Using Probabilistic Characterization to Reduce Runtime Faults in HPC Systems

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

These authors were 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.

Still, 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, if, indeed, they even receive notification.

Thus, the overall goal of our 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 and timely 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 proof of concept system for such characterizations and their utilization in resource management decisions. It includes 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.

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 fault tolerance 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 historic or platform specific data. We believe that use of 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. Today's HPC systems have aggregations of thousands of identical components. Our expectation is that these components should behave similarly 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. We then use these statistical characterizations in conjunction with application resource requirements to make resource usage decisions. This information is exchanged via an interface for communication amongst the the low level element characterization the application, and the scheduler/resource manager.

Our methodology allows us to target resources based upon their component parameters' distribution characteristics, whether or not the parameters are known to be directly indicative of future failure. 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 effects due to aging or changing environmental conditions.

In this section we first describe our tool, Ovis, for hardware analysis and element characterization, and our characterization methodology. We then present motivations for our statistical characterizations to be used as the basis for resource usage decisions. Finally, we present proposed interactions between an Application, Scheduler/RM, an Allocation Evaluator, and a Hardware Analysis and Monitoring System.

A. Hardware analysis and element characterization tool

For hardware analysis we use our distributed tool for scalable data collection and analysis, Ovis [7], [8]. Ovis uses historic and/or run time data to characterize hardware elements 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.

Currently, a person setting up the analyses on a system can specify in terms of standard deviation 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.

In the following subsections, we cover Ovis's data collection approach, its element characterization methodology, and its analysis engines' functionalities with emphasis on those aspects to be used in resource management decision support.

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 elements are critical in diagnosing system health, we attempt

to collect as much data as possible with as high a frequency as possible. Our goal is to discover critical health metrics and an optimal collection frequency with the hope that these will be a small subset of the the total 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. A simple example of the utility of this is in resource management 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 parameter 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 aforementioned 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. 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 is conducted by the means of statistical engines that can operate in parallel in order to address large data sets. The *execution modes* of these analysis engines – 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 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 behavior. Note that reportable cases may occur either when a particular event diverges from the model 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 abstract framework, OVIS currently implements three analysis engines. Note that not all of them implement each of the three conceptually possible execution modes. These analysis engines are:

- **Descriptive statistics:** this analysis engine 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 engine – 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, acceptable deviation, 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 the descriptive statistics engines.

- **Correlative statistics:** this analysis engine currently only implements the learn execution mode: the goal is 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 is not currently implemented 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 analysis engine implements all 3 execution modes within the context of parametric Bayesian inference modeling. 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. In validate mode, the analysis engine 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 defined 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. Details of the Bayesian engine methodology are outside the scope of this paper, but a brief illustration of a model generated in learn mode is presented in Figure 1 [8].

B. Statistical Characterizations for Resource Management Decision Support

We seek to utilize statistical characterizations in order to make intelligent resource management decisions. As mentioned previously, our statistical methodolgy allows us to target nodes based upon their parameter distribution characteristics, whether or not the parameter is known to be directly indicative of future failure. If a particular parameter has not been

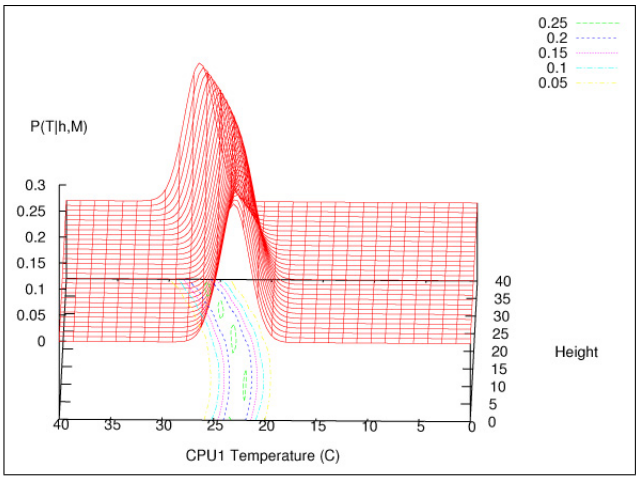

Fig. 1. Probabilistic model for CPU Temperatures in a particular rack in a particular cluster. The model consists of a Gaussian random variable whose mean is a unknown quadratic of relative height, and with unknown variance. The parameters of the quadratic function and the variance of the Gaussian were determined by the bivariate Bayesian inference engine using a small set of training data.

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 methodologies 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. We currently know of no relationship between voltage abnormality and failure for this element. 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, in the future, nodes having significant outlier characteristics in this parameter might then be removed from the pool for servicing.

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) We are currently monitoring a group of 160 hard drives

via smartctl [9]. This interface provides manufacturer defined thresholds for determining when disk parameters have degraded to a point that a disk should be removed from service. Consideration of a disk in singleton gives little to no indication if that disk's parameter's rate of approach to a threshold is reasonable, and, as the values are often expressed in scaled quantities, the values themselves are not intuitively comprehensible. Because our analysis system evaluates each disk's variable values in terms of the distributions for the whole set of disks, if a small subset were to degrade faster than the rest they would become an outliers and viewed as a less reliable components. This would result in a hint from the underlying monitoring and analysis system to applications as to their degraded degree of reliability.

Analyses such as those above would be performed by system engineers and administrators in order to discover system abnormalities and failure associations. The investigations would include both descriptive statistics upon the data collected in order to discover unsuspected issues (e.g., general temperature distributions in the system) and more targeted calculations based upon domain knowledge of potential issues in the system (e.g., geographical distributions that may arise as a result of the system set up and air flow design). As significant abnormalities and failure associations are discovered these can then be used as variables and parameters to be considered in determining resource allocations, as discussed in the next section.

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.

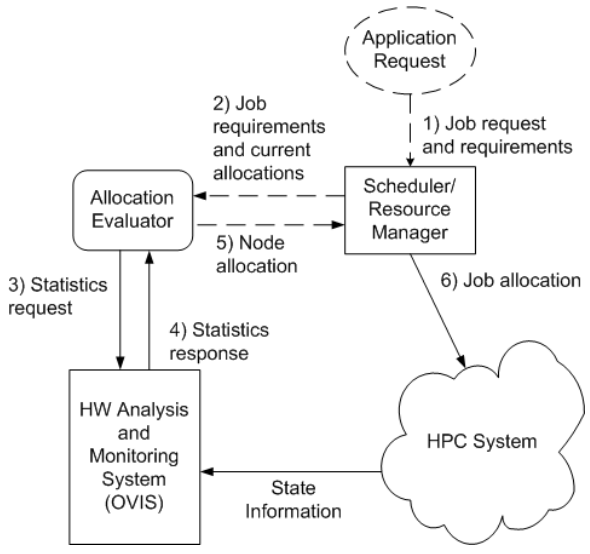

Fig. 2. Diagram of proposed interaction of Application, Scheduler/ResourceManager, and Analysis/Monitoring System (OVIS) by which an application submission can specify job requirements, a statistical analysis is done to determine nodes that will best satisfy that request in context of all the other jobs 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 an Analysis System with 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 queries the Analysis System for relevant statistics on resources meeting the parameters of the request. 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. It is not intended that the general user need have a detailed knowledge of the significant parameters for failure associations. Rather as those are determined by system administrators or engineers, significant parameters and values can be defined. Thus, a typical user would only have to indicate job priority, while the Allocation Evaluator would request statistics on the appropriate resources meeting predetermined probabilistic requirements. More system-aware users could still make higher complexity requests.

The Analysis System has 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 currently a script that is evoked in either the scheduler's prologue script or the job submission script. It 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 also determine and report system degradation with the intent of enabling preemptive resource reallocation. This is illustrated in Figure 3. In this case, there are a set of statistical requirements that the HPC system (or an assigned subset) must satisfy in order to be considered in good health. The Analysis and Monitoring System calculates running statistics on the quantities of interest and has an API to report when the statistics have changed in a way that indicates component degradation. This information is intended to be reported to the scheduler and the affected components 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. (Again, as in Figure 2, dashed lines indicate pieces

or interactions in development; solid lines indicate pieces or interactions which are currently in existence.)

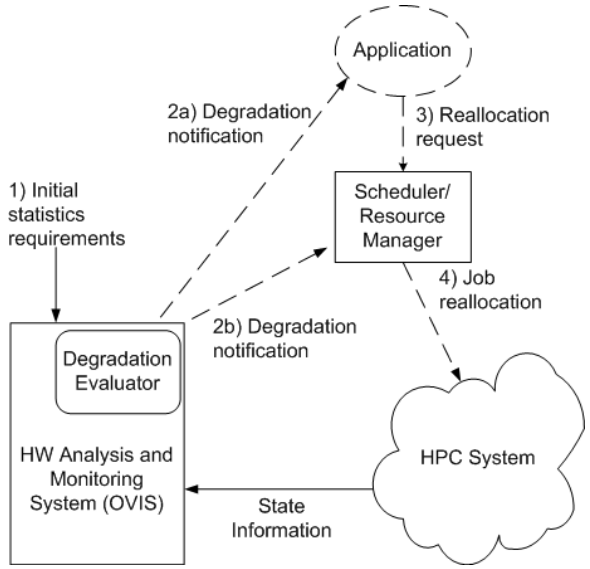

Fig. 3. Diagram of proposed interaction of Application, Scheduler/ResourceManager, and Analysis/Monitoring System (OVIS) by which system degradation can be determined and used to perform resource reallocation.

IV. Preliminary Findings

We currently have prototype deployments of the OVIS monitoring and analysis system on several production HPC systems at Sandia National Laboratories. Currently, we most frequently run in the learn and monitor modes in order to determine statistical distributions of run time values and detect outliers. Though we don't currently 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 characteristics and shifts in a particular node's relative position within that distribution. In our prototype deployment on a storage appliance 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 characterization of system parameters with respect to the particular application utilizing the elements being characterized. 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.

Finally we would like to explore taking into account an application's dominant 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. In this paper we present a scheme for changing this trend by increasing the effective MTTI 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 *Proc. IEEE 12th International Symposium on High-Performance Computer Architecture (2nd Workshop on High Performance Computing Reliability Issues)*, Austin, Texas, 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, Extended Abstract.

[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. 20th IEEE International Parallel & Distributed Processing Symposium (4th Workshop on System Management Techniques, Processes, and Services)*, Rhodes, Greece, Apr. 2006. [Online]. Available: http://www.ipdps.org

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

[9] "SMARTMONTOOLS," http://smartmontools.sourceforge.net.