Quantifying Effectiveness of Failure Prediction and Response in HPC Systems: Methodology and Example

James Brandt*, Frank Chen*, Vincent De Sapio*, Ann Gentile°, Jackson Mayo*,
Philippe Pébay*, Diana Roe°, David Thompson*, and Matthew Wong°
Sandia National Laboratories
MS *9159 / °9152
P.O. Box 969, Livermore, CA 94551 U.S.A.
ovis@sandia.gov

Abstract

Effective failure prediction and mitigation strategies in high-performance computing systems could provide huge gains in resilience of tightly coupled large-scale scientific codes. These gains would come from prediction-directed process migration and resource servicing, intelligent resource allocation, and checkpointing driven by failure predictors rather than at regular intervals based on nominal mean time to failure. Given probabilistic associations of outlier behavior in hardware-related metrics with eventual failure in hardware, system software, and/or applications, this paper explores approaches for quantifying the effects of prediction and mitigation strategies and demonstrates these using actual production system data. We describe context-relevant methodologies for determining the accuracy and cost-benefit of predictors.

1 Introduction

While many research studies have quantified the expected impact of growing system size, and the associated shortened mean time to failure (MTTF), on application performance in large-scale high-performance computing (HPC) platforms, there has been little if any work to quantify the possible gains from predicting system resource failures with significant but imperfect accuracy. This possibly stems from HPC system complexity and the fact that, to date, no one has established any good predictors of failure in these systems. Our work in the OVIS project [2–7] aims to discover these predictors via a variety of data collection techniques and statistical analysis methods that yield probabilistic predictions. The question then is, "How good or useful are these predictions?" We investigate methods for answering this question in a general setting, and illustrate them using a specific failure predictor discovered on a production system at Sandia.

2 Related work

Current MPI-based tightly coupled scientific simulation codes are inherently susceptible to process failure, particularly at large (tera-, peta-, exa-) scales with relatively short mean time to failure (MTTF). While checkpointing was a viable solution at smaller scale, it has become a significant bottleneck at larger scale. Research to date in this area has focused on optimizing checkpoint frequency using learned statistical distributions of failures within a system. This has been done at a system level using the learned MTTF to calculate an expected MTTF for a given fractional pool of resources [9]. There has also been work to determine MTTF on a per-node granularity, which can then be used to calculate a resource pool's MTTF or to allocate resources based on minimizing expected run time [15]. Work has also been done with the goal of predictive failure analysis on a per-resource basis using log-file analysis [16] as well as analysis of hardware-related metrics [4]. The utility of success in such work would lie in using this information to inform the system and application of impending failure, and to implement mechanisms in both enabling preemptive action to mitigate the consequences of the failure.

Work has also been done to investigate response methodologies without including an actual predictor. Scott et al. [13] have quantified response times where failure response consists of process-level virtual machine migration. Kalé et al. [8] have quantified the time savings of their Charm++ fast restart protocol over traditional checkpoint/restart. In general, such work does not include discovery of a failure predictor and hence cannot include quantification of prediction accuracy and cost.

Since to date there has been little success at identifying plausible failure predictors in HPC systems, little has been published on validation techniques for particular prediction methodologies. Some work [11], with respect to hard drive failures, uses receiver operating characteristic (ROC) curves to display true vs. false positive prediction rates. Inclusion of the cost of action/inaction is called out as what would be used for decision-making, though no specifics are given as it would vary depending on the actual installation. Vieira [17] discusses the need for validation of failure prediction methods and posits that this could be accomplished by comparing the precision and recall of various prediction algorithms.

Our preliminary formulation of failure prediction as a binary-classification problem takes advantage of much previous work on the general question of evaluating classifier performance. Originating in radar signal-detection theory, the concepts of confusion matrices and ROC curves are widely used in medical diagnostics, psychology, information retrieval, and machine learning [10, 12, 14]. A form of this approach has been pursued for HPC monitoring based on log files [16], but that work poses the primary task of failure *detection* (not necessarily before-the-fact prediction) and also does not attempt to directly quantify the costs and benefits of using a given classifier.

3 Our approach

The specific validation results reported here are based on an observed out-of-memory (OOM) failure mode in an HPC system, attributable to residual unreleased memory from completed jobs, as described further in Section 4.1. Before giving a more detailed account of the observations and results for this case study, we offer some general remarks on the validation of HPC failure prediction to provide a framework for present and future analyses.

3.1 Overview

Validation metrics should be guided by the intended applications of failure prediction and the objectives of decision-makers. The costs of action or inaction by an HPC prediction-response mechanism appear in various forms: waste of computing time, hardware, and labor. Because such costs are difficult to determine, it is appropriate to consider more general metrics, such as ROC curves, that reflect multiple objectives to be given specific weights later.

The simulated performance of various failure-prediction approaches can provide guidance on the implementation tradeoffs among data dimensionality, collection frequency, computational cost, and storage cost.

It is beneficial to have metrics that can confirm progress in modeling even before the results reach the stage of practical usefulness. A statistical model may have some predictive ability, but not enough for cost optimization to justify taking action based on it. Such a model is still a candidate for further development to enhance its performance.

3.2 Methodologies

The setting of HPC failure prediction poses intricate problems for validation. Tractable validation requires capturing a meaningful approximation of costs and benefits in a representation simple enough to treat with established statistical methods. A plausible representation, particularly for the example HPC failure mode we consider, is a binary-classification problem with variable misclassification costs. Based on some measurable properties of a system component, a classifier predicts either failure or non-failure. That is, we collapse what might well be a complicated probabilistic projection of failure, and a complicated decision process for taking action based on this projection, into an effective binary prediction representing the ultimate decision: Should action be taken in the expectation of a failure of this component? Whether this process is effective can then be assessed by attempting to estimate, from historical data, the costs incurred by following the predictions of a given classifier. These costs generally vary from one historical event to another (e.g., based on job scheduling), and can be quite difficult to estimate because of the complex temporal features of failure prediction and response.

A more reliably definable but somewhat less realistic measure of the effectiveness of failure prediction is based on conventional classifier metrics such as the confusion matrix and ROC curve. These involve the assignment of "actual" binary values to prediction test events, to represent the corresponding historical outcome ("failure" or "non-failure"), and the counting of true and false positives and negatives. The confusion matrix is simply a table of these counts for a particular instance of a classifier. Because failures can generally occur at any time, it is not obvious under what conditions to declare a binary prediction successful.

The conventional classification framework assumes a series of discrete and independent prediction tasks, each of which has an objective "actual" value that determines the cost of making each possible prediction. In the special case of fixed costs (a constant cost per false positive and another constant cost per false negative), the ROC curve translates directly into cost information. This curve summarizes the performance of a family of classifiers that are optimized for a range of possible cost tradeoffs by adjusting a threshold.

3.3 Challenges

A general complication lies in the assignment of useful "actual" values and misclassification costs for failure prediction. One simple approach would be to treat an alarm issued within a specific time window before a corresponding failure as a true positive and any other alarm as a false positive; the absence of an alarm in the window before a recorded failure would be a false negative. This provides at best a rough approximation to the dependence of cost on timing of alarms.

It may be more appropriate to consider the last relevant alarm (if any) issued before a failure as a true positive, and to assign it a cost proportional to the time interval between the alarm and the failure (in addition to any intrinsic cost of the response action itself). For example, if the response to the alarm is checkpointing of a program to guard against a possible upcoming failure, the interval between the alarm and the failure represents the computing time lost due to the true-positive event (in addition to the time spent in checkpointing). This checkpointing example has the simplification that the responses can be assumed not to affect the occurrence of failures, only their cost. Optimization of cost will lead to the standard strategy of checkpointing at regular intervals based on the currently estimated MTTF.

Our OOM study manifests another example of temporal and causal relations in failure prediction, with the property that the response (rebooting) is believed to alter the occurrence of failures. The simplifying features of this example allow a reasonable initial evaluation of predictivity and costs for the failure mode of interest, but various assumptions are still necessary, as will be discussed. The concrete application of our concepts to this example will offer guidance for further studies of other failure predictors.

4 Example of methodology

4.1 Failure mode

In previous work [5], we discovered a detectable precursor of a major failure mode on Sandia's Glory cluster, a 288-node, 4068-core Opteron cluster with an Infiniband interconnect. This precursor can be detected by automated means on timescales that can enable meaningful response. Here we summarize the relevant features of that work.

It was known that one of the major causes of application failure was the system running out of memory. We discovered that not only were some users' processes and/or allocated memory sometimes not being properly cleaned up upon job completion/exit, but that even in the absence of user processes the operating system would sometimes report abnormally high amounts of active memory. Active memory utilization during subsequent jobs is then that of the current job any lingering processes, memory used by the operating system, and memory the operating system believes is allocated. This combination can result in insufficient free memory available on the system and, consequently, system invocation of the OOM killer [1]. Job failure can then ultimately occur, either through direct killing of the job's processes or through killing of significant system processes.

This situation is shown in Fig. 1. Active memory (scaled to the total system memory) versus time for a particular node of the Glory cluster is shown in blue. No COMPLETED jobs occur in Fig. 1; CANCELLED or TIME(d)OUT jobs are shaded in alternating tones of yellow; jobs ending in the FAILED or NODE FAIL state are shaded in red; idle times are unshaded (white). Out-of-memory events are marked with red x's. Note that the vertical coordinates of the red x's are immaterial; the x's are placed on the active memory curve to facilitate relating events in time. OOM events occur during the idle times (unshaded times) and include the killing of user processes related to previous jobs.

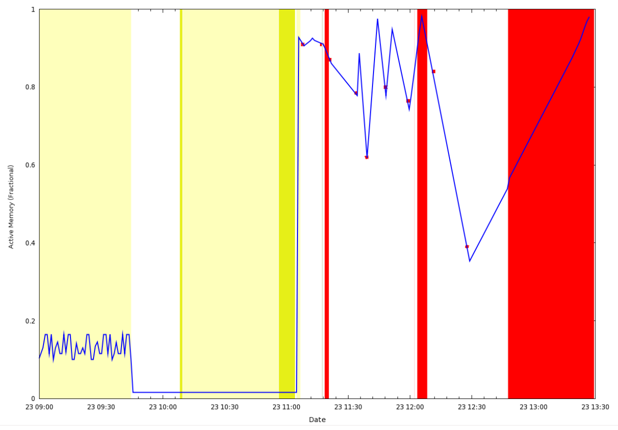

Figure 1. Active memory (scaled to the total system memory) vs. time for a particular node of the Glory cluster is shown in blue. Shaded times reflect job state as described in the text. Red x's indicate the times of OOM events.

We determined that nodes with anomalous values of active memory during idle time could be correlated with eventual OOM conditions resulting in failure scenarios. We proposed that such analysis, performed after job completion, could be used for determining subsequent job allocations and for invoking methods to resolve the problem.

4.2 Failure definition and attribution

As described in Section 3, the definition of failure and the attribution of the cause of failure are non-trivial. In this subsection, we discuss the availability of data and how it motivated the definitions and attributions used in this case study—illustrating an approach to such necessary tasks in the general attempt to quantify the effectiveness of failure prediction and response.

In this study, we collected data during a 16-day period between Feb. 12 and Feb. 28, 2009. We took data values for active memory at 60-second intervals continuously during this time period, except when it was rendered impossible by events such as the killing of our data-collection process by the OOM killer. Data were collected for 4178 jobs, including an ending state for each as reported by the scheduler. FAILED (562 jobs) and NODE FAIL (18 jobs), while clearly indicative of failure, are not necessarily accompanied by sufficient information from the scheduler or the logs to determine the cause of failure. Jobs intentionally cancelled result in the CANCELLED state (154 jobs). Jobs can be submitted with an estimated run time which, when exceeded, results in the TIMEOUT state (177 jobs). Finally, COMPLETED (3267 jobs) indicates that the termination was not otherwise abnormal.

Several effects are observed in our data that illustrate complexities of definition and attribution. First, high values of active memory during an idle time are often followed by high values into subsequent jobs. As a result, clearing the active memory condition before one job may improve the performance or even enable the completion of subsequent jobs. This is further understood in the context of Fig. 1, where lingering user processes from previous jobs claim memory on subsequent jobs and, most drastically, may result in invocation of the OOM killer, which may choose to kill a subsequent job or required system process outright. Moreover, possibly relatedly, after the onset of a high active memory condition, subsequent jobs may not end in FAILED or NODE FAIL conditions, but they frequently end up with CANCELLED or TIMEOUT states. Such terminations may be due to lack of sufficient progress on the job.

In support of the simplifying features discussed in Section 3, we chose to make the following definitions and attributions for this work: (1) Jobs that end in the COMPLETED state will be taken as successful; all others will be characterized as failed. (2) If a job fails, the node in that job that has the highest active memory during the preceding idle time will get full attribution for the failure of the job. (3) If there is no active memory data for the idle time immediately preceding the job on a given node, then data during the nearest preceding idle time with data will be used. (4) We treat all failures as OOM failures. That is, any failed job can be saved by intervention on the node to which the failure is attributed based on the memory condition as described in the above items.

The final item is not established by the data as presented. Failure modes other than OOM failures can of course occur. Some of the impact of this choice will be manifested in the results discussed in Section 4.4.

4.3 Applying prediction and response

In applying and evaluating prediction and response, we must consider all periods during which a response could be invoked and the potential costs and benefits of invoking that response during that period. Thus, rather than approaching the problem from a per-job standpoint, we must approach it from a per-actionable-period standpoint. In our example, given our attributions in Subsection 4.2, an actionable period is any idle time on a per-node basis preceding a job or jobs for which we have data. For simplicity, we will refer to these as idle periods, since idle periods during which we have no data are no longer in consideration. Invoking a response due to high active memory during an idle time on a given node can potentially result in a benefit only if that idle time on that node has been assigned responsibility for the fate of a subsequent job (had the highest active memory preceding the job) and if the job failed.

This has the result of transforming job-centric data into three cases of idle periods: (I) idle periods that have been assigned responsibility for the fate of a job that failed (479 periods in our 16-day sample); (II) idle periods that have been assigned responsibility for the fate of a job but for which no job has failed (540 periods); and (III) idle periods that have not been assigned responsibility for a job (2036 periods). Only for Case I would taking action result in a benefit.

4.4 Quantification

In Section 4.1, we proposed that statistical outliers in active memory can be used as predictors of potential failure. The observed distribution of active memory values during idle times is skewed and sharply peaked around low values of active memory. The mean is 0.06 and the standard deviation is 0.10.

The general OVIS approach to failure prediction [2–7] is based on detecting outliers (uncommon values) in observed data, with respect to a learned statistical distribution. The standard OVIS univariate analysis generates alarms for values that are either unusually low or unusually high. But our understanding of the HPC system indicates that only high, not low, values of active memory are OOM predictors.

For this work, we are choosing rebooting as the hypothetical response to a prediction. Rebooting a node successfully clears the memory and takes approximately 90 seconds. For the cost tradeoff in this work, then, if we choose to reboot during an idle time based upon the value of active memory, we consider that there is *no* cost if the duration of that idle period is greater than 90 seconds. If the duration of the idle period is less than 90 seconds, then the cost is the additional time that would have been lost in the reboot.

Figure 2 measures the accuracy of failure prediction via the tradeoff between never predicting failure (very high threshold, lower-left) and always predicting failure (very low threshold, upper-right). Such a ROC curve is generated by plotting the true positive rate (vertical axis: fraction of failure-attributable idle periods, Case I in Section 4.3, that were correctly alarmed) versus the false positive rate (horizontal axis: fraction of non-failure-attributable idle periods, Case II, that were incorrectly alarmed) as the classifier threshold (active memory threshold) is varied. If this analysis were done for all idle periods on all nodes (Cases I, II, and III), there would be a bias due to the attribution of job failure responsibility to highest-usage nodes. Thus the plot is limited to the 1019 idle periods in Cases I and II.

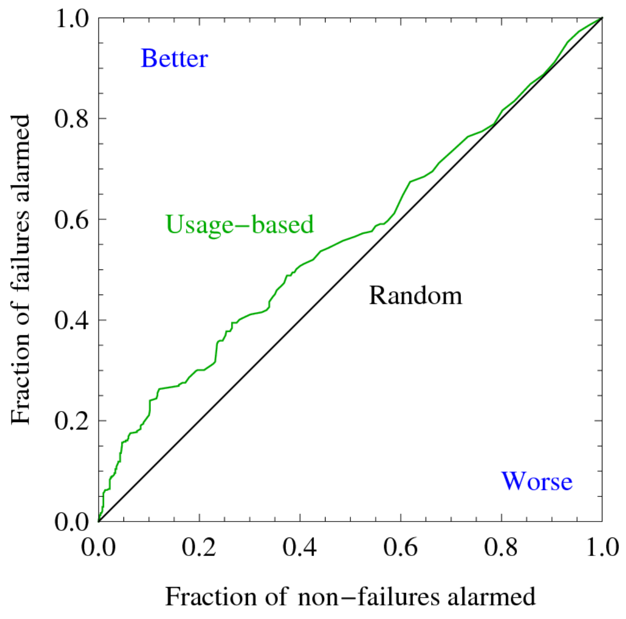

Figure 2. ROC curve (green) for failure prediction based on active memory during responsible idle periods (Cases I and II), compared to expected chance performance (black). The area under the green ROC curve is 0.562.

If there were no relation between usage and failure, we would obtain (on average) a straight diagonal line on the plot, because alarms would occur equally often for failures and non-failures. Our curve shows better-than-chance performance. Statistical significance was checked by creating 3000 randomly generated ROC curves from synthetic data with randomly distributed active memory values. The area under the ROC curve, a common summary measure of classifier performance, averages 0.5 for chance performance, and in only 2 of the 3000 synthetic cases did it exceed the value found for our data, 0.562. This indicates that our result would have about 1 chance in 1000 of occurring spuriously if there were no predictivity. For this case study, we have thus obtained clear validation that we can predict failure at a better-than-chance level using a single predictor. The modest level of accuracy is expected because many failures may be occurring for reasons unrelated to OOM.

Figure 3 measures the consequences of acting on the failure predictions by rebooting during idle periods. Here it is appropriate to include all idle periods with data (Cases I, II, and III), because it would not be known in advance which nodes have highest usage among the groups destined to receive certain jobs. Much as in the ROC curve, the threshold is swept to identify the tradeoff between no rebooting (left side) and routine rebooting (right side). The plot is made under the nominal assumption that *every* job failure could have been prevented by rebooting the node with highest usage during the preceding idle period, and that each failed job was a total loss, with all associated CPU-hours wasted. More realistic estimates could be approximated by scaling the vertical axis down by a suitable factor.

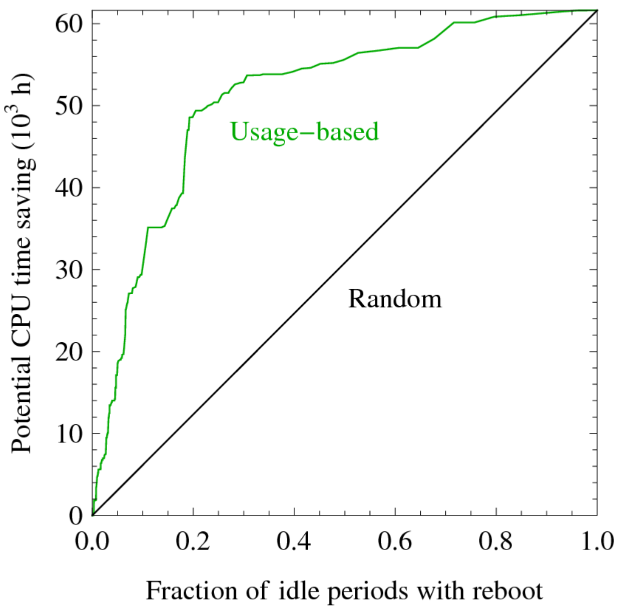

Figure 3. Net CPU time benefit from rebooting during idle times exceeding an active memory threshold, under assumptions in the text.

The monotonically rising green curve in Fig. 3 appears to indicate that routine rebooting (i.e., a very low threshold that effectively makes no use of the predictor) is optimal. But this does not negate the value of the predictive method, which is shown by the strong bend of the green curve (subject to the caveats of our assumptions). A threshold that results in rebooting 20% of the time would achieve about 80% of the potential benefits; this occurs for a scaled active memory threshold of 0.09. The addition of significant costs of rebooting not captured in our estimates would effectively shear the right-hand part of the plot downward. In that case, it can be seen that a portion of the green curve might well remain positive (a moderate amount of intelligent rebooting providing a net benefit) even if the black line (random rebooting) did not.

5 Conclusions

We have presented general methodologies for evaluating the effectiveness of failure prediction, and illustrated them with an example of an actual failure predictor (abnormally high active memory on a compute node) in a production system. We have shown evidence of its validity using a ROC curve. Additionally, we have quantified the cost savings (in CPU-hours) of using this information to take a particular action (rebooting the node during idle time), where we assigned negligible cost to the action but discussed how a known cost would affect the decision whether to take that action.

6 Future work

Our present initial quantification of HPC failure prediction is based on binary-classification theory. Although this framework relates failure prediction to a well-studied theoretical setting, further development of effectiveness metrics will benefit from a more complex and realistic picture of failure prediction. In particular, in contrast to the independent samples assumed in the classification setting, failure prediction is a process unfolding in time, where actions taken in response to an alarm can alter subsequent events. This raises the fundamental difficulty that the results of using a prediction-response mechanism cannot be directly evaluated from historical data in which the mechanism was not in use. To move beyond the simplifying assumptions in this work, such an evaluation could be based on real-world trials or on detailed models of system dynamics.

7 Acknowledgments

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-AC04-94AL85000.

References

[1] OOM KILLER. `http://linux-mm.org/OOM_Killer`.

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

[3] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pébay, D. Thompson, and M. Wong. OVIS 2: A robust distributed architecture for scalable RAS. In *Proc. 22nd IEEE Int'l. Parallel & Distributed Processing Symposium (4th Workshop on System Management Techniques, Processes, and Services)*, 2008.

[4] 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. 8th IEEE Symposium on Cluster Computing and the Grid (2008 Workshop on Resiliency in High-Performance Computing)*, 2008.

[5] J. Brandt, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong. Methodologies for advance warning of compute cluster problems via statistical analysis: A case study. In *Proc. 18th ACM Int'l. Symp. on High Performance Dist. Comp. (2009 Workshop on Resiliency in High-Performance Computing)*, 2009.

[6] 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 Int'l Parallel & Distributed Processing Symposium (2nd Workshop on System Management Techniques, Processes, and Services)*, 2006.

[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*, 2005. Extended Abstract.

[8] S. Chakravorty and L. Kalé. A fault tolerance protocol with fast fault recovery. In *Proc. 21st Int'l Parallel and Distributed Processing Symposium*, 2007.

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

[10] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[11] G. Hamerly and C. Elkan. Bayesian approaches to failure prediction for disk drives. In *Proc. 18th Int'l Conference on Machine Learning*, pages 202–209. Morgan, Kaufmann, 2001.

[12] W. J. Krzanowski and D. J. Hand. *ROC Curves for Continuous Data*. CRC Press, 2009.

[13] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. Scott. Proactive fault tolerance for HPC with Xen virtualization. In *Proc. ACM Int'l Conference on Supercomputing*, 2007.

[14] M. S. Pepe. *The Statistical Evaluation of Medical Tests for Classification and Prediction*. Oxford University Press, 2004.

[15] N. Raju, Y. L. Gottumukkala, C. B. Leangsuksun, R. Nassar, and S. Scott. Reliability analysis in HPC clusters. In *Proc. High Availability and Performance Computing Workshop 2006*, 2006.

[16] J. Stearley and A. J. Oliner. Bad words: Finding faults in Spirit's syslogs. In *Proc. 8th IEEE Symposium on Cluster Computing and the Grid (2008 Workshop on Resiliency in High-Performance Computing)*, 2008.

[17] M. Vieira, H. Madeira, I. Irrera, and M. Malek. Fault injection for failure prediction methods validation. In *Proc. 40th IEEE/IFIP Int'l Conference on Dependable Systems and Networks (5th Workshop on Hot Topics in System Dependability)*, 2009.