour. If, however, this technique shows promise we envision having a standard set of metrics that we would approach the HPC vendors about exporting using a standard out of band mechanism.

2) Element characterization methodology: A problem with the traditional HPC system monitoring tools is that they only provide information when a fault has already occurred or a critical threshold has been crossed necessitating a management system forced fault (*e.g.*, a shutdown or reboot). In either case the information is too late and the only useful outcome of having a management system at all is that it can cause the scheduler/RM to kill and restart the job as opposed to having it possibly hang until it times out or the owner notices it hasn't made progress. OVIS by contrast uses a variety of analysis engines to characterize the element's parameters in probabilistic terms based on their statistical distributions given their environments. The advantage of this is that, for failures that are evidenced by degradation in some element parameter, a measure of relative probability of anomalous behavior can be used to estimate stability of the element. Thus not only can the system be used to identify probabilistic outliers but conversely can identify relatively normal or well behaved elements for use in an application run. One of the goals of the OVIS project is to be able to quantify the probability of failure in a given time window for an element based on where it lies relative to its reference distribution and historic failure data of such elements. Thus a simple example would be that an application requesting 1000 nodes for 1000 hours might be handed nodes from a current free list starting with the one whose parameters weighted distance from the mean is the minimum and working out from there. Conversely an application requesting 10 nodes for an hour might well be handed the 10 whose parameters weighted distance from the mean was the greatest without being categorized as an outlier (outliers, being defined as elements whose paramenter values lie outside of acceptable bounds with respect to a reference model or distribution, would be removed from the "available resource pool" by the resource manager).

Additionally, unlike the afore mentioned traditional monitoring systems, since OVIS can track the runtime status of these parameters relative to their reference distributions and their starting point, it can notify the application of increased probability of failure based on a shift. This would allow the application to checkpoint a particular node's data and request a replacement upon completion of the next time step. If, with the addition of historic failure data, one can correlate distance from mean of some reference distribution with quantification of failure probability in a time window then not only could the scheduler be given the information to hand an application the currently optimal set of resources but the application could re-adjust its checkpoint frequency based on the allocated resource characteristics.

3) Analysis engines: Data analysis in OVIS 2 is conducted by the means of statistical engines, or *haruspices*, that can operate in parallel in order to address large data sets. The *execution mode* of these haruspices – in other words, the types of tasks that such engines may perform – can conceptually be classified as follows:

- Learn: in this mode, data is viewed as an absolute reference, from which a *model* is calculated or inferred. Such a model can take several forms, such as moments, estimators, PDF, *etc.*. The output of this execution mode is thus a model, or, more specifically in the context of OVIS 2, the most likely set of parameters of the model given the training data.
- Validate: here, data is still viewed as an absolute reference, but a model is now available; the goal is then to assess – and this assessment can be conducted in a variety of ways – the adequacy of this model to the data. The validate mode thus outputs the result of this assessment. Note that this *can*, but does not have to, be a number.
- Monitor: the roles are here interchanged with those of the learn mode: the unquestionable reference is now the model, with respect to which data is inspected. The output of the monitor mode is a collection of *reportable cases*, described in a way that allows for unambiguous and efficient retrieval of the particular components and times to which these correspond, when available; the output may also be presented as an ordered list so as to reflect a gradation in severity or abnormality of behaviour. Note that reportable cases may occur either when a particular event diverges from the model more than what has been set as acceptable (as is the case with the three haruspices detailed below), or because no (or fewer than expected) 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 abstract framework, OVIS 2 currently implements three haruspices. Note that not all of them implement each of the three conceptually possible execution modes. These haruspices are:

- **Descriptive statistics:** this haruspex offers learn, and monitor execution modes. In learn mode, descriptive statistics are calculated (estimators of the mean, standard deviation, skewness, kurtosis, as well as bounds) in order to provide a purely descriptive statistical characterization of the data set of interest. This information can be used *per se*, or be fed into the monitor mode, or be used elsewhere than within the descriptive statistics haruspex – for example, as a helper to specify priors for Bayesian parameter estimation. In monitor mode, the user can specify purely descriptive parameters – which may or may not come from an earlier execution of the learn mode – such as: nominal value and acceptable deviation wherefrom, and acceptable range. In particular, this mode enables outlier detection for two possible definitions (and use cases) of this concept: namely, variation from a value considered as "correct" or "normal", and thresholding. The latter use case alone replicates what current cluster monitoring tools typically offer. Note that there is currently no validate mode for descriptive statistics haruspices – but there may be one in the future.

- **Correlative statistics:** this haruspex is purely contemplative at this point, since it currently only implements the learn execution mode: the goal is here to evince linear correlation between two different metrics. 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. Even though a monitor mode has not been implemented yet, it will be the case soon since it is potentially of great interest to be able to report when 2 correlated variables begin to decorrelate – or, the other way around.
- **Bivariate Bayesian:** this haruspex implements all 3 execution modes within the context of parametric Bayesian inference modeling. For instance, 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 that mode, a parametric model as well as a prior must be provided to the haruspex before the calculation can proceed; the descriptive and correlative haruspices become here typically useful since they allow to user to come up with a "first cut" that is not completely uninformed – thus ensuring faster parameter identification and/or better accuracy. In validate mode, the haruspex conducts the comparison of the data of interest *vs.* the provided model along with a set of parameters, and returns a number that can be interpreted either as a sign that the model is not valid in this context (if one "trusts" the data – or defines as the norm for instance during a calibration process), or as an indication that the data as a whole is problematic (if one "trusts" the model instead – for instance if it has been inferred under comparable conditions). Finally, the monitor execution 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. For concision, the details of this methodology are not provided here, since this is clearly outside the scope of this paper, and only a brief illustration is presented in Figure 1. Meanwhile, the interested reader will find a complete explanation (in the context of OVIS 1) in [7].

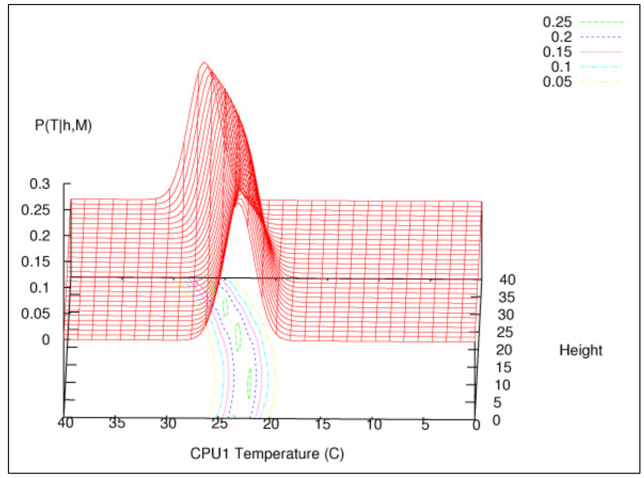

Fig. 1. Probabilistic model for CPU Temperatures in a particular rack in a particular cluster. The model consists of a quadratic mean as a function of relative height, with a Gaussian noise term about the mean. The parameters of the quadratic function and the standard deviation of the Gaussian are determined by bivariate Bayesian inference.

B. Statistical Characterizations for Resource Management Decision Support

We seek to utilize the statistical characterizations in order to make resource management decisions. As mentioned previously, our statisical methodolgy allows us to target nodes based upon their paramater distribution characteristics, whether or not the parameter is known to be directly indicative of future failure or not. If a particular parameter has not been conclusively tied to a failure mode, a node exhibiting statistical irregularly in that parameter can still be assigned to short-lived or low-priority jobs, for example, short initial test runs. Failure associations can then be discovered without pre-emptively taking any such nodes out of the pool, but with minimal impact to the higher-value jobs.

We list some examples of run-time statistical characterizations that we have performed that can be used to assist in resource management decision support. These are pre-failure diagnostics and would not be caught by methodogloies relying on historical failure data logs.

1) In our analysis of a particular HPC platform at Sandia, by using the descriptive statistics engine, we found that voltages of several thousand like components had approximately normal distributions overall. There were, however, consistently two statistical outliers, one six standard deviations from the mean and the other ten. There is currently no known relationship between this voltage abnormality and failure. In our proposed system, since there is the possibility of timing or other problems resulting from the deviations we would keep these nodes in the resource pool to be used for known short lived jobs. In addition, we would continue to actively monitor these nodes to see if this parameter can be used as an indicator of impending failure. If so, then in the future, nodes having significant outlier characteristics in this parameter might then be removed from the pool entirely.

2) In another example we found a node having a CPU

which, under load, had it's temperature fall five standard deviations below the mean of it's reference model. While this may not seem bad given that low temperatures (above the dew point) are generally viewed as good for the longevity of electronic components, it turned out to be indicative of a fan controller failure.

3) Getting disk data on the storage appliance we are monitoring via the SMART ctl we will be able to detect the gradual degradation of all spindle bearings based on the reported time to spin up. If one should degrade faster it would become an outlier and viewed as a less reliable component. This would result in a hint from the underlying monitoring system to applications as to its degraded degree of reliability.

C. Application, Scheduler/RM, OVIS *interaction*

This is in the planning phase and we are currently writing a proof of concept implementation and collecting reference data on several platforms. We describe here the currently planned architecture.

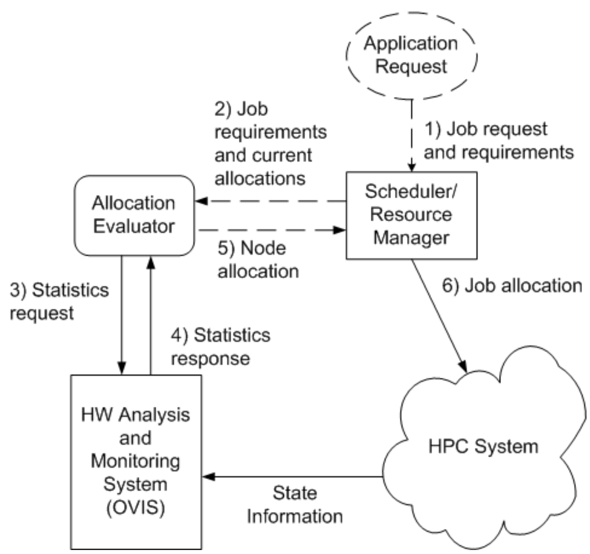

Fig. 2. Diagram of proposed interaction of Application, Scheduler/ResourceManager, and Analysis/Monitoring System (OVIS) by which an application submission can specify job requirements, a statistical analysis is done to determine nodes that will best satisfy that request in context of all the other jobs in the cluster, and the job allocation will occur.

A block diagram of interaction is illustrated in Figure 2. An application request can specify job requirements to the scheduler/RM. The requirements can be as simple as expected runtime, or if supported could include other requirements such as minimum acceptable MTTI. These more advanced requests would of course require something downstream to have access to applicable data coupled with the ability to perform the required calculations. The scheduler then passes the requirements on to an Allocation Evaluator. The Allocation Evaluator can query the Analysis System for relevant statistics. In the case of a high priority, long lived job, this may translate into a request for nodes with parameter characteristics closest to the mean of the distribution for a number of element parameters. The Analysis System supports an API by which a probabilistic distribution of nodes with respect to the requested constraints can be requested and returned. The Allocation evaluator can then weigh the returned statistics in order to determine the node allocation to recommend to the scheduler. In the event that there are not enough statistically satisfying nodes available, a job could be held until more satisfying nodes become available. An upper bound on this time could be determined by the current node allocation and current job time limit.

In Figure 2, dashed lines indicate pieces or interactions in development; solid lines indicate pieces or interactions which are currently in existence. Our Allocation Evaluator is a script that is evoked in the scheduler's prologue script. It currently passes the request to the HW Analysis system and chooses nodes either from "best to worst" or "worst to best" based on a requested node hour threshold. We currently define an API for invoking and returning descriptive and correlative statistics within OVIS.

The proof of concept system can determine and report system degradation to enable preemptive resource reallocation. This is illustrated in Figure 3. In this case, we are given a set of statistical requirements that the HPC system must satisfy in order to be considered in good health. The Analysis and Monitoring System calculates running statistics on the quantities of interest and report when the statistics have changed in a way that indicates system degradation. This information can be reported to the scheduler and these nodes taken out of the assignable pool of nodes. Applications that provide an interface for checkpointing can be instructed to checkpoint, or checkpoint more frequently, when degradation occurs.

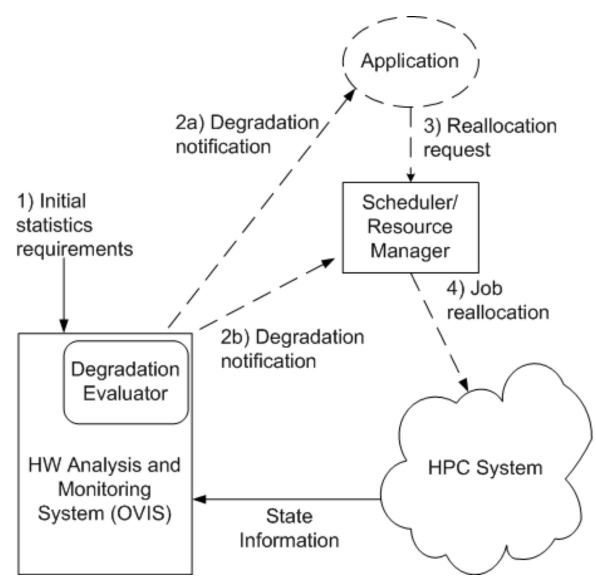

Fig. 3. Diagram of proposed interaction of Application, Scheduler/ResourceManager, and Analysis/Monitoring System (OVIS) by which system degradation can be determined and used to perform resource reallocation.

IV. Preliminary Findings

We currently have prototype deployments of the OVIS monitoring and analysis system on several production HPC systems at Sandia National Laboratories. Though we don't have enough failure data to make correlations between failures and the failed node characteristics with respect to their reference models and/or distributions, we have been able to quantify a measure of relative probability for node related element parameters. We are able to query for lists of nodes based on how far a particular parameter for each lies from the mean of that parameter's reference distribution. We can also discover shifts in the distribution's characterisitics and shifts in a particular node's relative position within that distribution.. We also have a prototype deployment on a storage appliance from which we are able to, in addition to the node level information, get disk operation parameter information. Since we retain all of the data we can, as failure data comes in, start to do correlative analysis between failures and the underlying hardware characteristics.

V. Future Work

We are currently working on a time series analysis engine which will be part of this framework. This will allow temporal charicterization of system parameters with respect to the particular application utilizing the elements being charcarized. We would like to expand the scope to include detection of elements exhibiting abnormal temporal behavior with respect to the similar elements participating in a particular application run. Along these same lines we would like to explore how different application input parameters affect this behavior.

Additionally we would like to explore taking into account an applications dominent communication patterns when doing node allocation so as to optimize run time application performance. I/O and storage component usage for an application could also be driven by requested storage hardware hints requested by the application and passed to it.

VI. Conclusion

Though we are seeing an explosion in node counts in new HPC platforms the failure rate of individual components seems to be staying constant. This combination is driving the system MTTI down thus burdening applications with the ever increasing overhead of higher checkpoint frequencies which in turn decreases the computational efficiency. We present a scheme for changing this trend by driving the effective MTTI back up through hardware level monitoring, analysis, and characterization. Since, according to studies over a significant number of platforms, greater than 50% of these interrupts are due to hardware faults we believe this approach can have a significant impact.

References

[1] J. T. Daly, "Performance challenges for extreme scale computing," `http://www.pdsi-scidac.org/publications/`, SDI/LCS Seminar Series, Oct. 2007.

[2] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," *SciDAC 2007 J. Phys.: Conf. Ser.*, vol. 78, no. 012022, 2007.

[3] 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.

[4] N. R. Gottumukkala, C. B. Leangsuksun, and S. Scott, "Reliability-aware approach to improve job completion time for large-scale parallel applications," in *2nd Workshop on High Performance Computing Reliability Issues*. Austin, Texas: IEEE 12th International Symposium on High-Performance Computer Architecture, Feb. 2006.

[5] "CIFTS," `http://www.mcs.anl.gov/research/cifts/`.

[6] 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.

[7] J. M. Brandt, A. C. Gentile, D. J. Hale, and P. P. Pébay, "OVIS: A tool for intelligent, real-time monitoring of computational clusters," in *Proc. IEEE International Parallel & Distributed Processing Symposium 2006*, Rhodes, Greece, Apr. 2006. [Online]. Available: http://www.ipdps.org

[8] "OVIS," `http://ovis.ca.sandia.gov`.