Near-Real-time Availability Monitoring and Modeling for HPC/HEC runtime systems

Hertong Song[1], Chokchai Box Leangsuksun[1], Narasimha Raju Gottumukkala[1], Raja Nassar[1], Stephen L. Scott[2], Andy Yoo[3]

[1]*eXtreme Computing Research Group*
College of Engineering & Science
Louisiana Tech University
Ruston, LA 71270, USA
{hso001, box, nrg003,nassar}@latech.edu

[2]*Network and Cluster Computing Group*
Oak Ridge National laboratory
Oak Ridge, TN 37830, USA

[3]*Center for Applied Scientific Computing*
Lawrence Livermore National Laboratory
Livermore, CA 94551, USA
ayoo@llnl.gov

Abstract

In this paper, we discuss a reliability-aware monitoring and modeling framework which provides near real-time system availability/reliability analysis and information for High Performance Computing / High End Computing (HPC/HEC) runtime systems. Our work aims to address issues in existing solutions in which HPC/HEC system management only considers performance aspects and leaves reliability to a reactive (i.e. addressing issues after they happen) or manual recovery approach. Our proposed framework dynamically obtains availability information such as failure and repair events of the individual nodes and is able to model and evaluate system availability for the overall and partial HPC system. With near-real-time availability evaluation, the framework enables runtime systems such as schedulers or resource managers to be aware of more accurate system reliability and hence better utilization and efficiency of the HPC systems. Lastly, we demonstrate usefulness of our approach to a scheduling runtime system based on the availability information provided by this framework. The failure and analysis model was reconstructed from system logs of the Lawrence Livermore National Laboratory Advanced Simulation and Computing (ASC) machines. The data set was used to understand system availability and validate how a scheduler can exploit such information to improve the overall completion time for parallel jobs in the presence of failures.

1. Introduction

High Performance Computing [1][2], especially in cluster form-factor, has become increasingly popular and is a significant scientific tool of choice. High availability (HA) on the other hand has helped business and mission critical environment in providing continuous services. Together, the combination of HA and HPC will clearly lead to even more benefits for critical computing infrastructure such as shared major HEC resource environments. In production environments, once a system is in operation, failures can happen anytime and may result in a disruption of services (service unavailability). Thus, the system must be monitored, especially for events of interest that may lead to outages. Any failure or recovery events must also be detected and recorded. Moreover, a system's availability should be reevaluated to reflect the actual outages and recovery events. Historical failure records may also help to identify system weakness, to analyze and locate the root cause of system (un)availability, which in turn helps improve the availability and performance aspects of the computing system.

In our research, we are interested in analysis of the mission-critical HPC system's Reliability Availability and Serviceability (RAS). Our studies focus on how reliability/availability will help improve overall performance and better HPC system utilization. Furthermore, we propose a reliability-aware monitoring and modeling framework that provides near real-time system availability/reliability analysis and information service for HPC/HEC runtime systems. The framework can also be extended to include other important reliability techniques such as detailed error classification, failure correlation, etc. In addition, the monitoring framework provides a platform for enabling the reliability/availability-aware runtime systems such as the scheduling/queuing or the check-pointing/restart mechanism in order to unleash HPC system closer to its maximum power.

The remainder of this paper is organized as follows: section 2 lays out the background and related work, section 3 presents the monitoring framework, section 4 describes the measure based on the event log from Lawrence Livermore

[1] The work of Song, Leangsuksun, and Gottumukkala is supported by DOE *fast*OS program, Grant # DE-FG02-04ER4614.
[2] Scott's research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.
[3] The work of Yoo was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48. Document UCRL-JC-147996.

Lab, section 5 demonstrates how the availability information acquired from this framework actually affects the job scheduling, and section 6 provides for the discussion of future work and conclusions.

2. Background

From desktop to supercomputer, availability and reliability are key attributes of any modern day computing system. Availability A_s, is the probability of system that is operated correctly during a period of time, and is measured by [9]:

$$A_s = \frac{Total\ Up\ Time}{Total\ Elapsed\ Time} \quad (1)$$

Reliability $R(t)$, is the probability the system is functioning at time t [9]. If we let X be the random variable representing the lifetime of a system, then the system reliability at time t can be depicted as [9]:

$$R(t) = \Pr(X > t) \quad (2)$$

Reliability and availability are terms that have often used as system fault tolerance attributes. However, they have slightly different meanings, namely, availability includes both reliability and repair time. In this paper, we interchangeably refer to these two terms.

In analytical models, failure and repair behaviors are assumed to be exponentially distributed, because exponential distribution provides concise mathematical expressions making the analysis easier. Mean time to fail (MTTF) and mean time to repair (MTTR) are two basic parameters required to evaluate a system's availability, and are heuristically taken by the analytical modeler based on historical data from other similar components or systems. On the other hand, system event logs can provide an insight and effective means of identifying defects which may lead to techniques for improving availability. The system monitoring mechanisms were originally developed to meet the needs to perform post-mortem analysis and corrective/preventive actions to better manage systems. These mechanisms have been evolved in time to meet the other needs. System logs are the typical records of historical events rather than being the result of a predefined plan. Event logs have been used in many ways, including long term trend analysis, online diagnosis for failure prediction and MTTF estimation. The log analysis process is highly dependent upon the quality and completeness of the event logs. If the information is incomplete or missing, it would be difficult or sometimes becomes impossible to interpret the event activities.

There is a wide variety of research on the analysis of event logs. Lin et al [3] analyzed the error log file on file servers to demonstrate that the log was composed of at least transient and intermittent processes. Wein and Sathaye [4] presented their experience with validation of complex computer system availability models. Ram et al [5] measured the failure rate in widely distributed software. Chillarge et al [6] presented a failure rate measurement technique on distributed software, based on classifying failure data into "failure windows". Moran et al [7] illustrated the availability monitoring facility developed at Digital Equipment International. These approaches are similar in performing data analyses; the differences in these approaches are the ways they classify errors, the correlation, distribution, and aims at different models.

The uniqueness of our approach is that our framework automatically performs data analysis, availability modeling and stores results in a repository when there are failure/repair events detected to provide near-real time availability information and inventory of the HPC system. Existing approaches either perform the analysis manually or retrieve the data from the database (logs) periodically in order to generate the analysis report. We envision that a reliability-aware runtime system can exploit near-real time availability information to improve efficiency and thus provide for a better HPC resource utilization.

In our near real-time availability analysis and repository (see figure 1), the availability inventory normally starts off with a default MTTF and MTTR value either by a manual calculation or taking directly from the equipment vendor. However, as times goes by, it may be discovered that actual failure and repair events are not accurately represented by the initial default values. Our technique considers the actual failure and repair events and normalizes current availability attributes with this actual information. We consider, given a set of numbers, representing default reliability information and near real-time dataset at time t is the sample space of an experiment, which is the set of all possible outcomes of that experiment [16][21]. Let $X_1, X_2, \cdots, X_n$ be the random variables from a random sample of size n, from some distribution, the sample mean $\overline{X}$ is given by:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \quad (3)$$

where the sample variance is given by:

$$S = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} X_i{}^2 - \frac{1}{n}\left(\sum_{i=1}^{n} X_i\right)^2\right] \quad (4)$$

It has been proved that $\overline{X}$ is the best estimator of the mean and S is the best estimator of the variance [20].

Goodness-of-Fit test. Let $N_i \quad i = 1,2,\cdots,k$ be the number of observations in the random sample, with the sample size $N = \sum_{i=1}^{k} N_i$, and let E_i be the expected value of type i. If the null hypothesis H_0 is true, and the sample size is large, then the distribution of the statistic $Q = \sum_{i=1}^{k}\left[\frac{(N_i - E_i)^2}{E_i}\right]$ will be approximately a χ^2 distribution with $k-1$ degree of freedom. It is desired to carry out the test at the significance level of α_0, the null hypothesis H_0 should be rejected if Q is in the $1-\alpha_0$ quantile of the χ^2 distribution with $k-1$ degree of freedom. The test is called the χ^2 test of goodness-of-fit test [20].

If the sample mean $\overline{X}$ is used to calculate the statistic Q, then the approximate distribution of Q when H_0 is true, and

lies between a χ^2 distribution with $k-2$ degrees of freedom, and this leads to the following formula.

$$\chi^2 = \sum_{i=1}^{k} \left[\frac{(N_i - E_i)^2}{E_i} \right] = \chi^2_{k-2,1-\alpha_0} \quad (5)$$

We apply the above technique to obtain $\overline{X}$, up-to-date MTTF and MTTR values.

3. The Reliability-aware Monitoring and Modeling Framework

The monitoring framework consists of two major parts, namely reliability-aware monitoring and system availability modeling and analysis. The reliability-aware monitoring is responsible for detecting the system's failure, performing recovery, and maintaining failure history. The system availability modeling module provides a near-real-time availability evaluation for both node-wise and overall system. Currently, we have constructed a proof-of-concept for each individual module. However, we plan to integrate our framework with the system availability information and configuration, and build an availability inventory and configuration database with normalization capability for the actual node-wise and system's MTTF and MTTR. Figure 1 shows the reliability-aware monitoring and modeling framework.

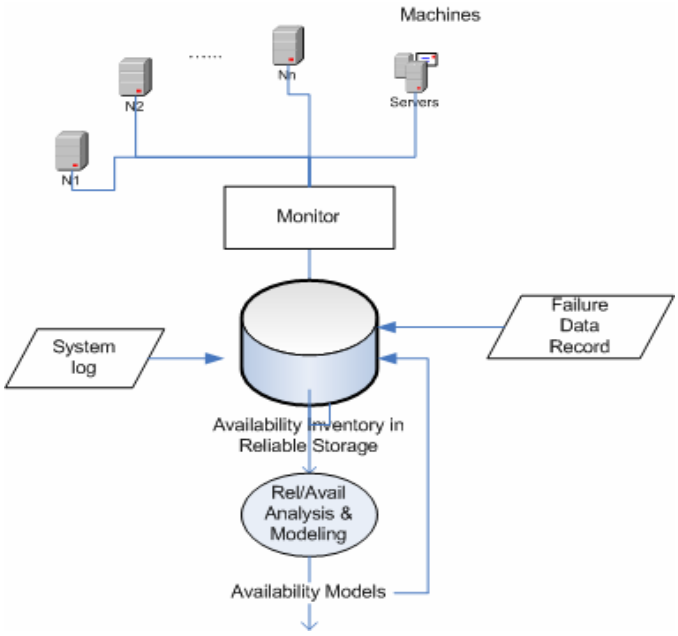

Figure 1 the reliability-aware monitoring and modeling framework.

In this framework, the monitoring facility is responsible for detections of failures and repair events, and recording these events into the system log. The failure data record (FDR) that is stripped from the system log file contains only the events that are necessary to evaluate a system's availability/reliability. The availability models are used to evaluate the system's availability which is stored in an XML file [18]. The system's log, failure data record and the availability model are stored in reliable disk storage. The Analysis & Solution modules are responsible for pulling the data from the FDR, performing the analysis and feeding the result into the availability models. The framework consists of three functionalities: (1) detection, which is responsible for detecting failure events based on the failure classification, (2) logging, which writes the failure events into the system log file and the system failure data record, and (3) analysis and update, which is responsible for failure analysis and normalization of the current system's availability.

Figure 2 shows the flow diagram inside the monitoring and analysis framework. Each arc in this diagram is associated with a number indicating the flow sequence, and a name denoting the action. The monitoring facility (MON) is responsible for watching system health. Once a failure or repair event is detected, MON writes this information into the system log, and invokes the availability update daemon (`AvailUpd`) to update the system's FDR record. After that, the `AvailUpd` invokes the analysis module, which queries the FDR to get the recent failure/repair activity, reevaluates the MTTF, MTTR and etc., and then updates both the mean time (MT) and the specification of the availability model with this new information. Finally, the `AvailUpd` invokes the solution module to solve this availability model. The result of availability solution is written back to the availability repository on a reliable storage.

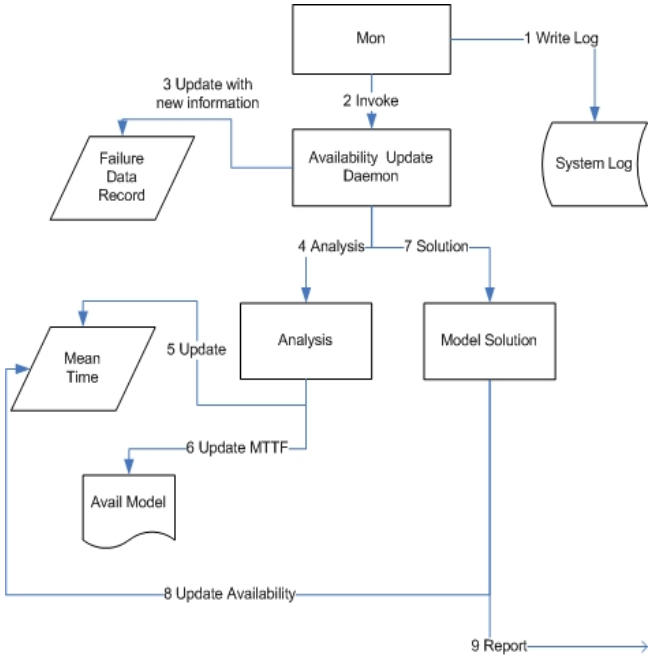

Figure 2 the reliability-aware runtime flow diagram

As mentioned earlier, the monitoring system maintains the system configuration and MTTF and MTTR information in the repository maintaining the availability property for each component. Figure 3 shows a snippet for a single instance in the availability and mean time records (MT). Each instance in the MT record has several fields: (1) the starting time of the

instance t_0, (2) the current time t_1, (3) the total elapsing time *T* in hours, which equals to $t_1 - t_0$, (3) total number of failures (*TF*) during this period of time, (4) the total downtime TDT which represents the total repair time, (5) the MTTF, which equals to *T/TF*, (6) the MTTR, which equals to *TDT*/*TF*, and (7) the steady state availability of the instance, which can be acquired from [9][14]. Among these fields, only t_1 and *TF* are recorded for each failure, and *TDT* is recorded for each repair. The rest of the fields are updated based on the changes of t_1, *TF* and *TDT*.

```
<SystemAvailability>0.999928</SystemAvailability>

<Servers>
    <Server 1>
        <t0>7/21/2000</t0>
        <t1>10/1/2004</t1/>
        <T>27960</T>
        <TF>9</TF>
        <TDT>1392.5</TDT>
        <MTTF>3106.27/MTTF>
        <MTTR>154.72</MTTR>
        <Availability>0.952</Availability>
    </Server 1>

        ……
</Servers>

<Nodes>
        ……
    </Node435>
        <t0>7/21/2000</t0>
        <t1>10/1/2004</t1/>
        <T>27960</T>
        <TF>33</TF>
        <TDT>1157</TDT>
        <MTTF>847.27/MTTF>
        <MTTR>35.06</MTTR>
        <Availability>0.952</Availability>
    </Node435>

    ……

</Nodes>
```

Figure 3 MT file for a single instance

Figure 4 shows a sample of system and availability model. The `AvailUpd` daemon first evaluates the node-wise and system-wise availability, and then updates the availability slots in the MT file. Once the availability is calculated, the `AvailUpd` evaluates the availability of the nodes and the entire system availability. For more details, please refer to [18]. The update facility performs the data analysis, and updates the MT file. The failure and repair events are assumed to be exponentially distributed, and computes the mean, variance and does the goodness-of-fit test. And finally, it generates a report and updates the system's availability.

```
    <System>

        <Servers>
            <Availability>CTMC(serverObjs.xml)</Availability>
        </Servers>

        <Nodes>
            <Availability>K-OUT-OF-N(K,N) </Availability>

        </Nodes>

    </System>
<Objects>
<Object Name="A1">
        <states>
                <state name="U"/>
                <state name="D"/>
        </states>
        <Good>
                <state name="U"/>
        </Good>
        <Status state="U"/>


        <Events>
                <Transition src="U" dst="D" rate="0.000321"/>
                <Transition src="D" dst="U" rate="0.006463"/>
        </Events>
</Object>


<Object Name="A2">
        <states>
                <state name="U"/>
                <state name="D"/>
        </states>
        <Good>
                <state name="U"/>
        </Good>
        <Status state="U"/>


        <Events>
                <Transition src="U" dst="D" rate="0.000357"/>
                <Transition src="D" dst="U" rate="0.00766"/>
        </Events>
</Object>

</Objects>
```

Figure 4 Service availability model

4. Evaluation and Analysis

We analyzed system logs of major HPC computing infrastructure from Lawrence Livermore National Laboratory. The system log file contains significant system events, from years past, collected from four ASC machines, namely White, Frost, Ice and Snow. We then performed a detailed analysis on these data sets. For the purpose of brevity, we present only the analysis result of White. White, the largest of among aforementioned systems, is a 512-node, 16-way symmetric multiprocessor (SMP) parallel computer. All nodes are of IBM's RS/6000 POWER3 symmetric multiprocessor 64-bit architecture. Each node is a stand-alone

Id	Type	Subtype	Wk-endng	TDT (hr)	Sect	Host list
1914	HW	HW-SSA_ADAPTER	7/28/2000	2	whit	265
1917	HW	HW-IO	7/28/2000	33	whit	275
1931	HW	HW-SWITCH	7/28/2000	72	whit	275
1913	HW	HW-SSA_ADAPTER	7/28/2000	2	whit	287
1968	HW	HW-SWITCH	8/4/2000	137	whit	017
1952	HW	HW-CPU	8/4/2000	19	whit	025
1938	HW	HW-CPU	8/4/2000	23	whit	026
1953	HW	HW-MEMORY	8/4/2000	21	whit	067
1954	HW	HW-MEMORY	8/4/2000	20	whit	100
1949	HW	HW-MEMORY	8/4/2000	21	whit	266
1986	HW	HW-CPU	8/11/2000	74	whit	010
1983	HW	HW-MOTHERBOARD	8/11/2000	76	whit	026
1969	HW	HW-OTHER	8/11/2000	48	whit	032
1970	HW	HW-CPU	8/11/2000	21	whit	052
1971	HW	HW-MOTHERBOARD	8/11/2000	20	whit	113
1985	HW	HW-IO	8/11/2000	74	whit	115
1980	SW	SW-COMM_SS	8/11/2000	172	whit	128
1972	HW	HW-CPU	8/11/2000	46	whit	194
1973	HW	HW-MEMORY	8/11/2000	48	whit	211
1974	HW	HW-MEMORY	8/11/2000	144	whit	241
2005	HW	HW-CPU	8/18/2000	144	whit	019
2002	HW	HW-IO	8/18/2000	188	whit	026

Table 1: Example of failure events in White

machine possessing its own memory, operating system (IBX AIX), local disk and 16 CPUs. Table 1 lists a sample of failure events in ASC White machine during the four year period, from 7/21/2000 to 10/1/2004. Table 1 also shows the failure id, the type and subtype of the failure, date of the failure discovery, total down time (TDT), the system (White) and the node affected by this event. Note that the TDT is the repair time for a given failure, which includes the response time, resolution time and the verification time. We analyzed the availability, MTTF and MTTR for each node in the system. The MTTF for the a node equals to (total elapsed time)/(number of failures). The average MTTF for each node in the system is approximately 7168.6 hours.

The MTTR is the (total down time)/(number of failures), which implies that it approximately needs this much time to recover from each failure event. The average MTTR for each node in the system is approximately 137 hours. The steady state availability for each node is 0.98. Figure 5 shows the availability density for each node in the White cluster system.

From Figure 5, we can see that the majority of the availability of the each node is above 0.95 with a few of them below 0.8. This indicates that, compared to others, some nodes manifest outages more. In fact, if the runtime systems are not aware of these nodes unreliability, it may result in low system total performance, extended application completion or failure. For the login nodes, the average MTTF is 1997.5 hours, and the average MTTR is 112.3 hours. Normally the login or service must be very reliable and have a quick repair time. Otherwise, it may lead to a single-point-of failure phenomenon, with a single failure taking down an entire machine or a rather large subset of a machine

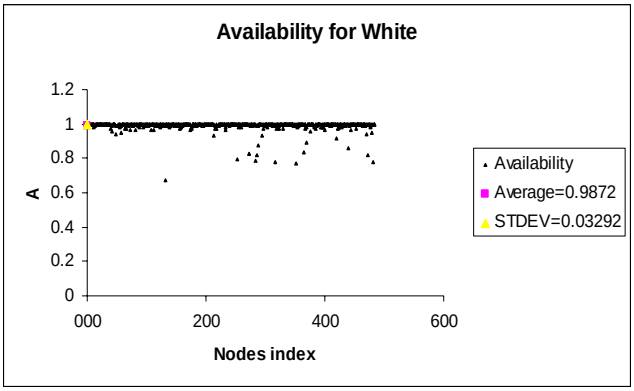

Figure 5 Availability density of each node in the ASC White

Figure 6 and Figure 7 show the mean time to failure and total down time (TDT) for each node in the cluster white, the means are 3293 and 355 hours, and the standard deviation is 1217 and 56, respectively. In Figure 6, we observe that the MTTF for each node varies significantly, namely, the

smallest MTTF is 230 hours, and the maximum is 5592 hours. In Figure 7, node downtime density indicates that the most of the total down time for each node is around 100 hours; some failure events cost more time due to a prolonged repair process, and thus increase the total average TDT.

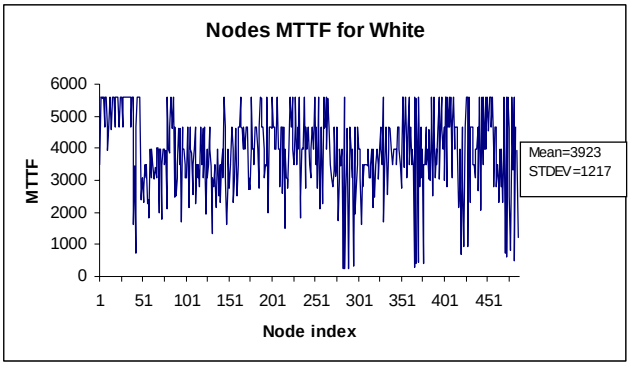

Figure 6 Node MTTF density (in hours)

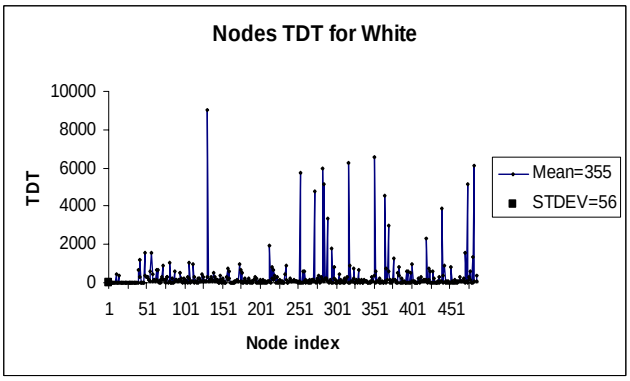

Figure 7 Node Downtime (in hours)

5. Reliability and Availability Aware Scheduling Algorithms

The monitoring framework provides up to date availability and reliability information for the overall system and also individual nodes. With this information, runtime services such as job scheduling can explore such information in order to improve an overall performance and system utilization. In this section, we present an experiment to demonstrate the affect of considering reliability and availability parameters in scheduling.

We have considered the system events of cluster White to reconstruct situations in order to validate an effectiveness of scheduling algorithms that use the availability information to schedule parallel jobs. The parameters MTTF, MTTR, and the elapsed time obtained from the information service are dynamically updated through the monitoring system. There are various important parameters that are significant in developing an effective scheduling algorithm such as job completion time, performance, throughput, utilization, reliability, safety, queuing time etc. Here we consider reliability as an important attribute for a scheduling algorithm and show how the job completion time is affected when choosing such an algorithm, since the reliability information can be acquired from the MTTF provided from the monitoring system.

Figure 8 illustrates a Gantt chart showing the effect of completion time for MPI jobs, in the presence of node failures. As the number of nodes increases, the probability that one or more of the nodes failing also increases, thus affecting the overall job completion time. In the case of parallel programs, the failure will impact the jobs running on all the systems. The MTTF for *n* nodes is given by the following equation:

$$MTTF(n,\lambda) = \frac{1}{n\lambda} \tag{6}$$

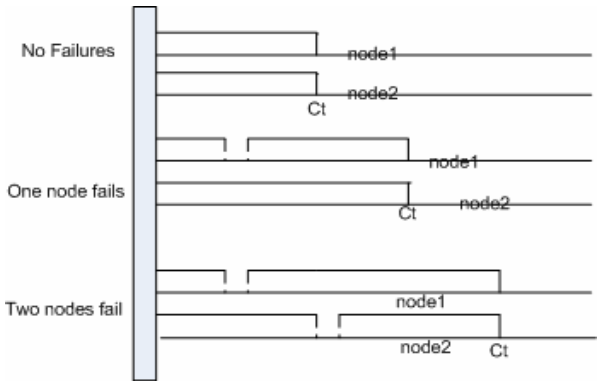

Figure 8 Completion time for a parallel job impacted by node failures

In a very large scale system, availability (or reliability) of the computing nodes becomes a very important factor in scheduling parallel jobs (such as MPI), because the job must be restarted and/or reallocated to a different set of nodes when failures occur. In this case, the completion time of the job will be affected in the event of failure, as shown in the Gantt chart above.

According to Amdahl's Law, the maximum speedup achievable is limited by the serial part of the program [22][19] The "speedup" of a parallel program is defined to be the ratio of the rate a job running on *N* processors to the rate at which the job is executed on one node. The speedup *S(N)* is given by

$$S(n) = \frac{1}{\frac{p}{n} + (1-p)} \tag{7}$$

where *p* is the fraction of code that can be made parallel (therefore, *1-p* is the code that has to be executed sequentially) and *n* is the number of nodes. The expected execution time on *n* nodes *T(n)* is given by

$$T(n) = \frac{T(1)}{S(n)} \tag{8}$$

In an MPI program the availability of n nodes participating in the program is given by

$$A(n) = A^n \quad (9)$$

where *n* is the number of computing nodes needed for the MPI program.

A job completion time can be described as

$$Ct = T(n) + T(f) \quad (10)$$

Where *T(n)* is the expected completion time and *T(f)* is the total time spent on the nodes that have failed to run the job. The fraction of code that is made parallel, *p*, is assumed to be 0.891 hereafter.

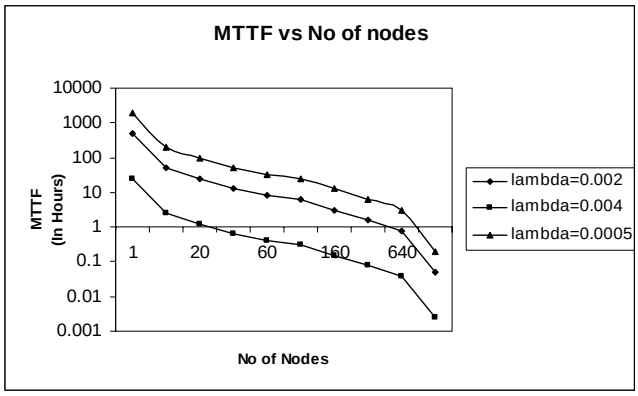

Figure 9: Numbers of nodes vs. system MTTF in the k of n reliability model (MPI program)

Node-wise MTTF and the number of cooperating nodes affect the total system availability and reliability as well as a completion time (see eq.6-10). For a given node-wise failure rate (λ), the total system MTTF decreases (i.e. the frequency of failure increases), as the number of nodes on which the job runs increases. At some points, scalability will approach a breakeven point where a long running job will not be able to finish due to its completion time being longer than the total system MTTF. Figure 9 shows how the MTTF varies with the increase in the number of nodes for three different failure rates. This phenomenon also elicits a conclusion that runtime systems, such as a scheduler aware of system reliability (MTTF), will benefit from MTTF information, especially more accurate MTTF information from our near real-time modeling approach.

6. Conclusion and Future Work

In this paper, we have presented the reliability-aware framework which is responsible for monitoring system health and automatically performs near-real-time system availability analysis. This framework not only is an important stepping stone to enable runtime systems to be aware of resource availability, but also ensures the more accurate result with a near real-time analysis approach, hence making better decisions in unleashing HPC/HEC power. We analyzed the actual data based on real-world production system logs from the Lawrence Livermore ASC machines and fed the analysis results to validate our approach. We have also demonstrated that it is possible to improve runtime performance via reliability/availability information.

Several improvements are considered to be included in the future work. The first enhancement is an event monitoring classification technique to further refine, identify and predict failure events. Currently the system's availability model is derived from crude model instances; for example, if the monitoring module does not receive response from a node, the node is considered as failed. This deficiency can be extended by monitoring and modeling more detailed events for each node, probing critical services or applications of interest, and to better estimates not only hardware but also application reliability/availability, as well as identifying the critical problems. Furthermore, other analysis techniques can be included, such as Weibull distribution [9] analysis, the Chapman-Smirnov test [16] among others, to study for more accurate models. Third, non-homogeneous Markov model and semi-Markov model [9][17] techniques should also be investigated to represent system behavior.

Since there are situations where a job will never complete due to relatively short MTTF, our future work will consider reliability-aware checkpoint/restart techniques that aim to derive an optimal interval to save application context based on the runtime system MTTF. Furthermore, we plan to investigate a checkpoint and restart time as one repair events and thus its time factor (penalty) will influent our total system availability model.

References

[1] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, and C. V. Packer. "Beowulf: A Parallel Workstation for Scientific Computation," Proceedings of International Conference on Parallel Processing, 1995.

[2] T. Naughton, S. L. Scott, Y. Fang, P. Pfeiffer, B. Ligneris and C. Leangsuksun, "The OSCAR Toolkit: Current and future developments," Dell Power Solutions, Nov. 2001.

[3] C. Leangsuksun, L. Shen, T. Liu, H. Song, and S. L. Scott, "Availability Prediction and Modeling of High Availability OSCAR Cluster," IEEE International Conference on Cluster Computing, December 1-4, 2003, Hong Kong.

[4] T. Y. Lin and D. P. Siewiorek, "Error log analysis: statistical modeling and Heuristic trend analysis," IEEE Trans. on Rel. vol. 39, no. 4, Oct. 1990, pp. 419-432.

[5] A. Wein and A. Sathaye, "Validating complex computer system availability models," IEEE Trans. on Rel., vol. 39, no. 4, Oct. 1990, pp.468-479.

[6] D. Tong and K. Iyer, "Dependability measurement and modeling of a multicomputer system," IEEE Trans. on Computers, vol. 42, no. 1, Jan, 1993, pp. 62-75.

[7] R. Chillarege, S. Biyani and J. Rosenthal, "Measurement of failure rate in widely distributed software," Proc. 25th International Symposium on Fault-Tolerant Computing (FTCS 25), Sendai, Japan, 1996.

[8] P. Moran, P. Gaffney, J. Melody, M. Condon and M. Hayden, "System Availability Monitoring," IEEE Transactions on Reliability, vol. 39, no. 4, Oct. 1990, pp. 480-485.

[9] M. Lanus, L. Yin and K. S. Trivedi, "Hierarchical Composition and Aggregation of State-Based Availability and Performability Models," IEEE Trans. on Reliability, vol. 52, no.1, March, 2003, pp. 44-52.

[10] Kishor S. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications. John Wiley & Sons, Inc. New York, 2002.

[11] J. Muppala, M. Malhotra, and K. S. Trivedi, "Markov Dependability Models of Complex Systems: Analysis Techniques," Reliability and Maintenance of Complex Systems, S. Ozekici (ed.), pp. 442-486, Springer-Verlag, Berlin, 1996.

[12] O.C. Ibe, R.C Howe and K.S. Trivedi, "Approximate availability analysis of VAXcluster systems," IEEE Trans. on Reliability, vol. 38, no.1, April, 1989, pp. 146-152.

[13] J. T. Blake and K. S. Trivedi, "Reliability Analysis of Interconnection Networks Using Hierarchical Composition," IEEE Trans. on Reliability, vol. 38, no.1, pp. 111-119, April, 1989.

[14] R. A. Sahner and K. S. Trivedi, "A hierarchical, combinatorial-Markov model of solving complex reliability models," Proceedings of 1986 ACM Fall joint computer conference, Dallas, Texas, United States, pp. 817 – 825.

[15] S. V. Amari, H. Pham, and Glenn Dill, "Optimal Design of k-out-of-n:G Subsystems Subjected to Imperfect Fault-Coverage", IEEE Trans on Reliability, vol. 53, no. 4, December 2004.

[16] M. H. DeGroot and M. J. Schervish, Probability and statistics, 3rd edition, Addition-Wesley, 2002.

[17] Pierre Bremaud, Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues. Springer, 1999.

[18] H. Song, C. Leangsuksun, R. Nassar, and Y. Liu "Availability Specification and Evaluation of HA-OSCAR Servers – An Object-Oriented Approach," Appearing in The 3rd International Conference on Computing, Communications and Control Technologies, July 24-27, Austin, Texas, USA. 2005

[19] Jeff Fier, "Origin™ 2001 and Onyx2® Performance Tuning and Optimization," Document Number 007-3430-001, Silicon Graphics.

[20] R. V. Hogg and A. T. Craig, Introduction to Mathematical Statistics, 4th Edition, Macmillan Publishing Co., Inc. 1978.

[21] J. L. Devore, Probability and statistics, 5th edition, Brook/Core, 1999.

[22] K., Hwang, 1993. Advanced Computer Architecture: Parallelism, Scalability, Programmability. McGraw-Hill, Inc., New York, NY.